INRACI

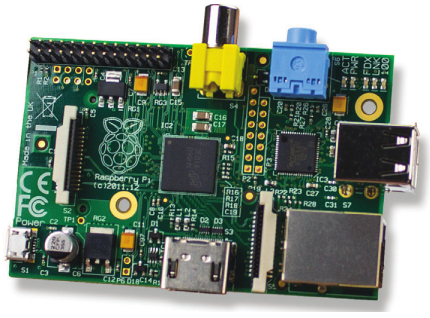
Informatique Embarquée

Syllabus de 6ème INFO
Premier Jet

python™

Notions de base, Tkinter & Raspberry

Tk



Edition : 2014

Professeurs : D. Mus, J. Pochet

Table des matières

1 La variable

1.1 La variable

Une variable a un nom et est destinée à contenir une information.
Une variable est associée à un type.

1.2 La déclaration de variable

Une variable est déclarée lorsqu'on précise son nom et qu'on lui donne une valeur (on affecte cette variable de la valeur). Le type sera automatiquement déduit de la valeur.
En python les variables sont dynamiquement typées (le type peut changer pour une même variable), cependant il est préférable de manipuler une variable de manière à ce qu'elle corresponde toujours au même type.

1.3 Le type de variable

On considérera trois catégories de types simples : entier, réel et booléen

1.4 L'affectation de la variable

Affecter une variable signifie lui donner une valeur.
En python le symbole d'affectation est « = ».
Une affectation va toujours de la droite vers la gauche, on met dans la variable à gauche la valeur qui est à droite.

Exemples de déclarations de variables

En python une déclaration de variable est : son nom suivi de « = » suivi de la valeur.

Exemples de déclarations de variables	Commentaires
a=1	On déclare une variable nommée « a » qui reçoit la valeur 1 → le type de a est déduit comme un entier
b=1.0	La variable nommée « b » est affectée de 1.0 (type réel)
c=True	La variable nommée « c » est affectée de True (type booléen)
somme=0	La variable « somme » a un nom évocateur

Même si en python la déclaration de variable n'est pas obligatoire, il est conseillé de prendre l'habitude de déclarer ses variables au début. Il faut que le nom de variable soit évocateur et ne contienne que des caractères valides (pas d'espaces, accents,…).
De plus, une variable doit toujours commencer par une lettre.

1.5 Incrémenter ou décrémenter une variable

Incrémenter une variable sous-entend augmenter cette variable d'une unité ; ce qui signifie « ajouter 1 à cette variable ».
Décrémenter une variable sous-entend diminuer cette variable d'une unité ; ce qui signifie « enlever 1 à cette variable ».

Comme on veut modifier une variable, on passe par l'affectation : « = » en python.
L'élément qui doit être modifié à gauche du signe « = » et la nouvelle valeur à droite.
Ainsi pour incrémenter une variable X on veut que celle-ci reçoive son ancienne valeur plus une unité. On écrira : X = X + 1

Exemple d'incrémentation et de décrémentation	Commentaires
nb1 = 8 nb2 = 5 nb1 = nb1 + 1 nb2 = nb2 - 1	Déclaration de nb1 Déclaration de nb2 Incrémentation de nb1 Décrémentation de nb2

Incrémentation et décrémentation d'un pas différent de 1

Lorsqu'on ne précise pas le pas, par défaut il s'agit de la valeur 1.
On peut cependant incrémenter (ou décrémenter) une variable en spécifiant un pas.

Exemple d'incrémentation et de décrémentation	Commentaires
nb1 = 8 nb2= 5 nb1= nb1 + 10 nb2 = nb2 - 2	Déclaration de nb1 Déclaration de nb2 Incrémentation de nb1 par un pas de 10 Décrémentation de nb2 par un pas de 2

2 L'instruction Ecrire et la fonction « print »

L'instruction Ecrire signifie que le programme renvoie au monde extérieur une information.
Le monde extérieur sera souvent représenté par l'utilisateur.
L'information renvoyée lors de l'instruction Ecrire est en général le contenu d'une variable.
Le plus souvent l'instruction Ecrire se fera par l'intermédiaire de la fonction « print ».

2.1 Afficher le type d'une variable

La fonction type() est une fonction préexistante qui renvoie une chaîne de caractère indiquant le type de son argument (ce qu'il y a entre les parenthèses).
Pour afficher le type d'une variable il faut d'abord avoir déclaré cette variable.

Exemple d'affichage du type d'une variable	Commentaires
cote_max=10 print type(cote_max)	Déclaration de l'entier « cote_max » Affichage du type de « cote_max »

2.2 Afficher du texte

Pour afficher du texte il suffit de mettre la chaîne de caractère entre " " ou entre ' ' et de le précéder du mot clé « print ».

Exemples d'affichage de texte	Commentaires
print "bonjour " print 'bonjour ' print "bonjour l'inraci "	On demande d'afficher la chaîne "bonjour " à l'écran Effet identique à la ligne précédente Une chaîne avec un ' sera plutôt entourée de " "

Dans certains cas on voudrait pouvoir afficher un texte qui comprend à la fois des « ' » et des « " ». On peut alors utiliser l'opérateur de concaténation « + » qui a pour effet de former une chaîne de caractère à partir de deux chaînes de caractères en les mettant bout à bout.

Exemples de concaténation	Commentaires
print "bonjour " + "l'inraci " print ' je dis : "bonjour ' + "l'inraci" +' ! "'	La chaîne "bonjour l'inraci " est formée puis affichée Sera affiché je dis : "bonjour l'inraci ! "

2.3 Afficher une variable

Pour afficher une variable il ne faut pas mettre les guillemets. Il faut simplement préciser le nom de la variable.

Exemple d'affichage de variable	Commentaires
cote_max=10 print cote_max	Déclaration d'un entier « cote_max » initialisé à 10 Affichage du contenu du moment de la variable « cote_max »

2.4 Affichage mixte

Par affichage mixte on entend un affichage à la fois de variable(s) et de chaîne(s) de caractères. Une façon simple de faire est de séparer les éléments par une virgule au sein de la fonction « print ».
Le texte doit toujours être entre " " ou entre ' ' mais pas les variables.
Attention les variables doivent avoir été préalablement déclarées.
La virgule sépare les éléments à afficher.

Exemple d'affichage mixte avec la virgule	Commentaires
cote_max=10 cote=9 print "Laurent a eu ", cote , "/ ", cote_max	Déclaration d'un entier « cote_max » initialisé à 10 Déclaration d'un entier « cote » initialisé à 9 Affichage mixte (« *Laurent a eu 9 / 10* »)

Une autre façon d'avoir un affichage mixte est d'utiliser l'opérateur de concaténation « + ».
Pour ce faire il faut que les éléments à concaténer soient tous les deux des chaînes de caractères.
Pas question de faire : print "ma cote : " + cote
Car la variable « cote » n'est pas une chaîne de caractères !

Ce qu'on peut par contre faire c'est utiliser la fonction str() qui va transformer le contenu de la variable en une chaîne de caractères (un string) correspondant à la valeur du moment de la variable.

Exemple d'affichage mixte avec la fonction str()	Commentaires
cote_max=10 cote=9 print "Laurent a eu " + str(cote) + " / " + str(cote_max)	Affichage mixte (« *Laurent a eu 9 / 10* »)

Remarques : Le séparateur « , » va ajouter un espace à l'affichage.
Avec la concaténation « + », l'espace doit être ajouté par l'utilisateur.

2.5 Utilisation de commentaire

Un commentaire est une explication que le programmeur ajoute qui ne sera pas prise en compte par le compilateur.
Sur une ligne, un commentaire est du texte placé après le caractère « # ».

Exemples de commentaires	Commentaires

#mini programme
nb1 = 8 #déclaration de nb1
nb1= nb1 + 1 #incrémentation de nb1
	Les commentaires rendent le code plus lisible.
	On peut placer un commentaire sur une ligne seule ou placer un commentaire pour détailler une ligne

3 L'instruction Lire

L'instruction Lire signifie que le programme, pendant son exécution, doit faire l'acquisition d'une donnée extérieure. En général la donnée extérieure sera fournie par l'utilisateur par l'intermédiaire du clavier.

3.1 Lire un nombre

Exemple de lecture et d'écriture	Commentaires
#somme de deux nombres	Commente l'énoncé de l'exercice
nb1 = 0 #déclaration de nb1	
nb2 = 0 #déclaration de nb2	
somme = 0 #déclaration de somme	
nb1= input("introduisez nb1 SVP: ") #Lire nb1	L'utilisateur voit le texte "introduisez nb1 SVP: " et le programme attend sa donnée pour l'affecter à nb1
nb2= input("introduisez nb2 SVP: ") #Lire nb2	L'utilisateur voit le texte "introduisez nb2 SVP: " et le programme attend sa donnée pour l'affecter à nb2
somme = nb1+nb2	La variable somme est affectée du résultat de l'opération
print "la somme de ",nb1, " et ",nb2, " = ",somme	L'affichage renvoie le résultat à l'utilisateur (Ecrire)

Pour lire un nombre nous avons utilisé la fonction « input ».

Cette fonction a pour but d'afficher sur écran un message destiné à l'utilisateur (afin qu'il sache la donnée à introduire) et à renvoyer cette donnée pour l'affecter à la variable qui doit la recevoir.
La transmission de donnée se fait grâce à la fonction « input » mais aussi grâce à l'affectation (« = » en python).

3.2 Lire une chaîne de caractères alphanumériques

Pour lire un caractère alphanumérique, on utilisera la fonction « raw_input ».

Exemple de lecture et d'écriture	Commentaires

#somme de deux nombres
nom=" "
#Lire nom
nom= raw_input("introduisez votre nom: ")
 	
#ecrire le contenu de la variable nom
print nom
	Commente l'énoncé de l'exercice
	
	L'utilisateur voit le texte "introduisez votre nom " et le programme attend sa donnée pour l'affecter à nom
	
	L'affichage renvoie le contenu de la variable nom à l'utilisateur (Ecrire)

4 Les opérateurs et l'affectation

4.1 Introduction

Les opérations sont formées à partir d'opérateurs. Les opérateurs arithmétiques sont :
+, -, *, / et permettent de constituer des opérations comme par exemple :
(nb1+nb2), (nb1-1), (10*nb2), (nb1/nb2),…
Les opérations sont constituées d'opérateurs mais aussi de variables et/ou constantes.

Exemple d'utilisation d'opérations	Commentaires

#tests sur les opérateurs arithmétiques
nb1 = 0
nb2 = 0
somme = 0
difference=0
produit =0
quotient=0
	Commente l'énoncé de l'exercice

nb1= input("introduisez nb1 SVP: ") #Lire nb1	Lire
nb2= input("introduisez nb2 SVP: ") #Lire nb2	Lire
somme = nb1+nb2	Affectation d'une opération
difference = nb1-nb2	Affectation d'une opération
produit = nb1*nb2	Affectation d'une opération
quotient = nb1/nb2	Affectation du quotient (entier)
print "les nombres sont ", nb1, " et ", nb2	Réaffichage des nombres lus
print "somme : ", somme, " et difference : ", difference	Ecrire somme et différence
print "produit : ", produit, " et quotient : ", quotient	Ecrire produit et quotient

La priorité des opérateurs est la priorité habituelle. On peut aussi ajouter des ( ).

Les opérations sont souvent liées à une affectation. On peut alors dire que le résultat de l'opération sera affecté à une variable.
Exemple : produit = nb1*nb2

Dans certains cas le résultat de l'opération n'est pas simplement affecté à une variable mais est utilisé par une fonction ; comme au sein de la fonction « print » par exemple.
Exemple : print "somme : ", nb1+nb2, " et difference : ", nb1-nb2

4.2 Particularité de l'opérateur « / »

L'opérateur « / » est particulier dans le sens où le résultat de l'opération n'est pas celui que l'on imagine naturellement.
Par exemple l'opération « 4 / 5 » fournit comme résultat « 0 » alors que si on divise 4 par 5 on doit trouver 0,8.
La raison de cette différence est que l'opérateur « / » va fournir un résultat de type entier si les tous les éléments intervenants au sein de l'opération sont de type entier.
Ici comme le résultat de la division est 0,8 et que la division entière ne tient pas compte de ce qu'il y a après la virgule, le résultat de la division « 4 / 5 » donne 0.

Comment obtenir la valeur exacte ?

Imaginons que l'on veuille calculer la moyenne de deux nombres entiers.
Comme le résultat est susceptible d'être une valeur réelle il faut trouver une astuce pour avoir le résultat exact. Une façon de faire est illustrée ci-dessous en multipliant un nombre par « 1.0 ». Cet artifice ne change rien à la valeur du nombre mais va indiquer à l'opérateur « / » qu'un élément est de type réel et donc que le résultat de l'opération devra être réel.

Exemple du calcul de la moyenne	Commentaires
#somme de deux nombres	Commente l'énoncé de l'exercice
nb1 = 0 #déclaration de nb1	
nb2 = 0 #déclaration de nb2	
moyenne = 0.0 #déclaration de moyenne	
nb1= input("introduisez nb1 SVP: ") #Lire nb1	L'utilisateur voit le texte "introduisez nb1 SVP: " et le programme attend sa donnée pour l'affecter à nb1
nb2= input("introduisez nb2 SVP: ") #Lire nb2	L'utilisateur voit le texte "introduisez nb2 SVP: " et le programme attend sa donnée pour l'affecter à nb2
moyenne = (nb1 + nb2 * 1.0) / 2	La moyenne sera le résultat d'une division réelle
print "la moyenne de ",nb1, " et ",nb2, " = ",moyenne	L'affichage renvoie le résultat à l'utilisateur (Ecrire)

4.3 L'opérateur modulo « % »

L'opérateur modulo est utilisé pour connaître le reste de la division entière.
Nous avions vu que pour la division entière, l'opération « 4 / 5 » fournit comme résultat « 0 » car « 4 » rentre 0 fois de manière entière dans « 5 ».

L'opérateur modulo va alors fournir le nombre entier qui correspond à ce qu'il reste à diviser lorsqu'on soustrait, du numérateur, le dénominateur autant de fois que le résultat de la division entière.
Dans notre exemple « 4 / 5 » la division entière est 0 et ce qu'il reste à diviser est 4.
Ainsi 4 MODULO 5 (noté 4%5) fournit comme résultat 4.

Autre exemple : 38%5 fournira comme résultat 3
En effet, la division entière « 38/5 » donne comme résultat 7 car 38 rentre 7 fois en entier dans 38. Ainsi, si l'on soustrait à 38 sept fois le dénominateur (autrement dit 35) il reste 3.

*4.4 Opérateur exposant « ** »*

L'opérateur exposant est composé des deux caractères « ** ».
Ces caractères ne peuvent être séparés d'un espace, ils doivent être collés.

Exemple : nb = 5
print nb ** 3 # 125 (= 5^3) sera affiché à l'écran

4.5 Attention aux types des opérandes

Les opérandes sont les paramètres d'une opération.
Par exemple dans l'opération « a + 1 » :
« a » et « 1 » sont les opérandes
« + » est l'opérateur
« a + 1 » est l'opération
L'opération correspondra à un résultat après l'exécution par la machine.

Il faut savoir qu'il est important de bien prendre en compte les types des opérandes.
En effet, le fonctionnement de l'opérateur sera différent suivant le type des opérandes ; un exemple vu précédemment est la particularité de l'opérateur « / ».
Pour rappel, avec l'opérateur « / », si toutes les opérandes sont des entiers le résultat est entier alors que si toutes les opérandes sont réelles le résultat est réel.

On peut également se demander ce qu'il se passera si les opérandes ont des types différents.
On a vu avec l'opérateur « / » que si nous avons un opérande entier et un opérande réel le résultat de l'opération sera réel mais ce n'est pas toujours aussi simple.

• Dans certains cas des opérandes différents provoquent une erreur.

Exemple, « 5 + ″bonjour″ » :
Il n'est pas possible d'additionner un nombre entier avec une chaîne de caractères.

• Dans certains cas des opérandes différents provoquent un comportement non désiré.

Exemple 1, « ″en effet″ + ″bonjour″ » :
On pourrait croire que l'opérateur « + » va tenter d'effectuer l'addition.
En réalité, avec des opérandes qui sont des chaînes de caractères (STRING), l'opérateur « + » effectue une concaténation.
La concaténation consiste à créer une chaîne de caractère composée d'autres chaînes de caractères.
Dans notre cas le résultat de l'opération « ″en effet″ + ″bonjour″ » donnera la chaîne de caractères : « ″en effetbonjour″ »

Exemple 2, « 5 * ″bonjour″ » :
On pourrait croire que l'opérateur « * » va tenter d'effectuer la multiplication.
En réalité, lorsqu'une des opérandes est une chaîne de caractères (STRING) et l'autre est un entier, l'opérateur « * » effectue une succession de concaténations.
Dans notre cas le résultat de l'opération « 5 * ″bonjour″ » donnera la chaîne de caractères : « ″ bonjourbonjourbonjourbonjourbonjour ″ »

Remarque : Si on souhaite concaténer un string et une constante, on peut transformer la constante en string avec la fonction str().
Exemple : print ″bonjour ″+ ″la ″ + str(5) + ″eme elec″

4.6 L'affectation

Des opérations telles que décrites ci-dessus ne sont jamais utilisées seules.
En effet une opération telle que « a + 1 » ne sera jamais utilisée seule car le résultat n'est pas exploité.
Une façon simple d'exploiter le résultat est de le transmettre à une variable par l'intermédiaire d'une affectation.
Le signe de l'affectation en python est le symbole « = ».
Une affectation consiste à placer ce qu'il y a à droite de son symbole au sein de l'élément qui se trouve à sa gauche.

```
Exemple :   a = a + 1      #on place dans la variable « a » le résultat de « a + 1 »
                           # la variable « a » voit alors sa valeur augmentée de 1
```

Il est alors classique d'avoir une ligne qui reprend à la fois une affectation et une ou plusieurs opérations.

Remarque : Le résultat d'une opération peut également être fournie à une fonction :
Exemples : res = math.fabs(a + b)
print cote1 + cote 2

5 La séquence

5.1 Les éléments de base dans une séquence

L'informatique permet de faire un traitement automatique des informations.
La séquence est une façon simple qui permet de montrer la solution à un problème informatique de manière séquentielle. L'ordre a donc de l'importance.

Pour pouvoir traiter une information, il faut en disposer. C'est l'instruction Lire qui va permettre de stocker cette information dans une variable. Classiquement il faut avoir préalablement déclaré les variables avant de pouvoir y stocker une information.
La séquence commencera alors typiquement par :

- La déclaration des variables
- La lecture des informations à traiter

Une fois toutes les informations nécessaires acquises, il faut les traiter.
Un traitement simple consiste à effectuer un ensemble d'opérations et de fournir le résultat à une variable.
Le traitement sera typiquement :

- Des opérations
- Une affectation

Une fois le traitement terminé le programme se doit de fournir le résultat au monde extérieur. La sortie des données après le traitement se fera via :

- L'écriture

En résumé les éléments de base d'une séquence sont :

- La déclaration des variables
- La lecture des informations à traiter
- Les opérations
- L'affectation
- L'écriture

5.2 La séquence en python

Exemple de séquence	Commentaires
moyenne = 0 nombre1 = 0 nombre2 = 0 nombre3 = 0	#Déclaration des variables #moyenne, nombre1, nombre2 et nombre3 # #
nombre1 = input("un nombre entier SVP : ") nombre2 = input("un nombre entier SVP : ") nombre3 = input("un nombre entier SVP : ")	#Lecture des trois nombres # #
moyenne = (nombre1 + nombre2 + nombre3) / 3	#traitement = opérations et affectation
print "la moyenne vaut : ", moyenne	# L'affichage renvoie le contenu de la variable moyenne à #l'utilisateur (Ecrire)

Le résultat risque d'être approximatif parce que le programme est conçu pour ne manipuler que des entiers. Même si le but du programme est de calculer la moyenne de trois nombres entiers, il faudrait prévoir une variable « moyenne » de la catégorie des réels et s'arranger lors de l'affectation de cette variable pour lui affecter la valeur réelle de la moyenne.

Exercice : Modifiez la séquence pour avoir un résultat réel pour la moyenne

Même si en python la déclaration de variables n'est pas obligatoire, nous recommandons vivement de déclarer toutes les variables en début de programme.
Et cela pour plusieurs raisons :

- Nous pourrons facilement transposer ce programme vers un autre langage où la déclaration est obligatoire.
- Nous pouvons facilement faire le rapprochement avec l'algorithme
- Le fait de visualiser les noms de variables (évocateurs bien sûr) nous apporte le contexte du programme et augmente la lisibilité.

5.3 La séquence en LDA

LDA signifie Langage de Description Algorithmique, il permet de mettre en évidence la structure du programme en restant indépendant du langage de programmation.
Le LDA est écrit en français, il utilise un certain nombre de mots clés qui doivent être soulignés. Il y a des mots clés tels que : Lire, Ecrire, Entiers, Réels… et des mots clés qui permettent de mettre en évidence les structures (voir plus tard).

• Pour déclarer une variable en LDA :

Il suffit d'écrire la catégorie de type suivi du nom de la variable.
Exemple de déclaration en LDA : Entier nombre1

Les trois catégories de types simples sont : Entier, Réel et Booléen

• Pour lire une donnée en LDA :

On écrit le mot clé « Lire » (en le soulignant) suivit de la (ou des) variable(s)
Exemple de lecture en LDA : Lire nombre1, nombre2, nombre3

Il est important que les variables aient été préalablement déclarées.

• Les opérations en LDA :

Les opérations sont constituées d'opérandes et d'opérateurs.
Les opérateurs les plus simples sont les opérateurs arithmétiques.
Les symboles en LDA sont simplement : + - * /
La division est considérée comme réelle.

• L'affectation en LDA :

L'affectation en LDA est symbolisée par une flèche dirigée vers la gauche. Ainsi on fait apparaître que c'est l'élément qui se trouve à gauche de la flèche qui se voit affectée du résultat de ce qui se trouve à droite.
Il est courant que l'affectation soit liée à une opération.

Exemple d'affectation en LDA : moyenne ← (nombre1 + nombre2 + nombre3) / 3

• Pour écrire une donnée en LDA :

On utilise le mot clé « Ecrire » (en le soulignant) suivit de la (ou des) variable(s)
Exemple d'écriture en LDA : Ecrire nombre1, nombre2, nombre3

Exemple de séquence en LDA	Commentaires
Entiers nombre1, nombre2, nombre3, moyenne	#Déclaration des variables #moyenne, nombre1, nombre2 et nombre3
Lire nombre1, nombre2, nombre3	#Lecture des trois nombres
moyenne ← (nombre1 + nombre2 + nombre3) / 3	#affectation et opération
Ecrire moyenne	# L'affichage renvoie le contenu de la variable moyenne au # monde extérieur (Ecrire)

Remarque : Le résultat risque d'être approximatif parce que le programme est conçu pour ne manipuler que des entiers.

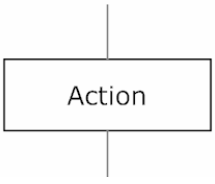
Exercice : ***Modifiez la séquence en LDA pour avoir un résultat réel pour la moyenne***

5.4 La séquence en ordinogramme

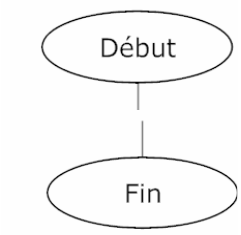
L'idée de l'ordinogramme est de faire apparaître, de manière visuelle, la structure du programme tout en restant indépendant du langage de programmation.

Pour une séquence pure (simple succession d'actions), l'ordinogramme ressemblera fort au LDA mais de manière visuelle.

Toutes les formes d'action sont représentées comme ici à droite par un rectangle.
Il peut s'agir d'une déclaration, d'une lecture, d'une écriture, d'une instruction ou d'un ensemble d'instructions.

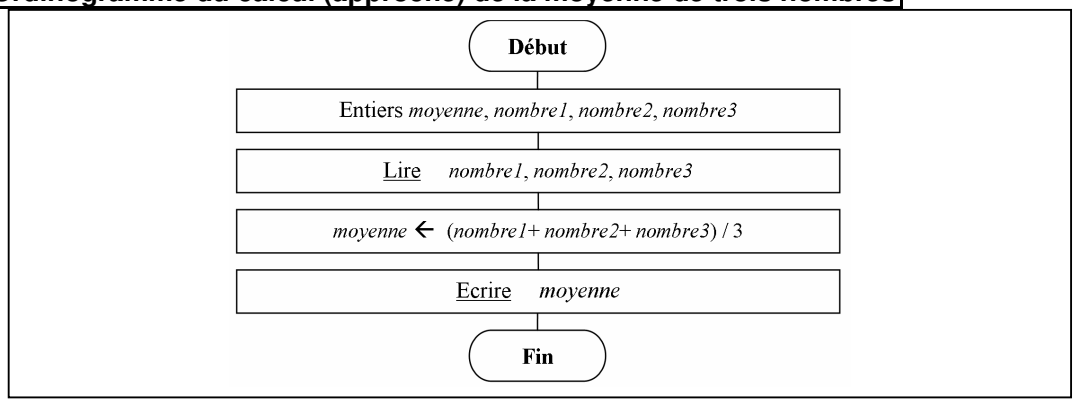


D'autre part, un ordinogramme commence toujours par l'entité



Et se termine toujours par

Ordinogramme du calcul (approché) de la moyenne de trois nombres



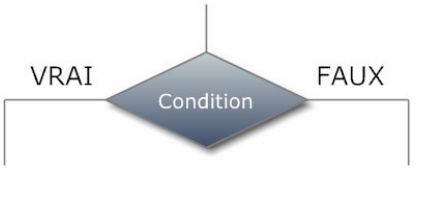
Remarque : Le résultat risque d'être approximatif parce que le programme est conçu pour ne manipuler que des entiers.

Exercice : ***Modifiez la séquence en ordinogramme pour avoir un résultat réel pour la moyenne***

Pour une séquence pure comme celle ci-dessus, l'ordinogramme n'apporte rien de plus que le LDA. En effet, les différentes actions sont notées de la même manière.
De plus, le LDA est plus concis ce qui le rend préférable à l'ordinogramme.

Cependant, il y a une autre entité de l'ordinogramme qui permet d'examiner une condition et poursuivre la suite du programme suivant le résultat vrai ou faux de la condition. Cette entité, un losange (avec une entrée et deux sorties), apporte un aspect visuel intéressant à l'ordinogramme que l'on ne retrouve pas dans le LDA.

Tous les tests sont représentés comme ici par un losange à l'intérieur duquel se trouve le prédicat (la condition) dont le résultat sera soit vrai soit faux selon que la condition est vérifiée ou non. Suivant le cas, la suite du programme s'exécutera soit dans un sens soit dans l'autre.



5.5 Exemple de séquence simple avec l'utilisation de délais

Exemple avec décompte

On peut faire un décompte en utilisant des délais et une décrémentation.
Ici notre exemple fonctionne mais devra être amélioré à l'aide d'une structure répétitive.

Exemple de décompte sans structure répétitive	Commentaires
import time nb_sec_rest=3 print "fini dans ", nb_sec_rest, " s" time.sleep(1) nb_sec_rest= nb_sec_rest – 1 print "fini dans ", nb_sec_rest, " s" time.sleep(1) nb_sec_rest= nb_sec_rest - 1 print "fini dans ", nb_sec_rest, " s" time.sleep(1) nb_sec_rest= nb_sec_rest - 1 print "FINI"	On répète trois fois : print "fini dans ", nb_sec_rest, " s" time.sleep(1) nb_sec_rest= nb_sec_rest – 1

En LDA ou en ordinogramme on se contentera d'écrire « Délai 1 seconde » pour symboliser l'action du délai d'une seconde.

6 La structure de choix simple

Lors de l'exécution du programme, il est possible d'examiner une condition afin d'exécuter un bloc d'actions que si cette condition est vraie.
Nous dirons qu'un bloc d'actions est simplement une suite successive de lignes.

Constituer une condition

Une condition est ce qui devra être examiné lors de l'exécution du programme afin de déterminer si elle est vraie ou fausse.
Après examen, une condition est donc un booléen (vrai ou faux).
Souvent une condition apparaît comme une opération dont le résultat est un booléen (vrai ou faux) ; Les opérateurs de comparaison le montrent bien.

Les opérateurs de comparaison en python sont :

< , <= , > , >= , == , !=

Exemples de conditions utilisant des opérateurs de comparaison :

(nb1<nb2), (nb1<=1), (10>nb2), (nb1>=nb2), (nb1==nb2), (nb1 !=nb2), …

Les conditions ci-dessus sont constituées d'opérateurs de comparaison mais aussi de variables et/ou de constantes. Les parenthèses ne sont en général pas nécessaires.
L'opérateur de comparaison « == » signifie «est la même chose que », à ne pas confondre avec l'opérateur d'affectation « = ».
L'opérateur de comparaison « != » signifie « est différent de ».

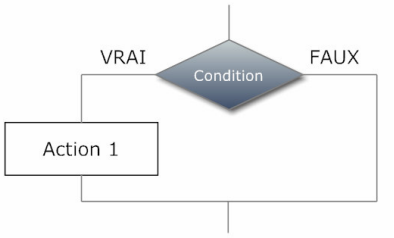
En LDA, tout comme pour les ordinogrammes, les opérateurs de comparaisons sont :
< , ≤, > , ≥ , = , ≠

6.1 La structure si-alors

Dans la mesure où il faut faire une action ou un bloc d'actions lorsqu'une condition est vérifiée, et qu'il ne faut rien faire lorsque cette même condition n'est pas vérifiée, on utilise la structure de choix simple de la forme « Si – Alors ».

Principe général :

Structure en ordinogramme :



En LDA

si condition alors
    Action 1

fsi

Syntaxe en python :

if condition :
    Action 1

Bien entendu le mot « condition » symbolise une condition (exemple : vitesse <30) et « Action 1 » symbolise une ou plusieurs lignes d'actions.

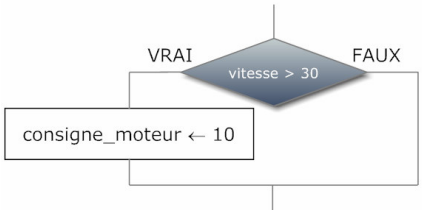
L'ordinogramme fait apparaître deux branches : une pour le vrai et une pour le faux.
Dans cette configuration « si-alors » on ne fait rien lorsque c'est faux.
La structure se termine après la recombinaison des deux branches.

Le LDA fait apparaître les mots clés « si », « alors » et « fsi ».
Ces mots clés doivent être soulignés ; le « fsi » indique la fin de la structure (la recombinaison des branches pour l'ordinogramme).

Le python utilise le « if », le « : » et l'indentation (décalage à droite).
Toutes les lignes d'actions liées à la branche « vrai » doivent être indentées (décalées vers la droite) par rapport à la position horizontale du « if ».
On considère que l'on est après la structure lorsque ce décalage disparaît.

Exemple :

Structure en ordinogramme :



En LDA

si vitesse > 30 alors
    consigne_moteur ← 10

fsi

Syntaxe en python :

if vitesse > 30 :
    consigne_moteur = 10

6.2 La structure si-alors-sinon

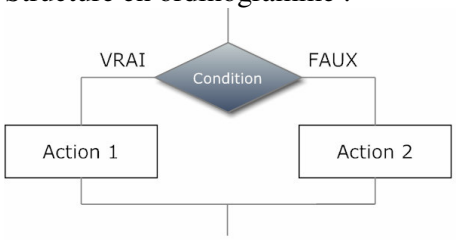
Nous avons vu que la structure de choix permet de préciser (en indentant) les lignes d'actions à exécuter lorsque la condition est examinée comme vraie au moment de l'exécution. Pour rappel la condition à examiner se trouve entre le « if » et le « : ».
Cependant il est également possible de définir un autre bloc d'actions à exécuter lorsque la condition est examinée comme fausse, pour ce faire on utilise le mot clé « else ».
Après le « else » on met « : » pour montrer que va commencer le bloc d'actions lié à une condition fausse ; toutes les lignes du bloc d'actions doivent être indentées.

Exemple d'utilisation du « if » et du « else »	Commentaires

a = 55
b = input("un entier SVP:")
if b>a:
  print "1ere ligne du bloc d'actions du if"
  print b," est plus grand que ",a
  print "derniere ligne du bloc d'actions du if "
else:
  print "1ere ligne du bloc d'actions du else"
  print b," est plus petit ou égal à ",a
  print "derniere ligne du bloc d'actions du else"
print "sans indentation, plus dans le else "
	Lire b
	Pendant l'exécution, la condition (b>a) est examinée
	
	← Ce n'est que lorsque la condition est vraie
	← que ce bloc d'actions (ici 3 lignes) est exécuté
	Attention plus de condition après le mot else
	
	← Ce n'est que lorsque la condition est fausse
	← que ce bloc d'actions (ici 3 lignes) est exécuté
	Ligne toujours affichée (peu importe la condition)

Principe général :

Structure en ordinogramme :



En LDA

si condition alors
    Action 1
sinon
    Action 2
fsi

Syntaxe en python :

```
if condition :
    Action 1
else :
    Action 2
```

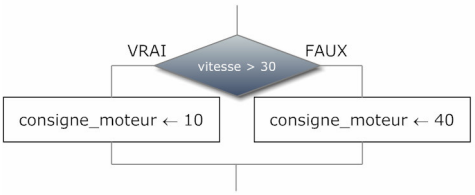
L'ordinogramme fait toujours apparaître deux branches, la structure se termine après leur recombinaison. Dans cette configuration « si-alors-sinon », la branche faux est liée à un bloc d'actions.

Le LDA fait apparaître le mot clé supplémentaire « sinon ».
Le « fsi » indique toujours la fin de la structure (la recombinaison des branches pour l'ordinogramme).

Le python fait apparaître le mot clé supplémentaire « else ».
Les différents blocs d'actions et la fin de la structure sont identifiés grâce à des indentations.

Exemple :

Structure en ordinogramme :



En LDA

```
si vitesse > 30 alors
   consigne_moteur ← 10
sinon
   consigne_moteur ← 40
fsi
```

Syntaxe en python :

```
if vitesse > 30 :
   consigne_moteur = 10
else :
   consigne_moteur = 40
```

Exemple complet avec utilisation d'une condition au sein de la structure de choix

Comme montré un peu plus tôt, il est apparu le besoin d'utiliser une condition avec la structure de choix. Nous verrons plus loin qu'il existe d'autres structures qui utilisent aussi une condition (comme la structure répétitive « tant que » par exemple).

Exemples de conditions au sein de structures de choix	Commentaires

cote = input("votre cote sur 10 SVP:")
if cote==10:
  print "bravo !"
  print "vous avez la cote max !"
print "je suis après la structure "
	Lire cote
	Si cote est la même chose que 10
	Alors afficher "bravo !"
	  afficher "vous avez la cote max !"
	afficher "je suis après la structure "

cote = input("votre cote sur 10 SVP:")
if cote !=10:
  print "snif !"
  print "vous n'avez pas la cote max !"
else :
  print "vous avez la cote max !"
print "je suis après la structure "
	Lire cote
	Si cote est différent de 10
	Alors afficher "snif !"
	  afficher "vous n'avez pas la cote max !"
	sinon
	  afficher "vous avez la cote max !"
	afficher "je suis après la structure "

6.3 Les structures imbriquées

Nous pouvons résumer le principe de la structure de choix (avec un else) comme :

Descriptif de la structure de choix (avec un else)	Commentaires

if condition :
  bloc_V
else:
  bloc_F
print "je suis après la structure "
	Si condition est vraie Alors
	  bloc_V
	Sinon
	  bloc_F
	afficher "je suis après la structure "

« bloc_V » correspond alors au bloc d'actions qui sera exécuté lorsque la condition est examinée comme vraie et « bloc_F » lorsqu'elle est examinée comme fausse.
Les deux blocs d'actions bloc_V et bloc_F représentent une suite successive de lignes. Il peut s'agir de 2 ,10 ou 1000 lignes. Un bloc d'actions peut même contenir lui-même une ou plusieurs autres structures. Il faut juste s'assurer que tout bloc d'actions contienne un nombre entier de structures (et pas de structures non achevées).

Lorsqu'on a une structure dans une structure on appelle ça des structures imbriquées.

Descriptif de la structure de choix (avec un else)	Commentaires

if a>=0 :
  if a==0 :
  print "a=0"
  else :
  print "a>0"
else :
  print "a<0"
	Si a>=0 alors
	  Si a est la même chose que 0 Alors
	  afficher "a=0
	  Sinon
	  afficher "a>0"
	Sinon
	  afficher "a<0"

Un if dans un else, une notation abrégée.

Le bloc d'actions associé à un « else » est un ensemble de lignes successives.
Si ce bloc d'actions doit commencer par une structure de choix, au lieu d'écrire « else : if … » on peut utiliser une notation abrégée notée « elif… ».

Un if dans un else, notation longue	Un if dans un else, notation abrégée

a = input("un entier SVP :")
if a>0 :
  print "positif"
else :
  if a==0 :
  print "nul"
  else :
  print "négatif"
	a = input("un entier SVP :")
	if a>0 :
	  print "positif"
	elif a==0 :
	  print "nul"
	else :
	  print "négatif"

Les deux exemples ci-dessus sont similaires (fonctionnement identique) cependant la version à droite est plus concise et n'exige pas deux niveaux d'indentation.

7 Les opérateurs logiques

Les opérateurs logiques permettent de constituer des conditions plus complexes.
Les opérateurs logiques sont : and (ET) or (OU) not(NON)

En LDA et en ordinogramme nous utiliserons les mots en français (ET, OU, NON).
En python nous utiliserons les mots clés réservés (and, or, not).

Les opérateurs logiques permettent de constituer une condition générale à partir d'autres conditions.

Par exemple la condition (a<=b and a<=c) est constituée de :
la condition a<=b , l'opérateur logique « and » , la condition a<=c

L'opérateur logique « and » constitue une condition qui sera vraie si et seulement si les deux autres conditions qui s'y rapportent sont vraies toutes les deux.
L'opérateur logique « or » constitue une condition qui sera vraie dès qu'une des deux autres conditions qui s'y rapportent est vraie (ou même le deux).

L'opérateur logique « not » ne se rapporte qu'à une seule condition.
Si cette dernière est « True », « not True » renvoie False, sinon « not False » renvoie True.

Les conditions constituées à l'aide d'opérateurs logiques sont en général utilisées au sein d'une structure (comme la structure de choix, la structure répétitive tant que,…).

Utilisations d'opérateurs logiques, exemple1 :	Commentaires

a = input("veuillez introduire le nombre a :")
b = input("veuillez introduire le nombre b :")
c = input("veuillez introduire le nombre c :")
if a<=b and a<=c:
  print a, " est le plus petit"
else:
  print a, " n'est pas le plus petit"
	Lire a
	Lire b
	Lire c
	Si a<=b et que a<=c:
	Alors afficher a, " est le plus petit"
	Sinon
	  afficher a, " n'est pas le plus petit"

Utilisations d'opérateurs logiques, exemple2 :	Commentaires

cote = input("votre cote sur 10 SVP:")
if not cote ==10:
  print "snif !"
  print "vous n'avez pas la cote max !"
else :
  print "vous avez la cote max !"
print "je suis après la structure "
	Lire cote
	Si on n'a pas cote qui est la même chose que 10
	Alors afficher "snif !"
	  afficher "vous n'avez pas la cote max !"
	sinon
	  afficher "vous avez la cote max !"
	afficher "je suis après la structure "

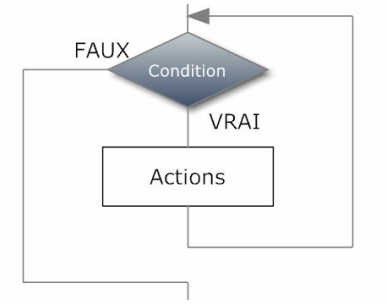
8 La tant que

8.1 Principe

Ci-dessous une phrase qui décrit le principe général de la boucle « tant que » :

Tant que condition répéter action.

Cela signifie que tant qu'une condition est vérifiée, on répète un bloc d'actions.
Si nous représentons la structure d'une boucle « tant que » à l'aide d'un ordinogramme, nous obtenons la figure ci-dessous :



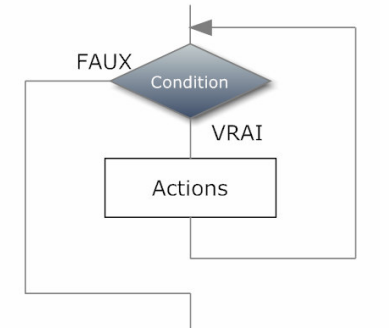
On remarque que la structure est en accord avec la description.

En effet, tant que la condition est vraie on répète le bloc d'actions.

On sort de la structure lorsque la condition est fausse.

Représentation générale :

Structure en ordinogramme :



En LDA

Tant que condition faire
    Actions
ftant

Syntaxe en python :

while condition :
    Actions

Comme pour la structure de choix, la structure « tant que » fonctionne avec une condition. La « tant que » est associée à un bloc d'actions qui sera exécuté tant que la condition est vraie. Il s'agit d'une structure répétitive car le bloc d'actions associé à la « tant que » est susceptible d'être répété si la condition continue à être vraie.

Contrairement à la structure de choix la condition peut être examinée plusieurs fois.
A la première exécution de la « tant que » la condition est examinée, si elle est fausse on sort de la structure et on passe à la suite (cas particulier où le bloc d'actions n'aura jamais été exécuté), si elle est vraie on effectue une fois le bloc d'actions et on réexamine la condition ; si au deuxième examen de la condition celle-ci est toujours vraie, on exécute pour la deuxième fois le bloc d'actions, on réexamine la condition et ainsi de suite. Tant que la condition sera vraie on va répéter le bloc d'actions, on ne sortira de la structure « tant que » que lorsque la condition sera examinée comme fausse (ou qu'une instruction break a eu lieu, voir plus tard).

Descriptif de la structure répétitive « tant que »	Commentaires

while condition :
  bloc
print "je suis après la structure "
	Tant que condition est vraie, répéter
	  bloc
	afficher "je suis après la structure "

Exemple d'utilisation de la "tant que"	**Commentaires**

#Attente de Lire la valeur 5
nb = input("Veuillez introduire un nombre entre 1 et 10")
while nb !=5 :
  print nb, "n'est pas le nombre attendu, recommencez"
  nb = input("Veuillez introduire un nombre entre 1 et 10")
print "après la structure la condition est fausse"
print "votre nombre (",nb, ")vaut bien la valeur 5"
	
	Lire nb
	tant qu'on n'a pas nb qui vaut 5 répéter
	  afficher "…, recommencez"
	  Lire nb
	afficher "… la condition est fausse"
	afficher "… vaut bien la valeur 5"

8.2 La « tant que » pour contrôler le nombre de répétitions

Il est possible d'utiliser la « tant que » pour répéter un bloc d'actions un nombre de fois donné. Nous avions vu un exemple de décompte qui montrait clairement que l'on répétait trois fois un bloc d'actions.
Pour rappel voici l'exemple à nouveau :

Exemple de décompte sans structure répétitive	Commentaires
import time nb_sec_rest=3 print "fini dans ", nb_sec_rest, " s" time.sleep(1) nb_sec_rest= nb_sec_rest – 1 print "fini dans ", nb_sec_rest, " s" time.sleep(1) nb_sec_rest= nb_sec_rest – 1 print "fini dans ", nb_sec_rest, " s" time.sleep(1) nb_sec_rest= nb_sec_rest – 1 print "FINI"	On répète trois fois : print "fini dans ", nb_sec_rest, " s" time.sleep(1) nb_sec_rest= nb_sec_rest – 1

On voit bien ici le bloc d'actions qui est répété plusieurs fois.
Il vaut mieux dans ce cas utiliser une structure répétitive.
L'exemple ci-dessous utilise la variable de décompte au sein de la condition.

Exemple de décompte avec structure répétitive	Commentaires

import time
nb_sec_rest=3
while nb_sec_rest>0:
  print "fini dans ", nb_sec_rest, " s"
  time.sleep(1)
  nb_sec_rest= nb_sec_rest – 1
print "FINI"
	On répète trois fois :
	
	print "fini dans ", nb_sec_rest, " s"
	time.sleep(1)
	nb_sec_rest= nb_sec_rest – 1

Une idée intéressante lorsqu'on veut répéter un bloc d'actions un nombre connu de fois est d'utiliser une variable dont le rôle est de compter le nombre de fois que le bloc d'actions a déjà été répété. On peut alors demander que « tant qu'on n'a pas atteint le nombre de répétitions souhaitées on répète le bloc d'actions », on utilise alors une structure répétitive « tant que » qui va utiliser une condition créée à partir de notre variable « compteur » et du nombre de répétitions souhaitées.

La condition sera par exemple (nb_repet_actu < nb_repet_a_faire) où :

nb_repet_actu	est la variable qui indique le nombre de répétitions qui a déjà eu lieu actuellement
nb_repet_a_faire	est une variable ou une constante qui indique le nombre total de répétitions que l'on souhaite faire.

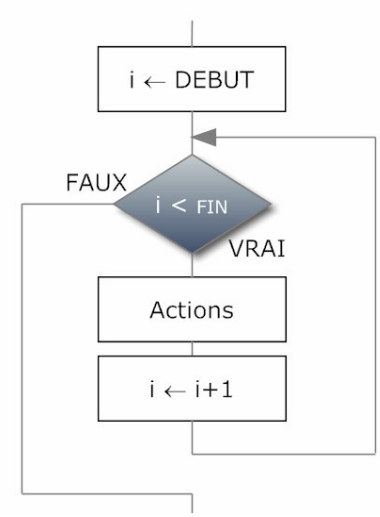
Pour que cette façon de faire fonctionne, il faut s'assurer que la variable *nb_repet_actu* soit initialisée à 0 et qu'elle soit incrémentée à chaque exécution du bloc d'actions.

La variable qui sert de compteur interne (nommée nb_repet_actu ci avant) est conventionnellement notée « i ».

Typiquement il y a une incrémentation de cette variable à la fin de chaque répétition du bloc d'actions. Il y a également un test propre à la boucle qui est effectué régulièrement de manière à déterminer s'il faut continuer à exécuter le bloc d'actions ou si le nombre de répétitions qui ont déjà été effectuées a atteint ce qui a été spécifié (dans ce cas on sort de la boucle).

Répétition d'un bloc d'actions un nombre donné de fois avec une « tant que »

Ordinogramme



En python

Principe :

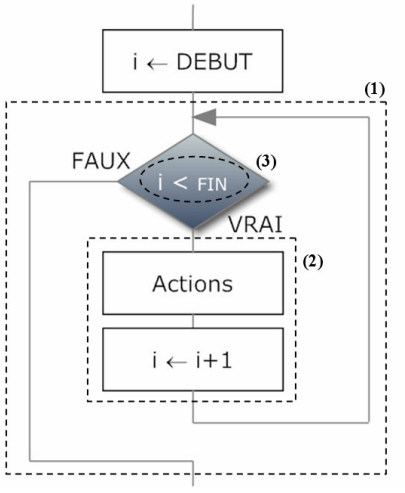
```
i =DEBUT
while i < FIN:
    Actions
    i = i– 1
#après la structure
```

Exemple (10 affichages de « Bonjour ») :

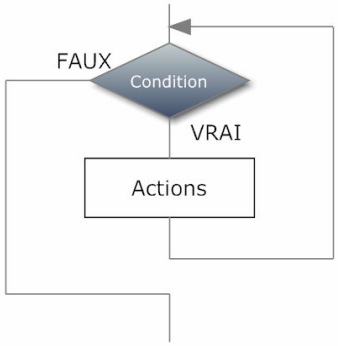
```
i =0
while i < 10:
    print ″bonjour″
    i = i– 1
#après la structure
```

On peut voir ci-dessous que La zone (1) fait bien apparaître une « tant que » qui englobe un bloc d'actions (2) et une condition (3).

Gestion des répétitions



Structure d'une « tant que »



Exemples de répétitions connaissant le nombre d'itérations	Commentaires

#Afficher bonjour 10 fois de suite
nb_repet_actu =0
while nb_repet_actu < 10 :
  print "bonjour !"
  nb_repet_actu = nb_repet_actu + 1
	On initialise nb_repet_actu à 0
	Tant qu'on n'a pas atteint 10 répétitions
	  Afficher "bonjour !"
	  « mettre à jour la variable nb_repet_actu »

#Afficher bonjour n fois de suite (n introduit par l'utilisateur)
nb_repet_a_faire = input("Combien de répétitions SVP ? ")
nb_repet_actu =0
while nb_repet_actu < nb_repet_a_faire :
  print "bonjour !"
  nb_repet_actu = nb_repet_actu + 1
	Lire *nb_repet_a_faire*
	On initialise nb_repet_actu à 0
	Tant qu'on n'a pas atteint *nb_repet_a_faire*
	  Afficher "bonjour !"
	  « mettre ajour la variable nb_repet_actu »

Cependant il existe une structure répétitive plus adaptée que la « tant que » lorsqu'on connaît le nombre de répétitions, il s'agit de la boucle FOR. Cette boucle utilise d'ailleurs le principe d'une variable « compteur » initialisée au début, mise à jour à chaque tour de boucle et testée régulièrement pour assurer le nombre de répétitions souhaitées.

8.3 L'usage du break

L'instruction break permet de sortir de la structure englobante la plus proche qui ne soit pas une structure de choix.
On peut très bien avoir une boucle « tant que » classique où tant que la condition est vraie on répète le bloc d'actions et dés que la condition devient fausse on sort de la boucle.
Cependant, on peut prévoir une façon de sortir de la boucle anticipativement grâce à l'instruction break.

Exemple d'utilisation du break au sein d'une tant que	Commentaires

#Somme de 5 chiffres introduits par l'utilisateur
print "un nombre n'est pas toujours un chiffre"
chiffre=0
somme=0
nb_repet_actu =0
while nb_repet_actu < 5 :
  chiffre = input("un nombre positif à un seul chiffre SVP : ")
  if chiffre>9 :
  print "vous avez introduits plusieurs chiffres"
  break
  somme=somme + chiffre
  nb_repet_actu = nb_repet_actu + 1
print "je suis après la structure "
if nb_repet_actu == 5 :
  print "la somme des 5 chiffres vaut : ", somme
	L'utilisateur doit introduire 5 fois de suite un nombre à un seul chiffre.
	On utilise alors une structure tant que et la gestion de la variable *nb_repet_actu* pour assurer 5 répétitions.
	Cependant l'exécution de la boucle peut s'interrompre anticipativement si l'utilisateur introduit un nombre à plusieurs chiffres.
	
	Une fois que nous sommes sortis de la boucle, la variable *nb_repet_actu* permet de déterminer si nous en sommes sortis anticipativement ou non.

9 L'algèbre de Boole

9.1 Généralités

L'algèbre de Boole est un outil mathématique qui permet de mettre en équation des fonctions combinatoires élémentaires (ET, OU,…) ou des fonctions combinatoires plus complexes (imbrication de fonctions combinatoires élémentaires).
Toute fonction combinatoire correspond à une expression mathématique appelée expression booléenne.

De la même manière qu'au cours de mathématique, les expressions booléennes sont constituées de variables, de constantes et d'opérateurs mathématiques. La grande différence réside dans le fait qu'avec l'algèbre de Boole les variables et les constantes ne peuvent prendre que 2 valeurs possibles : 0 et 1.

Souvent, pour représenter une fonction combinatoire, on utilise une équation qui contient la sortie dans le membre de gauche et l'expression booléenne correspondante dans le membre de droite. On trouvera par exemple : S = a + b

a, b et S sont des variables (ne peuvent donc prendre que les valeurs 0 et 1).
a et b sont les entrées.
S est la sortie, elle dépend de l'état des entrées et de l'expression booléenne.
a + b est l'expression booléenne de la fonction combinatoire de S.

9.2 Quelques particularités de l'algèbre de Boole

Non seulement les variables ne peuvent prendre que les valeurs 0 et 1 mais il en est de même pour les expressions. Ce qui signifie qu'une expression telle que (a+b) ne pourra contenir que les valeurs 0 et 1, même si a et b valent tous les deux 1.

$\boxed{1+1=1}$ Car le 2 n'existe pas et le résultat ne peut valoir que 0 ou 1.

$\boxed{a+a=a}$ Soit a vaut 0 et a + a = 0, soit a vaut 1 et a + a =1.

$\boxed{a+\overline{a}=1}$ $\boxed{a\times\overline{a}=0}$ Le trait horizontal au dessus signifie « complément de ».
On comprend alors que si $a=0$, $\overline{a}=1$ et si $a=1$, $\overline{a}=0$.

En examinant tous les cas, on peut montrer que les formules ci-dessus sont toujours vérifiées.

9.3 Les fonctions combinatoires élémentaires

Il y a trois fonctions combinatoires élémentaires : les fonctions ET, OU et NON.
Nous verrons que ces fonctions correspondent à des opérateurs particuliers au niveau de l'algèbre de Boole.

- La fonction ET correspond à une multiplication

L'opération $a \times b$ ne vaudra 1 que si $a = 1$ et $b = 1$
La table de vérité de la fonction ET est donnée ci-dessous :

a	b	a ET b	$S = a \times b$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Au point de vue schématique, on peut montrer que la fonction logique ET correspond à la mise en série.

- La fonction OU correspond à une addition

L'opération $a + b$ vaudra 1 dés que $a = 1$ ou $b = 1$ (ou non exclusif)
La table de vérité de la fonction OU est donnée ci-dessous :

a	b	a OU b	$S = a + b$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

Au point de vue schématique, on peut montrer que la fonction logique OU correspond à la mise en parallèle.

- La fonction NON correspond au trait horizontal au dessus

L'opération $\overline{a}$ vaudra 1 dés que $a = 0$ et inversement
La table de vérité de la fonction NON est donnée ci-dessous :

a	NON a	$S = \overline{a}$
0	1	1
1	0	0

9.4 Les théorèmes de De Morgan

Le complément d'une somme est égal au produit des compléments → $\boxed{\overline{a+b} = \bar{a} \times \bar{b}}$

Le complément d'un produit est égal à la somme des compléments → $\boxed{\overline{a \times b} = \bar{a} + \bar{b}}$

Ces lois traduisent le fait que :
- Le résultat d'une somme vaudra 1 dés qu'un des éléments de la somme vaut 1
- Le résultat d'un produit vaudra 0 dés qu'un des éléments du produit vaut 0

9.5 Exemple d'application de De Morgan en programmation

Dans certains cas on veut rester dans une boucle tant qu'on n'a pas une condition qui soit vraie. Par exemple on voudrait que l'utilisateur introduise un nombre positif à un chiffre (donc compris entre 0 et 9).
On doit alors faire une boucle qui demande à l'utilisateur de recommencer tant qu'on n'a pas une introduction correcte.

Exemple, demande d'introduction d'un nombre positif à un chiffre, version 1 :

```
nb_introduit = -1       # valeur qui assurera de rentrer dans la boucle
while (nb_introduit < 0) or (nb_introduit > 9) :
        nb_introduit  = input("Veuillez introduire un chiffre positif ou nul, SVP : ")
print "merci, vous avez introduit : ", nb_introduit
```

L'exemple ci-dessus fonctionne très bien mais il arrive parfois de se tromper au niveau des conditions. Si on remplace le « or » par un « and » on a une condition toujours fausse.
Pour augmenter la lisibilité, on peut utiliser De Morgan pour faire apparaître une condition équivalente

Exemple, demande d'introduction d'un nombre positif à un chiffre, version 2 :

```
nb_introduit = -1       # valeur qui assurera de rentrer dans la boucle
while not ((nb_introduit >= 0) and (nb_introduit <= 9)) :
        nb_introduit  = input("Veuillez introduire un chiffre positif ou nul, SVP : ")
print "merci, vous avez introduit : ", nb_introduit
```

Cette façon de faire illustre bien De Morgan de la forme : $\boxed{\overline{a \times b} = \bar{a} + \bar{b}}$

Avec	a	qui correspond à	(nb_introduit >= 0)
	b	qui correspond à	(nb_introduit <= 9)

Ceci augmente la lisibilité car on peut lire « tant qu'on n'a pas nb_introduit >= 0 et nb_introduit <= 9 (autrement dit un chiffre positif ou nul) on reste dans la boucle.

10 Les fonctions

Une fonction permet de contenir un ensemble d'instructions à exécuter.
Une fonction a un nom et permet d'effectuer une tâche.

10.1 Utilisation d'un délai (fonction existante)

Utilisation d'un délai (exemple manipulant une fonction de « time »)

Pour utiliser des utilitaires qui permettent d'attendre un délai, on écrira au début du programme « import time ».

Exemple d'utilisation d'un délai	Commentaires
import time	On écrit « import time » au début
print "salut"	On affiche « salut »
time.sleep(1)	On attend une seconde
print "tout le monde"	Lorsque la seconde est passée on affiche « tout le monde »

10.2 La définition et l'appel d'une fonction

Une fonction peut être appelée régulièrement au sein de votre programme mais doit avoir été préalablement définie. La définition de la fonction indique ce qu'il faut faire lorsque la fonction est appelée afin d'être exécutée. La définition est aussi appelée l'implémentation, on la reconnaît grâce au mot clé « def ».

L'appel d'une fonction correspond à l'utilisation proprement dite. Une fonction peut être appelée au sein du programme principal ou même au sein d'une autre fonction.
Ci-dessous un exemple de fonction dont le rôle est d'afficher un message, qui est définie (une seule fois bien sûr), mais qui est utilisée plusieurs fois.

Exemple de mise en œuvre d'une fonction (la déclaration et deux appels de fonctions).

```
#On commence à définir la fonction
def msg_derivee_fct():
    print "la dérivée d'une grandeur par rapport au temps correspond à la vitesse de variation de cette grandeur."
#On n'est plus dans la définition de la fonction car la ligne n'est plus indentée
print "la dérivée de l'énergie par rapport au temps est la puissance."
print "pour rappel :"
msg_derivee_fct()                                   #On appelle la fonction
print "la dérivée de la vitesse par rapport au temps est l'accélération."
print "Nous avions vu l'interprétation de la dérivée :"
msg_derivee_fct()                                   #On appelle la fonction
```

10.3 Fonction sans paramètres et sans valeurs de retours

L'exemple montré ci-dessus n'a pas de paramètre. En effet, tant au niveau de la définition de la fonction qu'au niveau de ses appels, il n'y a rien entre les parenthèses.

Les parenthèses permettent d'identifier une fonction mais aussi d'y placer des paramètres éventuels.
Des parenthèses vides décrivent alors des fonctions sans paramètre.

Les paramètres sont également appelés des arguments, ils permettent de transmettre des informations à la fonction pour que celle-ci puisse effectuer sa tâche.

Par ailleurs l'exemple précédent n'a pas non plus de valeur de retour, autrement dit l'appel de la fonction permet d'exécuter un certain nombre d'instructions mais la fonction ne renvoie pas un résultat qui pourrait être exploité.
Ci-dessous vous avez un autre exemple de mise en œuvre d'une fonction sans paramètre ni valeur de retour. La fonction se contente d'afficher la table de 5.

Deuxième exemple de mise en œuvre d'une fonction (la déclaration et deux appels de fonctions).

```
#On commence à définir la fonction
def affiche_table_5():
    facteur=1
    while facteur<=10:
        print facteur," fois 5 :",5*facteur
        facteur=facteur+1
print "voici la table de 5"
affiche_table_5()
print "et encore une fois :"
affiche_table_5()
```

Le nom de la fonction est « affiche_table_5 », elle est définie au début et est appelée ici deux fois dans le programme.

10.4 Fonction avec paramètre mais sans valeur de retour

Une fonction avec un paramètre doit recevoir une valeur au moment de l'appel pour qu'elle puisse s'exécuter. Par exemple lorsqu'on veut utiliser une fonction pour générer un délai, il est commode de préciser le délai que l'on souhaite attendre lors de l'appel de fonction. On peut par exemple imaginer alors une fonction nommée attendre_en_sec() où il suffit d'indiquer le nombre de secondes à attendre entre les parenthèses. Ainsi,

pour générer un délai de 5 secondes on écrit : attendre_en_sec(5)
pour générer un délai de 3 secondes on écrit : attendre_en_sec(3)

On comprend que le nombre de secondes d'attente sera paramétrable, c'est pour ça que l'information transmise à la fonction au moment de l'appel est appelée paramètre.

Pour définir une fonction avec un paramètre, il faut placer une variable entre les parenthèses de la définition et utiliser la variable au sein de l'implémentation de la fonction. Cette variable peut avoir n'importe quel nom valable et sa valeur* représente l'information qui sera transmise à la fonction au moment de l'appel.

* Le passage de paramètre se fait par valeur ou par copie pour les types primitifs. Il est à noter qu'il existe également ce qu'on appelle le passage par référence.

La variable utilisé au sein de la définition de la fonction est une variable locale à la fonction (ainsi que toutes les autres variables déclarées de façon classique au sein de la fonction).

Ci-dessous nous allons définir une fonction dont le rôle est d'afficher la table d'un entier passé en paramètre :

Exemple de mise en œuvre d'une fonction avec un paramètre

```
#On commence à définir la fonction
def affiche_table(nb):
    facteur=1
    while facteur<=10:
        print facteur," fois ",nb," : ",nb*facteur
        facteur=facteur+1
print "voici la table de 5"
affiche_table(5)
print "et la table de 9 :"
affiche_table(9)
```

Pour comprendre l'exécution du programme, il faut commencer par se pencher sur les quatre lignes concernées, ce sont les quatre dernières :

print "voici la table de 5"	→	Du texte est affiché sur écran
affiche_table(5)	→	La fonction affiche_table() est appelée, la constante 5 est communiquée à la variable « nb » de la fonction
print "et la table de 9 :"	→	Du texte est affiché sur écran
affiche_table(9)	→	La fonction affiche_table() est appelée, la constante 9 est communiquée à la variable « nb » de la fonction

Une fonction peut recevoir une variable comme paramètre au moment de l'appel

Il est possible de transmettre une variable à une fonction au moment de l'appel.
Ci-dessous on trouve un exemple presque identique au précédent sauf que l'appel fait apparaître une variable entre parenthèses.

Exemple d'appel d'une fonction en utilisant une variable comme paramètre	Commentaires

def affiche_table(nb):
  facteur=1
  while facteur<=10:
  print facteur," fois ",nb," : ",nb*facteur
  facteur=facteur+1
nb_choisi=input("Veuillez donner un nb entier SVP")
print "voici la table de ",nb_choisi
affiche_table(nb_choisi)
	Lorsque la dernière ligne est exécutée, c'est-à-dire :
	affiche_table(nb_choisi)
	
	la valeur de la variable *nb_choisi* sera communiquée à la variable *nb* de la définition de la fonction.

Il est possible d'utiliser le même nom de variable comme paramètre de l'appel et au sein de la définition de la fonction. Cependant la variable dans la fonction (variable locale) ne représente pas celle dans le programme principal (variable globale).

10.5 Fonction avec valeur de retour

Normalement une « vraie » fonction renvoie un résultat et donc possède une valeur de retour. Les fonctions qui ne possèdent pas de valeur de retour sont parfois appelées procédures.

L'avantage d'une fonction qui renvoie un résultat (donc avec une valeur de retour) et que ce résultat peut être exploité en dehors de la fonction.

On peut par exemple affecter la valeur de retour de la fonction à une variable.

Imaginons que l'on veuille calculer la température en Celcius d'une température donnée en Fahrenheit. On peut alors créer une fonction nommée *conversion_fahrenheit_celcius* qui reçoit comme paramètre une température en fahrenheit et qui calcule la température Celcius équivalente avant de fournir le résultat comme valeur de retour.

Pour la conversion on peut se baser sur la formule suivante : T(°C) = (T(°F) - 32)/1,8

Exemple d'utilisation d'une fonction avec une valeur de retour	Commentaires

def conversion_fahrenheit_celcius(temp_F):
  temp_C= (temp_F - 32)/9.0*5.0
  return temp_C
	
	Le mot clé *return* termine la fonction et donne le résultat

print "un petit test ..."	
temp_test=conversion_fahrenheit_celcius(82)	La valeur de retour est affectée à *temp_test*
print "82 Fahrenheit correspondent à ",temp_test," Celcius"	
temp_F_choisie=input("Une température en Fahrenheit SVP ")	
temp_C_convertie=conversion_fahrenheit_celcius(temp_F_choisie)	Le résultat est affecté à *temp_C_convertie*
print temp_F_choisie," F --> ",temp_C_convertie," C"	

10.6 Fonction avec plusieurs valeurs de retour

En python, contrairement à beaucoup de langage, il est possible d'avoir plusieurs valeurs de retour pour une fonction.

Cette fonctionnalité un peu particulière provient de la possibilité de faire des affectations multiples.
En python on peut par exemple écrire : a,b = 8, 20
Ça signifie que *a* est affecté de la valeur 8 et *b* de la valeur 20

On peut alors imaginer une fonction nommée *min_max* qui a plusieurs valeurs de retour et dont l'instruction *return* serait par exemple : return min, max
L'appel de la fonction sera alors lié à une affectation multiple comme par exemple :
nb_min, nb_max = *min_max(nb1, nb2, nb3, nb4, nb5, nb6)*

10.7 La fonction RANDOM (chiffre aléatoire)

Exemple d'utilisation d'une fonction de la librairie random	Commentaires
`x=random.randint`(a, b)	On affecte x d'un nombre entier aléatoire compris entre a et b (inclus).

11 Les chaînes de caractères en python

Pour expliquer l'utilisation des chaînes de caractères en python, nous décrirons la notion de tableau, la notion d'objet et l'utilisation de la boucle FOR avec des chaînes de caractères.

11.1 Introduction sur les tableaux

Bien que le python n'utilise pas vraiment de tableaux (il peut néanmoins se servir d'une liste comme d'un tableau), il est utile de connaître le principe d'un tableau ainsi que son utilité et sa mise en œuvre. Nous expliquerons ici le tableau de manière très succincte sans aborder la partie allocation de la mémoire (on suppose une taille donnée et fixe).

Un tableau permet, à l'aide d'un seul identifiant (le nom du tableau), de contenir plusieurs données qui ont le même type. Chaque donnée sera localisée au sein du tableau à l'aide d'un indice. L'indice commence à 0 et terminera à un indice qui dépend du nombre de données contenues dans le tableau.
S'il y a N données à stocker dans le tableau, le dernier indice sera N-1.

Descriptif du tableau dans un des questionnaires des olympiades d'informatiques

Pour manipuler plusieurs éléments avec une seule variable, on utilise un tableau.
Les éléments individuels d'un tableau sont indiqués par un index (que l'on écrit entre crochets après le nom du tableau).
Le premier élément d'un tableau ***tab*** *est d'indice 0 et est noté tab[0].*
Le second est celui d'indice 1 et le dernier est celui d'indice N - 1 si le tableau contient N éléments. Par exemple, si le tableau tab contient les 3 nombres 5, 9 et 12 (dans cet ordre), alors :

tab[0]= 5
tab[1]= 9
tab[2]= 12

La longueur est 3, mais l'index le plus élevé est 2.

11.2 Introduction sur la boucle FOR

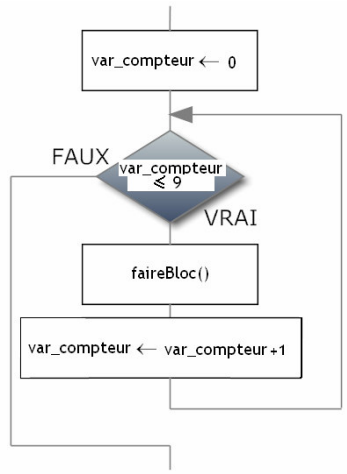
Il est intéressant de connaître le principe général de la boucle FOR avant d'étudier la mise en œuvre de celle-ci en python (qui est parfois un peu particulière).

La boucle FOR est une structure répétitive que l'on utilise en général lorsque le nombre de répétitions à faire est connu au préalable.

Le principe repose sur l'utilisation d'une variable qui servira de compteur afin d'assurer les répétitions souhaitées. Cette variable compteur est initialisée un première fois avant de démarrer, ensuite elle est testée régulièrement afin de déterminer s'il faut répéter le bloc d'actions ou si les répétitions sont terminées. Bien entendu la variable compteur est mise à jour (par pas de 1 en général) à chaque nouvelle exécution du bloc d'actions.

Ci-dessous un exemple pour 10 répétitions du bloc d'actions en commençant à 0, en vérifiant la condition *var_compteur* ≤ 9 et en incrémentant la variable à chaque tour.
Le bloc d'actions est symbolisé par la fonction faireBloc() :

Ordinogramme :



LDA : (anglais)

```
for (var_compteur ← 0 to 9 step 1) {
faireBloc()
}
```

LDA : (français)

```
pour var_compteur de 0 à 9 faire
        faireBloc()
fpour
```

Descriptif de la « FOR » dans un des questionnaires des olympiades d'informatiques

*Pour répéter du code, par exemple pour parcourir les éléments d'un tableau, on peut utiliser une boucle **for**. La notation **for** (i←a **to** b **step** k) représente une boucle qui sera répétée tant que i ≤ b, dans laquelle i commence à la valeur a, et est augmentée de k à la fin de chaque étape.*
L'exemple suivant calcule la somme des éléments du tableau tab en supposant que sa taille vaut N. La somme se trouve dans la variable sum à la fin de l'exécution de l'algorithme.

```
sum ← 0
for (i← 0 to N - 1 step 1)
{
sum ← sum + tab[i]
}
```

11.3 Les chaînes de caractères en python et la notion d'objet

Les chaînes de caractères également nommées « strings » en anglais s'apparentent dans certains langages à un tableau de caractères avec quelques spécificités particulières ; chaque case du tableau contient alors une donnée du type 'caractère' et tous ces caractères sont placés dans un ordre bien précis pour reconstituer la chaîne de caractères. Il est également courant de voir apparaître un caractère particulier à la fin des « strings » nommé caractère de fin de chaîne.

Les « strings », les tableaux ainsi que les listes et les dictionnaires que nous verrons plus tard, font partie de ce qu'on peut appeler des structures de données.

Cette appellation provient du fait que plusieurs données sont contenues au sein d'un seul élément et que ces données sont structurées suivant la convention adoptée.

Les structures de données présentées ici sont relativement simples dans la mesure où il s'agit simplement de séquences contenant une collection d'éléments.

Python est un langage orienté objet ce qui implique qu'il manipule des variables classiques (associées à un type tels que « entier », « réel » ou « booléen ») mais aussi des objets.
La notion d'objet repose en partie sur le principe de la structure de données, ainsi en python les « strings », les listes et les dictionnaires sont considérés comme des objets.

Les objets sont rattachés à un modèle qui décrit la structure des différentes données associées à ces objets ainsi que les fonctionnalités disponibles. Certains objets sont prédéfinis au sein du langage (comme les «strings », les listes et les dictionnaires en python), d'autres sont définis par le programmeur pour répondre à un besoin.

Un objet étend la notion de variable :

Une variable contient une donnée	↔	Un objet contient un ensemble de données
Une variable est associée à un type	↔	Un objet est associé à un modèle
On peut faire des opérations sur des variables	↔	On peut faire des opérations sur des objets
	↔	On peut appliquer des méthodes à des objets

Lorsqu'on crée une nouvelle donnée liée à une variable on dit qu'on déclare une variable.
La variable reçoit alors un nom au moment de sa création.
La variable est alors rattachée à un type (avec ou sans initialisation de donnée).

Lorsqu'on crée une nouvelle collection de données liées à un objet, on dit qu'on instancie un objet. L'instance de l'objet reçoit alors un nom au moment de sa création. L'instance de l'objet (également appelé objet) est alors rattachée à un modèle (avec ou sans initialisation de données).

Le modèle qui définit l'objet précise les attributs de l'objet (identifiants internes qui contiennent des données de l'objet) et les méthodes de l'objet (fonctionnalités internes à l'objet qui sont en quelques sorte des fonctions implémentées pour cet objet).

Un « string » est un objet immutable.
Ça signifie qu'il ne peut être modifié localement.

Autrement dit il faut passer par un affectation pour modifier un « string ».

En python il n'existe pas le type « caractère » mais il est possible de manipuler directement des « strings ».
Contrairement à d'autres langages il n'y a pas de caractère de fin de chaîne au sein des strings en python.

Une instance de l'objet string (que l'on appellera string) peut contenir une collection de caractères (0, 1 ou plus) qui sont ordonnés et accessibles individuellement de la même façon qu'un tableau.
Cependant il n'est pas possible de modifier un seul caractère d'un string individuellement car un string est immutable.

Exemple de mise en œuvre des « strings » (initialisation, concaténation et accès à un caractère)

```
my_str1= 'hello'                    #Lors de sa déclaration un string est initialisé
my_str2= "world"                    #entre " " ou entre ' '
my_str3= ""                         # un string peut-être vide
print  "mettons en oeuvre des strings"
print my_str1
print my_str1+my_str2               # l'opérateur + est la concaténation pour les strings
print my_str2[3]                    # le caractère d'indice 3 de my_str2 sera affiché
                                     # il s'agit du 4ème caractère donc le « l » car l'indice commence à 0
```

Comme indiqué ci-dessus un opérateur qui a un sens particulier peut avoir un autre sens pour des types ou objets différents. L'opérateur « + » qui signifie additionner pour des nombres sert à concaténer (joindre) des strings.

De la même manière l'opérateur « * » permet une répétition pour les strings (plusieurs concaténations successives).

Il est également possible d'accéder à une tranche d'un string ou d'utiliser une fonction de python qui indique la longueur du string :

Autre exemple sur les « strings » (l'opérateur « * », accès à une tranche et la fonction len())

```
my_str1= "hello world"
#my_str1[2]="r"                     #Provoquerait une erreur
my_str2= 'Ni'
my_str3= my_str2*3                  #my_str3 contient alors 'NiNiNi'
my_str2=my_str1[3:7]                #my_str2 contient la tranche de l'indice 3 à l'indice 7 non compris
print my_str1                       #affichera->hello world
print my_str2                       #affichera->lo w
print my_str3,"de longueur :",len(my_str3)   #affichera->NiNiNi de longueur : 6
```

Les tranches (slices) d'un string peuvent être présentés de différentes façons.
Un indice négatif par exemple fait référence à une position en partant de la fin.

Les tranches de « strings »

S="spam"	#résultat de l'affichage	#commentaires
print S[0]	s	
print S[-2]	a	#2ème caractère à partir de la fin
print S[1:3]	pa	#de l'index 1 au 3 non compris
print S[1:]	pam	#de l'index 1 à la fin
print S[:-1]	spa	#du début au dernier caractère non compris

Il existe des méthodes que l'on peut utiliser sur des objets strings. Une méthode est l'équivalent d'une fonction (avec ou sans paramètre) dédiée à un objet.

Pour appliquer une méthode sur une instance d'un objet, on précisera l'instance de l'objet suivi de « . » suivi du nom de la méthode (avec ou sans paramètre).

Ainsi une instance nommée « my_obj1 » pour laquelle on veut appliquer une méthode sans paramètre nommée « do_this() » sera notée : my_obj1.do_this()

Dans certains cas, la méthode peut modifier l'objet sur lequel elle est appliquée (ce qui ne sera pas le cas pour les strings car ils sont immutables).

Très souvent la méthode renvoie une valeur de retour exploitable (pour l'afficher, pour une affectation, au sein d'une condition,…).

Quelques méthodes de l'objet string

	#les affichages	#commentaires
S1="spam55"		#S1 est une instance de l'objet string
S2=S1.capitalize()		#La méthode capitalize() appliquée sur S1
		#renvoie un string comme valeur de retour
S3="98"		
print S1, S2	spam55 Spam55	#S2 comme S1 mais commence par une #majuscule
print S1.islower(), S2.islower()	True False	#La méthode islower() indique si tout en minuscule
print S1.isdigit(), S3.isdigit()	False True	#La méthode isdigit() indique si que des chiffres
print S1[4:].isdigit()	True	#Une méthode peut être appliquée à une tranche
S1=S1.replace("55","UR")		#replace() renvoie le string après remplacement
print S1,S1[4:].isdigit()	spamUR False	#S1 a été modifié grâce à l'affectation

11.4 La boucle FOR – utilisation avec des strings

Python permet de facilement parcourir les différents caractères d'un string avec la boucle FOR. Au sein de la boucle For en python, on précise un identifiant (au choix) qui contiendra successivement les différents caractères du string en cours de parcours.

L'exemple ci après utilise l'identifiant *my_id* pour contenir successivement les différents caractères de S1, nous aurions pu choisir un autre identifiant.

L'identifiant est en fait un string qui ne contient chaque fois qu'un seul caractère.

Mise en œuvre de la For pour un string (comptage du nombre de chiffres)

S1="spam55"	# affichage	#commentaires
nb_dig=0 #nombre de digits		
for my_id in S1:		#my_id contiendra successivement tous les
if my_id.isdigit():		#caractères de S1
nb_dig=nb_dig+1		
print S1,"a", nb_dig," chiffres"	spam55 a 2 chiffres	

12 Les listes

12.1 Principe des listes

Le principe des listes étend la notion de tableau ; fort connu dans un langage structuré comme le C. Un tableau permet de contenir un ensemble d'éléments de même type.

Chaque élément du tableau est identifiable grâce à un identifiant unique (le nom du tableau) et à sa position au sein du tableau (son indice).

En python, on utilise des listes lorsqu'on veut implémenter un tableau. Il faut cependant être très prudent car l'utilisation des listes en python est très souple, elle permet de faire certaines choses qui ne sont pas recommandées.
Les listes permettent par exemple de stocker des éléments de types différents au sein d'une même liste. Sauf exception, il faut éviter ça !!!

Une liste peut contenir tous types de variables ou d'objet. Une liste peut d'ailleurs elle-même contenir des listes ; ça ne pose pas de problème.

Il est cependant recommandé de s'arranger pour que chaque liste contienne des éléments de même type.
Par exemple tous des entiers, tous des strings, toutes des listes,…

• La déclaration d'une liste

Pour déclarer une liste on donne :
- son nom
- le signe « = »
- la valeur de départ de la liste entre [].
  On sépare les éléments par une virgule.

#exemples de déclaration	#commentaires
my_list_1 = [ ]	#on déclare une liste vide
my_list_2 = [1, 2, 3]	#on déclare une liste de trois entiers
my_list_3 = [0.1, 'a', 3]	#on déclare en donnant des éléments de types différents → à éviter
my_list_4 = [[ ], [ ]]	#on déclare une liste qui contient deux listes vides
my_list_5= [[8,9], [9,3]]	#on déclare une liste qui contient deux listes d'entiers

• L'affectation d'une liste

Une liste est un objet, une structure de données.
Lorsqu'on affecte une liste, en supposant que ce ne soit pas une déclaration, on fixe la nouvelle situation de la liste. La situation précédente est alors perdue.

#exemples d'affectation	#commentaires
my_list_2 = [9, 2, 8]	#on déclare une liste de trois entiers
my_list_2 = [ ]	# les 3 entiers de la déclaration n'existent plus

my_list_5= [[8,9], [9,3]] my_list_5= [5.2, 6.6]	#on déclare une liste qui contient deux listes d'entiers #la liste n'est plus une liste de deux listes d'entiers mais une liste de #deux nombres réels.

• L'ajout d'un élément à la fin d'une liste

Pour ajouter un élément à la fin d'une liste on utilise la méthode append().
On écrit le nom de la liste suivi de « . » suivi de « append() ».
On place alors de qu'on veut ajouter entre les parenthèses.

#exemples d'ajouts	#commentaires
my_list_2 = [9, 2, 8] my_list_2.append(54)	#on déclare une liste de trois entiers # la liste contient alors [9, 2, 8, 54]
my_list_3= [] my_list_3.append(2.2)	#on déclare une liste vide #la liste contient alors [2.2]
my_list_5= [[8,9], [9,3]] my_list_5.append([0, 8])	#on déclare une liste qui contient deux listes d'entiers #la liste contient alors [[8,9], [9,3], [0, 8]]

• L'affichage d'une liste

Pour afficher le contenu d'une liste, il suffit d'utiliser « print »

• Le nombre d'éléments d'une liste

Le nombre d'éléments d'une liste peut-être obtenu grâce à la fonction « len() »

#exemple illustrant len()	#commentaires
my_list_2 = [9, 2, 8] x = len(my_list_2)	#on déclare une liste de trois entiers # x contient le nombre d'éléments de my_list_2
print x print my_list_2	#sera affiché 3 #sera affiché [9, 2, 8]
my_list_2.append(4) print len(my_list_2)	#La liste contient alors [9, 2, 8, 4] # sera affiché 4

12.2 L'affectation et la mutation d'un objet

Lorsqu'on fait une affectation, on fait en réalité correspondre l'identifiant à un nouvel objet ; l'identifiant ne fait plus référence à l'ancien objet.

Illustration :

#affectation d'un objet	#commentaires
my_list = [9, 2, 8]	#L'objet [9, 2, 8] est créé, appelons-le OBJET1

	#L'identifiant « my_list » fait référence à OBJET1
my_list = [8 ,8, 5]	#L'objet [8,8,5] est créé, appelons-le OBJET2 #L'identifiant « my_list » fait référence à OBJET2 #OBJET1 n'est plus référencé par l'identifiant « my_list »

Lorsqu'on fait une affectation d'un objet1 vers un autre objet2, on dit que l'objet2 fait référence à l'objet1 !

Illustration :

#affectation d'un objet	#commentaires
my_list = [9, 2, 8]	#L'objet [9, 2, 8] est créé, appelons-le OBJET1 #L'identifiant « my_list » fait référence à OBJET1
my_id = my_list	#L'identifiant « my_id » fait référence à l'objet référencé par « my_list » #L'identifiant « my_id » fait référence à OBJET1
my_list = [8 ,8, 5]	#L'objet [8,8,5] est créé, appelons-le OBJET2 #L'identifiant « my_list » fait référence à OBJET2 #OBJET1 n'est plus référencé par l'identifiant « my_list »
print my_list, my_id	# il sera affiché « [8 ,8, 5] [9, 2, 8] »

Lorsqu'on effectue une mutation à un objet, on modifie des données de cet objet mais on considère que c'est toujours le même objet !

Illustration :

#affectation d'un objet	#commentaires
my_list = [9, 2, 8]	#L'objet [9, 2, 8] est créé, appelons-le OBJET1 #L'identifiant « my_list » fait référence à OBJET1
my_id = my_list	#L'identifiant « my_id » fait référence à l'objet référencé par « my_list » #L'identifiant « my_id » fait référence à OBJET1
my_list[0] = 100	# OBJET1 [9, 2, 8] est muté et devient [100, 2, 8] #L'identifiant « my_list » fait toujours référence à OBJET1
print my_list, my_id	# il sera affiché « [100, 2, 8], [100, 2, 8]» #la valeur affichée de « my_id » a changé car **fait référence** à un objet #qui a simplement muté !

Une autre façon d'avoir un comportement similaire est lorsqu'on applique une méthode sur un objet.

Illustration (avec la méthode « append() »):

#affectation d'un objet	#commentaires
my_list = [9, 2, 8]	#L'objet [9, 2, 8] est créé, appelons-le OBJET1 #L'identifiant « my_list » fait référence à OBJET1
my_id = my_list	#L'identifiant « my_id » fait référence à l'objet référencé par « my_list »

	#L'identifiant « my_id » fait référence à OBJET1
my_list.append(47)	# OBJET1 [9, 2, 8] est modifié et devient [9, 2, 8 , 47] #L'identifiant « my_list » fait toujours référence à OBJET1
print my_list, my_id	# il sera affiché « [9, 2, 8 , 47], [9, 2, 8 , 47]» #la valeur affichée de « my_id » a changé car **fait référence** à un objet #qui a simplement été modifié !

12.3 Exemples d'utilisation de listes en pratique

• *Exemple 1, introduction de deux cotes dans une liste et affichage de la moyenne :*

```
liste_cotes = []
cote = input("une cote SVP : ")
liste_cotes.append(cote)
cote = input("une cote SVP : ")
liste_cotes.append(cote)
moyenne = (liste_cotes[0] + liste_cotes[1])/2.0
print "votre moyenne vaut : ", moyenne
```

On a commencé par créer une liste vide.
On a ensuite ajouté (avec « append ») deux cotes au sein de notre liste
On calcule la moyenne en utilisant les éléments de la liste grâce à leurs index.
On affiche la moyenne.

• *Exemple 2, moyenne de 5 cotes avec initialisation de la liste :*

```
liste_cotes = [0, 0, 0, 0, 0]
total_cotes = 0
i = 0

#insertion des cotes
indice_cote = 0
while indice_cote < 5 :
    cote = input("une cote SVP : ")
    liste_cotes[indice_cote] = cote
    indice_cote = indice_cote + 1

#calcul de la somme des cotes
i = 0
while i < 5 :
    total_cotes = total_cotes + liste_cotes[i]
    i = i + 1

moyenne = total_cotes /5.0
print "votre moyenne vaut : ", moyenne
```

On peut remarquer que nous avons initialisé la liste avec 5 valeurs.
Cette façon de faire nous a permis d'insérer nos 5 cotes sans utiliser la méthode « append » mais simplement en donnant la valeur à un élément de la liste : liste_cotes[indice_cote] = cote

Un risque de vouloir insérer directement une valeur à un élément de la liste est de tenter d'accéder un élément qui n'existe pas.
Si nous avions tapé « liste_cotes[5] = cote » il y aurait eu une erreur ; en effet, l'indice maximum dans notre exemple est 4.

En résumé, on peut donner une valeur à un élément d'une liste qui a déjà été initialisé mais il faut faire attention à ne pas tenter de faire référence à un élément qui n'est pas existant (dont l'indice est hors de portée).

D'autre part on peut remarquer que nous utilisons une while pour parcourir les 5 éléments de la liste. Nous verrons plus tard qu'une boucle « FOR » permet de simplifier l'itération.

• *Exemple 3, moyenne de n cotes sans initialisation de la liste :*

```
liste_cotes = []
total_cotes = 0
n = 0
i = 0

#lecture de la valeur de n
n = input("indroduisez le nombre de cotes N, SVP : ")

#insertion des cotes
i = 0
while i < n :
    cote = input("une cote SVP : ")
    liste_cotes.append(cote)
    i = i + 1

#calcul de la somme des cotes puis de la moyenne
i = 0
while i < len(liste_cotes) :
    total_cotes = total_cotes + liste_cotes[i]
    i = i + 1
moyenne = total_cotes / (1.0 * len(liste_cotes) )
print "votre moyenne vaut : ", moyenne
```

On demande à l'utilisateur d'introduire la valeur de « n ».
On utilise la valeur de « n » pour s'assurer de lire les cotes et de les placer dans la liste à l'aide de la méthode « append() ».
On parcourt toute la liste pour calculer la somme des cotes puis la moyenne.
On peut voir que l'on n'utilise plus « n » mais la longueur de la liste « len(liste_cotes) ». Cette façon de faire est recommandée car elle fonctionne même si « n » a changé et ne correspond plus au nombre d'éléments de la liste. Il faut juste s'assurer que lorsqu'on parcourt tous les éléments de la liste (dans la boucle), on ne modifie pas le nombre d'éléments de la liste.

• *Exemple 4, trier les éléments d'une liste :*

```
liste_cotes = []
n = 0
i = 0

#lecture de la valeur de n
n = input("indroduisez le nombre de cotes N, SVP : ")

#insertion des cotes
i = 0
while i < n :
   cote = input("une cote SVP : ")
   liste_cotes.append(cote)
   i = i + 1

#trier la liste
liste_cotes.sort()

#afficher la liste
print liste_cotes
```

Une fois les « n » cotes introduites, la méthode « sort() » trie la liste dans l'ordre croissant.

Une explication exhaustive des listes n'est pas l'objet de ce cours ; il y a de nombreuses fonctionnalités que nous ne détaillerons pas.
Cependant, voici un petit aperçu de fonctionnalités supplémentaires :

- Comme pour les « strings », les listes peuvent être manipulées par tranches
  Exemple : print liste_cotes[2 :]
- Deux listes peuvent être combinées en une seule avec l'opérateur « + »
- On peut avoir des listes à deux dimensions ou plus.
- Il existe des méthodes telles que extend(), remove(), reverse(), …
- Certaines fonctions de python acceptent une liste comme argument.
  Exemples : len(), sorted(), reversed(),…

12.4 La boucle FOR, utilisation avec des listes

Il existe une fonction en python qui est capable de créer une liste ordonnée de nombres entiers, il s'agit de la fonction « range() ».
Par exemple lorsqu'on écrit : print range(5) # Il apparaît [0, 1, 2, 3, 4]

Cette fonction est couramment utilisée pour créer une boucle « FOR ».
Pour rappel, une boucle « FOR » est une boucle que l'on utilise lorsqu'on connaît à l'avance le nombre de répétitions.

Si on veut par exemple afficher 5 fois « bonjour », on écrira :

```
for i in range(5):
   print "bonjour"
```

• *Exemple, introduction de cinq cotes dans une liste et affichage de la moyenne :*

```
liste_cotes = []
cote = 0
total_cotes = 0
moyenne = 0
for i in range(5):
    cote = input("une cote SVP : ")
    liste_cotes.append(cote)
for i in range(len(liste_cotes)):
    total_cotes = total_cotes + liste_cotes[i]
moyenne =  1.0 * total_cotes / len(liste_cotes)
print "votre moyenne est de : ", moyenne
```

Dans certains cas on souhaite utiliser une liste ordonnée de nombres mais qui ne commencent pas forcément à 0.
Dans ce cas on peut donner deux paramètres à la fonction « range() », exemple :

```
liste_nb = range(1, 11)          # la liste vaut alors [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

• *Exemple, affichage de la table de multiplication (jusqu'à 10) d'un nombre :*

```
nombre = 0
print "table de multiplication "
nombre = input("veuillez introduire votre nombre : ")
for facteur in range(1, 11):
    print facteur, " * ", nombre, " = ", facteur * nombre
```

Remarque : Il est possible d'utiliser la fonction range() pour manipuler directement une liste sans la boucle « FOR ». Exemple : liste_nombres = range(100)

13 Les fichiers

Des fonctionnalités sont prévues en python pour créer, lire et écrire dans des fichiers.
Il existe en python des objets de type <file> qui vont permettre de manipuler directement des fichiers.
On créera alors un fichier en utilisant la fonction open().

13.1 Création d'un fichier

On commence par ouvrir le fichier en python.

Pour ouvrir le fichier on utilise la fonction *open* et on donne le mode d'ouverture
'r' pour une ouverture en lecture
'w' pour une ouverture en écriture (le contenu du fichier est écrasé)
'a' pour une ouverture en mode ajout

Exemple d'ouverture d'un fichier	Commentaires
my_file = open("data_base.txt","r")	On ouvre le fichier data_base.txt en mode lecture et on lui donne le nom my_file

Le nom que l'on donne à l'objet qui représente le fichier (« my_file » dans l'exemple ci-dessus) va nous permettre de faire appel à des méthodes pour manipuler le fichier.

13.2 Ecriture dans un fichier, la méthode write()

Pour pouvoir écrire dans un fichier on utilise la méthode write() sur un objet de type fichier. L'objet doit donc avoir été créé préalablement avec la fonction open().
Il est nécessaire d'avoir un fichier dont le mode d'ouverture est 'w' (écriture) ou 'a' (ajout) pour lui appliquer la méthode write().
Tenter d'appliquer la méthode write() à un fichier ouvert en mode lecture provoquera une erreur !

Pour écrire dans un fichier :

Exemple d'écriture dans un fichier	Commentaires
my_file = open("data_base.txt","w") my_file .write("texte à mettre dans le fichier") my_file.close()	On ouvre le fichier data_base.txt en mode écriture On écrit « texte à mettre dans le fichier » dans le fichier data_base.txt (en écrasant le contenu précédant) On ferme le fichier

Pour ajouter une information dans un fichier :

Exemple d'ajout dans un fichier	Commentaires
my_file = open("data_base.txt","a") my_file .write("texte à mettre dans le fichier") my_file.close()	On ouvre le fichier data_base.txt en mode ajout On écrit « texte à mettre dans le fichier » dans le fichier data_base.txt (à la suite du contenu précédant) On ferme le fichier

Dans les deux exemples donnés ci-dessus, si le fichier n'existe pas dans le dossier courant (celui qui contient votre fichier python) il sera créé automatiquement.

13.3 Fermeture d'un fichier, la méthode close()

Les deux exemples du point précédent montraient bien qu'on terminait la manipulation du fichier à l'aide de la méthode close().
Ce qui permet à python de libérer son emprise sur le fichier.

13.4 La gestion des répertoires des fichiers

Il faut savoir que la gestion des répertoires (les dossiers) se fait par défaut à partir du dossier courant (celui qui contient votre fichier python).
Si on demande de créer un dossier, celui-ci sera donc créé par défaut au sein du dossier qui contient votre programme python.

• Création d'un dossier

Pour créer un nouveau dossier, on utilisera les fonctionnalités de l' « operating system ». On importera la librairie python « os » afin de pouvoir utiliser la méthode mkdir() qui permettra de créer un nouveau répertoire (mkdir ↔ « make directory »).

On écrira par exemple :

```
import os
os.mkdir("mon_dossier1")
```

Il apparaîtra alors un nouveau dossier nommé « mon_dossier1 » au sein du dossier courant. Il est important de savoir que si ce dossier existait déjà, les lignes ci-dessus provoqueraient une erreur !

Une façon de s'assurer de ne pas avoir une erreur lors de la création d'un dossier est de vérifier si un dossier spécifique existe. Pour ce faire on utilise la méthode « isdir() » sur un objet qui représente le dossier courant.

On peut tester par exemple :

```
import os
if os.path.isdir("mon_dossier1"):
    print "dossier existe"
else:
    print "dossier n'existe pas"
```

On remarquera alors un affichage qui indique si le dossier spécifié existe.

Pour créer un dossier sans risque d'erreur :

```
import os
if not os.path.isdir("mon_rep"):
        os.mkdir("mon_rep")
```

On a alors créé le dossier « mon_rep » que dans le cas où celui-ci n'existe pas.

Il est également possible de donner comme paramètre aux méthodes isdir() et mkdir() non pas le nom d'un répertoire mais tout un chemin.

On peut par exemple écrire :

```
import os
os.mkdir("dos_glob/sous_dos1/repertoire5")
```

Lors de l'exécution, dans la mesure où « dos_glob » existe dans le dossier courant et qu'il contient un dossier nommé « sous_dos1 », il sera créé un nouveau dossier « repertoire5 » au sein de « sous_dos1 ».

Dans les autres cas, l'exécution provoquera une erreur.

On peut aussi écrire :

```
import os
if not os.path.isdir("dos_glob/sous_dos1/repertoire5") :
   print  "le chemin existant n'est pas complet"
   print  "au minimum un dossier n'existe pas"
else :
   print  "le chemin existant est complet"
```

13.5 Changement du répertoire courant

• Chemins relatifs et absolus

Les exemples précédents montraient comment créer des dossiers vis-à-vis du répertoire courant (celui qui contient votre fichier python).

D'autre part, lorsqu'on donnait un chemin qui reflète l'organisation interne des dossiers on précisait un chemin relatif ; ce qui veut dire que l'on précisait un chemin relatif vis-à-vis du répertoire courant.

Un gros avantage de travailler avec un chemin relatif est d'assurer une portabilité du code. Ainsi, même si on déplace le répertoire courant, le chemin relatif reste valable.

Il est également possible de manipuler ce qu'on appelle des chemins absolus.
Voici un exemple de chemin absolu : "C:\Documents and Settings\All Users\Bureau"
C'est un chemin qui commence à partir de la racine.

• La méthode chdir()

La méthode chdir() signifie « change directory » et permet de changer le répertoire considéré comme le répertoire courant.
Au départ le répertoire courant est celui qui contient votre fichier python.

Lorsqu'on va préciser un chemin comme paramètre de chdir(), le programme considèrera le répertoire mentionné comme le dossier courant.
Il faut bien sûr que le chemin donné corresponde à un chemin déjà existant.

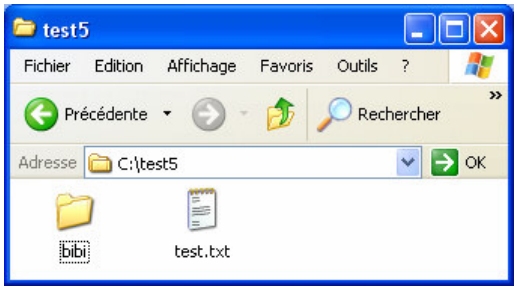
Ensuite, tous les dossiers et fichiers créés, s'ils utilisent des chemins relatifs, seront créés à un emplacement relatif au dernier dossier courant spécifié par chdir().

Exemple :

```
import os
os.mkdir("C:/test5")
os.chdir("C:/test5")
os.mkdir("bibi")
my_file = open("test.txt","w")
my_file .write("texte pour voir")
my_file.close()
```

Dans cet exemple, un nouveau dossier « test5 » sera créé au sein de « C : » (on suppose qu'il n'existait pas).
Ce dossier sert ensuite de dossier courant.
On crée alors un directory « bibi » relatif au dossier courant.
On crée ensuite un fichier « test.txt » au sein du dossier courant.

La capture d'écran ci-dessous montre alors le résultat de l'exécution.



13.6 Lecture d'un fichier

Pour lire un fichier, il faut l'ouvrir en mode lecture et accéder au contenu à l'aide de la méthode read(). On peut ensuite affecter à une variable la valeur de retour de la méthode read().

Exemple de lecture et d'affichage d'un fichier :

```
import os
my_file = open("test.txt","r")
contenu = my_file.read()
print contenu
my_file.close()
```

La variable « contenu » de notre exemple sera en fait une variable de type « string » qui contient sous forme d'une chaîne de caractère l'ensemble du fichier.

14 Notions d'algorithmes

Un algorithme permet de mettre en évidence une résolution d'un problème informatique.
L'algorithme est en général rédigé en LDA (langage de description algorithmique).

Une fois que l'algorithme est établi, on peut le transposer dans le langage informatique de son choix.

Lorsqu'on étudie les principes algorithmiques, on commence souvent par étudier quelques algorithmes de tri afin de pouvoir comparer l'efficacité de ces algorithmes.

Les deux critères primordiaux d'efficacité des algorithmes sont :
L'optimisation de l'espace mémoire.
L'optimisation du temps d'exécution

Nous tenterons ici simplement de sensibiliser l'aspect « temps d'exécution » en montrant que, suivant la manière dont l'algorithme est conçu, l'exécution du programme sera plus ou moins rapide.

14.1 Algorithme de tri, principe

L'idée est de tenter de trier un ensemble d'éléments par ordre (croissant en général).
Pour ça il faut que les éléments soient ordonnables (par exemple des nombres ou des mots) et qu'ils soient disponibles pour la routine qui se chargera du tri ; le plus simple est de considérer que tous les éléments se trouvent au sein d'un tableau.

On appellera N le nombre d'éléments à trier.
Chaque élément sera identifiable grâce à un indice.
En LDA on considère souvent les indices allant de 1 à N alors qu'au sein d'un langage de programmation les indices iront en général de 0 à N-1.

14.2 Algorithme de tri, deux exemples

1) Tri par sélection

L'idée du tri par sélection est assez simple. On tente d'abord de trouver l'élément le plus petit est de le placer en 1ère position. On se charge ensuite de trouver l'élément le 2ème plus petit est de le mettre en 2ème position et ainsi de suite.

La description et l'algorithme de ce type de tri sont donnés ci-dessous :

Description générale [modifier | modifier le code]

Sur un tableau de *n* éléments (numérotés de 1 à *n*), le principe du tri par sélection est le suivant :

- rechercher le plus petit élément du tableau, et l'échanger avec l'élément d'indice 1 ;
- rechercher le second plus petit élément du tableau, et l'échanger avec l'élément d'indice 2 ;
- continuer de cette façon jusqu'à ce que le tableau soit entièrement trié.

En pseudo-code, l'algorithme s'écrit ainsi :

```
procédure tri_selection(tableau t, entier n)
    pour i de 1 à n - 1
        min ← i
        pour j de i + 1 à n
            si t[j] < t[min], alors min ← j
        fin pour
        si min ≠ i, alors échanger t[i] et t[min]
    fin pour
fin procédure
```

http://fr.wikipedia.org/wiki/Tri_par_sélection

2) Tri - Bulle

L'idée du tri-bulle (ou tri à bulle) se base une succession d'échanges d'éléments jusqu'à ce que tout le tableau soit trié.
Le principe est simple : on parcourt tous les éléments et on compare chaque fois deux éléments contigus pour voir si l'indice le plus faible correspond à la valeur la moins élevée ; si ce n'est pas le cas on échange les deux éléments.

Imaginons N éléments :
On commence par comparer l'élément d'indice 1 et l'élément d'indice 2 et, si l'élément d'indice 1 contient la valeur la plus élevée, on échange les deux éléments.
L'idée est, qu'après la comparaison et le traitement des valeurs d'indices 1 et 2, l'indice le plus élevé contient la valeur la plus élevée ; autrement dit : la valeur la plus grande est à droite.
On continue le procédé avec les indices 2 et 3 ; après traitement la valeur la plus élevée est à droite.
On continue le procédé avec tous les autres indices, à la fin du procédé la valeur la plus élevée est à droite : on a placé le plus grand à sa place (tout à droite).
On va ensuite répéter tout le procédé à partir de l'indice 1 pour placer le deuxième plus grand en avant dernier.
Suivant la même idée on s'assurera de placer le 3ème plus grand, puis le 4ème plus grand et ainsi de suite jusqu'à ce que tout le tableau soit trié.

La description et l'algorithme de ce type de tri sont donnés ci-dessous :

L'algorithme parcourt le tableau, et compare les couples d'éléments successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, ils sont échangés. Après chaque parcours complet du tableau, l'algorithme recommence l'opération. Lorsqu'aucun échange n'a lieu pendant un parcours, cela signifie que le tableau est trié. On arrête alors l'algorithme.

```
procédure tri_bulle(tableau T)
    n = longueur(T)
    pour i de n - 1 à 1
        aucun_échange = vrai
        pour j de 1 à i
            si T[j] > T[j + 1], alors
                échanger T[j] et T[j + 1]
                aucun_échange = faux
            fin si
        fin pour
        si aucun_échange : fin procédure
    fin pour
fin procédure
```

http://fr.wikipedia.org/wiki/Tri_à_bulles

14.3 Algorithme de tri, étude simplifiée de l'efficacité

Nous allons analyser brièvement l'efficacité du tri par sélection.
Il est intuitif que le temps nécessaire à trier des éléments dépendra du nombre d'éléments N.
D'autre part l'ordre de départ des éléments influencera également le temps d'exécution mais comme on n'est pas responsable de cet ordre de départ on supposera un cas défavorable.

Pour simplifier on supposera que la gestion des structures (if, while, for) ne prennent pas de temps ; on se contentera de prendre en compte les affectations.
Toujours pour simplifier, on va négliger les échanges (swap) et compter le nombre d'affectations de la variable « min ».
On supposera le cas défavorable où les comparaisons sont telles que les affectations de la variable « min » ont lieu à chaque fois.

Comptons le nombre d'affectations maximum de la variable « min » :

- Pour trouver le plus petit, on risque d'affecter N fois la variable « min ».
- Pour trouver le 2ème plus petit, on risque d'affecter N - 1 fois la variable « min ».
- Pour trouver le 3ème plus petit, on risque d'affecter N - 2 fois la variable « min ».
- …
- Pour trouver le 2ème plus grand, on risque d'affecter 2 fois la variable « min ».

Une fois tous les points précédents faits, le plus grand est d'office à sa place.

Au total, le nombre maximum d'affectations de la variable « min » est donc :

$$N + (N - 1) + (N-2) + (N-3) + \ldots + 3 + 2 = ((N + 2) / 2)* (N - 1) = 0,5\ N^2 + 0,5\ N - 1$$

Si on suppose que N vaut 1 million (10^6) et que la durée d'une affectation dure 0, 01µs (10^{-8} s) on peut estimer la durée du tri :

$$\begin{aligned} \text{Durée approximative} = & \quad (10^{-8}\text{ s})* (0,5 * (10^6)^2 + 0,5*(10^6) - 1) \\ \approx & \quad (10^{-8}\text{ s})* (0,5 * (10^6)^2) \\ \approx & \quad (10^{-8}\text{ s})* (0,5 * (10^6)^2) \\ \approx & \quad 0,5 * 10^4\text{ s} \\ \approx & \quad 5\ 000\text{ s} \\ \approx & \quad 83,\ 3\text{ minutes} \end{aligned}$$

14.4 Algorithme de tri, approche du tri fusion (merge sort)

Le but ici est de sensibiliser sur le critère de temps d'exécution d'un algorithme.
Sans rentrer dans l'implémentation complète du tri fusion (merge sort) on va faire ressortir une idée très intéressante qui consiste à diviser les éléments à traiter en plusieurs parties et les traiter séparément pour diminuer la durée totale d'exécution.

Imaginons l'idée suivante :

- On divise le tableau d'éléments à trier en deux parties égales.
- On trie chaque partie du tableau séparément.
- On constitue l'ensemble trié sur base des deux « demis tableaux » triés.

Analyse de l'efficacité :

Considérons le même exemple que celui analysé avec le tri par sélection.
Dans notre exemple nous avions considéré N = 10^6, le temps d'une affectation = 10^{-8} s et nous étions arrivé à une durée approximative d'exécution de 83,3 minutes.

Considérons maintenant l'idée de trier deux « demis tableaux » séparément.
Chaque « demi tableau » correspond alors à un nombre d'éléments qui vaut N/2.
Toujours avec le tri par sélection, la durée d'exécution de chaque « demi tableau » devient :

$$\begin{aligned} \text{Durée approximative (50\%)} = & \quad (10^{-8}\text{ s})* (0,5 * (0,5\ 10^6)^2 + 0,5*(0,5\ 10^6) - 1) \\ \approx & \quad (10^{-8}\text{ s})* (0,5 * (0,5 *10^6)^2) \\ \approx & \quad (10^{-8}\text{ s})* (0,5 * (10^6)^2) \\ \approx & \quad 0,125 * 10^4\text{ s} \\ \approx & \quad 1\ 250\text{ s} \quad \approx \quad 20,\ 8\text{ minutes} \\ \text{Durée approximative (100\%)} \approx & \quad 41,\ 6\text{ minutes} \end{aligned}$$

Comme chaque « demi tableau » met environ 20,8 minutes à être trié, le tri des deux « demis tableaux » mettra environ 41,6 minutes au total.

La durée du tri des deux « demis tableaux » est donc significativement plus petite que la durée du tri du tableau complet. Cependant il faudra encore du temps pour pouvoir reconstituer le tableau complet trié à partir des deux « demis tableaux » triés.

Le jeu en a-t-il valu la chandelle ?

Dans ce cas la réponse est oui car la fusion des deux « demis tableaux » triés en un seul tableau trié ne prendrait dans notre cas qu'une fraction de seconde.
L'idée de cet exemple est simplement de montrer un exemple d'artifice qui permet de diminuer significativement la durée d'exécution.

15 L'interface graphique Tkinter

Lorsqu'on prend en charge une interface graphique, on doit pouvoir gérer ce qu'on appelle des évènements.

Les évènements sont par exemple :

- Un click de souris
- La validation d'un bouton
- L'introduction d'une valeur dans une zone spécifique
- …

15.1 Gestion d'évènements

Pour prendre en charge des évènements, le programme doit être prévu pour et mettre en œuvre ce qu'on appelle de la programmation évènementielle (par opposition à la programmation procédurale).

Avec de la programmation évènementielle, le programme va passer son temps à attendre… qu'un évènement survienne.

A contrario, la programmation procédurale repose sur le principe de gérer ses tâches les unes après les autres dans le temps par l'intermédiaire de procédures.
Une fois les tâches terminées, le programme atteint sa phase de terminaison.

Tous les programmes informatiques possèdent généralement trois phases :

- Une phase d'initialisation
- Une phase centrale
- Une phase de terminaison

Programmation procédurale :

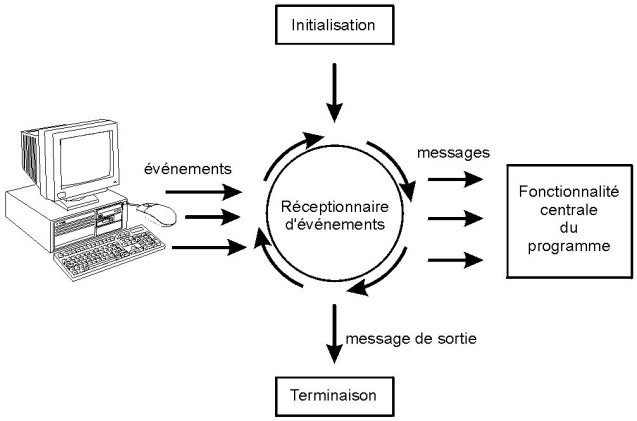
Dans un programme de type procédural, ces trois phases sont organisées suivant un schéma linéaire comme indiqué ci-contre.

Dans un programme qui utilise une interface graphique, l'organisation interne est différente. On dit qu'il est piloté par des évènements.

Programmation événementielle :

Ici après sa phase d'initialisation, le programme se met en attente.
Un réceptionnaire d'évènements scrute en permanence les périphériques et réagit dès qu'un évènement est détecté.

Dès qu'un évènement est détecté, la routine de prise en charge de cet évènement va gérer l'évènement.



15.2 Interface graphique Tkinter - Introduction

Pour introduire la mise en œuvre d'une interface graphique Tkinter, nous allons partir d'un exemple simple (voir page suivante).

- Le programme commencera par créer une fenêtre.

- Ensuite les différents éléments de la fenêtre seront également créés et rattachés à notre fenêtre. C'est également lors de ce rattachement que l'on précisera éventuellement le nom de la fonction qui se chargera de prendre en charge la routine gestionnaire de l'évènement associé.

- La fenêtre doit être associée à une écoute d'évènements (attente).

- Les fonctions gérant les routines gestionnaires d'évènements doivent être implémentées.

Illustration par l'exemple :

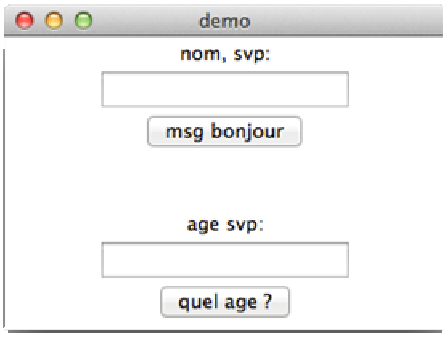
- Création de la fenêtre, des ses éléments et du nom des routines qui gèrent les évènements

Le morceau de programme ci-dessous n'est que la partie du programme qui créée la fenêtre :

Morceau de programme

```
my_frame = Tk()
my_frame.title("demo")
my_frame.minsize(300, 200)

Label(text="nom, svp:").pack()
case_nom = Entry()
case_nom.pack()
Button(text = "msg bonjour", command = bonjour).pack()

Label(text="\n").pack()

Label(text="age svp:").pack()
case_age = Entry()
case_age.pack()
Button(text = "quel age ?", command = affiche_age).pack()
```

Aperçu



Les « labels » ne sont que des éléments de texte.
Les éléments créés à l'aide de la fonction « Entry() » sont des zones conçues pour que l'utilisateur puisse y introduire des données.
Les « Buttons » permettent d'être associés à une routine de gestion de click. Par exemple, le bouton sur lequel est indiqué « msg bonjour » est associé à une fonction nommée « bonjour » qui sera appelée lorsqu'on cliquera sur ce bouton.
Tous les éléments sont associés à la fenêtre à l'aide d'un « manager » ; dans notre exemple le gestionnaire utilisé est l'utilitaire « pack() ».

- Association de la fenêtre à une écoute d'évènements

Après avoir créé une fenêtre (my_frame dans notre exemple), on peut l'associer à une attente d'évènement en notant :

```
my_frame.mainloop()
```

- Les routines gestionnaires d'évènements.

Les deux boutons de notre exemple sont associés à des fonctions qui s'exécuteront lorsqu'on cliquera sur le bouton correspondant.
Ces deux fonctions sont « bonjour() » et « affiche_age() ».
Ces fonctions peuvent être implémentées comme des fonctions habituelles.
Cependant, il s'agit ici de fonctions sans paramètre.

L'exemple complet est donné ci-dessous :

Exemple manip2.demo.py	Explications
`from Tkinter import *`	Import Tkinter
`nom = ''` `age = ''`	Variables globales

`def bonjour():`
`  global nom`
`  nom = case_nom.get()`
`  case_nom.delete(0, END)`
`  print "Bonjour", nom`
	Fonction qui affiche bonjour suivi du contenu de la case_nom
	La case est ensuite effacée

`def affiche_age():`
`  global age`
`  age = case_age.get()`
`  case_age.delete(0, END)`
`  print "vous avez", age, "ans"`
	Fonction qui affiche un message avec contenu de la case_age
	La case est ensuite effacée

`my_frame = Tk()`
`my_frame.title("demo")`
`my_frame.minsize(300, 200)`
	On crée une fenêtre avec un titre et une taille minimale

`Label(text="nom, svp:").pack()`
`case_nom = Entry()`
`case_nom.pack()`
`Button(text = "msg bonjour", command = bonjour).pack()`
	On joint un label à la fenêtre
	On crée une zone case_nom
	On crée un bouton et on lui associe la fonction bonjour()

`Label(text="\n").pack()`	Saut de ligne

`Label(text="age svp:").pack()`
`case_age = Entry()`
`case_age.pack()`
`Button(text = "quel age ?", command = affiche_age).pack()`
	On joint un label à la fenêtre
	On crée une zone case_age
	On crée un bouton et on lui associe la fonction affiche_age()

`my_frame.mainloop()`	On se place en attente d'événements

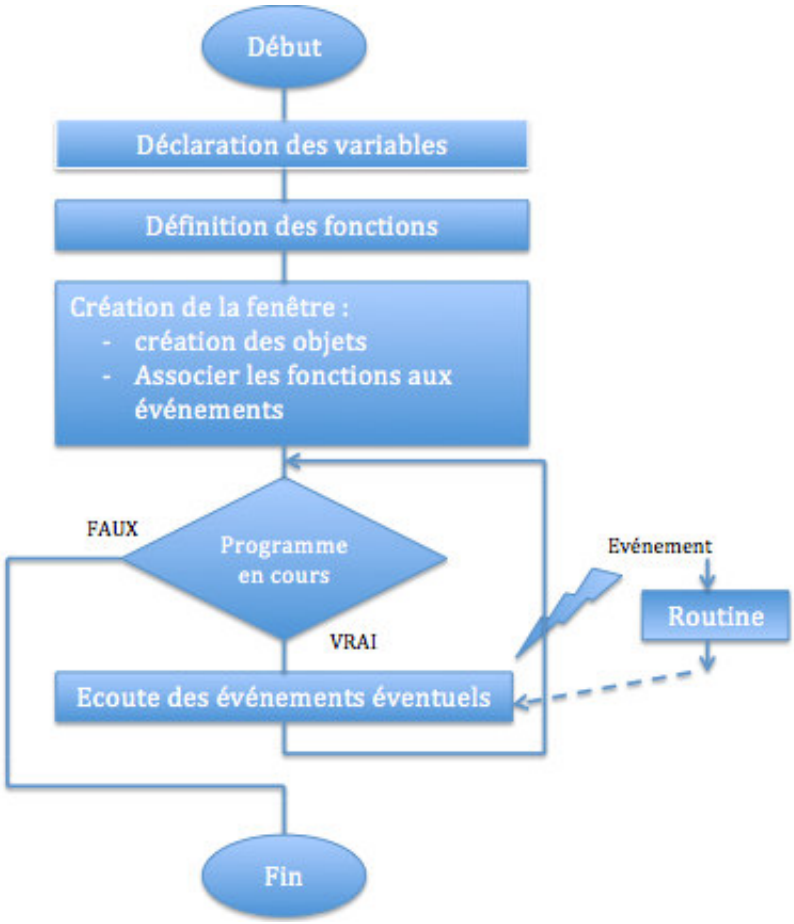
Cet exemple de base permet d'afficher un message particulier lorsqu'on clique sur un bouton. Comme le montrent les fonctions « bonjour() » et « affiche_age() », un clic sur un bouton va aller sauver dans une variable le contenu de la case associée (avec la méthode get()), va effacer la case (avec la méthode delete()) et afficher un message utilisant l'information de la case.

On peut remarquer que l'exemple fait apparaître :

- La déclaration de variables
- La définition des fonctions
- La création de la fenêtre avec : - La création des objets
  - L'association des objets aux évènements
- La boucle d'écoute des évènements.

L'ordinogramme montré à la page suivante illustre cette structure

Structure d'un programme évènementiel :



Le programme passe la majorité de son temps à attendre.
Lorsqu'un événement extérieur survient (click de souris, click sur un bouton, enfoncement d'une touche,…) le programme exécute la routine qui lui est associée.

15.3 Création d'une fenêtre

Exemple de création d'un fenêtre	Commentaires
from Tkinter import * fen=Tk() fen.mainloop()	Importe la bibliothèque de fonction Tkinter On crée un objet Tkinter (une fenêtre) On ouvre la fenêtre

15.4 Création d'un Label

Exemple de création d'un fenêtre	Commentaires
from Tkinter import * fen=Tk() Label(text = " texte du label ").pack() fen.mainloop()	Importe la bibliothèque de fonction Tkinter On crée un objet Tkinter (une fenêtre) Le message " texte du label " sera affiché On ouvre la fenêtre

15.5 Création d'une zone de texte

Exemple de création d'un fenêtre	Commentaires
from Tkinter import * fen=Tk() ma_case = Entry() ma_case.pack() fen.mainloop()	Importe la bibliothèque de fonction Tkinter On crée un objet Tkinter (une fenêtre) Une zone de texte est créée La zone est liée à la fenêtre On ouvre la fenêtre

Pour plus tard accéder au contenu de la zone de texte, il faudra appeler la méthode get() sur l'identifiant de la zone de texte.
Dans l'exemple ci-dessus on manipulera « ma_case.get() ».
Pour effacer le contenu on utilisera la méthode « delete() ».

15.6 Création d'un bouton

Exemple de création d'un fenêtre	Commentaires
from Tkinter import * #... fen=Tk() Button(text = "clic1" command = fct1).pack() fen.mainloop()	Importe la bibliothèque de fonction Tkinter #... On crée un objet Tkinter (une fenêtre) Il sera indiqué « clic1 ». La routine associée est fct1() On ouvre la fenêtre

Si besoin est, il est possible de préciser que la routine qui se charge de gérer le clic d'un bouton recevra un paramètre au moment de l'évènement. Voici un exemple :

```
Button(text = "clic2" command = lambda:fct2(mon_parametre)).pack()
```

Il en résultera qu'au moment où l'on cliquera sur le bouton « clic2 », la fonction fct2() sera appelée avec « mon_parametre » comme paramètre.

15.7 Création d'un Canvas

Un « Canvas » est une zone de dessin que l'on peut ajouter à une fenêtre.
Cette zone permet d'y ajouter différentes formes et de préciser des propriétés à ces formes.
On peut par exemple placer un rond rouge au sein du Canvas et lier le Canvas à la fenêtre.
Rien n'empêche, par la suite de modifier des propriétés de certaines formes du Canvas.

Plusieurs exemples sont donnés dans les manipulations ; le plus complet est le labyrinthe.

Exemple de création d'un objet (oval)	Commentaires
from Tkinter import * fen=Tk() can=Canvas (fen,width=400,height=300,bg='ivory') can.pack() balle=can.create_oval(20,20,40,40,fill='red') fen.mainloop()	 Définition d'une fenêtre Définition d'une zone de dessin de largeur 400 et hauteur 300 de couleur ivoire On ouvre la zone de dessin Création d'un ovale dont les extrémités et la couleur son fixé (x-r,y-r,x+r,y+r,couleur)

15.8 L'évènement « touche enfoncée »

La gestion d'un évènement « touche enfoncée » possède deux aspects :

1.) Créer un écouteur pour un enfoncement clavier

```
my_frame.bind("<Key>", gestion_touche_enfoncee)
```

Une fois la fenêtre créée, on peut préciser qu'il faut être à l'écoute d'un événement « enfoncement d'une touche ». Ci-dessus « <Key> » précise le type d'événement et gestion_touche_enfoncee la fonction qui sera appelée.

2.) Définir la routine qui prend en charge l'enfoncement d'une touche (exemple)

```
def gestion_touche_enfoncee(touche):
    print "repr(touche.char) : ", repr(touche.char), "touche.char : ", touche.char   #affichages pour comprendre
    touche_str = touche.char
    if touche_str in ['0', '1']:
        click(int(touche_str))
```

Même si elle n'est pas clairement explicitée lors de la création de l'écouteur « enfoncement d'une touche », la fonction qui prend en charge la touche utilise un paramètre.
Le paramètre est un objet particulier qui représente la touche enfoncée.
Pour avoir un string qui représente la touche on peut utiliser le paramètre suivi de « .char ».
Dans notre cas ça sera « touche.char ».
Afin de vous aider à comprendre n'hésitez pas à utiliser des affichages.

15.9 Le manager grid() – Exemple de la calculette

Dans les exemples précédents le « manager » de la disposition des éléments au sein de la fenêtre était réalisé à l'aide de l'utilitaire « pack() ».
Un autre « manager » intéressant est l'utilitaire « grid() » qui permet de disposer les éléments de la même manière qu'une grille.

L'exemple de la calculette illustre bien cette façon de faire dans la mesure où les différentes touches sont disposées suivant des lignes et des colonnes de la même manière qu'une grille.

Le rôle et la création des différents éléments (label, button,…) ne change pas, seul le manager de la fenêtre changera. Il est cependant conseillé d'utiliser le même manager pour tous les éléments de la fenêtre. Autrement dit, on utilise partout des « pack() » ou partout des « grid() » mais pas les deux en même temps au sein d'une même fenêtre.

Exemple de la calculette :

La demo destinée à la 1ère manipulation de la calculette illustre bien l'utilisation du « manager » grid().
Le code est donné à la page suivante et est commenté juste après.

```
# calculette 2 termes
# prend en compte des clicks sur boutons
# demo1 pour la 1ere seance

from Tkinter import *

# Les chaines constantes possibles pour la variable operateur
STR_ADD = ' + '
STR_SOUSTR = ' - '
STR_MUL = ' * '
STR_DIV = ' / '

#variables globales
nb1 = ''        #une chaine vide au depart, un nombre ensuite
nb2 = ''        #une chaine vide au depart, un nombre ensuite
operateur = ''  #une chaine vide au depart, une chaine operateur ensuite
egal = ''       #une chaine vide au depart, ' = ' ensuite
result = ''     #une chaine vide au depart, un nombre ensuite

calcul = ''     #sera utilisee comme StringVar() comme variabletext d'un label

#les fonctions
def click_2():
    #lorsqu'on click sur un chiffre
    global nb1, nb2, result
    if operateur == '':         #l'operateur n'est pas introduit
        if nb1 == '':           # -- ces 2 lignes seront utiles
            nb1 = 0             # -- plus tard
        nb1 = 2
        calcul.set(str(nb1))
    else:
        if nb2 == '':           # -- ces 2 lignes seront utiles
            nb2 = 0             # -- plus tard
        nb2 = 2
        calcul.set(str(nb1) + operateur + str(nb2))

def click_operateur(str_operateur):
    #lorsqu'on click sur un operateur
    global operateur
    if egal == '':                          #le calcul n'est pas fini
        if operateur == '' and nb1 != '':   #l'operateur n'est pas introduit et nb1 existe
            operateur = str_operateur
            calcul.set(str(nb1) + operateur)

def click_egal():
    #lorsqu'on click sur egal
    global result, egal
    if nb2 != '':
        egal = ' = '
        if operateur == STR_ADD:
            result = nb1 + nb2
        else:
            result = 999 #a modifier bien sur
        calcul.set(str(nb1) + operateur + str(nb2) + egal + str(result))

#creation de la fenetre
my_frame = Tk()
my_frame.title("calculette 2 termes")
my_frame.minsize(200, 200)

#la 1ere ligne contient un label dont le texte est associe a la variable 'calcul'
calcul = StringVar()
Label(textvariable = calcul, height = 2).grid(row=0, column=0, columnspan=4, sticky=W)
calcul.set('')

#la 2eme ligne
Button(text = STR_DIV, width = 3, command = lambda:click_operateur(STR_DIV)).grid(row=1, column=3)

#la 3eme ligne
Button(text = STR_MUL, width = 3, command = lambda:click_operateur(STR_MUL)).grid(row=2, column=3)

#la 4eme ligne
Button(text = ' ', width = 3).grid(row=3, column=0)
Button(text = "2", width = 3, command = click_2).grid(row=3, column=1)
Button(text = ' ', width = 3).grid(row=3, column=2)
Button(text = STR_SOUSTR, width = 3, command = lambda:click_operateur(STR_SOUSTR)).grid(row=3, column=3)

#la derniere ligne
Button(text = ' ', width = 3).grid(row=4, column=0)
Button(text = STR_ADD, width = 3, command = lambda:click_operateur(STR_ADD)).grid(row=4, column=3)
Button(text = " = ", width = 20, command = click_egal).grid(row=5, column=0, columnspan=4)

#la phase d'attente d'un evenement
my_frame.mainloop()
```

Description des notions utilisées :

- Création d'une fenêtre (éléments utilisés)

```
my_frame = Tk()
my_frame.title("calculette 2 termes")
my_frame.minsize(200, 200)
```

On crée une fenêtre (ici nommée my_frame), on lui donne un titre et une taille minimum. Selon le contenu, la taille peut dépasser la taille minimum.

- Création d'un label ()

```
calcul = StringVar()
Label(textvariable = calcul, height = 2).grid(row=0, column=0, columnspan=4, sticky=W)
calcul.set('')
```

Un label est simplement une zone de texte qui montre une chaine de caractère.
Si on veut que la chaîne de caractère puisse être une variable, on déclare un StringVar() avant et on précise le nom du StringVar() via le paramètre textvariable du label.
Pour placer le label dans la fenêtre on peut utiliser le manager grid() qui va gérer la disposition des éléments dans la fenêtre suivant un tableau (lignes et colonnes).
On peut décider d'étaler l'élément sur plusieurs cases ; columnspan permet d'étaler sur plusieurs colonnes. Sticky = W permet que le texte du label se colle à gauche (ouest = W).

- Création d'un bouton

```
Button(text = ' ', width = 3).grid(row=3, column=0)
```

On peut mettre du texte sur le bouton et préciser sa largeur (width) en caractères.
Il est possible, comme pour le label, d'étaler le bouton sur plusieurs cases.

- Associer une fonction à l'événement click d'un bouton

```
Button(text = " = ", width = 20, command = click_egal).grid(row=5, column=0, columnspan=4)
Button(text = STR_ADD, width = 3, command = lambda:click_operateur(STR_ADD)).grid(row=4, column=3)
```

Les deux exemples ci-dessus permettent d'associer une fonction à l'événement click d'un bouton. Le 1er exemple utilise la fonction sans paramètre click_egal(), le 2ème exemple utilise une fonction avec un paramètre. On reconnaît que c'est une fonction avec paramètre grâce au « lambda : » et la valeur du paramètre qui sera passé lors du click est ici la valeur de STR_ADD. Cette façon de faire permet d'avoir une même fonction associée à des clicks de boutons différents.
Cependant, rien n'interdit de faire appel à ces fonctions en dehors d'une routine qui gère un événement.

16 Comment rendre un programme python exécutable ?

Les explications données ici sont extraites du lien suivant :
http://fr.openclassrooms.com/uploads/fr/ftp/livre/python/extrait_chap33.pdf

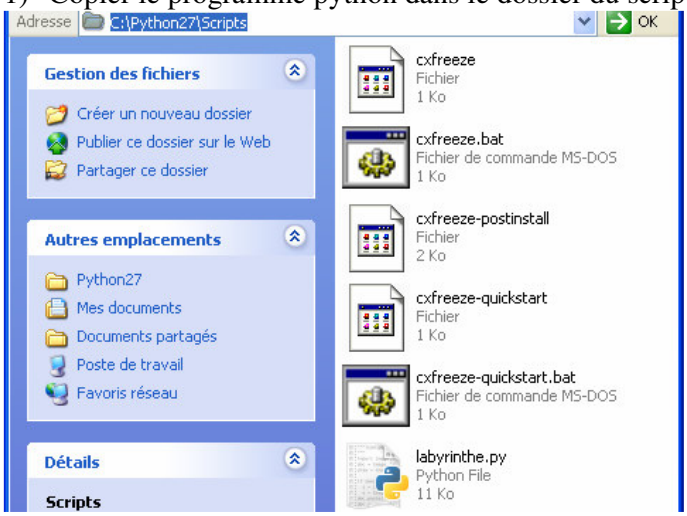
L'idée est de constituer un fichier exécutable sur une machine qui n'a pas forcément python qui soit installé. Le processus de transformation du programme python en fichier exécutable créera le fichier exécutable lui-même ainsi que d'autres fichiers nécessaires au bon fonctionnement de notre programme.

Il y a de nombreuses façons de créer un fichier exécutable, nous allons ici utiliser l'utilitaire cx_freeze. Cet utilitaire fonctionne sur Windows, Linux et Mac OS.
Montrons ci-dessous la procédure de transformation sur Windows.

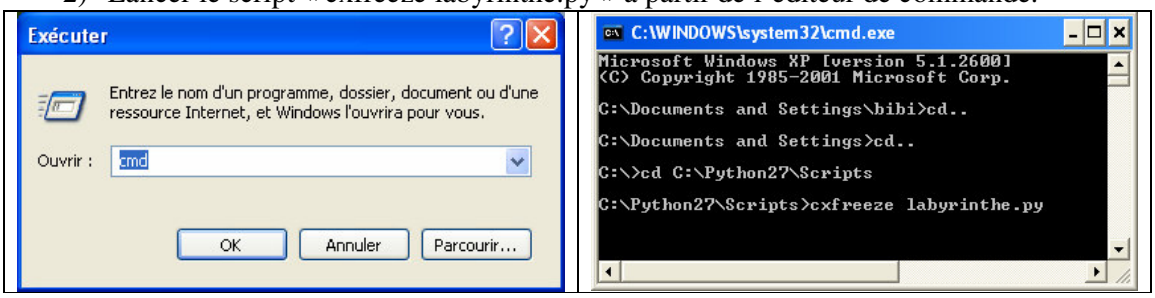
Comment transformer un fichier python (labyrinthe.py dans notre exemple) en fichier exécutable sur Windows ?

- Installer cx_freeze via http://cx-freeze.sourceforge.net/
- Sélectionner la version de python souhaitée et les caractéristiques de l'ordinateur.
- Mettre en œuvre l'utilitaire cx_freeze (exemple montré pour un fichier labyrinthe.py)

1) Copier le programme python dans le dossier du script :



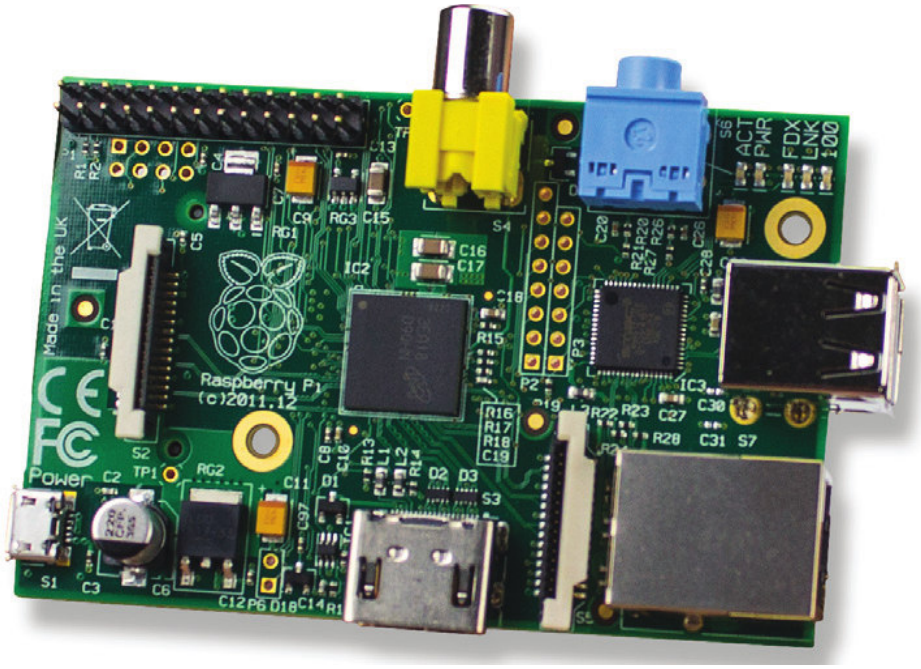
2) Lancer le script « cxfreeze labyrinthe.py » à partir de l'éditeur de commande.



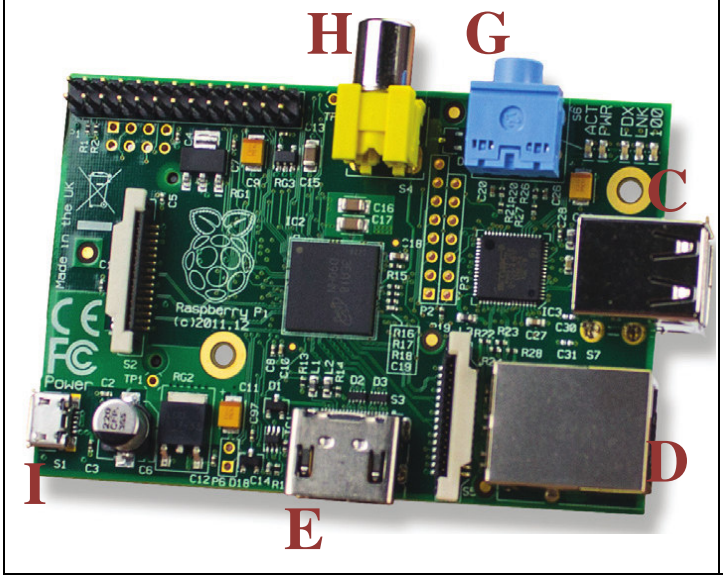
17 Raspberry

17.1 Introduction au Raspberry

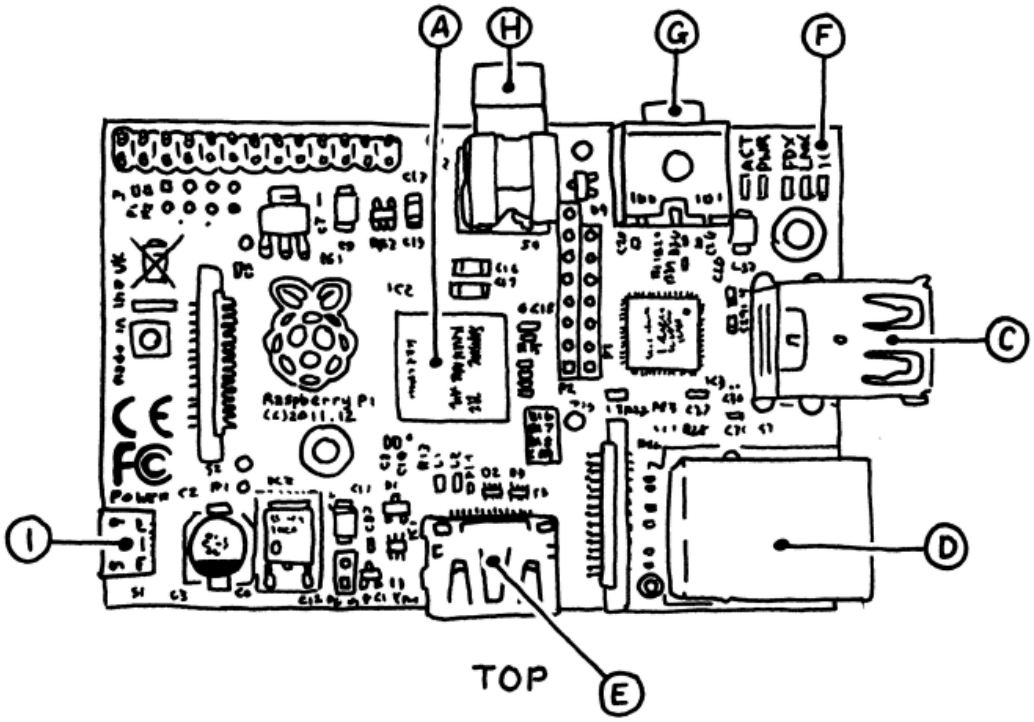
Le Raspberry est l'équivalent d'un mini-ordinateur de dimension réduite et à prix démocratique. Il est prévu pour fonctionner avec une version adaptée de la distribution Debian de Linux (appelée Raspbian). Ci-dessous une photo du Raspberry modèle B.



L'interconnexion de base avec le monde extérieur :

Interconnexions de base	Description
	C Double usb (clavier et souris) D Port ethernet (réseau) E Port hdmi (écran) G Mini-jack (audio) H RCA (vidéo composite) I Micro-usb (alim)

Interconnexion, éléments supplémentaires :



Description

A

Processeur, ARM11

B : Pas sur la vue du dessus, sur le « bottom », une carte SD

Mémoire type flash, configurée pour servir de DD

C

Double usb (clavier et souris)

D

Port ethernet (réseau)

E

Port hdmi (écran)

F : Leds de debug

ACT	Green	Lights when the SD card is accessed (marked OK on earlier boards)
PWR	Red	Hooked up to 3.3V power
FDX	Green	On if network adapter is full duplex
LNK	Green	Network activity light
100	Yellow	On if the network connection is 100Mbps (some early boards have a 10M misprint)

Led verte, connexion SD
Led rouge, 3V3
Led verte, ON si fullduplex
Led verte, activité réseau
Led, jaune, mode rapide

G

Mini-jack (audio), destiné à des sources hautes-impédances (-> alimentées)

H

RCA (vidéo composite)

I

Micro-usb (alim)

Pour commencer une manip Raspberry il nous faut :

Bien entendu	Le raspberry type B
Pour l'OS	Une carte SD
Pour l'alimentation	- Un « transfo » avec une fiche micro-usb Et/Ou - Une batterie avec une fiche micro usb
Pour l'écran	- Un écran avec une entrée dvi - Un câble de conversion hdmi/dvi
Les entrées interface utilisateur	- Un clavier - Une souris
Vers l'extérieur	- Une connexion réseau Mais aussi - Des interconnexions avec des cartes d'interface

Fonctionnement général du Raspberry :

Où est l'OS, quel est l'OS ?
L'OS est une version debian de linux.
L'OS est stocké dans la carte SD

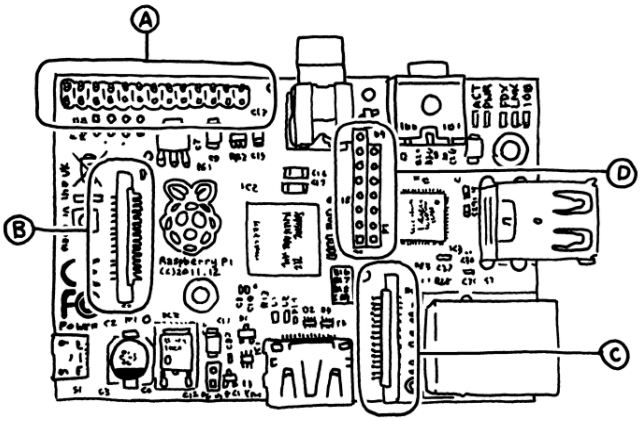
Où est la RAM ?
La RAM est la RAM interne du processeur (512Mo pour le type B)

Où est l'alimentation ?
Le « transfo » apporte 5V à la carte.
La carte utilise les 5V et à l'aide de composants électroniques, crée du 3V3 et du 1V8 pour utilisation interne.

Où est la carte réseau ?
La gestion réseau est faite sur la carte à l'aide de composants électroniques, il est géré par le composant lan9512 qui est accompagné d'un quartz à 25MHz, de résistances, de condensateurs,…

Où est la carte graphique ? Où est la carte son ?
La gestion graphique ainsi que la gestion audio se fait sur la carte grâce à des composants électroniques.

Les extensions possibles à l'aide de connecteurs :

Emplacement des connecteurs	Description
	A GPIO General Purpose Input and Output B SDI The Display Serial Interface Pour LCD or OLED C CSI The Camera Serial Interface D testing headers cf Broadcom chip et LAN9512

Détail du connecteur GPIO :

Notations classiques				n° de pin				Notations « fonctionnelles »			
1	3V3	5V	2	1		2	1	3V3	5V	2
3	SDA	--	4	3		4	3	SDA	5V	4
5	SCL	GND	6	5		6	5	SCL	Gnd	6
7	#4	TXD	8	7		8	7	GPIO_GCLK	Txd0	8
9	--	RXD	10	9		10	9	Gnd	Rxd0	10
11	#17	#18	12	11		12	11	GPIO_GEN0	GPIO_GEN1	12
13	#21	--	14	13		14	13	GPIO_GEN2	Gnd	14
15	#22	#23	16	15		16	15	GPIO_GEN3	GPIO_GEN4	16
17	--	#24	18	17		18	17	3V3	GPIO_GEN5	18
19	MOSI	--	20	19		20	19	SPI_MOSI	Gnd	20
21	MISO	#25	22	21		22	21	SPI_MISO	GPIO_GEN6	22
23	SCLK	CS0	24	23		24	23	SPI_SCLK	SPI_CE0_N	24
25	--	CS1	26	25		26	25	gnd	SPI_CE1_N	26

Pour démarrer le Raspberry :

- Tout connecter puis brancher le micro-usb

- Par défaut le login est « pi » et le password est « raspberry »
  Remarque, au départ le clavier est en querty

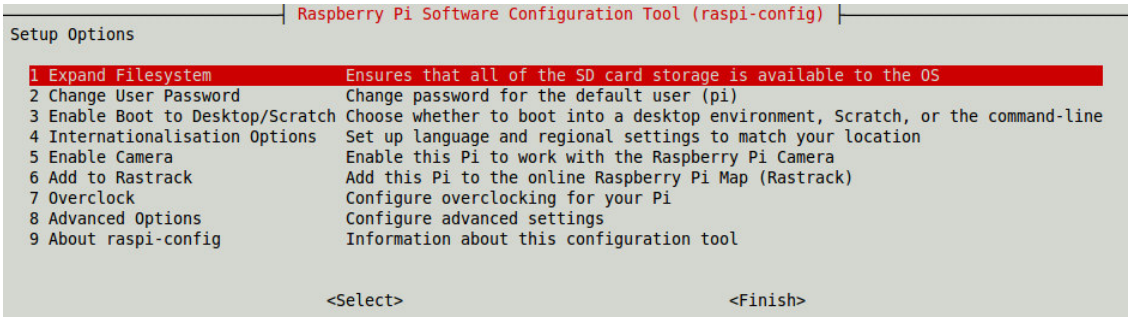
- help montre un ensemble de commandes

- sudo permet de faire des commandes en tant que super-utilisateur (root)
  Ex : sudo raspi-config
  Sudo nano /etc/default/keyboard
  Sudo nano test.py
  Sudo python test.py
  sudo halt

17.2 Prise en main du Raspbian

Raspbian est l'appellation pour une distribution linux Debian optimisée pour le Raspberry
Raspberry + Debian donne Raspbian.

- **Au premier démarrage**, avec une carte SD spécifique, il apparaît le message :
  « writing image to SD (1513 MB) »
  Une fois qu'on click sur OK il apparaît un ensemble de lignes de commandes à l'écran suivi de la fenêtre « Setup Options » ci-dessous :

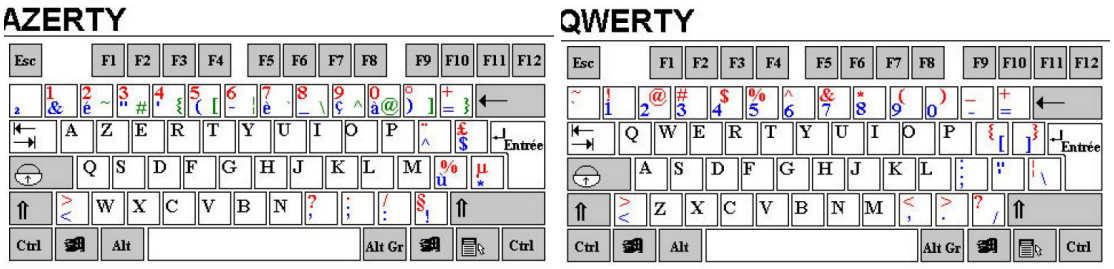


- **En imaginons qu'on ne modifie rien** (on click sur « finish »), il apparaît :
  pi@raspberrypi~$

  En tapant « help » il apparaît un ensemble de commandes.
  Remarque : par défaut le clavier est en querty → difficultés pour les commandes.
  On peut demander des informations sur une commande spécifique, exemple :
  En tapant « help alias » ou plutôt «help qliqs », il apparaît des infos sur la commande « alias ».

  Par défaut le **login** est **« pi »** et le **password** est **« raspberry »**
  Remarque, au départ le clavier est en querty
  → Il faut alors taper le login « pi » et le password « rqspberry »
  En tapant « sudo halt » ou plutôt «sudo hqlt », on éteint proprement.

- Comparaison des **claviers AZERTY et QWERTY** ci-dessous :



  On peut remarquer que les changements majeurs sont :
  A et Q (inversés), Z et W (inversés), M et « , » (inversés)

- **« sudo »** permet de faire des commandes en tant que super-utilisateur (root)
  Exemples : sudo raspi-config
  sudo halt

- **Pour modifier le clavier** à partir du mode lignes de commandes:
  Tapez « sudo nano /etc/default/keyboard » ou plutôt :
  « sudo nqno /etc/defqult/keyboqrd »

  « nano » est un utilitaire qui permet d'éditer un fichier en mode texte.
  Pour modifier le clavier, il faudra changer « gb » par « fr » puis sauver (« CTRL+X » et taper « enter »).
  On tape ensuite « sudo reboot » pour redémarrer.
  Au redémarrage on remarque que le clavier est changé.

- **python en mode interactif** (à l'aide des lignes de commande)
  Si on tape simplement la commande « python », on doit normalement constater que la version 2.7.3 est installée. On peut alors faire des tests en mode interactif.
  Exemples : print ″coucou″
  a = 5
  print a + 9
  On tape alors « CTRL + D » pour quitter le mode python interactif

- Edition d'un programme **python via « nano »** (à l'aide des lignes de commande)
  Si on tape la commande « sudo nano test.py ».
  On arrive dans une fenêtre d'environnement nano (exemple : GNU nano 2.2.6)

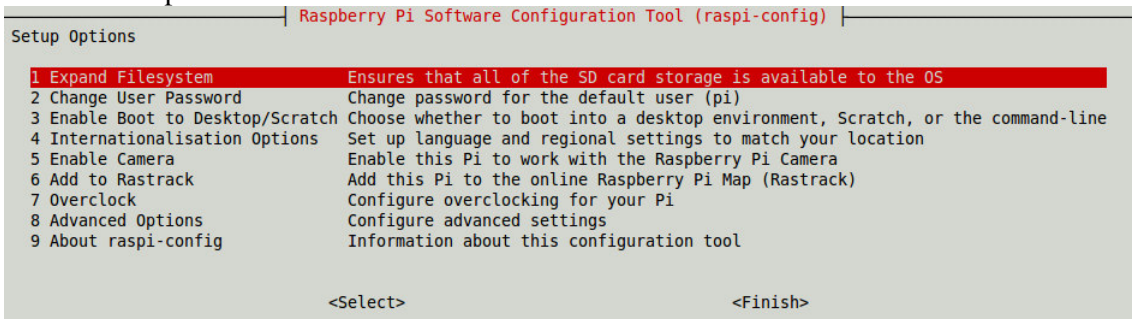
  On tape par exemple : print ″ test ″
  a = input(″un nb SVP : ″)
  print ″votre nb : ″, a

  Puis « CTRL + O » → write out
  « enter » → wrote 3 lines
  « CTRL + X » → exit

  Pour ensuite lancer le programme on tape « sudo python test.py »

  On peut si on le désire modifier notre programme existant en tapant à nouveau :
  « sudo nano test.py ».
  Remarque : on utilise 4 espaces pour l'indentation.

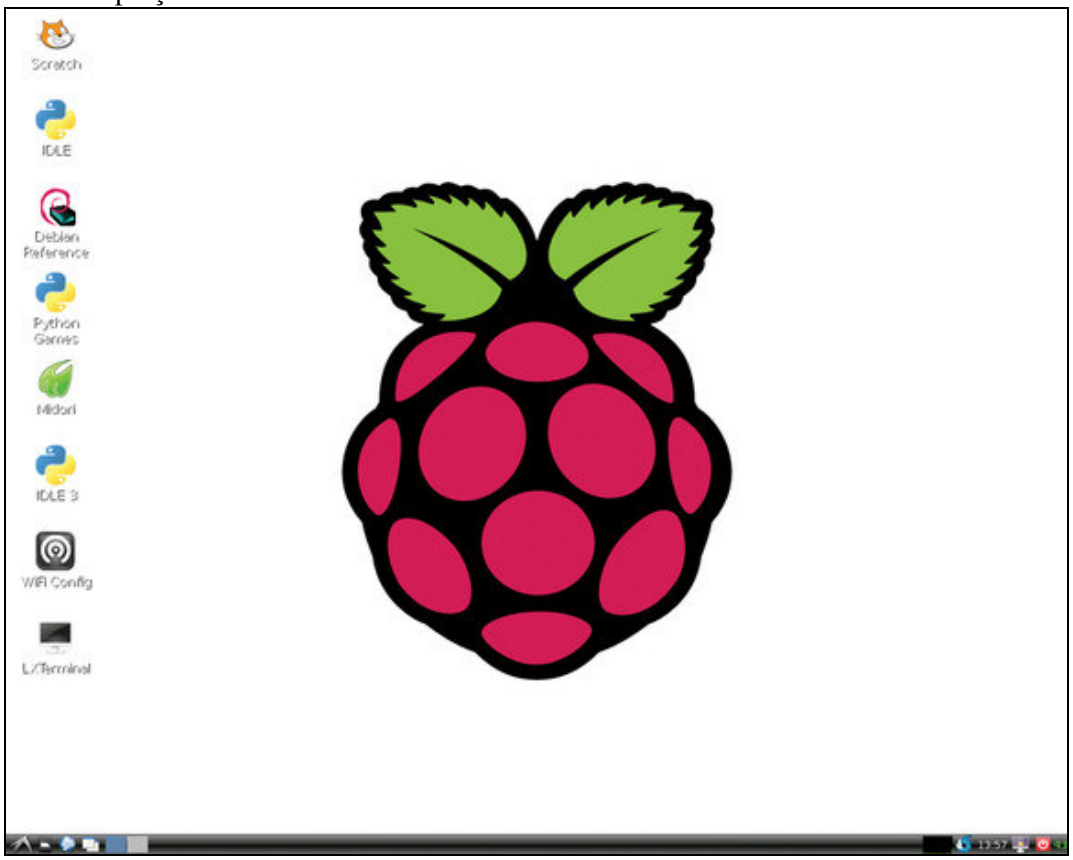
- **Modifier les options** (boot, langue,…) à l'aide des lignes de commande.
  Pour arriver sur la fenêtre de configuration, tapez « sudo raspi-config ».
  Vous pourrez alors avoir une fenêtre comme ci-dessous :



  Le plus simple est de paramétrer l'option 3 en « Boot to desktop ».

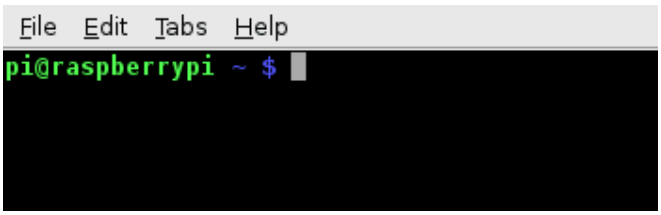
17.3 Desktop Raspbian

- Aperçu de l'environnement :



- LXTerminal
  En cliquant sur l'icône de LXTerminal, on arrive dans une fenêtre qui permet d'introduire des lignes de commandes (voir ci-dessous).

  Icône : 

  Fenêtre : 

  On peut alors taper les mêmes commandes que lorsqu'on était en mode commandes.
  Exemple : « sudo nano test.py ».

- IDLE
  De la même manière que sur un ordinateur à la maison ou au laboratoire, on peut ouvrir l'IDLE Python afin de gérer ses programmes.
  Dans certains cas, il faut lancer l'exécution du programme via le Terminal plutôt que via l'environnement python car les droits paramétrés y sont différents.

17.4 Le module d'extension PiFace

Le module d'extension PIFACE DIGITAL I/O est un module spécialement conçu pour le Raspberry. Disponible chez le fournisseur Farnell via la référence (221-8566), la carte PiFace se connecte directement sur le Raspberry et permet de mettre en œuvre des LEDS, des boutons poussoirs, des entrées/sorties supplémentaires et des relais.

Vous trouverez ci-dessous une description en anglais :

PIFACE - PIFACE DIGITAL - I/O EXPANSION BOARD, FOR RASPBERRY PI

Designed to plug on to the SPIO of your Raspberry Pi, allowing you to sense and control the real world. With PiFace Digital you can detect a state of a switch, for example from a door sensor, a pressure pad or any number of other switch types. Once this state has been detected, you can write your own software for Raspberry Pi that determines how to respond to that switch state. You can drive outputs to power motors, actuators, LEDs or anything you can imagine.

- Plugs directly onto Raspberry Pi GPIO socket
- 2 changeover relays
- 4 tactile swtiches
- 8 digital inputs
- 8 open-collector outputs
- 8 LED indicators
- Easy to program in Python, Scratch and C
- Graphical emulator and simulator

17.5 Le module d'extension Gertboard

Le module d'extension Gertboard est un module spécialement conçu pour le Raspberry. Disponible chez le fournisseur Farnell via la référence (225-0034), la carte se connecte directement sur le Raspberry et permet de mettre en œuvre des LEDS, des boutons mais surtout un ensemble d'éléments extérieurs comme des moteurs, des signaux analogiques,…

Vous trouverez ci-dessous une description en anglais :

GERTBOARD - ASSEMBLED GERTBOARD, FOR RASPBERRY PI

Gertboard is an input/output (I/O) extension board for the Raspberry Pi computer. It fits onto the GPIO pins of the Raspberry Pi (the double row of pins on the upper left corner) via a socket on the back of the Gertboard. A bit of care is required when putting the two devices together. It is easy to insert just one row of pins into the socket, but all of the pins need to be connected. The Gertboard gets its power from those GPIO pins, so user will need a power supply for the Raspberry Pi (RPi) that is capable of supplying a current of at least 1A. It is a flexible experimenter board that plugs directly into Raspberry Pi, and out into the physical world, allowing user to detect and respond to external physical events. Detect and output analogue voltages. Drive powerful motors. Detect switch presses. Illuminate LEDs and drive relays. Jumper cables allow user to hook up different parts of the circuit in many different ways, allowing total flexibility. All controlled by Raspberry Pi.

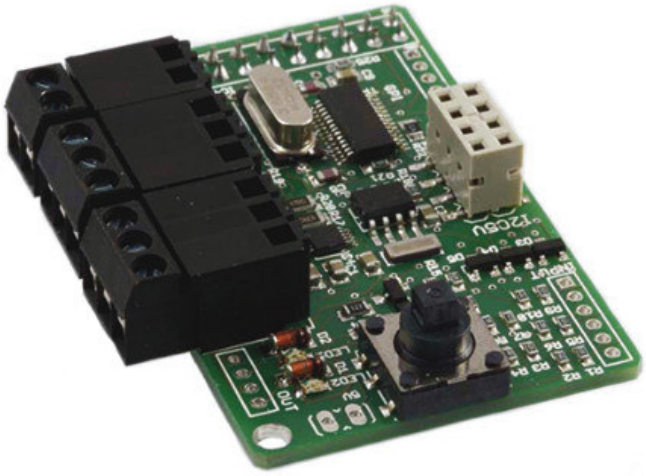
- Plugs directly onto Raspberry Pi GPIO socket
- Motor controller, capable controlling a motor bidirectionally, delivering 18V and 2A maximum
- Dual channel D/A converter, 8bit
- Dual channel A/D converter, 10bit
- Onboard Atmel ATmega328ATmega328 MCU for running off board programmes
- 6 x open collector outputs
- 12 x LED indicators
- 3 x momentary push switches
- 10 x strap cables and 18 x jumpers included
- Software and manuals available to operate and learn how to use Gertboard

17.6 Le module d'extension Raspicomm

Le module d'extension Raspicomm est un module spécialement conçu pour le Raspberry. Disponible chez le fournisseur RS via la référence (772-2974), la carte se connecte directement sur le Raspberry et permet de mettre en œuvre un joystick, une Real Time Clock ainsi que différents types de bus de communication.

Vous trouverez ci-dessous une description en anglais :

RasPiComm extension for Raspberry Pi



Main Features

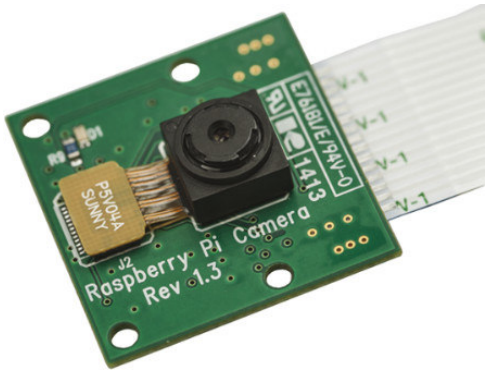
- RS-485 port – for control over stepper motors, etc.
- RS-232 port – connect to serial port devices like a modem or your PC
- I2C connector – directly connect a display or sensors
- real-time clock with battery backup
- 5-way joystick
- piggyback mounting – no cables or additional space required
- driver support and sample applications code
- fully assembled including the backup-battery

17.7 Le module d'extension Camera

Le module d'extension Camera est un module spécialement conçu pour le Raspberry. Disponible chez le fournisseur RS via la référence (775-7731), la carte se connecte sur le Raspberry par l'intermédiaire d'un « flat cable » et permet de mettre en œuvre une camera.

Vous trouverez ci-dessous une description en anglais :

Raspberry Pi Video Module Raspberry Pi Camera Board



Raspberry Pi HD Camera

High-Definition video camera for Raspberry Pi Model A, B, B+ and Compute.

Omnivision 5647 sensor in a fixed-focus module
5MPixel sensor
Integral IR filter
Still picture resolution: 2592 x 1944
Max video resolution: 1080p
Max frame rate: 30fps
Size: 20 x 25 x 10mm
Connection by flat ribbon cable to 15-pin MIPI Camera Serial Interface (CSI) connector S5 on Raspberry Pi computer board

17.8 Le module d'extension Emission / Réception (E/R)

Le module d'extension E/R est un module qui permet d'être connecté à un port USB pour permettre une communication sans fil entre différents dispositifs.
Disponible chez le fournisseur RS via la référence (783-9662), le module est un Emetteur-récepteur RF de fréquence 868 MHz. Il est pratique de disposer de HUB USB pour l'interconnexion au Raspberry.

Vous trouverez ci-dessous une description en anglais :

Low Power Radio Solutions Remote Control Base Module ERA-CONNECT2-PI, Transceiver 869.75 MHz, 915 MHz, 5V

EasyRadio Advanced Connect2 Pi,868MHz

The LPRS USB dongle offers an easy to use plug and play wireless bridge between a PC and a Raspberry Pi, Raspberry Pi/Raspberry Pi (Pi2P) or devices that support USB serial communications.
The LPRS easy Radio kit operates in the 868MHz (UK and Europe) and 915MHz (US) Industrial Scientific & Medical (ISM) bands. The ERA-CONNECT2-PI is also compatible with Arduino in conjunction with the LPRS arduino shield see RS stock number (7839616).

LPRS easyRadio RF transceiver technology
Carrier frequency: 868MHz (RS 783-9662), 868/915MHz (RS 783-9666)
Bi-directional link, no RF protocol software required
Built-in temperature sensor
Transmit & Receive LEDs
SMA antenna connector
Low power: can be powered from Raspberry Pi
Operating temperature range: -40 to +85°C
Compatible with Windows OS, Mac OS and Linux

Exercices

1 Prise en main

Les notions associées sont :
- La variable, sa déclaration et son affectation
- L'affichage de texte, d'une variable d'un type
- L'utilisation de délais
- Opérateurs, incrémentation, décrémentation
- La lecture d'une donnée avec input

1.1 Hello World

Affichez "Hello World" sur écran	Difficulté : 1

1.2 Affichage d'une variable initialisée

Commencez par déclarer une variable nommée *ma_variable* et initialisez sa valeur à 20 Affichez ensuite la valeur de *ma_variable*	Difficulté : 1

1.3 Affichage mixte et initialisation d'une variable

Commencez par déclarer une variable nommée *ma_variable* et initialisez sa valeur à 100 Affichez ensuite "La valeur est : " suivi de la valeur de *ma_variable*	Difficulté : 1

1.4 Affichage mixte et plusieurs variables

Commencez par déclarer trois variables nommées *nb1*, *nb2* et *somme* et initialisez respectivement leurs valeurs à 10, 20 et 0. Utilisez ensuite l'opérateur + pour affecter à la variable *somme* la somme de *nb1* et *nb2*. Affichez ensuite "Si on ajoute " suivi de la valeur de *nb1* suivi de " et " suivi de la valeur de *nb2* suivi de " on obtient " suivi de la valeur de *somme*	Difficulté : 1

1.5 Somme de deux nombres introduits par l'utilisateur

Commencez par déclarer trois variables nommées *nb1*, *nb2* et *somme* et initialisez les à 0. Utilisez la fonction input pour demander à l'utilisateur d'introduire la valeur de *nb1*. Utilisez la fonction input pour demander à l'utilisateur d'introduire la valeur de *nb2*. Affectez la variable *somme* de la somme des deux nombres introduits par l'utilisateur. Affichez un message convivial qui informe l'utilisateur de la valeur des deux nombres introduits ainsi que leur somme.	Difficulté : 2

1.6 Affichage mixte et affichage de type

Commencez par déclarer trois variables nommées *var_entier*, *var_reel* et *var_bool*. Affectez les respectivement de 5 , 5.0 et True Affichez ensuite la valeur de *var_entier* et son type Affichez ensuite la valeur de *var_reel* et son type Affichez ensuite la valeur de *var_bool* et son type Qu'observez-vous ?	Difficulté : 1

1.7 Décompte 5 secondes

Commencez par déclarer une variable *sec* et initialisez la à 5. Affichez "Bonjour, décompte dans 3 secondes" Attendez 3 secondes Répétez ensuite 5 fois de suite (simple copier/coller des lignes) les actions suivantes : Afficher "il reste " suivi de la valeur de *sec* suivi de "s. " Attendre une seconde Affichez "Décompte terminé"	Difficulté : 1

1.8 Différence de deux nombres introduits par l'utilisateur

Commencez par déclarer trois variables nommées *nb1*, *nb2* et *diff* et initialisez les à 0. Utilisez la fonction input pour demander à l'utilisateur d'introduire la valeur de *nb1*. Utilisez la fonction input pour demander à l'utilisateur d'introduire la valeur de *nb2*. Affectez la variable *diff* de la différence des deux nombres introduits par l'utilisateur. Affichez un message convivial qui informe l'utilisateur de la valeur des deux nombres introduits ainsi que leur différence	Difficulté : 2

1.9 Table d'un nombre introduit par l'utilisateur

Le programme consiste à demander à l'utilisateur d'introduire un nombre entre 1 et 9 et de fournir la table de multiplication de ce nombre. Imaginons par exemple que l'utilisateur introduise le nombre 5, il sera affiché : 1 fois 5 = 5 2 fois 5 = 10 … 10 fois 5 = 50 On remarque que 10 lignes sont affichées les 10 premiers multiples de 5. Commencez par déclarer trois variables nommées *multiple*, *nb* et *result* . Initialisez *multiple* à 1 et les autres variables à 0. Demandez à l'utilisateur un nombre entre 1 et 9 (sans vérifier ici si c'est correct). Tentez de trouvez les lignes d'actions qui seront répétées 10 fois de manière à afficher les 10 premiers multiples du nombre introduit par l'utilisateur. Il faut que l'ensemble de ces lignes soit identique pour constituer un bloc d'actions que nous pourrons plus tard lier à une structure répétitive.	Difficulté : 2

2 La structure de choix

Les notions associées sont :
- Les notions vue en 1.) Prise en main
- La structure de choix simple
- L' opérateur '/' sur entiers

2.1 Minimum de deux nombres

Commencez par déclarer trois variables : une variable pour le 1er nombre, une variable pour le 2ème nombre et une variable qui contiendra le plus petit des deux nombres. Demandez à l'utilisateur d'introduire la valeur du 1er nombre. Demandez à l'utilisateur d'introduire la valeur du 2ème nombre. Affectez à votre autre variable le plus petit des deux nombres introduits par l'utilisateur. Affichez la valeur des 2 nombres et un message informant la valeur du plus petit nombre.	Difficulté : 2

2.2 Addition ou soustraction de deux nombres

Commencez par déclarer cinq variables : une variable pour le 1er nombre, une variable pour le 2ème nombre et une variable pour la somme, une pour la différence et une pour le choix. Demandez à l'utilisateur d'introduire la valeur du 1er nombre. Demandez à l'utilisateur d'introduire la valeur du 2ème nombre. Affichez ensuite "Tapez 1 pour + ou n'importe quoi pour -" en mémorisant le choix. Affichez ensuite avec un message convivial fournissant : la somme des deux nombres si « 1 » a été introduit la différence des deux nombres si « 1 » n'a pas été introduit	Difficulté : 2

2.3 Produit de deux nombres différents

Demandez à l'utilisateur d'introduire la valeur du 1er nombre. Demandez à l'utilisateur d'introduire la valeur du 2ème nombre en précisant qu'il doit être différent du 1er nombre. Si les deux nombres sont différents, affichez un message convivial indiquant le produit. Si les deux nombres sont identiques, dites poliment à l'utilisateur qu'il n'a pas compris.	Difficulté : 2

2.4 Cinq fois « Bonjour » mais trois vitesses possibles

L'idée est qu'il apparaisse "Bonjour" cinq fois de suite mais avec un délai fixe entre les affichages successifs. Cependant ce délai doit être un choix de l'utilisateur qui commence par voir sur écran : Veuillez entrer la vitesse : normal (tapez 1) – lent (tapez 2) – très lent (tapez 3) Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de : 1 c'est qu'il faut un délai de 1 seconde entre chaque "Bonjour" 2 c'est qu'il faut un délai de 2 secondes entre chaque "Bonjour" 3 c'est qu'il faut un délai de 3 secondes entre chaque "Bonjour" Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris.	Difficulté : 3

2.5 Tapez 1 pour l'addition et 2 pour la soustraction

L'idée est que l'utilisateur introduise deux nombres puis choisisse l'opération + ou - Demandez à l'utilisateur d'introduire la valeur du 1er nombre. Demandez à l'utilisateur d'introduire la valeur du 2ème nombre. L'utilisateur doit ensuite voir apparaître : Veuillez entrer votre choix : addition (tapez 1) – soustraction (tapez 2) Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de : 1 c'est qu'il faut lui afficher le résultat de l'addition 2 c'est qu'il faut lui afficher le résultat de la soustraction Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris.	Difficulté : 3

2.6 Maximum de deux nombres

Demandez à l'utilisateur d'introduire deux nombres. Affichez la valeur des 2 nombres et un message informant la valeur du plus grand nombre.	Difficulté : 2

2.7 Minimum ou maximum de deux nombres

Demandez à l'utilisateur d'introduire deux nombres. Affichez ensuite "Tapez 1 pour MIN ou n'importe quoi pour MAX" en mémorisant le choix. Affichez ensuite avec un message convivial fournissant : Le minimum des deux nombres si « 1 » a été introduit Le maximum des deux nombres si « 1 » n'a pas été introduit	Difficulté : 2

2.8 Tapez 1 pour le minimum et 2 pour le maximum

L'idée est que l'utilisateur introduise deux nombres puis choisisse MIN ou MAX Demandez à l'utilisateur d'introduire deux nombres. L'utilisateur doit ensuite voir apparaître : Veuillez entrer votre choix : MIN (tapez 1) – MAX (tapez 2) Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de : 1 c'est qu'il faut lui afficher le plus petit des deux nombres 2 c'est qu'il faut lui afficher le plus grand des deux nombres Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris.	Difficulté : 3

2.9 Table d'un nombre (à un chiffre) introduit par l'utilisateur

L'idée est que l'utilisateur introduise un nombre compris entre 1 et 9 afin que le programme puisse afficher la table de ce nombre (les 10 premiers multiples). Cependant on souhaite vérifier que le nombre introduit est bien compris entre 1 et 9. Vérifiez que ce nombre est bien compris entre 1 et 9 sans utiliser d'opérateur logique. L'utilisateur doit voir apparaître : Veuillez introduire un nombre entre 1 et 9 : Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de : Un nombre entre 1 et 9, affichez la table de ce nombre. Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris.	Difficulté : 3

2.10 Différence positive de deux nombres

Demandez à l'utilisateur d'introduire deux nombres. Affichez ensuite la différence de deux nombres mais en vous assurant de faire le plus grand moins le plus petit (pour que le résultat soit positif ou nul).	Difficulté : 2

2.11 Maximum de trois nombres

Demandez à l'utilisateur d'introduire trois nombres. Affichez ensuite, sans utiliser d'opérateurs logiques, le plus grand parmi ces trois nombres.	Difficulté : 3

2.12 Parité d'un nombre

En exploitant le fait que l'opérateur '/' donne le quotient de la division (et non le résultat exact) lorsque les nombres sont entiers, essayez d'afficher la parité d'un nombre entier introduit par l'utilisateur. Astuce : Diviser un nombre pair par 2 tombe juste (le résultat fois 2 redonne le nombre).	Difficulté : 2

2.13 Trier du plus petit au plus grand

Demandez à l'utilisateur d'introduire trois nombres. Affichez ensuite, sans utiliser d'opérateurs logiques, le trois nombres triés du plus petit au plus grand.	Difficulté : 3

3 Les opérateurs logiques

Les notions associées sont :
- Les notions vue en 1.) Prise en main
- Les notions vue en 2.) La structure de choix
- Les opérateurs logiques

3.1 Trier du plus petit au plus grand

Demandez à l'utilisateur d'introduire trois nombres. Affichez ensuite, en exploitant les opérateurs logiques, le trois nombres triés du plus petit au plus grand.	Difficulté : 3

3.2 Maximum de trois nombres

Demandez à l'utilisateur d'introduire trois nombres. Affichez ensuite, en exploitant les opérateurs logiques, le plus grand parmi ces trois nombres.	Difficulté : 3

3.3 Table d'un nombre (à un chiffre) introduit par l'utilisateur

L'idée est que l'utilisateur introduise un nombre compris entre 1 et 9 afin que le programme puisse afficher la table de ce nombre (les 10 premiers multiples). Cependant on souhaite vérifier que le nombre introduit est bien compris entre 1 et 9. Vérifiez que ce nombre est bien compris entre 1 et 9 en exploitant les opérateurs logiques. L'utilisateur doit voir apparaître : Veuillez introduire un nombre entre 1 et 9 : Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de : Un nombre entre 1 et 9, affichez la table de ce nombre. Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris.	Difficulté : 3

3.4 Produit de deux nombres différents (utilisation de « not »)

Demandez à l'utilisateur d'introduire la valeur du 1er nombre. Demandez à l'utilisateur d'introduire la valeur du 2ème nombre en précisant qu'il doit être différent du 1er nombre. Si les deux nombres sont différents, affichez un message convivial indiquant le produit. Si les deux nombres sont identiques, dites poliment à l'utilisateur qu'il n'a pas compris. Une contrainte supplémentaire, vous devez utiliser l'opérateur « not » au sein de votre condition.	Difficulté : 2

3.5 Prix du billet suivant l'age de l'utilisateur

Demandez à l'utilisateur d'introduire son age et d'introduire le prix du billet. Si son age est inférieur à 24 ans ou que son age est supérieur à 65 ans il bénéficie d'une réduction de 20%. Affichez le prix qu'il devra effectivement payer.	Difficulté : 2

4 La structure répétitive « tant que »

Les notions associées sont :
- Les notions vue en 1.) Prise en main
- Les notions vue en 2.) La structure de choix
- Les notions vue en 3.) Les opérateurs logiques
- La structure répétitive « tant que »

4.1 Cinq fois « Bonjour »

L'idée est qu'il apparaisse "Bonjour" cinq fois de suite mais avec un délai de 1 seconde entre les affichages successifs. Il faut éviter de faire des copier/coller de lignes équivalentes et utiliser la structure répétitive « tant que » pour gérer les répétitions souhaitées	Difficulté : 3

4.2 Tapez 1 pour l'addition et 2 pour la soustraction

L'idée est que l'utilisateur introduise deux nombres puis choisisse l'opération + ou - Demandez à l'utilisateur d'introduire la valeur du 1er nombre. Demandez à l'utilisateur d'introduire la valeur du 2ème nombre. L'utilisateur doit ensuite voir apparaître : Veuillez entrer votre choix : addition (tapez 1) – soustraction (tapez 2) Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de : 1 c'est qu'il faut lui afficher le résultat de l'addition 2 c'est qu'il faut lui afficher le résultat de la soustraction Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris et demandez à nouveau d'introduire un choix valide (et ce tant que nécessaire).	Difficulté : 3

4.3 Cinq fois « Bonjour » mais trois vitesses possibles

L'idée est qu'il apparaisse "Bonjour" cinq fois de suite mais avec un délai fixe entre les affichages successifs. Cependant ce délai doit être un choix de l'utilisateur qui commence par voir sur écran : Veuillez entrer la vitesse : normal (tapez 1) – lent (tapez 2) – très lent (tapez 3) Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de : 1 c'est qu'il faut un délai de 1 seconde entre chaque "Bonjour" 2 c'est qu'il faut un délai de 2 secondes entre chaque "Bonjour" 3 c'est qu'il faut un délai de 3 secondes entre chaque "Bonjour" Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris et demandez à nouveau d'introduire une vitesse valide (et ce tant que nécessaire). Attention, il faut éviter de faire des copier/coller de lignes équivalentes et utiliser la structure répétitive « tant que » pour gérer les répétitions souhaitées.	Difficulté : 4

4.4 Table d'un nombre (à un chiffre) introduit par l'utilisateur

L'utilisateur doit voir apparaître : Veuillez introduire un nombre entre 1 et 9 : Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de : Un nombre entre 1 et 9, affichez la table de ce nombre. Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris et demandez à nouveau d'introduire un choix valide (et ce tant que nécessaire). Attention, il faut éviter de faire des copier/coller de lignes équivalentes et utiliser la structure répétitive « tant que » pour gérer la table complète.	Difficulté : 4

4.5 Décompte du nombre de secondes introduit par l'utilisateur

Demandez à l'utilisateur d'introduire un nombre qui correspondra au nombre de secondes souhaitées pour le décompte. Démarrez ensuite un décompte d'autant de secondes que le nombre qu'a introduit l'utilisateur.	Difficulté : 3

4.6 Décompte considérant les minutes et les secondes

Demandez à l'utilisateur d'introduire une valeur de départ pour les minutes et les secondes. Démarrez ensuite un décompte démarrant de cette valeur. Pour éviter d'attendre trop longtemps pour les tests, vous pouvez par exemple, temporairement, remplacer les délais de 1s par des délais de 0,1 s.	Difficulté : 4

4.7 Produit de deux nombres différents

Demandez à l'utilisateur d'introduire la valeur du 1er nombre. Demandez à l'utilisateur d'introduire la valeur du 2ème nombre en précisant qu'il doit être différent du 1er nombre. Si le 2ème nombre est identique au 1er, dites poliment à l'utilisateur qu'il n'a pas compris et demandez à nouveau d'introduire un nombre différent du 1er (et ce tant que nécessaire).	Difficulté : 3

4.8 Attente d'un nombre pair

Demandez à l'utilisateur d'introduire un nombre pair. Si le nombre est pair affichez cette information sur écran. Si le nombre, n'est pas pair, dites poliment à l'utilisateur qu'il n'a pas compris et demandez à nouveau d'introduire un nombre pair (et ce tant que nécessaire).	Difficulté : 3

4.9 Ping-Pong

L'idée est que l'équipe 1 va jouer contre l'équipe 2 au tennis de table. L'utilisateur doit voir apparaître : Si l'équipe 1 marque tapez 1, si l'équipe 2 marque tapez 2 Au fur et à mesure de la partie le score (X-Y) doit se mettre à jour (X représente les points de l'équipe 1 et Y les points de l'équipe 2) et ce tant que la partie n'est pas terminée. Si l'utilisateur n'a pas enfoncé un choix correct (1 ou 2), il faut lui demander de recommencer son choix. Les points s'incrémentent chaque fois d'une unité jusqu'à ce que la partie soit terminée. La partie se termine dés qu'une équipe atteint 11 points avec au moins deux points d'écarts sur l'autre équipe. Une fois la partie terminée, vous devez afficher sur écran le nom du vainqueur.	Difficulté : 4

4.10 Ping-Pong et 1, 2 ou 3 sets gagnants

Le principe est le même que l'exercice précédent mais l'utilisateur doit commencer par préciser le nombre de sets gagnants souhaités (1, 2 ou 3). Si l'utilisateur n'a pas enfoncé un choix correct (1, 2 ou 3), il faut lui demander de recommencer son choix. Une fois le nombre de sets gagnants introduits, il faudra mettre à jour le score actuel en précisant le nombre de sets gagnés pour chaque équipe et le score du set en cours. A la fin de chaque set il faudra indiquer le vainqueur du set et mettre à jour les sets gagnés de chaque équipe. Une fois qu'une équipe a atteint le nombre de sets à gagner, il faudra indiquer l'équipe qui a gagné le match.	Difficulté : 5

4.11 Décomposition d'un nombre en produit de nombres premiers

Demandez à l'utilisateur d'introduire un nombre compris entre 1 et 30. Si le nombre introduit n'est pas un entier entre 1 et 30, dites poliment à l'utilisateur qu'il n'a pas compris et demandez à nouveau d'introduire le nombre (et ce tant que nécessaire). Une fois qu'un nombre valide a été introduit, tentez de décomposez ce nombre en produit de nombres premiers sachant que les nombres premiers compris entre 1 et 30 sont : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29. Essayez de résoudre cet exercice avec les outils actuels.	Difficulté : 5

4.12 Somme de n nombres

Demandez à l'utilisateur d'introduire le nombre de nombres dont il vaut calculer la somme. Stockez ce nombre dans une variable n. Demandez ensuite à l'utilisateur d'introduire les n nombres dont on doit calculer la somme. Affichez les n nombres et leur somme.	Difficulté : 3

4.13 Devinez le nombre introduit par l'utilisateur précédent

Imaginons un programme destiné à un jeu « devinez le nombre que l'utilisateur précédent a introduit». Le programme se déroule en deux étapes : - la 1ère étape le 1er utilisateur introduit un nombre entre 1 et 100 - La 2ème étape le 2ème utilisateur tente de trouver ce nombre avec le minimum d'essais. A chaque essai le programme renvoie « trop petit » ou « trop grand » selon le cas. Dés que le nombre est correct il est indiqué en combien d'essais il a été trouvé.	Difficulté : 4

5 Fonction sans paramètre

Les notions associées sont : - Les notions vue en 1.), 2.), 3.), et 4.)
- Les notions de base sur les fonctions sans paramètre

5.1 Une fonction pour Hello World

Définissez une fonction nommée fct_HW dont le rôle est d'afficher "Hello World" sur écran Appelez ensuite trois fois de suite la fonction au sein du programme principal.	Difficulté : 2

5.2 Une fonction decompte_5s

Définissez une fonction nommée decompte_5s dont le rôle est de faire apparaître un décompte de 5 à 0 par intervalles de 1seconde. Affichez ensuite « ça va commencer … » puis appeler la fonction. Affichez encore« et encore un décompte… » et appeler à nouveau la fonction.	Difficulté : 3

5.3 Une fonction affiche_20_facteurs_table7

Définissez une fonction nommée affiche_20_facteurs_table7 dont le rôle est de faire apparaître la table de multiplication de 7 allant de 1 à 20. Utilisez ensuite cette fonction au sein de votre programme principal pour s'assurer de son bon fonctionnement.	Difficulté : 3

5.4 Trois fonctions bonjour_5

Définissez trois fonctions proches mais pourtant différentes. Les trois fonctions doivent afficher « Bonjour » 5 fois de suite mais l'intervalle de temps entre chaque affichage n'est pas identique. Les trois fonctions sont donc : bonjour5_norm() → Affichage des « Bonjour » à vitesse normale (délais de 1s.) bonjour5_lent() → Affichage des « Bonjour » à vitesse lente (délais de 2s.) bonjour5_tres_lent() → Affichage des « Bonjour » à vitesse très lente (délais de 3s.) Ensuite l'utilisateur doit voir apparaître sur écran : Veuillez entrer la vitesse : normal (tapez 1) – lent (tapez 2) – très lent (tapez 3) Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de : 1 c'est qu'il faut un délai de 1 seconde entre chaque "Bonjour" 2 c'est qu'il faut un délai de 2 secondes entre chaque "Bonjour" 3 c'est qu'il faut un délai de 3 secondes entre chaque "Bonjour" Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris. Gérez ce cas en utilisant les fonctions définies avant.	Difficulté : 3

5.5 Ping-Pong en utilisant une fonction et des variables globales

L'idée est que l'équipe 1 va jouer contre l'équipe 2 au tennis de table. L'utilisateur doit voir apparaître : Si l'équipe 1 marque tapez 1, si l'équipe 2 marque tapez 2 Au fur et à mesure de la partie le score (X-Y) doit se mettre à jour (X représente les points de l'équipe 1 et Y les points de l'équipe 2) et ce tant que la partie n'est pas terminée. Si l'utilisateur n'a pas enfoncé un choix correct (1 ou 2), il faut lui demander de recommencer son choix. Les points s'incrémentent chaque fois d'une unité jusqu'à ce que la partie soit terminée. La partie se termine dés qu'une équipe atteint 11 points avec au moins 2 points d'écarts sur l'autre équipe (dans certains cas il faudra plus de 11 points pour avoir 2 points d'écarts). Une fois la partie terminée, vous devez afficher sur écran le nom du vainqueur. Attention, l'exercice ici consiste également à mettre en œuvre une fonction, cette fonction est nommée « affiche_score() » qui ne reçoit pas de paramètre mais qui pourra afficher le score parce que les variables liées au score sont des variables globales. Essayer de voir les avantages et inconvénients de cette méthode.	Difficulté : 4

5.6 Ping-Pong et 1, 2 ou 3 sets gagnants utilisant une fonction pour le score

Le principe est le même que l'exercice précédent mais l'utilisateur doit commencer par préciser	Difficulté : 5

le nombre de sets gagnants souhaités (1, 2 ou 3). Si l'utilisateur n'a pas enfoncé un choix correct (1, 2 ou 3), il faut lui demander de recommencer son choix. Une fois le nombre de sets gagnants introduits, il faudra mettre à jour le score actuel en précisant le nombre de sets gagnés pour chaque équipe et le score du set en cours. A la fin de chaque set il faudra indiquer le vainqueur du set et mettre à jour les sets gagnés de chaque équipe. Une fois qu'une équipe a atteint le nombre de sets à gagner, il faudra indiquer l'équipe qui a gagné le match. Attention, tout comme l'exercice précédent, l'exercice ici consiste à mettre en œuvre une fonction, cette fonction est nommée « affiche_score() » qui ne reçoit pas de paramètre mais qui pourra afficher le score parce que les variables liées au score sont des variables globales. Le score est maintenant le score du match en cours, à savoir, les sets gagnés pour chaque équipe et les points du set en cours. Essayer de voir les avantages et inconvénients de cette méthode.	

6 Fonction avec un paramètre, sans valeur de retour

Les notions associées sont : - Les notions vue de 1.) → 5.)
- Les fonctions avec un paramètre

6.1 Affiche le carré d'un nombre

Définissez une fonction (choisissez un nom évocateur) qui reçoit un nombre comme paramètre et qui affiche un message informant de la valeur du carré de ce nombre. Au sein du programme principal qui sera exécuté, faites un test de fonctionnement de la fonction en affichant le carré d'une constante suivi de l'affichage d'un nombre introduit par l'utilisateur.	Difficulté : 2

6.2 Affiche la parité d'un nombre

Définissez une fonction (choisissez un nom évocateur) qui reçoit un nombre comme paramètre et qui affiche un message informant de la parité de ce nombre. Au sein du programme principal qui sera exécuté, faites un test de fonctionnement de la fonction en affichant la parité d'une constante paire (au choix), puis la parité d'une constante impaire (au choix) et enfin la parité d'un nombre introduit par l'utilisateur.	Difficulté : 3

6.3 Affiche un nombre et son opposé

Définissez une fonction (choisissez un nom évocateur) qui reçoit un nombre comme paramètre et qui affiche un message reprenant la valeur de ce nombre suivi de la valeur de son opposé. Au sein du programme principal qui sera exécuté, faites un test de fonctionnement de la fonction en demandant à l'utilisateur d'introduire un nombre puis en faisant appel à votre fonction pour afficher la valeur de ce nombre et de son opposé.	Difficulté : 2

6.4 Affiche un nombre et son inverse

Définissez une fonction (choisissez un nom évocateur) qui reçoit un nombre comme paramètre et qui affiche un message reprenant la valeur de ce nombre suivi de la valeur de son inverse (l'inverse d'un nombre est 1 divisé par ce nombre). Au sein du programme principal qui sera exécuté, faites un test de fonctionnement de la fonction	Difficulté : 3

en demandant à l'utilisateur d'introduire un nombre puis en faisant appel à votre fonction pour afficher la valeur de ce nombre et de son inverse. Attention l'opérateur « / » sur entier donne le quotient et non le résultat exact de la division. D'ailleurs la valeur « 1 » est considérée comme entier mais « 1.0 » comme réel.	

6.5 Affiche n fois bonjour où n est le paramètre

Définissez une fonction (choisissez un nom évocateur) qui reçoit un nombre comme paramètre et qui affiche « Bonjour » autant de fois que la valeur du paramètre reçu. Au sein du programme principal qui sera exécuté, faites un test de fonctionnement de la fonction en demandant à l'utilisateur d'introduire un nombre puis en faisant appel à votre fonction pour afficher « Bonjour » autant de fois que le nombre introduit.	Difficulté : 3

6.6 Trois fonctions bonjour avec paramètre

Définissez trois fonctions proches mais pourtant différentes. Les trois fonctions doivent afficher « Bonjour » autant de fois de suite que le paramètre introduit mais l'intervalle de temps entre chaque affichage n'est pas identique. Les trois fonctions sont donc : bonjour_norm(…) → Affichage des « Bonjour » à vitesse normale (délais de 1s.) bonjour_lent(…) → Affichage des « Bonjour » à vitesse lente (délais de 2s.) bonjour_tres_lent(…) → Affichage des « Bonjour » à vitesse très lente (délais de 3s.) Les « … » des fonctions montrent l'utilisation d'un paramètre (le nombre de répétions). Ensuite l'utilisateur doit voir apparaître sur écran : Veuillez entrer la vitesse : normal (tapez 1) – lent (tapez 2) – très lent (tapez 3) Le choix de l'utilisateur doit bien sûr être mémorisé, s'il s'agit de : 1 c'est qu'il faut un délai de 1 seconde entre chaque "Bonjour" 2 c'est qu'il faut un délai de 2 secondes entre chaque "Bonjour" 3 c'est qu'il faut un délai de 3 secondes entre chaque "Bonjour" Une autre valeur, dites poliment à l'utilisateur qu'il n'a pas compris. Gérez ce cas en utilisant les fonctions définies avant.	Difficulté : 4

6.7 Fonction décompte avec paramètre

Définissez une fonction (choisissez un nom évocateur) qui provoque un décompte (décrémentation à chaque seconde) qui commence à partir du nombre passé en paramètre. Testez ensuite votre programme en commençant par demander à l'utilisateur d'introduire un nombre et en démarrant le décompte à partir de ce nombre.	Difficulté : 3

6.8 Affiche la table du paramètre

Définissez une fonction dont le rôle est d'afficher la table (jusqu'à 10) du nombre passé en paramètre. Testez ensuite votre programme en demandant à l'utilisateur d'introduire un nombre et en faisant appel à votre fonction avec ce nombre comme paramètre.	Difficulté : 3

6.9 Affiche la table de 3 jusqu'au paramètre

Définissez une fonction dont le rôle est d'afficher la table de 3 mais plus forcément jusqu'à 10, le nombre de facteurs peut être différent. La fonction doit alors avoir un paramètre qui est le nombre de facteurs qu'il faudra afficher pour la table de 3. Testez ensuite votre programme en demandant à l'utilisateur d'introduire un nombre (le nombre de facteurs) et en faisant appel à votre fonction avec ce nombre comme paramètre.	Difficulté : 3

6.10 Décompte sous forme « mm :ss » suivant le nombre de secondes reçues

L'idée est d'avoir une fonction qui est prête à accepter un nombre supérieur à 60 comme paramètre et de réaliser un décompte indiquant les minutes et les secondes. Le décompte devra être mis sous la forme « mm :ss » où « mm » représente les deux chiffres possibles des minutes en cours et « ss » les deux chiffres possibles des secondes en cours. Testez ensuite votre programme en demandant à l'utilisateur d'introduire un nombre (le nombre de secondes) et en faisant appel à votre fonction pour tester le décompte. Remarque : Un test peut s'avérer être très long, il alors est intéressant, juste pour le test, de par exemple modifier le délai de 1seconde en un délai de 0,1 seconde. On écrira alors time.sleep(0.1) pour gagner du temps de test (on remet « 1 » une fois testé).	Difficulté : 4

6.11 Décomposition d'un nombre en produit de nombres premiers

Définissez et utilisez une fonction qui affiche la décomposition d'un nombre en un produit de nombres premiers. On suppose que le nombre à décomposer est compris entre 1 et 30. La fonction reçoit le nombre à décomposer comme paramètre. Demandez à l'utilisateur d'introduire un nombre compris entre 1 et 30. Si le nombre introduit n'est pas un entier entre 1 et 30, dites poliment à l'utilisateur qu'il n'a pas compris et demandez à nouveau d'introduire le nombre (et ce tant que nécessaire). Une fois qu'un nombre valide a été introduit, tentez de décomposez ce nombre en produit de nombres premiers sachant que les nombres premiers compris entre 1 et 30 sont : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29. Essayez de résoudre cet exercice avec les outils actuels.	Difficulté : 5

7 Fonction avec plusieurs paramètres

Les notions associées sont : - Les notions vue de 1.) → 6.)
- Les fonctions avec plusieurs paramètres

7.1 Affiche le minimum de deux paramètres

La fonction doit recevoir deux paramètres et afficher le plus petit des deux paramètres. Définissez la fonction et utilisez la au sein du programme en demandant à l'utilisateur d'introduire les deux nombres dont on voudrait déterminer le minimum.	Difficulté : 3

7.2 Affiche les trois paramètres par ordre croissant

La fonction doit recevoir trois paramètres et afficher ces paramètres par ordre croissant. Définissez la fonction et utilisez la au sein du programme en demandant à l'utilisateur d'introduire les trois nombres qu'on voudrait afficher par ordre croissant.	Difficulté : 3

7.3 Affiche des bonjour mais deux paramètres

L'idée est de créer une fonction (choisissez un nom évocateur) qui a deux paramètres : Le 1er indique le nombre de répétitions de « bonjour » Le 2ème indique le délai en secondes entre les différentes répétitions. Définissez cette fonction et mettez la en œuvre pour que l'utilisateur puisse choisir le nombre de répétitions ainsi que le délai souhaité entre les répétitions. Testez ensuite des possibilités comme (3,4) et (10,1) comme combinaisons de paramètres.	Difficulté : 3

7.4 Affiche la table en utilisant deux paramètres

L'idée est de créer une fonction qui, en quelque sorte, généralise les exercices 6.8 et 6.9. La fonction contient deux paramètres : Le 1er indique le nombre dont on veut afficher la table Le 2ème indique le nombre de facteurs souhaités (jusqu'où on affiche la table) Définissez cette fonction et mettez la en œuvre pour que l'utilisateur puisse choisir quel nombre utiliser pour la table ainsi que le nombre de facteurs souhaités. Testez ensuite des possibilités comme (3,10), (3,20), (3,8) et (8,3) comme combinaisons de paramètres.	Difficulté : 3

7.5 Ping-Pong en utilisant une fonction sans manipuler de variable globale

L'idée est que l'équipe 1 va jouer contre l'équipe 2 au tennis de table. L'utilisateur doit voir apparaître : Si l'équipe 1 marque tapez 1, si l'équipe 2 marque tapez 2 Au fur et à mesure de la partie le score (X-Y) doit se mettre à jour (X représente les points de l'équipe 1 et Y les points de l'équipe 2) et ce tant que la partie n'est pas terminée. Si l'utilisateur n'a pas enfoncé un choix correct (1 ou 2), il faut lui demander de recommencer son choix. Les points s'incrémentent chaque fois d'une unité jusqu'à ce que la partie soit terminée. La partie se termine dés qu'une équipe atteint 11 points avec au moins 2 points d'écarts sur l'autre équipe (dans certains cas il faudra plus de 11 points pour avoir 2 points d'écarts). Une fois la partie terminée, vous devez afficher sur écran le nom du vainqueur. Attention, l'exercice ici consiste également à mettre en œuvre une fonction qui reçoit comme paramètres les points des équipes 1 et 2 et qui affiche sur écran le score en cours. Essayer de voir les avantages et inconvénients de cette méthode.	Difficulté : 4

7.6 Ping-Pong et 1, 2 ou 3 sets gagnants utilisant une fonction pour le score

Le principe est le même que l'exercice précédent mais l'utilisateur doit commencer par préciser le nombre de sets gagnants souhaités (1, 2 ou 3). Si l'utilisateur n'a pas enfoncé un choix correct (1, 2 ou 3), il faut lui demander de recommencer son choix. Une fois le nombre de sets gagnants introduits, il faudra mettre à jour le score actuel en précisant le nombre de sets gagnés pour chaque équipe et le score du set en cours. A la fin de chaque set il faudra indiquer le vainqueur du set et mettre à jour les sets gagnés de chaque équipe. Une fois qu'une équipe a atteint le nombre de sets à gagner, il faudra indiquer l'équipe qui a gagné le match. Attention, tout comme l'exercice précédent, il faut utiliser une fonction qui reçoit suffisamment de paramètres pour afficher tous les détails du match en cours.	Difficulté : 5

7.7 Affiche le max des trois paramètres

Définissez une fonction qui affiche le maximum parmi trois nombres passés comme paramètre. Imaginez ensuite, au sein de votre programme principal, une demande d'introduction de ces trois nombres auprès de l'utilisateur afin de pouvoir tester le bon fonctionnement de votre fonction.	Difficulté : 3

7.8 Affiche le décompte recevant les minutes et les secondes

Définissez une fonction qui a deux paramètres : Le 1er indique les minutes Le 2ème indique les secondes Et qui gère un compteur sous la forme « mm :ss », « mm » représente les deux chiffres possibles des minutes en cours	Difficulté : 4

« ss » les deux chiffres possibles des secondes en cours. Votre programme principal doit alors demander à l'utilisateur d'introduire les minutes et les secondes pour que puisse démarrer le compteur après un appel de votre fonction. Remarque : Un test peut s'avérer être très long, il alors est intéressant, juste pour le test, de par exemple modifier le délai de 1seconde en un délai de 0,1 seconde. On écrira alors time.sleep(0.1) pour gagner du temps de test (on remet « 1 » une fois testé).	

7.9 Affiche la moyenne de quatre paramètres

Définissez une fonction qui affiche la moyenne des quatre nombres passés en paramètre. Imaginez ensuite, au sein de votre programme principal, une demande d'introduction de ces nombres auprès de l'utilisateur afin de pouvoir tester le bon fonctionnement de votre fonction. Attention, le résultat de « a/b » risque de ne pas tomber juste si « a » est un entier. Par contre le fait de multiplier un entier par un réel permettra d'interpréter le produit comme un réel.	Difficulté : 3

*7.10 Affiche un produit sans l'opérateur « * »*

Définissez une fonction qui affiche le produit de deux nombres entiers passés en paramètres. Attention, vous n'avez pas le droit d'utiliser l'opérateur « * ». Rem : Une multiplication peut être exprimée sous la forme d'une succession d'additions.	Difficulté : 3

*7.11 Affiche une puissance sans l'opérateur « ** »*

Définissez une fonction qui affiche une puissance créée à partir des deux nombres entiers passés en paramètres. Le premier nombre est la base et le deuxième est l'exposant. Attention, vous n'avez pas le droit d'utiliser l'opérateur « ** ». Rem : Une puissance peut être exprimée sous la forme d'une succession de multiplications.	Difficulté : 3

8 Fonction avec paramètre(s) et une valeur de retour

Les notions associées sont : - Les notions vue de 1.) → 7.)
- Les fonctions avec une valeur de retour

8.1 Renvoyer le minimum des deux paramètres

La fonction doit recevoir deux paramètres et renvoyer le plus petit des deux paramètres. Définissez la fonction et utilisez la au sein du programme en demandant à l'utilisateur d'introduire les deux nombres dont on voudrait déterminer le minimum.	Difficulté : 3

8.2 Renvoyer le maximum des trois paramètres

La fonction doit recevoir trois paramètres et renvoyer le plus grand des trois paramètres. Définissez la fonction et utilisez la au sein du programme en demandant à l'utilisateur d'introduire les trois nombres dont on voudrait déterminer le maximum.	Difficulté : 3

8.3 Renvoyer la moyenne des quatre paramètres

Définissez et mettez en œuvre une fonction qui renvoie la moyenne des quatre nombres passés en paramètre. Attention avec l'opérateur « / », un résultat tel que « a/b » risque de ne pas tomber juste si « a » est un entier.	Difficulté : 3

Par contre multiplier un entier par un réel fera interpréter le produit comme un réel.	

8.4 Renvoyer le produit sans l'opérateur « * »

Définissez une fonction qui renvoie le produit de deux nombres entiers passés en paramètres. Attention, vous n'avez pas le droit d'utiliser l'opérateur « * ». Rem : Une multiplication peut être exprimée sous la forme d'une succession d'additions.	Difficulté : 3

8.5 Renvoyer une puissance sans l'opérateur « ** »

Définissez une fonction qui renvoie une puissance créée à partir des deux nombres entiers passés en paramètres. Le premier nombre est la base et le deuxième est l'exposant. Attention, vous n'avez pas le droit d'utiliser l'opérateur « ** ». Rem : Une puissance peut être exprimée sous la forme d'une succession de multiplications.	Difficulté : 3

8.6 Renvoyer un booléen pour la parité

Définissez une fonction qui renvoie un booléen pour indiquer si son paramètre est pair ou impair. Le booléen devra renvoyer la valeur « True » si le paramètre est pair et la valeur « False » s'il ne l'est pas. Mettez ensuite en œuvre cette fonction pour afficher un message indiquant à l'utilisateur si le nombre qu'il vient d'introduire est pair ou impair.	Difficulté : 4

8.7 Renvoyer le résultat d'une opération parmi quatre

Définissez une fonction qui a trois paramètres ; Le 1er indique l'opération souhaitée Le 2ème indique le 1er nombre à prendre en compte pour l'opération Le 3ème indique le 2ème nombre à prendre en compte pour l'opération Le 1er paramètre a quatre valeurs différentes possibles : « 1 » pour une addition, on fait le 2ème paramètre + le 3ème paramètre « 2 » pour une soustraction, on fait le 2ème paramètre - le 3ème paramètre « 3 » pour une multiplication, on fait le 2ème paramètre * le 3ème paramètre « 4 » pour une multiplication, on fait le 2ème paramètre ** le 3ème paramètre Tout autre valeur doit engendrer un message d'erreur à l'utilisateur et demander de réintroduire un choix correct. Trouvez un moyen de mettre en œuvre cette fonction de manière à susciter l'intervention de l'utilisateur.	Difficulté : 4

8.8 Renvoyer le résultat d'une fonction logique donnée

Un circuit combinatoire est l'équivalent de l'expression booléenne suivante : $S = \overline{a} \cdot b + \overline{b \cdot c} + \overline{a} \cdot \overline{b} \cdot \overline{c}$ Définissez une fonction qui renvoie S en tenant compte des trois paramètres a, b et c. Remarques : a, b, c et S sont des variables booléennes un trait au dessus correspond à l'opérateur logique « NOT » une multiplication est l'équivalent de l'opérateur logique « and » une addition est l'équivalent de l'opérateur logique « or » Trouvez ensuite une manière conviviale de tester votre fonction.	Difficulté : 3

8.9 Poules et moutons

Un fermier a une drôle de façon de compter ses poules. Le fermier possède des poules et des moutons et pour déterminer le nombre de poules il compte le nombre total de têtes (poule comme mouton) ainsi que le nombre total de pattes (poule comme mouton également). Ensuite il note sur un morceau de papier les deux nombres : Le 1er nombre est le nombre de têtes Le 2ème nombre est le nombre de pattes Il détermine enfin par raisonnement le nombre de poules présentes à partir de son morceau de papier. Cependant sa femme, informaticienne, souhaite définir une fonction qui renvoie le nombre de poules présentes à partir des deux nombres passés en paramètres. Définissez et mettez en œuvre une telle fonction sachant que le 1er paramètre est le nombre de têtes et le 2ème le nombre de pattes.	Difficulté : 3

9 Fonction avec paramètre et plusieurs valeurs de retour

Les notions associées sont : - Les notions vue de 1.) → 8.)
- Les fonctions avec plusieurs valeurs de retour

Il faut savoir qu'il n'est pas habituel en programmation d'avoir plusieurs valeurs de retour. Le langage python permet cet artifice grâce au principe d'affectation multiple.

9.1 Fonction swap

Définissez une fonction qui reçoit deux paramètres et qui renvoie ces deux mêmes paramètres mais dans l'ordre inverse. Une affectation multiple combinée à un appel de fonction permet alors de faire ce qu'on appelle un « swap », c'est-à-dire de commuter les valeurs des deux variables.	Difficulté : 3

9.2 Renvoyer les trois paramètres ordonnés

Définissez une fonction qui reçoit trois paramètres et qui renvoie ces trois mêmes paramètres ordonnés du plus petit au plus grand. Trouvez ensuite une manière conviviale de tester votre fonction.	Difficulté : 3

9.3 Renvoyer les trois paramètres ordonnés

Définissez une fonction qui reçoit un entier représentant le nombre total de secondes et qui a deux valeurs de retour : La 1ère représente les minutes La 2ème représente les secondes L'idée est de s'aider de cette fonction pour réaliser un décompte sous la forme « mm : ss ». « mm » représente les deux chiffres possibles des minutes en cours « ss » les deux chiffres possibles des secondes en cours. Trouvez ensuite une manière conviviale de tester votre fonction au sein d'un décompte.	Difficulté : 4

9.4 Renvoyer le résultat d'une opération parmi quatre avec une variable d'état

Définissez une fonction qui a deux valeurs de retour, la 1ère est un booléen indiquant si la fonction s'est déroulée sans erreur et la 2ème est le résultat de l'opération. L'idée du paramètre booléen ('True' s'il n'y a pas eu d'erreurs, 'False' autrement) est de pouvoir gérer, en dehors de la fonction, la demande d'introduction des valeurs tant que celle-ci n'est pas correcte. Sachant également que la fonction a trois paramètres ; Le 1er indique l'opération souhaitée Le 2ème indique le 1er nombre à prendre en compte pour l'opération Le 3ème indique le 2ème nombre à prendre en compte pour l'opération Que le 1er paramètre a quatre valeurs différentes possibles : « 1 » pour une addition, on fait le 2ème paramètre + le 3ème paramètre « 2 » pour une soustraction, on fait le 2ème paramètre - le 3ème paramètre « 3 » pour une multiplication, on fait le 2ème paramètre * le 3ème paramètre « 4 » pour une multiplication, on fait le 2ème paramètre ** le 3ème paramètre Tout autre valeur doit engendrer une valeur de retour 'False' pour le 1er paramètre de retour. Trouvez un moyen de gérer une introduction correcte des données de l'utilisateur en mettant en œuvre cette fonction, affichez ensuite le résultat de l'opération.	Difficulté : 4

9.5 Poules et moutons avec une variable d'état

Le problème est le même que l'exercice 8.9 (poules et moutons) sauf qu'ici on souhaite avoir deux valeurs de retour : La 1ère est un booléen indiquant si le résultat est possible ('True' si c'est possible) La 2ème représente le nombre de poules Dans certains cas, les valeurs des paramètres introduits ne sont pas possibles (par exemple 5 têtes et 2 pattes), il faut alors mettre la 1ère valeur de retour à 'False' pour permettre à l'utilisateur de corriger les données. Trouvez une manière conviviale de tester votre fonction en assurant une introduction de données correctes par l'utilisateur.	Difficulté : 4

10 Les listes

10.1 Liste de noms

Définissez une fonction qui demande le nom et le prénom. Ecrire un programme qui demande, si on affiche la liste existante, puis si on encode un nouveau nom, le nouveau nom doit être ajouté à la liste.	Difficulté : 3

10.2 Liste de plaque d'immatriculation

Le principe est le même que l'exercice précèdent. Définissez une fonction qui demande la plaque d'immatriculation et la date de 1er mise en circulation (véhicule). Ecrire un programme qui demande, si on affiche la liste existante, puis si on encode un nouveau véhicule, le nouveau véhicule doit être ajouté à la liste. Pour votre facilité démarrer avec une liste préétablie de trois ou quatre véhicules.	Difficulté : 3

10.3 Triez la liste de plaque d'immatriculation

L'exercice complète l'exercice précèdent. Il faut pouvoir trier la liste des véhicules par date décroissante de mise en circulation.	Difficulté : 4

10.4 Gestion de la liste de plaque d'immatriculation

A partir de l'exercice précédent, Ecrire une fonction qui permet de sélectionner tous les véhicules immatriculés entre deux dates. Ecrire une fonction qui permet d'effacer tous les véhicules immatriculés entre deux dates.	Difficulté : 4

11 Les Fichiers

11.1 Création d'un fichier

Ecrire un programme qui créé le fichier « data_base.txt » dans un répertoire existant. Ecrire une liste dans le fichier. A l'aide du bloc-note ou de word, vérifier l'existence du fichier et son contenu.	Difficulté : 3

11.2 Création d'un fichier dans un nouveau répertoire

Ecrire un programme qui créé un fichier « date_base.txt » dans un nouveau répertoire appelé « dossier_perso ». Ecrire une liste dans le fichier : liste=['jean',' pierre',' jules'] A l'aide du bloc-note ou de word, vérifier l'existence du fichier et son contenu.	Difficulté : 3

11.3 Test l'existence d'un fichier

Modifier le programme existant qui teste d'abord l'existence du fichier avant de le créer.	Difficulté : 3

11.4 Remplacement d'un fichier existant

A partir du programme précédent, remplacer la liste existante par la nouvelle liste : liste=['anne','juliette','aurélie']	Difficulté : 3

11.5 Modification d'un fichier existant

A partir du programme précédent, A l'aide d'une fonction ajouter un nom dans la liste si le nom n'est pas encore présent.	Difficulté : 3

11.6 Gestion d'un fichier

A partir du programme précédent, A l'aide d'une fonction ajouter un nom dans la liste et créer un nouveau fichier indexé (ce qui permet de garder le fichier précédent non modifié).	Difficulté : 4

Manipulations en Python

Les différentes manipulations en python sont conçues pour une durée de 2*50 minutes.

Ces manipulations sont prévues pour être cotées ou non ; certaines sont d'ailleurs des exemplaires d'interrogations repris tels quels.

Certaines manipulations sont plus conséquentes et ont donc été divisées en plusieurs séances de 100 minutes.

Certaines manipulations sont corrigées, cependant il est très fortement conseillé de se pencher sur la correction qu'après avoir réalisé la manipulation par soi-même !

Manip 1.1 : Série1, les listes – 1.) Introduction

1.) On voudrait utiliser les fonctionnalités des listes pour gérer les cotes des élèves dans une classe. Pour écourter l'encodage, on supposera une classe avec 5 élèves seulement. L'idée est de gérer l'encodage des noms et de deux cotes ce ces élèves.

Gérez les étapes ci-dessous dans l'ordre :

- Les 5 noms d'élèves sont introduits par l'utilisateur et stockés dans une liste.
- Les 5 cotes respectives (dans le même ordre des élèves) sont stockées dans une liste.
- 5 autres cotes (toujours dans l'ordre des élèves) sont stockées dans une liste.
- Affichez un compte rendu pour l'ensemble des élèves formaté comme ci-dessous :

  « Durand a obtenu les cotes de 4 et 9, ce qui donne un total de 13/20 »

  Où les éléments soulignés proviennent des trois listes créées précédemment.
  La cote totale est déduite des deux cotes des deux listes.

2.) Gestion d'une base de données d'une bibliothèque à l'aide de listes

a.) Gestion de l'identifiant

Nous appellerons l'identifiant la clé unique qui permet d'identifier une donnée dans une liste. Bien entendu l'indice de la liste permet déjà de jouer ce rôle mais ici on imaginera l'identifiant comme le numéro de référence d'un livre d'une bibliothèque.

Même s'il existe en python une structure de donnée plus adaptée (les dictionnaires), nous allons prendre en charge la gestion des ouvrages de la bibliothèque à l'aide de listes.

Dans un premier temps nous n'allons gérer que les identifiants des ouvrages.
L'utilisateur doit pouvoir alors ajouter ou supprimer des identifiants d'ouvrages.

L'idée est que le programme demande régulièrement à l'utilisateur :
« Avez-vous fini votre gestion ? o/n ? ».
Il faut alors que, tant que l'utilisateur n'a pas fini sa gestion, le programme permette d'ajouter et/ou de supprimer un identifiant. L'ajout d'un identifiant ne pourra alors se faire que si cet identifiant n'existe pas encore au sein de la liste. Suivant la même idée, la suppression d'un identifiant ne pourra se faire que si celui-ci existe déjà dans la liste.

A partir du moment où l'utilisateur répond oui (o) à la question « Avez-vous fini votre gestion ? o/n ? », le programme doit afficher le contenu de la liste avec tous les identifiants encodés.

b.) Gestion complète

La gestion complète permet de gérer des données d'ouvrages suivant 4 champs :
L'identifiant (cf point précédent), l'auteur, le titre de l'ouvrage, l'année d'édition.
Il est possible de créer une liste de listes pour stocker les données.

Manip 1.2 : Série1, les listes – 2.) Traitements

Dans un premier temps essayer de gérer les traitements ci-dessous sans utiliser des outils prédéfinis sur les listes (comme sort(),…) afin de vous entraîner à gérer le traitement par vous-même.

1.) Gestion d'une liste de nombres entiers

Commencer par créer une liste de nombres entiers de manière automatique afin de pouvoir tenter d'effectuer un traitement sur celle-ci.
Voici un exemple permettant de créer une liste de manière automatique :

```
from random import *

my_list = []
random_nb = 0
for i in range(10):
    random_nb = randint(1,20)
    my_list.append(random_nb)
print my_list
```

a.) Trier une liste

- Implémentez une procédure de tri manuelle
- Vérifiez son bon fonctionnement en affichant la liste avant et après le tri
- Créez une fonction qui gère le tri pour vous
- Testez la méthode sort() pour l'implémentation du tri

b.) Présence d'un nombre

- Implémentez une procédure qui vous dit si un nombre est présent dans une liste
- Vérifiez son bon fonctionnement en testant sur plusieurs exemples bien choisis.
- Créez une fonction qui renvoie un booléen pour indiquer la présence.

c.) Existence doublon(s)

On considèrera qu'une liste n'a pas de doublon si tous les éléments sont différent entre eux.

- Implémentez une procédure qui indique s'il y a doublon(s)
- Vérifiez son bon fonctionnement en testant sur plusieurs exemples bien choisis.
- Créez une fonction qui renvoie un booléen pour indiquer la présence de doublon(s)

2.) Liste de strings

Créez une liste avec 6 noms et une autre liste avec 6 prénoms.
Traitez ensuite ces deux listes pour :

- Créer une liste dont chaque élément est une liste de deux données : prénom et nom
- Créer une liste dont chaque élément est le nom complet recréé par concaténation
- Triez la liste ci-dessus.

Manip 2.1 : Série2, les fichiers – 1.) Introduction

Cette manipulation est tirée de l'interrogation du 22 novembre 2013.

1°) En Python, développez un programme qui demande d'introduire un nom, un prénom et une adresse (rue, numéro et code postal) puis qui écrit, les données comme sur une lettre. Les étapes sont les suivantes :

- Lire le nom (l'utilisateur introduit son nom)
- Lire le prénom (l'utilisateur introduit son prénom)
- Lire la rue, le numéro (l'utilisateur introduit la rue et le numéro)
- Lire le code postal (l'utilisateur introduit le code postal)
- Créer un fichier avec le résultat formaté comme pour une lettre

2°) Modifier le programme pour qu'à la fin de celui-ci, le programme crée un fichier « liste_eleve.txt ».

3°) Vérifier si le fichier a bien été créé.

4°) Modifier le programme pour que celui-ci affiche le contenu du fichier.

5°) Modifier le programme pour que celui demande au début si on veut effacer le contenu du fichier, puis si on veut ajouter un nom supplémentaire :

- Voulez-vous effacer le fichier? O/o ou N/n ?

- Voulez-vous encoder des élèves supplémentaires? O/o ou N/n ?

Le programme doit afficher la totalité du fichier avant de terminer.

6°) A l'aide de la bibliothèque **time** et de la fonction **strftime()** afficher la date au début du programme.

Strftime() utilise les arguments suivants :

%d = donne la date
%m= donne le mois
%Y= donne l'année
%H=donne l'heure
%M= donne les minutes
%S=donne les secondes

Bon travail !

Manip 2.2 : Série2, les fichiers – 2.) Base de données, partie 1

Créez un nouveau dossier pour toute la manip avec votre nom suivi de database1.

1.) Créez un fichier encodage1.py pour que son exécution fasse apparaître :

Veuillez introduire un ID (3 chiffres) :
Veuillez introduire votre nom :
Veuillez introduire votre prenom :
Veuillez introduire votre age :

Il doit apparaître dans votre dossier courant un dossier « data » qui contient le fichier « data_base.txt » avec les données qui viennent d'être encodées compte tenu de :

Encadrez l'ID (nommez le « my_id ») des caractères # (exemple : #134#)
Les différents éléments sont suivis d'un passage à la ligne
A la fin de la série d'encodage un 2ème passage à la ligne apparaît
Plusieurs séries d'encodage peuvent être mises au sein du même fichier.

2.) Toujours dans votre dossier de cette manip, créez un fichier encodage2.py et complétez-le afin qu'il ajoute des fonctionnalités par rapport au point précédent :
Au départ il doit apparaître :

Si vous désirez faire un encodage, tapez 'o' sinon tapez 'n'

Tant que l'utilisateur choisit oui ('o') un encodage tel qu'en 1.) doit apparaître.
Dés que l'utilisateur choisit non ('n') l'encodage se termine et le programme doit :

Lire tout le fichier data_base.txt et l'afficher
Afficher un passage à la ligne
Lire tout le fichier data_base.txt et afficher tous les ID (un ID par ligne).

On peut bien sûr exploiter la manière dont les données sont dans le fichier :

Exemple avec 1 record	Caractéristiques des données dans le fichier
#123# Jo fugtniol 65	Un record commence par l'ID mis entre # et # Un record se termine par un saut de ligne

Le premier # d'un record indique donc le début de l'ID
Le # suivant indique la fin de l'ID

Quelques pistes :

Pour savoir si on est dans la partie de l'ID on peut utiliser un flag
Pour reconnaître le caractère '#' on peut mettre deux FOR en cascade,

```
current_id = ''
reading_an_id = False
for a_line in my_file:
    for a_char in a_line:
        if reading_an_id:
            if a_char == '#':       #then finish reading
```

3.) Toujours dans votre dossier de cette manip, créez un fichier id_et_nom.py et complétez-le afin qu'il parcoure le fichier « data_base.txt » et qu'il constitue, toujours dans le dossier dat a, un fichier « id_et_nom.txt » avec les ID et les noms.

4) Créez un fichier encodage3.py qui est similaire au fichier encodage1.py mais en s'assurant de ne pas encoder deux records avec le même ID.

Manip 3.1 : Série3, listes et les fichiers – 1.) Synthèse

Créez un nouveau dossier pour toute la manip avec votre nom suivi de manip_3_1.

1.) L'idée est que l'utilisateur va introduire un nombre entier N qui va correspondre au nombre de nombres aléatoires qui seront générés. Par exemple, si l'utilisateur introduit le nombre 25 pour N, il y aura 25 nombres aléatoires générés.
Chaque nombre sera un nombre entier compris entre 1 et 100.
Différents points doivent être implémentés :

- Création de la liste : Lorsque le nombre N est introduit le programme doit se créer une liste nommée *mes_nombres* avec les nombres aléatoires.
- Deux autres listes : Sur base de la liste *mes_nombres*, créez une liste qui contient les nombres pairs et une autre pour les nombres impairs.
- Trois fichiers : Créez trois fichiers pour contenir respectivement : Tous les nombres, les nombres pairs, les nombres impairs

2.) Le programme doit créer une liste de 5 nombres aléatoires différents.
Une fois la liste créée, le programme doit constituer un fichier avec ces nombres.

3) Implémentez une fonction qui va se charger d'afficher des statistiques sur un fichier.
Il est naturel que le nom du fichier soit mis en paramètre à la fonction.
Vous pouvez construire votre fonction progressivement, les statistiques à afficher sont :

- Le nombre d'éléments du fichier
- La somme de tous les nombres du fichier
- La moyenne des nombres du fichier
- L'existence ou non de doublons

Pour vérifier le bon fonctionnement de votre programme, il est conseillé de constituer un ensemble de fichiers de tests bien choisis afin de vérifier les statistiques générées par votre programme.

4) Implémentez une fonction qui se chargera du tri manuel d'une liste.
Vérifiez ensuite le bon fonctionnement de votre fonction sur des exemples de listes.

Manip 3.2 : Série3, listes et les fichiers – 2.) Récapitulatif

Les questions ci-dessous sont de type interrogation lorsque les listes et les fichiers ont été étudiés.

1) Création de deux listes et de deux fichiers de 10 nombres aléatoires.

Créez une liste de 10 nombres entiers aléatoires puis affichez-la. Parcourez ensuite votre liste afin de créer un fichier avec ces nombres.
Répéter l'opération avec au sein d'une autre liste puis d'un autre fichier

2.) Reconstituez la somme au sein d'une liste puis d'un fichier

Créer une liste « somme » qui contiendra également 10 nombres entiers.
Chaque nombre situé à l'indice « i » sera la somme des deux nombres de même indice au sein des listes créées précédemment.
Affichez ensuite la liste et créez un fichier avec son contenu.

3.) Calcule de la somme des nombres d'un fichier

Implémentez une fonction qui calculera la somme de tous les nombres entiers contenu dans un fichier et qui fournira cette somme comme valeur de retour. Il est conseillé de mettre le nom du fichier comme paramètre à cette fonction.
Profitez-en pour tester votre fonction sur vos trois fichiers (vos deux fichiers de 10 nombres aléatoires et votre fichier avec la somme).

4.) Calcule de la somme des nombres d'une liste

Implémentez une fonction qui calculera la somme de tous les nombres entiers contenu dans une liste et qui fournira cette somme comme valeur de retour. Il est naturel de mettre le nom de la liste comme paramètre.
Profitez-en pour tester votre fonction sur vos trois listes (vos deux listes de 10 nombres aléatoires et votre liste avec la somme).
Comparez vos résultats avec ceux de la question 3.)

5) Tri manuel d'un fichier

Tentez d'implémentez une procédure de tri sur un fichier de nombres entiers. Le fichier sera modifié après la procédure dans le sens où l'ordre des nombres sera modifié pour faire apparaître un nombre croissant. Cependant tous les nombres du fichier doivent subsister.
Il est ensuite possible de créer une fonction prenant en charge le tri et de tester le bon fonctionnement de cette fonction sur base de fichiers existants.

Manip 4.1 : Série4, Tkinter – 1.) Base de données, partie 2

Créez un nouveau dossier pour toute la manip avec votre nom suivi de database2.

1.) Créez un fichier tkinter1.py pour que son exécution fasse apparaître :

Une fenêtre avec une zone pour l'ID, une zone pour le prénom, une zone pour le nom et une zone pour l'âge. Il doit également figurer un bouton avec l'inscription « Affiche_dernier » qui se charge d'afficher sur la console l'ID, le prénom, le nom et l'âge du dernier record formaté comme indiqué ici :

Exemple avec 1 record	Caractéristiques des données dans le fichier
#123# Jo fugtniol 65	Un record commence par l'ID mis entre # et # Un record se termine par un saut de ligne

Après l'affichage les cases avec les données introduites doivent être vides.

2.) Toujours dans votre dossier, créez une copie de votre programme tkinter1.py, sauvez le sous encodage_tk1.py et complétez-le d'un bouton supplémentaire avec l'inscription « Encode ».
Suite à un click sur ce bouton, il faut qu'il apparaisse dans votre dossier courant un dossier « data » qui contient le fichier « data_base.txt » avec les données qui viennent d'être encodées.
Le fichier doit être formaté de la même manière, doit pouvoir être complété à chaque nouvel encodage et montre un saut de ligne après chaque encodage.

3.) Partez de votre programme précédent que vous sauvez sous encodage_tk2.py.
Le programme contient les mêmes cases mais il y a maintenant 4 boutons :
- Un bouton avec l'inscription « Affiche »
  Le contenu des cases est conservé, ce bouton a maintenant pour but d'afficher le record courant sur la console. Rien n'est ajouté à notre fichier.
- Un bouton avec l'inscription « efface »
  Ici le bouton a pour unique but d'effacer le contenu des cases, rien ne doit s'afficher et rien ne doit s'ajouter à notre fichier.
- Un bouton avec l'inscription « encode »
  Ici le record sera ajouté à notre fichier « data_base.txt », rien ne doit s'afficher sur la console mais les cases doivent être effacées afin de permettre l'introduction d'un nouveau record.
- Un bouton avec l'inscription « lire fichier »
  Ce bouton se charge simplement d'afficher sur la console le contenu du fichier « data_base.txt »

4.) Imaginez une extension du point 3 que vous nommez affiche_id.py et qui contient un bouton supplémentaire avec l'inscription « affiche id». Un click sur ce bouton doit afficher l'ensemble des ID du fichier, si vous savez le faire, ajoutez l'option « afficher les ID triés ».

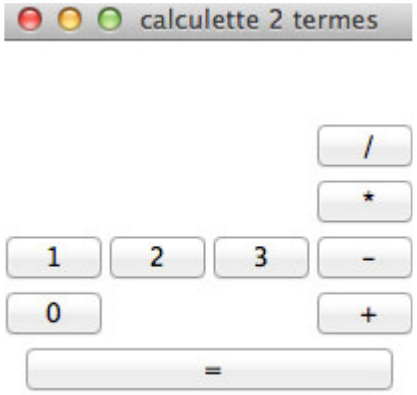
Manip 4.2 : Série4, Tkinter – 2.) Calculette, partie 1

La manipulation sur la calculette repose sur le fait de compléter un programme existant.

Il s'agit de partir du programme demo1.py (voir page suivante), de le comprendre, le modifier et le compléter pour obtenir les fonctionnalités demandées.

Enoncé de la manip :

Au départ la fenêtre n'est pas complète, il faudra dans un premier temps la compléter pour avoir une fenêtre comme celle ici à droite.



Il faudra également compléter les créations de boutons et les fonctions associées pour faire fonctionner la calculette avec seulement ces boutons.

On ajoutera les autres boutons par la suite.

1) /20
Modifiez la fonction click_2() pour permettre de constituer un nombre de plusieurs chiffres.

2) /20
Ajouter un test pour qu'on ne puisse pas compléter nb2 une fois que le égal est fait.
Compléter également les boutons 0, 1 et 3 et leurs fonctions respectives.

3) /20
Compléter click_egal() pour que tous les operateurs fonctionnent.

4) /20
Implémenter une fonction click(chiffre) où « chiffre » est le paramètre qui représente le chiffre cliqué.
Dans click_1() faites un simple appel de click(), dans click_2() idem, ...

5) /20
Modifier, lors de la création des boutons 0, 1, 2 et 3, les fonctions associées pour utiliser une même fonction click avec comme paramètre la valeur du chiffre du bouton enfoncé.

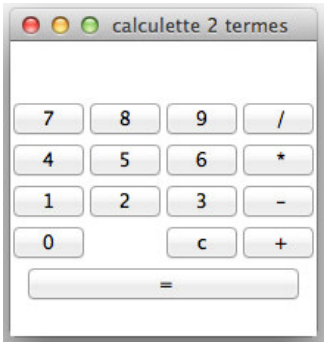
```
# calculette 2 termes
# prend en compte des clicks sur boutons
# demo1 pour la 1ere seance

from Tkinter import *

# Les chaines constantes possibles pour la variable operateur
STR_ADD = ' + '
STR_SOUSTR = ' - '
STR_MUL = ' * '
STR_DIV = ' / '

#variables globales
nb1 = ''        #une chaine vide au depart, un nombre ensuite
nb2 = ''        #une chaine vide au depart, un nombre ensuite
operateur = ''  #une chaine vide au depart, une chaine operateur ensuite
egal = ''       #une chaine vide au depart, ' = ' ensuite
result = ''     #une chaine vide au depart, un nombre ensuite

calcul = ''     #sera utilisee comme StringVar() comme variabletext d'un label

#les fonctions
def click_2():
    #lorsqu'on click sur un chiffre
    global nb1, nb2, result
    if operateur == '':         #l'operateur n'est pas introduit
        if nb1 == '':           # -- ces 2 lignes seront utiles
            nb1 = 0             # -- plus tard
        nb1 = 2
        calcul.set(str(nb1))
    else:
        if nb2 == '':           # -- ces 2 lignes seront utiles
            nb2 = 0             # -- plus tard
        nb2 = 2
        calcul.set(str(nb1) + operateur + str(nb2))

def click_operateur(str_operateur):
    #lorsqu'on click sur un operateur
    global operateur
    if egal == '':                          #le calcul n'est pas fini
        if operateur == '' and nb1 != '':   #l'operateur n'est pas introduit et nb1 existe
            operateur = str_operateur
            calcul.set(str(nb1) + operateur)

def click_egal():
    #lorsqu'on click sur egal
    global result, egal
    if nb2 != '':
        egal = ' = '
        if operateur == STR_ADD:
            result = nb1 + nb2
        else:
            result = 999 #a modifier bien sur
        calcul.set(str(nb1) + operateur + str(nb2) + egal + str(result))

#creation de la fenetre
my_frame = Tk()
my_frame.title("calculette 2 termes")
my_frame.minsize(200, 200)

#la 1ere ligne contient un label dont le texte est associe a la variable 'calcul'
calcul = StringVar()
Label(textvariable = calcul, height = 2).grid(row=0, column=0, columnspan=4, sticky=W)
calcul.set('')

#la 2eme ligne
Button(text = STR_DIV, width = 3, command = lambda:click_operateur(STR_DIV)).grid(row=1, column=3)

#la 3eme ligne
Button(text = STR_MUL, width = 3, command = lambda:click_operateur(STR_MUL)).grid(row=2, column=3)

#la 4eme ligne
Button(text = ' ', width = 3).grid(row=3, column=0)
Button(text = "2", width = 3, command = click_2).grid(row=3, column=1)
Button(text = ' ', width = 3).grid(row=3, column=2)
Button(text = STR_SOUSTR, width = 3, command = lambda:click_operateur(STR_SOUSTR)).grid(row=3, column=3)

#la derniere ligne
Button(text = ' ', width = 3).grid(row=4, column=0)
Button(text = STR_ADD, width = 3, command = lambda:click_operateur(STR_ADD)).grid(row=4, column=3)
Button(text = " = ", width = 20, command = click_egal).grid(row=5, column=0, columnspan=4)

#la phase d'attente d'un evenement
my_frame.mainloop()
```

Manip 4.3 : Série4, Tkinter – 3.) Calculette, partie 2

La partie 2 de la manip calculette est la suite de la manipulation précédente. Elle repose également sur le fait de compléter un programme existant.

Il s'agit de partir du programme demo2.py (voir ci-après), de le comprendre, le modifier et le compléter pour obtenir les fonctionnalités demandées.

Enoncé de la manip :



1) /20
Compléter la fenêtre pour qu'elle soit complète.
Il faut les boutons comme ici à droite.

2) /20
Compléter les fonctions associées à tous les boutons (sauf le cancel) pour faire fonctionner la calculette.

3) /20
Compléter la fonction associée au bouton cancel pour réinitialiser la calculette

4) /20
Gérer l'écoute de l'enfoncement des touches correspondant aux lettres et aux opérateurs arithmétiques.

5) /20
Gérer l'écoute de l'enfoncement de touche permettant de faire « cancel » ou « égal ».

BONUS : Ajouter un bouton « virgule » entre le 0 et le C permettant de manipuler des nombres à virgule.

```
# calculette 2 termes
# prend en compte des clicks sur boutons
# demo2 pour la 2eme seance

from Tkinter import *

# Les chaines constantes possibles pour la variable operateur
STR_ADD = ' + '
STR_SOUSTR = ' - '
STR_MUL = ' * '
STR_DIV = ' / '

#variables globales
nb1 = ''          #une chaine vide au depart, un nombre ensuite
nb2 = ''          #une chaine vide au depart, un nombre ensuite
operateur = ''    #une chaine vide au depart, une chaine operateur ensuite
egal = ''         #une chaine vide au depart, ' = ' ensuite
result = ''       #une chaine vide au depart, un nombre ensuite
```

```
calcul = ''     #sera utilisee comme StringVar() comme variabletext d'un label

#les fonctions
def click_2():
    #lorsqu'on click sur un chiffre
    global nb1, nb2, result
    if operateur == '':         #l'operateur n'est pas introduit
        if nb1 == '':           # -- ces 2 lignes seront utiles
            nb1 = 0             # -- plus tard
        nb1 = 2
        calcul.set(str(nb1))
    else:
        if nb2 == '':           # -- ces 2 lignes seront utiles
            nb2 = 0             # -- plus tard
        nb2 = 2
        calcul.set(str(nb1) + operateur + str(nb2))

def click_operateur(str_operateur):
    #lorsqu'on click sur un operateur
    global operateur
    if egal == '':                          #le calcul n'est pas fini
        if operateur == '' and nb1 != '':   #l'operateur n'est pas introduit et nb1 existe
            operateur = str_operateur
            calcul.set(str(nb1) + operateur)
def click(touche):
    #juste un exemple, a completer
    if touche == 0:
        print 0
    elif touche == 1:
        print 1

def click_egal():
    #lorsqu'on click sur egal
    global result, egal
    if nb2 != '':
        egal = ' = '
        if operateur == STR_ADD:
            result = nb1 + nb2
        else:
            result = 999 #a modifier bien sur
        calcul.set(str(nb1) + operateur + str(nb2) + egal + str(result))



def click_cancel():
    #definir en global les variables globales qui seront modifiees
    #reinitialiser les variables et le calcul
    pass   #completer


#la routine qui prend en charge l'enfoncement d'une touche
def gestion_touche_enfoncee(touche):
    print "repr(touche.char) : ", repr(touche.char), "touche.char : ", touche.char
    touche_str = touche.char
    if touche_str in ['0', '1']:
        click(int(touche_str))


#creation de la fenetre
my_frame = Tk()
my_frame.title("calculette 2 termes")
my_frame.minsize(200, 200)

#ecoute d'un enfoncement clavier
my_frame.bind("<Key>", gestion_touche_enfoncee)
```

Manip 4.4 : Série4, Tkinter – 4.) Prise en main Canvas

Cette manipulation est tirée de l'interrogation du 25 avril 2013.

Petit exercice graphique utilisant Tkinter

Vous pouvez utiliser votre cahier ou même repartir d'exercices résolus (les votre ou ceux des professeurs).
Vous allez créer une fenêtre Tkinter avec une zone de dessin qui contient un rond vert au départ. Au départ le rond vert a un diamètre de 120 (taille maximale).
La touche « d » permet de diminuer la taille.
La touche « b » permet de diriger le rond vers le bas
Le rond doit rester dans la limite de la zone de dessin.

a.) Modifier la taille /15
Au départ le rond vert doit avoir une taille de 120 (taille maximale) et apparaître au sein du Canvas (à une position arbitraire).
La touche « d » doit permettre de diminuer de taille en diminuant le diamètre par pas de 20.
Si la taille du rond atteint un diamètre de 20 (taille minimale), un enfoncement de la touche « d » doit faire passer le carré à la taille maximale.

b.) Déplacer la forme /15
La touche « b » permet de descendre le rond en le déplaçant par pas de 20 vers le bas.
Si le déplacement est sur le point de faire sortir le rond de la zone de dessin, il faut s'arranger pour que le déplacement n'ait pas lieu afin que le rond reste à tout moment dans la zone de dessin.

Manip 4.5 : Série4, Tkinter – 5.) Qa du 21/03/2013

Cette manipulation est tirée du questionnaire A de l'interrogation du 21 mars 2013.

Petit exercice graphique utilisant Tkinter

Vous pouvez utiliser votre cahier ou même repartir d'exercices résolus (les votre ou ceux des professeurs).
Vous allez créer une fenêtre Tkinter avec deux boutons, une zone de dessin et une valeur numérique informant sur un aspect du système.
La forme sera un rond de diamètre 50 ou 100, la touche « r » permet de choisir le diamètre.
L'ordonnée de la forme (de son centre) sera fixe.
Deux boutons permettront de changer l'abscisse de la forme
Une valeur numérique sera constamment affichée pour indiquer l'abscisse de la forme.

a.) Modifier la taille /10
Au départ un rond de diamètre 50 apparaît au sein du Canvas (à une position arbitraire).
La touche « r » doit permettre de changer de taille.
Si le rond a un diamètre de 50, le diamètre doit passer à 100 ; Si le rond a un diamètre de 100, le diamètre doit passer à 50

b.) Modifier l'abscisse /10
Votre fenêtre doit contenir deux boutons avec les indications respectives :
« abscisse - » et « abscisse + ».
Un click sur « abscisse - » doit décaler sensiblement la forme vers la gauche alors qu'un click sur « abscisse + » décaler sensiblement la forme vers la droite. La taille de la forme et l'ordonnée de la forme doivent rester inchangés.

c.) Afficher l'abscisse actuelle /10
Arrangez-vous pour que votre fenêtre contienne un emplacement qui indique l'abscisse actuelle du centre de la forme.

Manip 4.6 : Série4, Tkinter – 6.) Qb du 21/03/2013

Cette manipulation est tirée du questionnaire B de l'interrogation du 21 mars 2013.

Petit exercice graphique utilisant Tkinter

Vous pouvez utiliser votre cahier ou même repartir d'exercices résolus (les votre ou ceux des professeurs).
Vous allez créer une fenêtre Tkinter avec deux boutons, une zone de dessin et une valeur numérique informant sur un aspect du système.
La forme sera un carré, la touche « c » permet de choisir la couleur.
La position du carré (de son centre) sera fixe.
Deux boutons permettront de changer la taille (le coté) du carré.
Une valeur numérique sera constamment affichée pour indiquer la taille (le coté) du carré.

a.) Modifier la couleur /10
Au départ un carré rouge ('red') apparaît au sein du Canvas (à une position arbitraire).
La touche « c » doit permettre de changer de couleur.
Si le carré est rouge, il devient vert ('green') ; si le carré est vert, il doit redevenir rouge.

b.) Modifier la taille /10
Votre fenêtre doit contenir deux boutons avec les indications respectives :
« taille - » et « taille + ».
Un click sur « taille - » doit diminuer sensiblement la taille du carré alors qu'un click sur « taille + » doit augmenter sensiblement la taille du carré.
La position du centre du carré et la couleur doivent rester inchangés.

c.) Afficher la taille actuelle /10
Arrangez-vous pour que votre fenêtre contienne un emplacement qui indique la taille (le coté) actuelle du carré.

Manip 4.7 : Série4, Tkinter – 7.) Labyrinthe, partie1

La manipulation sur le labyrinthe repose sur le fait de compléter un programme existant. Il s'agit de partir du programme demo1.py (voir ci-après), de le comprendre, le modifier et le compléter pour obtenir les fonctionnalités demandées.

Description de certaines notions :

- Utilisation d'un tuple

Un Tuple est comme une liste mais n'est pas modifiable une fois créé. Il est constitué d'un ensemble d'éléments séparés par une virgule. Le tuple est délimité par des parenthèses et est adressable (en lecture) grâce à un index.

Exemple : my_tuple = (87, 21, 55)
print my_tuple[1] # il sera affiché 21

- Manipulation de labels variables au sein d'une fenêtre

Montrons un exemple :

```
str_var_1 = StringVar()
Label(textvariable = str_var_1, height = 2).grid(row=1, column=0, sticky=W)
str_var_1.set('')
```

La variable str_var_1 est une variable particulière (construite via StringVar()) qui permet de mettre à jour le texte d'un label. Pour mettre à jour le texte, il suffit d'écrire le nom de la variable, suivi de set() avec notre texte en paramètre.

Il y a aussi la possibilité d'utiliser par exemple str_var_1.get() qui renverra le texte actuel du label (que l'on peut alors affecter à une variable).

- Afficher l'instant actuel

Pour afficher l'instant actuel sous forme de « hh :mm :ss » où :

hh correspond à l'heure, mm aux minutes et ss aux secondes,

nous allons écrire utiliser la valeur de retour de strftime("%H:%M:%S").

Pour ça il faut avoir écrit au sein de programme une ligne telle que :

```
from time import strftime
```

Programme demo1.py :

```
from time import strftime
from Tkinter import *

# les chaines key.keysym.lower() pour les touches de deplacement
HAUT = 'up'
BAS = 'down'
GAUCHE = 'left'
DROITE = 'right'

#dimensions de la zone de dessin
LARGEUR_CANVAS = 400
HAUTEUR_CANVAS = 100

# dimensions des zones (depart et arrivee) et du rayon de l'element
DIM_ZONE = 30
RAYON = 10

# constantes de la zone de depart (coordonnées du centre et couleur)
X_D = DIM_ZONE
Y_D = HAUTEUR_CANVAS / 2
COL_D = 'green'
```

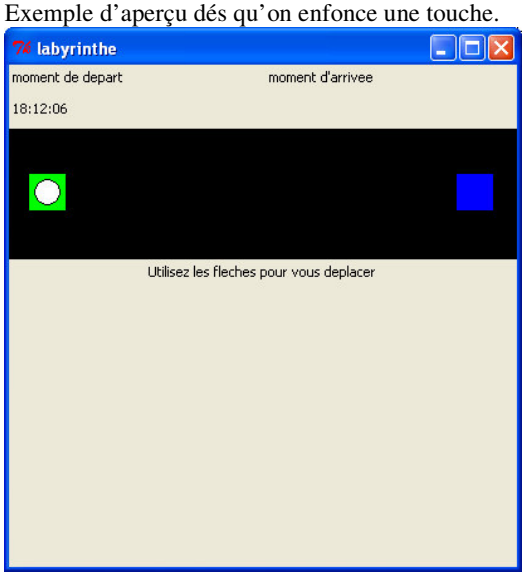
```
# constantes de la zone d'arrivee (coordonnées du centre et couleur)
X_A = LARGEUR_CANVAS - DIM_ZONE
Y_A = HAUTEUR_CANVAS / 2
COL_A = 'blue'

# les boites definissent les zones d'encombrement des elements de la zone de dessin
# on precise Xmin, Ymin, Xmax, Ymax
BOX_D = (X_D-DIM_ZONE/2, Y_D-DIM_ZONE/2, X_D+DIM_ZONE/2, Y_D+DIM_ZONE/2)
BOX_A = (X_A-DIM_ZONE/2, Y_A-DIM_ZONE/2, X_A+DIM_ZONE/2, Y_A+DIM_ZONE/2)

# variables globales
col = 'white'           #couleur de l'objet
x, y = 30, 15           #a modifier pour avoir l'objet dans la zone de depart

#les StringVar() qui seront utilises comme variabletext d'un label
str_var_moment_depart = ''
str_var_moment_arrivee = ''

def renvoie_box_cercle(x, y, rayon):
    #a modifier pour renvoyer le box de la balle
    return (5, 8, 15, 18)

def gestion_touche(key):
    global col, x, y

    ma_touche = key.keysym.lower()
    #print ma_touche   #on active pour le debugg

    #la touche 'c' modifie la couleur de l'objet
    if ma_touche == 'c':
        if col =='white':
            col = 'red'
        else:
            col = 'white'
        zone_dessin.itemconfig(balle, fill=col)
    elif ma_touche == BAS or ma_touche == HAUT:
        print "haut ou bas"

    #on cree le box
    box_obj = renvoie_box_cercle(x, y, RAYON)

    #on met a jour l'objet
    zone_dessin.coords(balle, box_obj)

#creation de la fenetre
my_tk = Tk()
my_tk.title("labyrinthe")
my_tk.minsize(400, 400)

#ecoute d'un enfoncement clavier
my_tk.bind("<Key>", gestion_touche)
```

```
#les lignes d'indice 0 et 1 contiennent les instants de depart et arrivee
Label(text="moment de depart ").grid(row=0, column=0, sticky=W)
Label(text="moment d'arrivee ").grid(row=0, column=2, sticky=W)
str_var_moment_depart = StringVar()
Label(textvariable = str_var_moment_depart, height = 2).grid(row=1, column=0, columnspan=2, sticky=W)
str_var_moment_depart.set('')
str_var_moment_arrivee = StringVar()
Label(textvariable = str_var_moment_arrivee, height = 2).grid(row=1, column=2, columnspan=2, sticky=W)
str_var_moment_arrivee.set('')

#la ligne d'indice 2 est la zone de dessin
zone_dessin = Canvas(my_tk, width=LARGEUR_CANVAS, height=HAUTEUR_CANVAS, bg = 'black')
zone_dessin.grid(row=2, column=0, columnspan=4)
zone_dessin.focus_set()
zone_dessin.create_rectangle(BOX_D, fill=COL_D) #depart
zone_dessin.create_rectangle(BOX_A, fill=COL_A) #arrivee
#la balle
balle = zone_dessin.create_oval(renvoie_box_cercle(x, y, RAYON), fill=col)

#la ligne d'indice 9 explique a l'utilisateur
Label(text = " Utilisez les fleches pour vous deplacer ").grid(row=9, column=0, columnspan=4)

my_tk.mainloop()
```

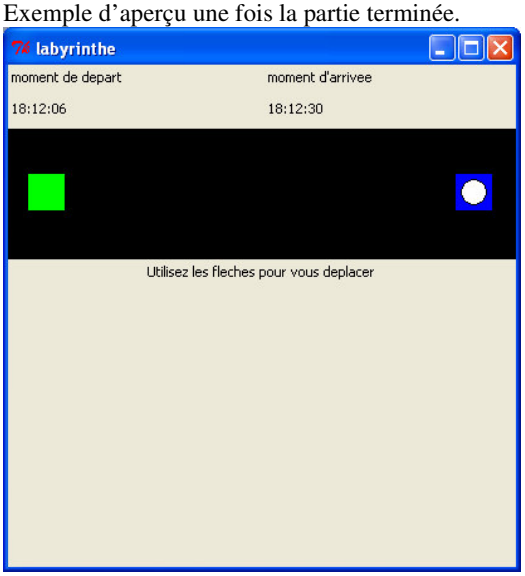
Labyrinthe, énoncé de la manip 1 :

Petit aperçu de la fenêtre de jeu :

Exemple d'aperçu dés qu'on enfonce une touche.



Exemple d'aperçu une fois la partie terminée.



1) /20 Vérifiez la détection des flèches

Assurez-vous que vous pouvez détecter l'enfoncement des touches « haut », « bas », « gauche » et « droite » et de faire une action propre à chaque touche.
Testez cette fonctionnalité simplement en plaçant des « print » pour les différents cas.

2) /20 Placez la balle au départ grâce à la fonction renvoie_box_cercle(x, y, rayon)

Commencez par faire un premier test qui consiste à simplement placer la balle dans la zone de départ. Une fois que c'est fait, implémentez la fonction renvoie_box_cercle(x, y, rayon) pour qu'elle renvoie un tuple avec les coordonnées extrêmes (Xmin, Ymin, Xmax, Ymax).
Utilisez cette fonction pour placer votre balle au départ pour vérifier si ça fonctionne.
Utilisez une variable qui indique le nombre de fois qu'on s'est trouvé hors canvas.
Mettez également à jour la valeur indiquée dans votre fenêtre.

3) /20 Affichez l'instant de départ lors du 1er enfoncement d'une touche

Commencez par tenter d'afficher le moment de départ sur la console dés le 1er enfoncement d'une touche. Une fois que c'est fait, mettez à jour le str_var_moment_depart afin de le voir dans votre fenêtre Tkinter.

4)/20 Déplacez la balle avec les flèches

Arrangez-vous pour déplacer les coordonnées de la balle par pas de 5 à l'aide des flèches. On peut alors déterminer les nouvelles données du « box » de la balle et mettre à jour les coordonnées avec une ligne telle que :

zone_dessin.coords(balle, box_obj)

5) /20 Affichez le moment d'arrivée

Une fois arrive dans la zone d'arrivée, affichez le moment d'arrivée dans la fenêtre

Manip 4.8 : Série4, Tkinter – 8.) Labyrinthe, partie2

La partie 2 de la manip labyrinthe est la suite de la manipulation précédente.
Elle repose également sur le fait de compléter un programme existant.
Il s'agit de partir du programme demo2.py (voir ci-après), de le comprendre, le modifier et le compléter pour obtenir les fonctionnalités demandées.

Décodage de deux strings formatés pour en déduire une durée

Lorsqu'on utilise des strings formatés pour en déduire une durée, il faut connaître le formatage du string. Nous choisirons hh :mm :ss où hh correspond à l'heure, mm aux minutes et ss aux secondes.
Il est alors utile d'utiliser des outils qui permettent de sélectionner une partie (ou tranche) d'un string. Exemples :
my_string[:5] #renvoie la tranche allant du début jusqu'à l'indice 5 non compris
int(my_string [1:4]) # prend la tranche où 1<= indice < 4 et la convertit en entier.

Programme demo2.py :

```
from time import strftime
from Tkinter import *

# les chaines key.keysym.lower() pour les touches de deplacement
HAUT = 'up'
BAS = 'down'
GAUCHE = 'left'
DROITE = 'right'

#dimensions de la zone de dessin
LARGEUR_CANVAS = 400
HAUTEUR_CANVAS = 100

# dimensions des zones (depart et arrivee) et du rayon de l'element
DIM_ZONE = 30
RAYON = 10

# constantes de la zone de depart (coordonnées du centre et couleur)
X_D = DIM_ZONE
Y_D = HAUTEUR_CANVAS / 2
COL_D = 'green'

# constantes de la zone d'arrivee (coordonnées du centre et couleur)
X_A = LARGEUR_CANVAS - DIM_ZONE
Y_A = HAUTEUR_CANVAS / 2
COL_A = 'blue'
# les boites definissent les zones d'encombrement des elements de la zone de des
# on precise Xmin, Ymin, Xmax, Ymax
BOX_D = (X_D-DIM_ZONE/2, Y_D-DIM_ZONE/2, X_D+DIM_ZONE/2, Y_D+DIM_ZONE/2)
BOX_A = (X_A-DIM_ZONE/2, Y_A-DIM_ZONE/2, X_A+DIM_ZONE/2, Y_A+DIM_ZONE/2)

# variables globales
col = 'white'           #couleur de l'objet
x, y = 30, 15           #a modifier pour avoir l'objet dans la zone de depart
penalites_canvas = 0
```

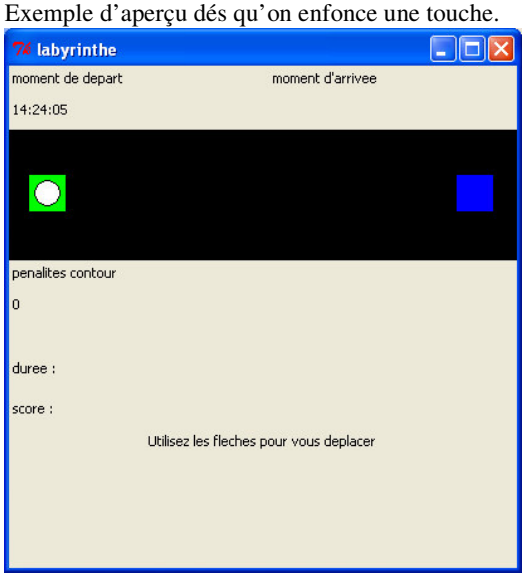
```
#les StringVar() qui seront utilises comme variabletext d'un label
str_var_moment_depart = ''
str_var_moment_arrivee = ''
str_var_penalites_canvas = ''

def renvoie_box_cercle(x, y, rayon):
    #a modifier pour renvoyer le box de la balle
    return (5, 8, 15, 18)

def box_hors_canvas(box):
    hors_canvas = False
    x_min = box[0]
    #completer les coordonnees extremes
    #et ...
    #mettre a jour le flag hors_canvas si necessaire
    return hors_canvas

def calcule_score(duree, penalites_canvas):
    #la duree est en secondes
    score = 0
    #completer
    return score

def gestion_touche(key):
    global col, x, y
    global penalites_canvas
    x_avant_deplacement = x
    y_avant_deplacement = y

    ma_touche = key.keysym.lower()
    #print ma_touche   #on active pour le debugg

    #la touche 'c' modifie la couleur de l'objet
    if ma_touche == 'c':
        if col =='white':
            col = 'red'
        else:
            col = 'white'
        zone_dessin.itemconfig(balle, fill=col)
    elif ma_touche == BAS or ma_touche == HAUT:
        #on memorise l'emplacement et on prepare le deplacement
        print "haut ou bas"

    #on cree le box qu'aurait l'objet si deplace
    box_obj = renvoie_box_cercle(x, y, RAYON)

    #on remet l'emplacement initial si on sortait du canvas
    if box_hors_canvas(box_obj):
        #completer ici
        str_var_penalites_canvas.set(str(penalites_canvas))

    #on met a jour l'objet
    zone_dessin.coords(balle, box_obj)
```

```
#creation de la fenetre
my_tk = Tk()
my_tk.title("labyrinthe")
my_tk.minsize(400, 400)

#ecoute d'un enfoncement clavier
my_tk.bind("<Key>", gestion_touche)

#les lignes d'indice 0 et 1 contiennent les instants de depart et arrivee
Label(text="moment de depart ").grid(row=0, column=0, sticky=W)
Label(text="moment d'arrivee ").grid(row=0, column=2, sticky=W)
str_var_moment_depart = StringVar()
Label(textvariable = str_var_moment_depart, height = 2).grid(row=1, column=0, c
str_var_moment_depart.set('')
str_var_moment_arrivee = StringVar()
Label(textvariable = str_var_moment_arrivee, height = 2).grid(row=1, column=2,
str_var_moment_arrivee.set('')

#la ligne d'indice 2 est la zone de dessin
zone_dessin = Canvas(my_tk, width=LARGEUR_CANVAS, height=HAUTEUR_CANVAS, bg = '
zone_dessin.grid(row=2, column=0, columnspan=4)
zone_dessin.focus_set()
zone_dessin.create_rectangle(BOX_D, fill=COL_D) #depart
zone_dessin.create_rectangle(BOX_A, fill=COL_A) #arrivee
#la balle
balle = zone_dessin.create_oval(renvoie_box_cercle(x, y, RAYON), fill=col)

#les lignes d'indice 3 et 4 indiquent les penalites
Label(text="penalites contour ").grid(row=3, column=0, sticky=W)
str_var_penalites_canvas = StringVar()
Label(textvariable = str_var_penalites_canvas, height = 2).grid(row=4, column=0
str_var_penalites_canvas.set(penalites_canvas)

#la ligne d'indice 9 explique a l'utilisateur
Label(text = " Utilisez les fleches pour vous deplacer ").grid(row=9, column=0,

my_tk.mainloop()
```

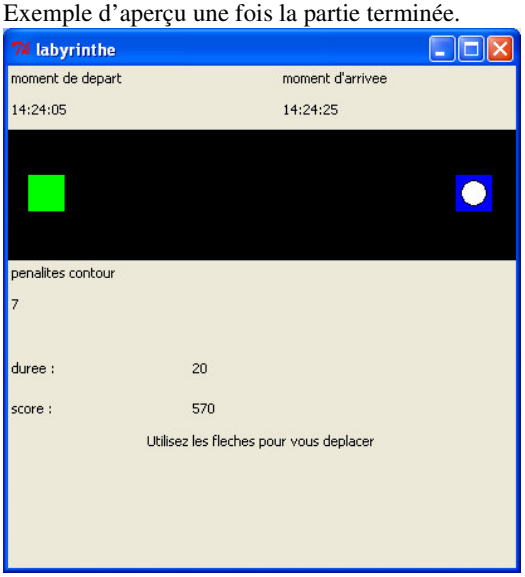
Labyrinthe, énoncé de la manip 2 :

Petit aperçu de la fenêtre de jeu :

Exemple d'aperçu dés qu'on enfonce une touche.



Exemple d'aperçu une fois la partie terminée.



1) /20 Implémentez une fonction box_hors_canvas(box)

Le paramètre « box » est en fait un tuple (comme une liste mais délimitée par des parenthèses et est non modifiable) de quatre valeurs.

Les valeurs sont respectivement : Xmin, Ymin, Xmax, Ymax

Où Xmin et Xmax sont les abscisses min et max de la zone

et Ymin et Ymax sont les ordonnées min et max de la zone

La fonction box_hors_canvas(box) doit renvoyer « True » si la zone est en dehors du canvas et « False » autrement.

Testez ensuite votre fonction simplement en plaçant des « print » lorsqu'il y a la balle qui sort du canvas.

2) /20 Affichez les pénalités dues au contour

Utilisez une variable qui indique le nombre de fois qu'on s'est trouvé hors canvas.

Mettez également à jour la valeur indiquée dans votre fenêtre.

3) /20 Ne déplacez votre balle que si on reste dans le canvas

Lorsqu'un déplacement est sur le point de se trouver hors canvas, incrémenter la variable qui contient les pénalités dues au contour, mettez à jour votre fenêtre mais n'effectuez pas le déplacement de la balle (remettez x, y à leurs anciennes valeurs).

4) /20 Affichez la durée une fois dans la zone d'arrivée.

Lorsque votre balle est dans la zone d'arrivée vous devez afficher le moment d'arrivée et calculer la durée du parcours. Pour cela il faudra décoder le string qui contient les moments de départ et d'arrivée pour calculer le nombre de secondes du parcours.

5) /20 Calculez le score à l'arrivée

Implémentez la fonction calcule_score() pour calculer un score à partir de la durée et des pénalités dues au contour. Calculez le score et affichez le dans la fenêtre.

Manip 4.9 : Série4, Tkinter – 9.) Labyrinthe, partie3

La partie 3 de la manip labyrinthe est la suite de la manipulation précédente.
Elle repose également sur le fait de compléter un programme existant.
Il s'agit de partir du programme demo3.py (voir ci-dessous), de le comprendre, le modifier et le compléter pour obtenir les fonctionnalités demandées.

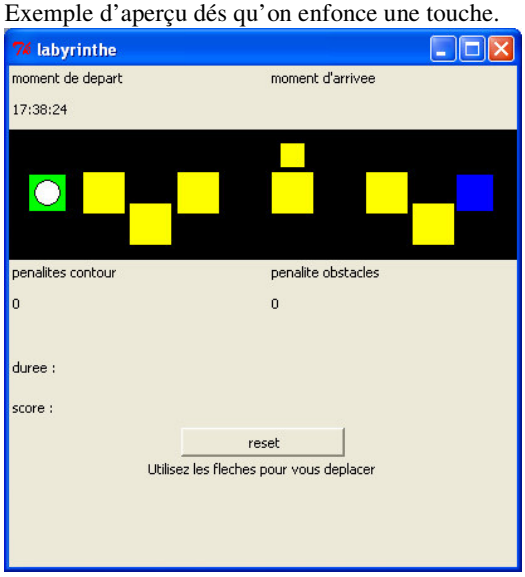
```
from time import strftime
from Tkinter import *

# les chaines key.keysym.lower() pour les touches de deplacement
HAUT = 'up'
BAS = 'down'
GAUCHE = 'left'
DROITE = 'right'

#dimensions de la zone de dessin
LARGEUR_CANVAS = 400
HAUTEUR_CANVAS = 100

# dimensions des zones (depart et arrivee) et du rayon de l'element
DIM_ZONE = 30
RAYON = 10

# constantes de la zone de depart (coordonnées du centre et couleur)
X_D = DIM_ZONE
Y_D = HAUTEUR_CANVAS / 2
COL_D = 'green'

# constantes de la zone d'arrivee (coordonnées du centre et couleur)
X_A = LARGEUR_CANVAS - DIM_ZONE
Y_A = HAUTEUR_CANVAS / 2
COL_A = 'blue'

# les boites definissent les zones d'encombrement des elements de la zone de dessin
# on precise Xmin, Ymin, Xmax, Ymax
BOX_D = (X_D-DIM_ZONE/2, Y_D-DIM_ZONE/2, X_D+DIM_ZONE/2, Y_D+DIM_ZONE/2)
BOX_A = (X_A-DIM_ZONE/2, Y_A-DIM_ZONE/2, X_A+DIM_ZONE/2, Y_A+DIM_ZONE/2)

# variables globales
col = 'white'           #couleur de l'objet
x, y = 30, 15           #a modifier pour avoir l'objet dans la zone de depart
penalites_canvas = 0

#liste des "box" des obstacles carres.
obstacles_carres = []  #Les obstacles sont ajoutes plus loin dans la creation de la fenetre

#les StringVar() qui seront utilises comme variabletext d'un label
str_var_moment_depart = ''
str_var_moment_arrivee = ''
str_var_penalites_canvas = ''

def renvoie_box_cercle(x, y, rayon):
    #a modifier pour renvoyer le box de la balle
    return (5, 8, 15, 18)

def renvoie_box_carre(x, y, cote):
    #a modifier pour renvoyer le box de la boite
    return (5, 8, 15, 18)
def box_hors_canvas(box):
    hors_canvas = False
    x_min = box[0] # exemple de lecture d'un element de box
    #completer les coordonnees extremes et ...
    #mettre a jour le flag hors_canvas si necessaire
    return hors_canvas

def il_y_a_collision(box1, box2):
    collision = True
    obj1_x_min = box1[0] # exemple de lecture d'un element de box
    #completer les coordonnees extremes et ...
    #mettre a jour le flag collision si necessaire
    return collision
```

```
def box_hors_canvas(box):
    hors_canvas = False
    x_min = box[0] # exemple de lecture d'un element de box
    #completer les coordonnees extremes et ...
    #mettre a jour le flag hors_canvas si necessaire
    return hors_canvas

def il_y_a_collision(box1, box2):
    collision = True
    obj1_x_min = box1[0] # exemple de lecture d'un element de box
    #completer les coordonnees extremes et ...
    #mettre a jour le flag collision si necessaire
    return collision

def calcule_score(duree, penalites_canvas, penalites_obstacles):
    #la duree est en secondes
    score = 0
    #completer ...
    return score

def gestion_touche(key):
    global col, x, y
    global penalites_canvas
    x_avant_deplacement = x
    y_avant_deplacement = y

    ma_touche = key.keysym.lower()
    #print ma_touche   #on active pour le debugg

    #la touche 'c' modifie la couleur de l'objet
    if ma_touche == 'c':
        if col =='white':
            col = 'red'
        else:
            col = 'white'
        zone_dessin.itemconfig(balle, fill=col)

    elif ma_touche == BAS or ma_touche == HAUT:
        #on memorise l'emplacement et on prepare le deplacement
        print "haut ou bas"     #exemple d'action liee a une fleche

    #on cree le box qu'aurait l'objet si deplace
    box_obj = renvoie_box_cercle(x, y, RAYON)

    #on remet l'emplacement initial si on sortait du canvas
    if box_hors_canvas(box_obj):
        #completer ici
        str_var_penalites_canvas.set(str(penalites_canvas))

    #on met a jour l'objet
    zone_dessin.coords(balle, box_obj)
#creation de la fenetre
my_tk = Tk()
my_tk.title("labyrinthe")
my_tk.minsize(400, 400)

#ecoute d'un enfoncement clavier
my_tk.bind("<Key>", gestion_touche)

#les lignes d'indice 0 et 1 contiennent les instants de depart et arrivee
Label(text="moment de depart ").grid(row=0, column=0, sticky=W)
Label(text="moment d'arrivee ").grid(row=0, column=2, sticky=W)
str_var_moment_depart = StringVar()
Label(textvariable = str_var_moment_depart, height = 2).grid(row=1, column=0, columnspan=2, sticky=W)
str_var_moment_depart.set('')
str_var_moment_arrivee = StringVar()
Label(textvariable = str_var_moment_arrivee, height = 2).grid(row=1, column=2, columnspan=2, sticky=W
str_var_moment_arrivee.set('')

#la ligne d'indice 2 est la zone de dessin
zone_dessin = Canvas(my_tk, width=LARGEUR_CANVAS, height=HAUTEUR_CANVAS, bg = 'black')
zone_dessin.grid(row=2, column=0, columnspan=4)
zone_dessin.focus_set()
zone_dessin.create_rectangle(BOX_D, fill=COL_D) #depart
zone_dessin.create_rectangle(BOX_A, fill=COL_A) #arrivee
#la balle
balle = zone_dessin.create_oval(renvoie_box_cercle(x, y, RAYON), fill=col)

#les lignes d'indice 3 et 4 indiquent les penalites
Label(text="penalites contour ").grid(row=3, column=0, sticky=W)
str_var_penalites_canvas = StringVar()
Label(textvariable = str_var_penalites_canvas, height = 2).grid(row=4, column=0, sticky=W)
str_var_penalites_canvas.set(penalites_canvas)

#la ligne d'indice 9 explique a l'utilisateur
Label(text = " Utilisez les fleches pour vous deplacer ").grid(row=9, column=0, columnspan=4)
my_tk.mainloop()
```

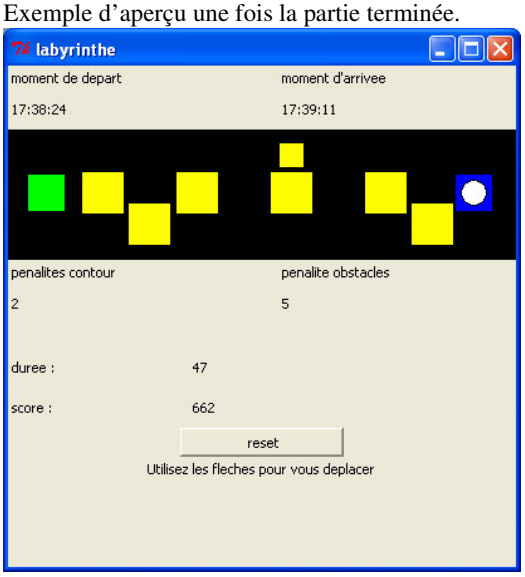
Labyrinthe, énoncé de la manip 3 :

Petit aperçu de la fenêtre de jeu :

Exemple d'aperçu dés qu'on enfonce une touche.



Exemple d'aperçu une fois la partie terminée.



1) /20 Implémentez une fonction renvoie_box_carre(x, y, cote)

Les paramètres « x » et « y » correspondent aux coordonnées du centre de la zone carrée. Le paramètre « cote » (le coté) est la longueur du coté de la zone carrée.
La fonction renvoie un tuple (comme une liste mais délimitée par des parenthèses et est non modifiable) de quatre valeurs.
Les valeurs sont respectivement : Xmin, Ymin, Xmax, Ymax

Où Xmin et Xmax sont les abscisses min et max de la zone
et Ymin et Ymax sont les ordonnées min et max de la zone

Utilisez ensuite cette fonction pour créer un obstacle carré dans la zone de dessin.

2) /20 Implémentez une fonction fonction il_y_a_collision(box1, box2)

Cette fonction doit renvoyer « True » s'il y a collision et « False » autrement.
Les deux paramètres correspondent à des zones telles que décrites au point 1).
Il est possible d'accéder aux limites de ces zones avec une syntaxe telle que box1[0]
Testez ensuite votre fonction simplement en plaçant des « print » lorsqu'il y a une collision entre votre balle et votre obstacle.

3) /20 Affichez les pénalités dues aux obstacles

Utilisez une variable qui indique le nombre de fois qu'une collision s'est produite.
Mettez également à jour la valeur indiquée dans votre fenêtre.

4) /20 Ajoutez un ensemble d'obstacles pour créer votre labyrinthe

Lorsque vous ajoutez des éléments dans votre zone de dessin, ajoutez à votre liste obstacles_carres l'ensemble des obstacles. Vérifiez lorsque vous déplacez votre balle s'il n'y a pas d'obstacles.

5) /20 Calculez le score à l'arrivée et ajoutez un reset

Modifiez votre fonction calcule_score() pour calculer un score à partir de la durée et des pénalités dues au contour et aux obstacles. Ajoutez également un bouton reset.

Manip 5.1 : Série5, Exécutable – 1.) Rendre exécutable

Pour créer un fichier exécutable à partir d'un fichier python finalisé.

Installer l'utilitaire cx_freeze

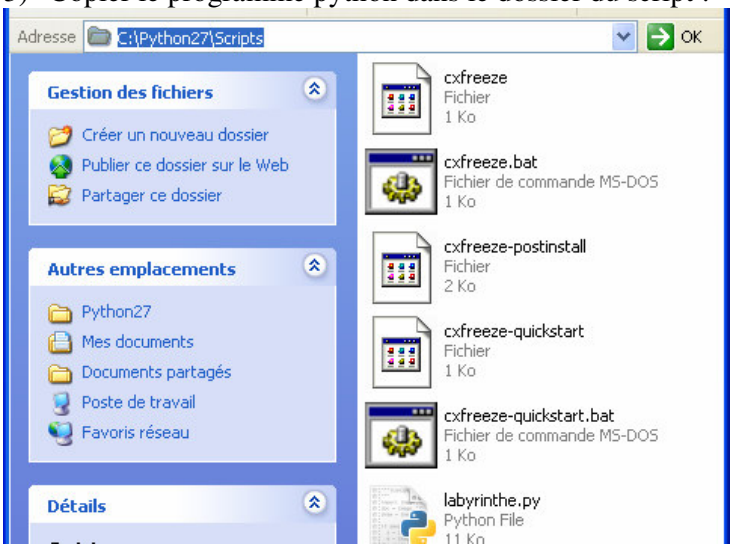
- Installer cx_freeze via http://cx-freeze.sourceforge.net/
- Sélectionner la version de python souhaitée et les caractéristiques de l'ordinateur.

Mettre en œuvre l'utilitaire cx_freeze

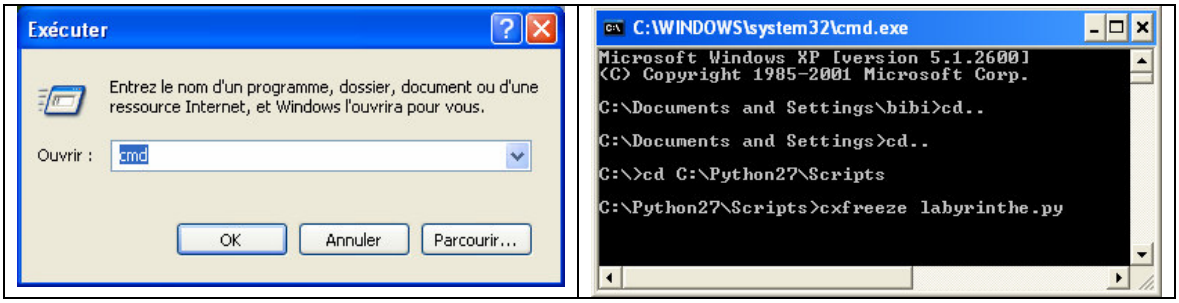
Vous trouverez des explications via :
http://fr.openclassrooms.com/uploads/fr/ftp/livre/python/extrait_chap33.pdf

La procédure ci-dessous a été faite pour le labyrinthe :

3) Copier le programme python dans le dossier du script :



4) Lancer le script de votre programme python» à partir de l'éditeur de commande.
Si votre programme python se nomme par exemple « labyrinthe.py », tapez :
cxfreeze labyrinthe.py



Manip 6.1 : Série6, Récapitulatif – 1.) Qa du 23/05/2013

1) Petite fonctionnalité au démarrage /15

a) Introduction d'une donnée dans un intervalle donné. /5

b) Première action dépendant de la donnée introduite. /10

Entrez un nombre « n » compris entre 10 et 20 et affichez une décrémentation par pas de « n » en démarrant à 100 et en allant jusqu'à 0. Si par exemple le « n » introduit vaut 15, il sera affiché « 100, 85, …, 10».

2) Choix d'un menu parmi deux possibilités. /5

Affichez « si vous souhaitez le menu de la question 3 tapez 'ok' sinon c'est le menu de la question 4 qui sera lancé ». Une sélection de « ok » amène alors au menu de la question 3 où dans un premier temps il sera affiché « 6eme info Q3 bienvenue » alors qu'un autre choix amène au menu de la question 4 où il sera affiché « 6eme info Q4 bienvenue».

3) Implémentation de la 1ère possibilité de menu /20

L'idée est de respecter la fonctionnalité demandée en utilisant des outils Tkinter.

Après l'affichage de la question 3 (« 6eme info Q3 bienvenue»), vous devez gérer : L'affichage d'une fenêtre avec une zone de dessin (Canvas). La zone de dessin contient un rond rouge. Le rond rouge est au départ à l'extrême droite de la zone de dessin. Le fait d'appuyer sur la touche « g » déplace le rond vers la gauche.

4) Implémentation de la 2ème possibilité de menu /40

L'idée est de respecter la fonctionnalité demandée en utilisant une fonction avec paramètre (20 points) puis en manipulant un fichier (20 points)

Après l'affichage de la question 4 (« 6eme info Q4 bienvenue») vous devez gérer les éléments suivants (dans l'ordre): a.) Mise en œuvre d'une fonction (20 points) - Demander à l'utilisateur d'introduire un nombre réel entre -273,15 et 3000 - Une fois un nombre correct introduit, faites appel à une fonction qui reçoit le nombre introduit comme paramètre. La fonction doit alors afficher la température en Fahrenheit en utilisant le paramètre représentant des Celcius. (Temp_F moins 32 vaut 9/5 de Temp_C) Pour vérifier on peut vérifier que si Temp_C vaut 0, Temp_F vaut 32. On peut aussi vérifier que si Temp_C vaut 100, Temp_F vaut 212. b.) Mise en pratique de gestion de fichiers (20 points) - Créez hors programme un fichier avec 15 nombres entiers (1 nb par ligne) - Lire par programme tous les nombres et mémoriser ceux divisibles par 3. - Indiquez combien de nombres sont divisibles par 3 et affichez les.

Manip 6.2 : Série6, Récapitulatif – 2.) Qb du 23/05/2013

1) Petite fonctionnalité au démarrage /15

a) Introduction d'une donnée dans un intervalle donné. /5

b) Première action dépendant de la donnée introduite. /10

Entrez un nombre « n » compris entre 10 et 20 et affichez toutes les puissances positives de 2 inférieures à 2 exposant « n ». Commencez à 2 exposant « n » (noté 2**n en python) et affichez les puissances inférieures de 2 jusqu'à 2 exposant 0.

2) Choix d'un menu parmi deux possibilités. /5

Affichez « si vous souhaitez le menu de la question 4 tapez 'o' sinon tapez 'n' ». Un enfoncement de la touche 'o' amène alors au menu de la question 4 où dans un premier temps il sera affiché « 6eme info Q4 bonjour » alors qu'un autre choix amène au menu de la question 3 où il sera affiché « 6eme info Q3 bonjour ».

3) Implémentation de la 1ère possibilité de menu /20

L'idée est de respecter la fonctionnalité demandée en utilisant des outils Tkinter.

Après l'affichage de la question 3 (« 6eme info Q3 bonjour »), vous devez gérer : L'affichage d'une fenêtre avec une zone de dessin (Canvas). La zone de dessin contient un rond jaune. Le rond jaune est au départ tout au dessus de la zone de dessin. Le fait d'appuyer sur la touche « b » déplace le rond vers le bas.

4) Implémentation de la 2ème possibilité de menu /40

L'idée est de respecter la fonctionnalité demandée en utilisant une fonction avec paramètre (20 points) puis en manipulant un fichier (20 points)

Après l'affichage de la question 4 (« 6eme info Q4 bonjour ») vous devez gérer les éléments suivants (dans l'ordre): a.) Mise en œuvre d'une fonction (20 points) - Demander à l'utilisateur d'introduire un chiffre (donc entre 0 et 9) - Une fois un chiffre introduit, faites appel à une fonction qui reçoit le chiffre introduit comme paramètre. La fonction doit alors afficher la table jusqu'à 20 du chiffre introduit. b.) Mise en pratique de gestion de fichiers (20 points) - Créez hors programme un fichier avec 12 nombres entiers (1 nb par ligne) - Lire par programme tous les nombres et mémoriser les impairs. - Indiquez combien de nombres sont impairs et affichez les.

Manip 6.3 : Série6, Récapitulatif – 3.) Qc du 23/05/2013

1) Petite fonctionnalité au démarrage /15

a) Introduction d'une donnée dans un intervalle donné. /5

b) Première action dépendant de la donnée introduite. /10

Entrez un nombre « n » compris entre 10 et 20 et affichez toutes les puissances positives de 2 inférieures à 2 exposant « n ». Commencez à 2 exposant « n » (noté 2**n en python) et affichez les puissances inférieures de 2 jusqu'à 2 exposant 0.

2) Choix d'un menu parmi deux possibilités. /5

Affichez « si vous souhaitez le menu de la question 4 tapez 'o' sinon tapez 'n' ». Un enfoncement de la touche 'o' amène alors au menu de la question 4 où dans un premier temps il sera affiché « QC question 4, bonjour » alors qu'un enfoncement de la touche 'n' amène au menu de la question 3 où il sera affiché 'QC question 3, bonjour ». Si l'utilisateur ne choisit ni 'o' ni 'n', il faut lui demander de réintroduire son choix.

3) Implémentation de la 1ère possibilité de menu /20

L'idée est de respecter la fonctionnalité demandée en utilisant une fonction sans paramètre. Comme la fonction ne peut pas utiliser de passage de paramètre, si vous devez partager des données communes avec le programme principal vous pouvez utiliser le mot clé « global » (10 points pour la fonction et 10 points pour l'exercice).

Après l'affichage de la question 3 (« QC question 3, bonjour »), vous devez gérer les éléments suivants : - Afficher « comptage rapide pendant 3 minutes » - Une fois l'affichage fait, faites appel à une fonction sans paramètre qui prend en charge un comptage rapide (0.05s de délai) sous la forme « m :ss » où « m » correspond au nombre de minutes et « ss » au nombre de secondes. Le comptage commence alors par 0 :00 puis 0 :01 jusqu'à 2 :59 - Une fois le comptage terminé, affichez « Fin Qc question 3 »

4) Implémentation de la 2ème possibilité de menu /20

L'idée est de respecter la fonctionnalité demandée en utilisant une fonction avec paramètre (10 points pour la fonction et 10 points pour l'exercice).

Après l'affichage de la question 4 (« QC question 4, bonjour »), vous devez gérer les éléments suivants : - Demander à l'utilisateur d'introduire un chiffre (donc entre 0 et 9) - Une fois un chiffre introduit, faites appel à une fonction qui reçoit le chiffre introduit comme paramètre. La fonction doit alors afficher la table jusqu'à 20 du chiffre introduit.

Manip 6.4 : Série6, Récapitulatif – 4.) Qd du 23/05/2013

1) Petite fonctionnalité au démarrage **/15**

a) Introduction d'une donnée dans un intervalle donné. /5

b) Première action dépendant de la donnée introduite. /10

Entrez un nombre « n » entre 250 et 1000 puis affichez une décrémentation par pas de 10 en démarrant de « n » et en allant jusqu'à 0.

2) Choix d'un menu parmi deux possibilités. **/5**

Affichez « si vous souhaitez le menu de la question 3 tapez '3' sinon tapez '4' ». Un enfoncement de la touche '3' amène alors au menu de la question 3 où dans un premier temps il sera affiché « QB question 3, bienvenue » alors qu'un enfoncement de la touche '4' amène au menu de la question 4 où il sera affiché « QB question 4, bienvenue ». Si l'utilisateur ne choisit ni '3' ni '4', il faut lui demander de réintroduire son choix.

3) Implémentation de la 1ère possibilité de menu **/20**

L'idée est de respecter la fonctionnalité demandée en utilisant une fonction sans paramètre. Comme la fonction ne peut pas utiliser de passage de paramètre, si vous devez partager des données communes avec le programme principal vous pouvez utiliser le mot clé « global » (10 points pour la fonction et 10 points pour l'exercice).

Après l'affichage de la question 3 (« QB question 3, bienvenue »), vous devez gérer les éléments suivants : - Afficher « les 16 premières puissances de 2 » - Une fois l'affichage fait, faites appel à une fonction sans paramètre qui affiche les 16 1ères puissances de 2, il apparaîtra alors : 2exposant 1 = 2 2exposant 2 = 4 … 2exposant 16 = 65536 - Une fois les affichages des terminés, affichez « Fin Qb question 3 »

4) Implémentation de la 2ème possibilité de menu **/20**

L'idée est de respecter la fonctionnalité demandée en utilisant une fonction avec paramètre (10 points pour la fonction et 10 points pour l'exercice).

Après l'affichage de la question 4 (« QB question 4, bienvenue »), vous devez gérer les éléments suivants : - Demander à l'utilisateur d'introduire un nombre entier positif inférieur à 100. - Une fois un nombre correct introduit, faire appel à une fonction qui reçoit le nombre introduit comme paramètre. La fonction doit alors afficher si oui ou non le nombre est divisible par 3.

Manipulations Raspberry

Les différentes manipulations ci-après sont basées sur l'utilisation du Raspberry.

Suivant la manipulation, une manipulation peut durer jusqu'à 4 ou 5 séances de 100 minutes.

Une séance de base comme une première prise en main du Raspberry peut s'envisager sur une séance de 100 minutes alors qu'une manipulation qui met en œuvre un ou plusieurs modules d'extension Raspberry nécessitera plusieurs séances de 100 minutes.

Manip R1.1 : 1.) Raspberry – 1.) Prise en main

- Prenez soin de lire de la documentation sur le Raspberry avant de commencer.

- Rassemblez tout le matériel nécessaire à la mise en œuvre du Raspberry, à savoir :

    - Ecran + cables
    - clavier et souris USB
    - Alimentation DC compatible
    - Carte SD avec le système d'exploitation
    - Une clé USB si vous en disposez
    - Eventuellement une connexion Internet via câble

- Branchez l'ensemble en terminant par l'alimentation

- Prenez soin de noter toutes les opérations que vous faites pour pouvoir les refaire.

- Entraînez-vous à configurer le mode de démarrage (mode commandes ou graphique)

- Examinez et testez la procédure à faire pour changer la configuration du clavier

- Tentez d'éditer un fichier texte (.txt) à partir du mode de commande

- Tentez d'éditer un fichier texte (.txt) à partir du mode du terminal du mode graphique

- Tentez, à partir du mode de commande, d'éditer un fichier python (.py) et de le lancer

- Tentez, à partir du mode graphique, d'éditer un fichier python (.py) et de le lancer

- Au sein du mode graphique, explorez l'emplacement où vos fichiers sont stockés

- Lancez votre environnement de développement python et testez-le

- Si vous disposez d'une clé USB, essayez d'y copier vos fichiers (.txt et .py)

- Si vous disposez d'une connexion Internet, essayez des outils de navigation

- Toujours si vous disposez d'Internet, tentez de trouver des outils de capture d'écran

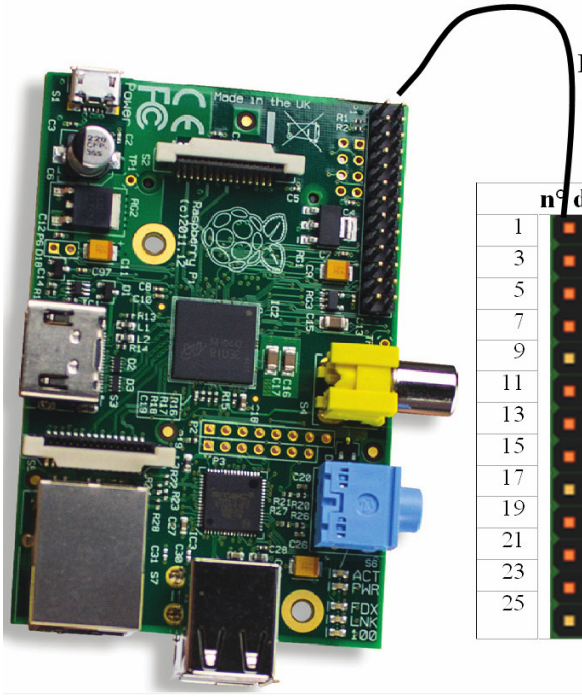
- Enfin, copiez votre fichier capture d'écran sur votre clé USB

Manip R2.1 : 2.) Modules Raspberry – 1.) Feu rouge

La manipulation « feu rouge » permet de mettre en œuvre de la soudure de composants électronique de manière à créer un module d'extension de base.

Comme pour un feu de circulation, Il y a trois LEDS : une rouge, une orange et une verte.
Chaque LED sera vera mise en série avec une résistance dédiée.
Les valeurs des résistances dépendent normalement de la couleur de la LED ainsi que de sa consommation typique ; pour simplifier on considèrera des résistances identiques de 330 Ω.

L'extension « feu rouge » est simplement un connecteur sécable femelle de deux rangées qui se connectera sur le Raspberry sur le connecteur décrit ci-dessous.



n° de pin			Notations « fonctionnelles »			
1		2	1	3V3	5V	2
3		4	3	SDA	5V	4
5		6	5	SCL	Gnd	6
7		8	7	GPIO_GCLK	Txd0	8
9		10	9	Gnd	Rxd0	10
11		12	11	GPIO_GEN0	GPIO_GEN1	12
13		14	13	GPIO_GEN2	Gnd	14
15		16	15	GPIO_GEN3	GPIO_GEN4	16
17		18	17	3V3	GPIO_GEN5	18
19		20	19	SPI_MOSI	Gnd	20
21		22	21	SPI_MISO	GPIO_GEN6	22
23		24	23	SPI_SCLK	SPI_CE0_N	24
25		26	25	gnd	SPI_CE1_N	26

Sur le connecteur femelle qui se connectera au Raspberry, on soudra les trois LEDS et les trois résistances suivant la convention ci-dessous :

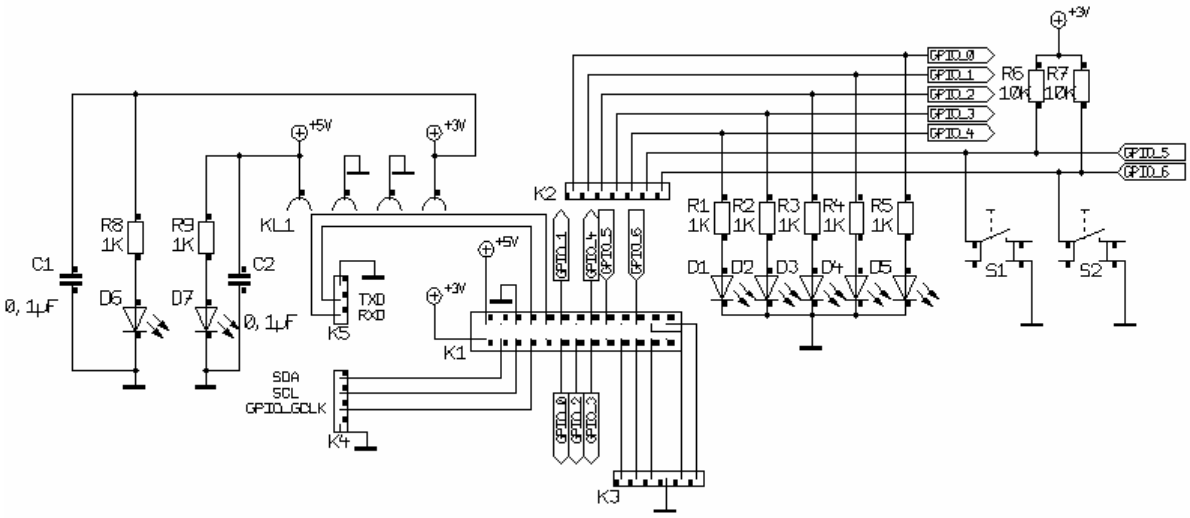
- L'anode de la LED rouge se connectera sur la patte 22
- L'anode de la LED orange se connectera sur la patte 18
- L'anode de la LED verte se connectera sur la patte 16
- Chaque LED (au niveau de la cathode) est soudée en série avec sa résistance
- Les pattes des résistances qui ne sont pas connectées aux LEDS seront soudées entre elles et soudées à la patte 6 pour la masse (Gnd)

La présentation des trois LEDS doit suggérer un feu de circulation.
Le programme écrit en python doit montrer un comportement semblable à celui d'un feu de circulation.

Manip R2.2 : 2.) Modules Raspberry – 2.) Module perso

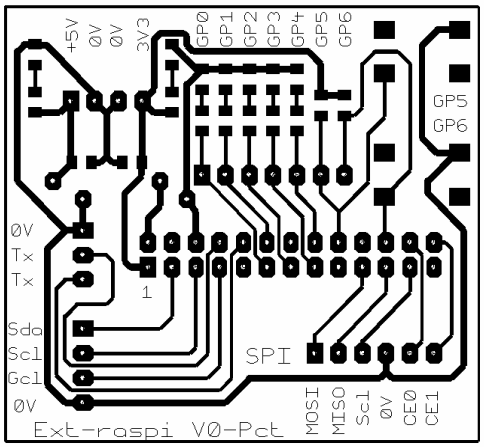
La manipulation « module perso » illustre un module de base personnalisé conçu par un professeur (J. Pochet) pour une petite prise en main du connecteur GPIO du Raspberry.

Le schéma est donné ci-dessous :

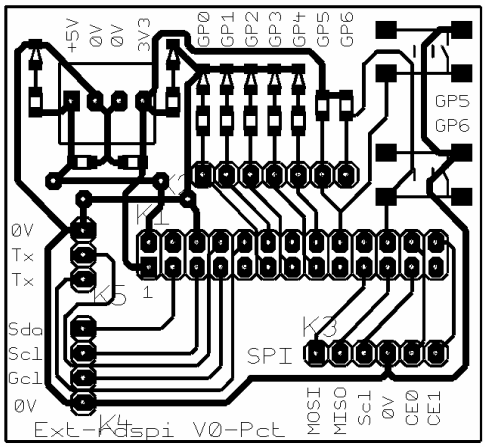


Le PCB correspondant est donné ci-dessous :

Vue des pistes et pastilles



Vue de l'ensemble



Tout comme la manipulation « feu rouge » (manipulation précédente R2.1), l'idée est de connecter l'extension sur le connecteur GPIO du Raspberry.

La manipulation consiste à comprendre le schéma et de mettre au point un programme en python qui permet de vérifier le bon fonctionnement de toutes les entrées et sorties du module d'extension.

Manip R2.3 : 2.) Modules Raspberry – 3.) Pi-face

A l'aide du Raspberry (type B), de ses périphériques habituels (clavier, souris, écran), d'une connexion Internet câblée et du module extension PiFace (référence Farnell 221-8566) :

- Connectez le Raspberry sans l'extension pour vérifier vous matériel

Vous pouvez vérifier que vous avez bien l'alimentation du Raspberry, que la carte SD contient bien un système d'exploitation prêt à être utilisé, que votre souris et votre clavier sont reconnus et fonctionnels et que votre écran est bien connecté (avec le câble adéquat). Profitez-en pour vérifier que vous avez une connexion Internet câblée fonctionnelle.

- Rassemblez toute la documentation nécessaire sur le Raspberry et l'extension.

Il est intéressant de parcourir toute la documentation et d'imprimer les parties les plus utiles à la manipulation

- Prenez en main votre extension de manière organisée et progressive.

Lorsqu'on manipule pour la première fois un dispositif embarqué, il est conseillé de se fixer des objectifs réfléchis. Une approche est de mener à bien les différentes fonctionnalités petit à petit en se fixant des objectifs simples.
L'idée sera ici de se fixer trois objectifs progressifs à tenter d'atteindre dans l'ordre.
Le 1er objectif sera au plus simple (1ere prise de contact avec la carte), le deuxième sera une étape intermédiaire et le 3ème objectif un petit challenge.

Prenez soin de noter tout ce que vous faites, voici les 3 objectifs :

1.) Allumer / Eteindre une LED

Une fois que cette étape fonctionne, vous pouvez vérifier que vous êtes en mesure d'agir sur toutes les LEDS de la carte.

2.) Détecter un enfoncement / relâchement d'un bouton

Une fois que cette étape fonctionne, vous pouvez vérifier que vous êtes en mesure de détecter tous les boutons de la carte.

3.) Interfacer un élément extérieur

Tentez de mener à bien l'interconnexion d'un élément extérieur (entrée et/ou sortie) et sa gestion par programmation.

- Constituer un dossier explicatif et un « directory » avec les fichiers informatiques

Rédigez un dossier qui reprend le cahier des charges (les objectifs), la documentation nécessaire et sa description ainsi que toutes les actions mise en œuvre pour répondre au cahier des charges ; accompagnez votre dossier d'explications et d'une conclusion.
Constituez également un « directory » tous les fichiers informatiques liés à la manipulation.

Manip R2.4 : 2.) Modules Raspberry – 4.) Gertboard

A l'aide du Raspberry (type B), de ses périphériques habituels (clavier, souris, écran), d'une connexion Internet câblée et du module extension Gertboard (référence Farnell 225-0034) :

- Connectez le Raspberry sans l'extension pour vérifier vous matériel

Vous pouvez vérifier que vous avez bien l'alimentation du Raspberry, que la carte SD contient bien un système d'exploitation prêt à être utilisé, que votre souris et votre clavier sont reconnus et fonctionnels et que votre écran est bien connecté (avec le câble adéquat). Profitez-en pour vérifier que vous avez une connexion Internet câblée fonctionnelle.

- Rassemblez toute la documentation nécessaire sur le Raspberry et l'extension.

Il est intéressant de parcourir toute la documentation et d'imprimer les parties les plus utiles à la manipulation

- Prenez en main votre extension de manière organisée et progressive.

Lorsqu'on manipule pour la première fois un dispositif embarqué, il est conseillé de se fixer des objectifs réfléchis. Une approche est de mener à bien les différentes fonctionnalités petit à petit en se fixant des objectifs simples.
L'idée sera ici de se fixer trois objectifs progressifs à tenter d'atteindre dans l'ordre.
Le 1er objectif sera au plus simple (1ere prise de contact avec la carte), le deuxième sera une étape intermédiaire et le 3ème objectif un petit challenge.

Prenez soin de noter tout ce que vous faites, voici les 3 objectifs :

1.) Allumer / Eteindre une LED

Une fois que cette étape fonctionne, vous pouvez vérifier que vous êtes en mesure d'agir sur toutes les LEDS de la carte.

2.) Détecter un enfoncement / relâchement d'un bouton

Une fois que cette étape fonctionne, vous pouvez vérifier que vous êtes en mesure de détecter tous les boutons de la carte.

3.) Interfacer un élément extérieur

Tentez de mener à bien l'interconnexion d'un élément extérieur (entrée et/ou sortie) et sa gestion par programmation.

- Constituer un dossier explicatif et un « directory » avec les fichiers informatiques

Rédigez un dossier qui reprend le cahier des charges (les objectifs), la documentation nécessaire et sa description ainsi que toutes les actions mise en œuvre pour répondre au cahier des charges ; accompagnez votre dossier d'explications et d'une conclusion.
Constituez également un « directory » tous les fichiers informatiques liés à la manipulation.

Manip R2.5 : 2.) Modules Raspberry – 5.) Raspicomm

A l'aide du Raspberry (type B), de ses périphériques habituels (clavier, souris, écran), d'une connexion Internet câblée et du module extension Raspicomm (référence RS 772-2974) :

- Connectez le Raspberry sans l'extension pour vérifier vous matériel

Vous pouvez vérifier que vous avez bien l'alimentation du Raspberry, que la carte SD contient bien un système d'exploitation prêt à être utilisé, que votre souris et votre clavier sont reconnus et fonctionnels et que votre écran est bien connecté (avec le câble adéquat). Profitez-en pour vérifier que vous avez une connexion Internet câblée fonctionnelle.

- Rassemblez toute la documentation nécessaire sur le Raspberry et l'extension.

Il est intéressant de parcourir toute la documentation et d'imprimer les parties les plus utiles à la manipulation

- Prenez en main votre extension de manière organisée et progressive.

Lorsqu'on manipule pour la première fois un dispositif embarqué, il est conseillé de se fixer des objectifs réfléchis. Une approche est de mener à bien les différentes fonctionnalités petit à petit en se fixant des objectifs simples.
L'idée sera ici de se fixer trois objectifs progressifs à tenter d'atteindre dans l'ordre.
Le 1er objectif sera au plus simple (1ere prise de contact avec la carte), le deuxième sera une étape intermédiaire et le 3ème objectif un petit challenge.

Prenez soin de noter tout ce que vous faites, voici les 3 objectifs :

1.) Afficher la direction du joystick

Affichez dans un premier temps un message à l'écran informant de la direction du joystick.

2.) Afficher et encoder l'heure de la real time clock

A l'aide du clavier et de l'écran, gérez l'affichage et l'encodage de l'heure.

3.) Encoder l'heure à l'aide du joystick

A l'aide du joystick et de l'écran, gérez l'affichage et l'encodage de l'heure.
.

- Constituer un dossier explicatif et un « directory » avec les fichiers informatiques

Rédigez un dossier qui reprend le cahier des charges (les objectifs), la documentation nécessaire et sa description ainsi que toutes les actions mise en œuvre pour répondre au cahier des charges ; accompagnez votre dossier d'explications et d'une conclusion.
Constituez également un « directory » tous les fichiers informatiques liés à la manipulation.

Manip R2.6 : 2.) Modules Raspberry – 6.) Camera

A l'aide du Raspberry (type B), de ses périphériques habituels (clavier, souris, écran), d'une connexion Internet câblée et du module extension Camera (référence RS 775-7731) :

- Connectez le Raspberry sans l'extension pour vérifier vous matériel

Vous pouvez vérifier que vous avez bien l'alimentation du Raspberry, que la carte SD contient bien un système d'exploitation prêt à être utilisé, que votre souris et votre clavier sont reconnus et fonctionnels et que votre écran est bien connecté (avec le câble adéquat). Profitez-en pour vérifier que vous avez une connexion Internet câblée fonctionnelle.

- Rassemblez toute la documentation nécessaire sur le Raspberry et l'extension.

Il est intéressant de parcourir toute la documentation et d'imprimer les parties les plus utiles à la manipulation

- Prenez en main votre extension de manière organisée et progressive.

Lorsqu'on manipule pour la première fois un dispositif embarqué, il est conseillé de se fixer des objectifs réfléchis. Une approche est de mener à bien les différentes fonctionnalités petit à petit en se fixant des objectifs simples.
L'idée sera ici de se fixer trois objectifs progressifs à tenter d'atteindre dans l'ordre.
Le 1er objectif sera au plus simple (1ere prise de contact avec la carte), le deuxième sera une étape intermédiaire et le 3ème objectif un petit challenge.

Prenez soin de noter tout ce que vous faites, voici les 3 objectifs :

1.) Stocker un fichier image

Une camera peut bien sûr prendre une photo ; commencez par prendre une photo et d'observer le contenu de votre fichier image.
Analysez les formats de fichiers et paramètres de capture d'image.

2.) Détecter une feuille noir ou blanche

Prenez en photo une feuille blanche puis une feuille noire et observez les différences au niveau des contenus de fichiers. Tentez de rédiger une routine qui permettra de déterminer si une feuille est noire ou blanche.

3.) Détecter la couleur d'une feuille

Etendez le point précédent pour tenter de détecter une couleur.

- Constituer un dossier explicatif et un « directory » avec les fichiers informatiques

Rédigez un dossier qui reprend le cahier des charges (les objectifs), la documentation nécessaire et sa description ainsi que toutes les actions mise en œuvre pour répondre au cahier des charges ; accompagnez votre dossier d'explications et d'une conclusion.
Constituez également un « directory » tous les fichiers informatiques liés à la manipulation.

Manip R2.7 : 2.) Modules Raspberry – 7.) E/R

A l'aide de 2 Raspberry's (types B), de ses périphériques habituels (claviers, souris, écrans), d'une connexion Internet câblée et de 2 modules extensions E/R (référence RS 783-9662) :

- Connectez le Raspberry sans l'extension pour vérifier vous matériel

Vous pouvez vérifier que vous avez bien pour les 2 cartes Raspberry :
L'alimentation, la carte SD avec un système d'exploitation prêt à être utilisé, une souris et un clavier reconnus et fonctionnels, un HUB USB et un écran (avec le câble adéquat).
Profitez-en pour vérifier que vous avez une connexion Internet câblée fonctionnelle.

- Rassemblez toute la documentation nécessaire sur le Raspberry et l'extension.

Il est intéressant de parcourir toute la documentation et d'imprimer les parties les plus utiles à la manipulation

- Prenez en main votre extension de manière organisée et progressive.

Lorsqu'on manipule pour la première fois un dispositif embarqué, il est conseillé de se fixer des objectifs réfléchis. Une approche est de mener à bien les différentes fonctionnalités petit à petit en se fixant des objectifs simples.
L'idée sera ici de se fixer trois objectifs progressifs à tenter d'atteindre dans l'ordre.
Le 1er objectif sera au plus simple (1ere prise de contact avec la carte), le deuxième sera une étape intermédiaire et le 3ème objectif un petit challenge.

Prenez soin de noter tout ce que vous faites, voici les 3 objectifs :

1.) Le premier poste émet, le second reçoit le caractère émis

Un Raspberry est configuré en émission et se charge d'émettre un caractère.
L'autre Raspberry est configuré en réception et affiche le caractère reçu.

2.) L'émission réception doit être bidirectionnelle

Les deux modules sont configurés de la même façon, la transmission peut se faire dans les deux sens.

3.) Emission d'un paquet (message complet)

Etendez le point précédent pour transmettre un message plus volumineux.

- Constituer un dossier explicatif et un « directory » avec les fichiers informatiques

Rédigez un dossier qui reprend le cahier des charges (les objectifs), la documentation nécessaire et sa description ainsi que toutes les actions mise en œuvre pour répondre au cahier des charges ; accompagnez votre dossier d'explications et d'une conclusion.
Constituez également un « directory » tous les fichiers informatiques liés à la manipulation.

Quelques corrections

• *Correction de la manip 2.1 : Les fichiers - introduction*

```
# Interro du 22 novembre 2013
# -*- coding: cp1252 -*-
import os
import time
from time import strftime

os.chdir("c:\python27")

def efface_fichier():
    reponse=raw_input("Voulez-vous effacer le fichier? ")
    if reponse=="O" or reponse=="o":
        fichier=open("liste_eleve","w")
        fichier.write("")
        fichier.close()
        print "efface"

def encodage_nom(da):
    reponse=raw_input("Voulez-vous encoder un nom? ")
    if reponse=="O" or reponse=="o":
        a,b,c,d,e=intr_nom()
        ecrire_fichier(da,a,b,c,d,e)
        return(0)
    else:
        return(1)

def intr_nom():
    nom=raw_input("introduire le nom: ")
    prenom=raw_input("introduire le prénom :")
    rue=raw_input("introduire la rue :")
    numero=raw_input("introduire le numéo :")
    code_post=raw_input ("introduire le code postale: ")
    return(nom,prenom,rue,numero,code_post)

def ecrire_fichier(date,nom,prenom,adresse,numero,code_post):
    texte=[date," ",nom," ",prenom," ",adresse," ",numero," ",code_post]
    print texte

    fichier=open("liste_eleve","a")
    fichier.write(date)
    fichier.write(" ")
    fichier.write(nom)
```

```
    fichier.write(" ")
    fichier.write(prenom)
    fichier.write(" ")
    fichier.write(adresse)
    fichier.write(" ")
    fichier.write(numero)
    fichier.write(" ")
    fichier.write(code_post)
    retour_chariot="\n"
    fichier.write(retour_chariot)
    fichier.close()

def test(k,l,m,n,o):
    print k,l,m,n,o

def lire_heure():
    heure=strftime("%d/%m/%Y")
    print heure
    return(heure)

#début ddu programme
flag=0
dat=lire_heure()
efface_fichier()
while flag==0:
    flag=encodage_nom(dat)
```

• *Correction de la manip 2.2 : Les fichiers – Base de donnée, partie1*

Exercice 1 :

```
import os

CHAR_ID = "#"
CHAR_NEW_LINE = "\n"
my_id = ""
prenom = ""
nom = ""
age = ""

def demande_info():
    global my_id, prenom, nom, age
    my_id=raw_input("Veuillez introduire un ID :")
    prenom=raw_input("Veuillez introduire votre prenom :")
    nom=raw_input("Veuillez introduire votre nom :")
    age=raw_input("Veuillez introduire votre age:")
```

```
def select_dossier(nom_dossier):
    #si le dossier n'existe pas on le cree
    if not os.path.isdir(nom_dossier):
        os.mkdir(nom_dossier)
    os.chdir(nom_dossier)

def ajoute_au_fichier():
    my_file = open("data_base.txt","a")
    my_file.write(CHAR_ID)
    my_file.write(my_id)
    my_file.write(CHAR_ID + CHAR_NEW_LINE)
    my_file.write(prenom + CHAR_NEW_LINE)
    my_file.write(nom + CHAR_NEW_LINE)
    my_file.write(age + CHAR_NEW_LINE)
    my_file.write(CHAR_NEW_LINE)
    my_file.close()

demande_info()
select_dossier("data")
ajoute_au_fichier()
```

Exercice 2 :

```
import os

CHAR_ID = "#"
CHAR_NEW_LINE = "\n"
NOM_FICHIER = "data_base.txt"
encodage_en_cours = True
my_id = ""
prenom = ""
nom = ""
age = ""

def select_dossier(nom_dossier):
    #si le dossier n'existe pas on le cree
    if not os.path.isdir(nom_dossier):
        os.mkdir(nom_dossier)
    os.chdir(nom_dossier)

def demande_si_encodage():
    nouvel_encodage = False
    reponse = raw_input("Si vous voulez faire un encodage, tapez o sinon tapez n : ")
    if reponse == 'o' or reponse == 'O':
        nouvel_encodage = True
    return nouvel_encodage
```

```
def demande_info():
    global my_id, prenom, nom, age
    my_id = raw_input("Veuillez introduire un ID :")
    prenom = raw_input("Veuillez introduire votre prenom :")
    nom = raw_input("Veuillez introduire votre nom :")
    age = raw_input("Veuillez introduire votre age:")

def ajoute_au_fichier(nom_fichier):
    my_file = open(nom_fichier,"a")
    my_file.write(CHAR_ID)
    my_file.write(my_id)
    my_file.write(CHAR_ID + CHAR_NEW_LINE)
    my_file.write(prenom + CHAR_NEW_LINE)
    my_file.write(nom + CHAR_NEW_LINE)
    my_file.write(age + CHAR_NEW_LINE)
    my_file.write(CHAR_NEW_LINE)
    my_file.close()

def affiche_fichier(nom_fichier):
#on parcourt tout le fichier et on l'affiche
    my_file = open(nom_fichier,"r")
    my_record = ''
    for a_line in my_file:
        if a_line == CHAR_NEW_LINE: # un "\n" seul à chaque fin de record
            print my_record        #comme print crée un saut de ligne automatique
            my_record = ''         #on ne print qu'une fois par record
        else:
            my_record += a_line
    my_file.close()

def affiche_id_fichier(nom_fichier):
#on parcourt tout le fichier et on affiche l'id
    my_file = open(nom_fichier,"r")
    current_id = ''
    reading_an_id = False
    for a_line in my_file:
        for a_char in a_line:
            if reading_an_id:
                if a_char == CHAR_ID:       #then finish reading
                    print current_id
                    current_id = ''
                    reading_an_id =  False
                else:
                    current_id += a_char
            elif a_char == CHAR_ID:         #beginning an ID reading
                reading_an_id =  True
    my_file.close()
```

```
#début du programme
select_dossier("data")
while(encodage_en_cours):
    demande_info()
    ajoute_au_fichier(NOM_FICHIER)
    encodage_en_cours = demande_si_encodage()
affiche_fichier(NOM_FICHIER)
affiche_id_fichier(NOM_FICHIER)
```

• *Correction des manip 4.2 et 4.3 : Tkinter - la calculette*

```
# calculette 2 termes
# prend en compte des clicks sur boutons et une lecture clavier

from Tkinter import *

# Les chaines constantes possibles pour la variable operateur
STR_ADD = ' + '
STR_SOUSTR = ' - '
STR_MUL = ' * '
STR_DIV = ' / '

#variables globales
nb1 = ''       #une chaine vide au depart, un nombre ensuite
nb2 = ''       #une chaine vide au depart, un nombre ensuite
operateur = ''  #une chaine vide au depart, une chaine operateur ensuite
egal = ''      #une chaine vide au depart, ' = ' ensuite
result = ''    #une chaine vide au depart, un nombre ensuite

calcul = ''    #sera utilisee comme StringVar() comme variabletext d'un label

#les fonctions
def click(chiffre):
    #lorsqu'on click sur un chiffre
    global nb1, nb2, result
    if egal == '':                #le calcul n'est pas fini
        if operateur == '':       #l'operateur n'est pas introduit
            if nb1 == '':
                nb1 = 0
            nb1 = 10*nb1 + chiffre
            calcul.set(str(nb1))
        else:
            if nb2 == '':
                nb2 = 0
            nb2 = 10*nb2 + chiffre
            calcul.set(str(nb1) + operateur + str(nb2))
```

```
def click_operateur(str_operateur):
    #lorsqu'on click sur un operateur
    global operateur
    if egal == '':                      #le calcul n'est pas fini
        if operateur == '' and nb1 != '':   #l'operateur n'est pas introduit et nb1 existe
            operateur = str_operateur
            calcul.set(str(nb1) + operateur)

def click_egal():
    #lorsqu'on click sur egal
    global result, egal
    if nb2 != '':
        egal = ' = '
        if operateur == STR_ADD:
            result = nb1 + nb2
        elif operateur == STR_SOUSTR:
            result = nb1 - nb2
        elif operateur == STR_MUL:
            result = nb1 * nb2
        elif operateur == STR_DIV:
            result = nb1*1.0 / nb2
        calcul.set(str(nb1) + operateur + str(nb2) + egal + str(result))

def click_cancel():
    #lorsqu'on click sur cancel
    global nb1, nb2, operateur, egal, result
    nb1 = ''
    nb2 = ''
    operateur = ''
    egal = ''
    result = ''
    calcul.set('')

#la routine qui prend en charge l'enfoncement d'une touche
def gestion_touche_enfoncee(touche):
    print "repr(touche.char) : ", repr(touche.char), "touche.char : ", touche.char
    touche_str = touche.char
    if touche_str in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']:
        click(int(touche_str))
    elif touche_str == '+':
        click_operateur(STR_ADD)
    elif touche_str == '-':
        click_operateur(STR_SOUSTR)
    elif touche_str == '*':
        click_operateur(STR_MUL)
    elif touche_str == '/':
        click_operateur(STR_DIV)
```

```
    elif touche_str == ' ' or touche_str == '\r':
        click_egal()
    elif touche_str == '\x7f':
        click_cancel()

#creation de la fenetre
my_frame = Tk()
my_frame.title("calculette 2 termes")
my_frame.minsize(200, 200)

#ecoute d'un enfoncement clavier
my_frame.bind("<Key>", gestion_touche_enfoncee)

#la 1ere ligne contient un label dont le texte est associe a la variable 'calcul'
calcul = StringVar()
Label(textvariable = calcul, height = 2).grid(row=0, column=0, columnspan=4, sticky=W)
calcul.set('')

#la 2eme ligne
Button(text = "7", width = 3, command = lambda:click(7)).grid(row=1, column=0)
Button(text = "8", width = 3, command = lambda:click(8)).grid(row=1, column=1)
Button(text = "9", width = 3, command = lambda:click(9)).grid(row=1, column=2)
Button(text = STR_DIV, width = 3, command = lambda:click_operateur(STR_DIV)).grid(row=1, column=3)

#la 3eme ligne
Button(text = "4", width = 3, command = lambda:click(4)).grid(row=2, column=0)
Button(text = "5", width = 3, command = lambda:click(5)).grid(row=2, column=1)
Button(text = "6", width = 3, command = lambda:click(6)).grid(row=2, column=2)
Button(text = STR_MUL, width = 3, command = lambda:click_operateur(STR_MUL)).grid(row=2, column=3)

#la 4eme ligne
Button(text = "1", width = 3, command = lambda:click(1)).grid(row=3, column=0)
Button(text = "2", width = 3, command = lambda:click(2)).grid(row=3, column=1)
Button(text = "3", width = 3, command = lambda:click(3)).grid(row=3, column=2)
Button(text = STR_SOUSTR, width = 3, command = lambda:click_operateur(STR_SOUSTR)).grid(row=3, column=3)

#la derniere ligne
Button(text = "0", width = 3, command = lambda:click(0)).grid(row=4, column=0)
Button(text = "c", width = 3, command = click_cancel).grid(row=4, column=2)
Button(text = STR_ADD, width = 3, command =
lambda:click_operateur(STR_ADD)).grid(row=4, column=3)
Button(text = " = ", width = 20, command = click_egal).grid(row=5, column=0, columnspan=4)

my_frame.mainloop()                    #la phase d'attente d'un evenement
```

• Correction de la manip 4.4 : Tkinter – prise en main du Canvas

```
#interro du 25 avril 2014 Qb

from Tkinter import *

def cercle (x,y,r,coul):
    global balle
    #on cree la balle
    balle=can.create_oval(x-r,y-r,x+r,y+r,fill=coul)

def gestion_touche(key):
    global x,r,y,coul,balle

    #assigne la variable ma_touche avec la valeur enfoncé
    ma_touche = key.keysym.lower()


    if ma_touche=='d':
        y=y+10
        # test la limite inférieure
        if y>=280:
            y=280

    if ma_touche=='r':
        r=r-10
        # test le diamètre
        if r<=20:# teste le diamètre
            r=120

    #on modifie la place de la balle
    can.coords(balle,x-r,y-r,x+r,y+r)

#debut du programme

#definition des variables
x=200
y=200
r=120
coul="green"

#creation de la fenetre
fen1=Tk()

#declenche la gestiondes touches
fen1.bind("<Key>", gestion_touche)
```

can=Canvas(fen1,width=400,height=400,bg='ivory')
can.pack(side=TOP, padx=5,pady=5)

cercle(x,y,r,coul)
fen1.mainloop()

• *Correction de la manip 4.5 : Tkinter – Qa du 21/03/2013*

```
from Tkinter import *

#dimensions de la zone de dessin
LARGEUR_CANVAS = 400
HAUTEUR_CANVAS = 400

# Coordonnee de depart (coordonnées du centre)
X_D = LARGEUR_CANVAS / 2
Y_D = HAUTEUR_CANVAS / 2

# dimension du rayon de l'element
rayon = 25

# les boites definissent les zones d'encombrement des elements de la zone de dessin
# on precise Xmin, Ymin, Xmax, Ymax
BOX_D = (X_D-rayon, Y_D-rayon, X_D+rayon, Y_D+rayon)

# variables globales
col = 'white'           #couleur de l'objet
x, y = X_D, Y_D         #coordonnees de l'objet initialise dans la zone de depart

#les StringVar() qui seront utilises comme variabletext d'un label
str_abscisse = ''

def renvoie_box_cercle(x, y, rayon):
    box = (x-rayon, y-rayon, x+rayon, y+rayon)
    return box

def abs_moins():
    global x
    x = x - 10
    box_obj = renvoie_box_cercle(x, y, rayon)
    zone_dessin.coords(balle, box_obj)
    str_abscisse.set(x)

def abs_plus():
    global x
    x = x + 10
    box_obj = renvoie_box_cercle(x, y, rayon)
    zone_dessin.coords(balle, box_obj)
    str_abscisse.set(x)

def gestion_touche(key):
    ma_touche = key.keysym.lower()
    #print key.keysym.lower()                                          #on active pour le debugg
    #print "key.char : " + key.char + " repr(key.char) : " + repr(key.char) #on active pour le debugg
    global rayon, x, y

    pas_px = 10                 #nombre de pixels pour chaque deplacement

    #la touche 'r' modifie la couleur de l'objet
    if ma_touche == 'r':
        if rayon == 25:
            rayon = 50
        else:
            rayon = 25
    box_obj = renvoie_box_cercle(x, y, rayon)
    zone_dessin.coords(balle, box_obj)
```

```
#creation de la fenetre
my_tk = Tk()
my_tk.title("interro")
my_tk.minsize(400, 600)

#ecoute d'un enfoncement clavier
my_tk.bind("<Key>", gestion_touche)

#les lignes d'indice 0 et 1 contiennent les boutons

Button(text = "abscisse - ", width = 15, command = abs_moins).grid(row=0, column=0)
Button(text = "abscisse + ", width = 15, command = abs_plus).grid(row=1, column=0)

#la ligne d'indice 2 est la zone de dessin
zone_dessin = Canvas(my_tk, width=LARGEUR_CANVAS, height=HAUTEUR_CANVAS, bg = 'black')
zone_dessin.grid(row=2, column=0, columnspan=4)
zone_dessin.focus_set()

#la balle
balle = zone_dessin.create_oval(renvoie_box_cercle(x, y, rayon), fill='white')

#les lignes d'indice 3 et 4 indiquent l'abscisse
Label(text="abscisse").grid(row=3, column=0, sticky=W)
str_abscisse = StringVar()
Label(textvariable = str_abscisse, height = 2).grid(row=4, column=0, sticky=W)
str_abscisse.set(x)


my_tk.mainloop()
```

• *Correction de la manip 4.6 : Tkinter – Qb du 21/03/2013*

```
from Tkinter import *

#dimensions de la zone de dessin
LARGEUR_CANVAS = 400
HAUTEUR_CANVAS = 400

# Coordonnee de depart (coordonnées du centre)
X_D = LARGEUR_CANVAS / 2
Y_D = HAUTEUR_CANVAS / 2

# dimension du rayon de l'element
est_rouge = True
hauteur = 50

# les boites definissent les zones d'encombrement des elements de la zone de dessin
# on precise Xmin, Ymin, Xmax, Ymax
#ici hauteur = largeur car rond ou carre
BOX_D = (X_D-hauteur/2, Y_D-hauteur/2, X_D+hauteur/2, Y_D+hauteur/2)

# variables globales
col = 'white'           #couleur de l'objet
x, y = X_D, Y_D         #coordonnees de l'objet initialise dans la zone de depart

#les StringVar() qui seront utilises comme variabletext d'un label
str_hauteur = ''

def renvoie_box(x, y, hauteur):
    box = (x-hauteur/2, y-hauteur/2, x+hauteur/2, y+hauteur/2)
    return box
```

```
def hauteur_moins():
    global hauteur
    if hauteur >20:
        hauteur = hauteur - 10
    box_obj = renvoie_box(x, y, hauteur)
    zone_dessin.coords(carre, box_obj)
    str_hauteur.set(hauteur)

def hauteur_plus():
    global hauteur
    if hauteur <300:
        hauteur = hauteur + 10
    box_obj = renvoie_box(x, y, hauteur)
    zone_dessin.coords(carre, box_obj)
    str_hauteur.set(hauteur)

def gestion_touche(key):
    ma_touche = key.keysym.lower()
    #print key.keysym.lower()                                          #on active pour le debugg
    #print "key.char : " + key.char + " repr(key.char) : " + repr(key.char) #on active pour le debugg
    global est_rouge

    #la touche 'c' modifie la couleur de l'objet
    if ma_touche == 'c':
        if est_rouge :
            est_rouge = False
            zone_dessin.itemconfig(carre, fill='green')
        else:
            est_rouge = True
            zone_dessin.itemconfig(carre, fill='red')

#creation de la fenetre
my_tk = Tk()
my_tk.title("interro")
my_tk.minsize(400, 600)

#ecoute d'un enfoncement clavier
my_tk.bind("<Key>", gestion_touche)

#les lignes d'indice 0 et 1 contiennent les boutons

Button(text = "hauteur - ", width = 15, command = hauteur_moins).grid(row=0, column=0)
Button(text = "hauteur + ", width = 15, command = hauteur_plus).grid(row=1, column=0)

#la ligne d'indice 2 est la zone de dessin
zone_dessin = Canvas(my_tk, width=LARGEUR_CANVAS, height=HAUTEUR_CANVAS, bg = 'black')
zone_dessin.grid(row=2, column=0, columnspan=4)
zone_dessin.focus_set()

#la forme
carre = zone_dessin.create_rectangle(renvoie_box(x, y, hauteur), fill='red')

#les lignes d'indice 3 et 4 indiquent l'abscisse
Label(text="hauteur : ").grid(row=3, column=0, sticky=W)
str_hauteur = StringVar()
Label(textvariable = str_hauteur, height = 2).grid(row=4, column=0, sticky=W)
str_hauteur.set(hauteur)


my_tk.mainloop()
```

• Correction des manip 4.7 et 4.8 : Tkinter – Labyrinthe

```
"""
LABYRINTHE :
Il y a une zone de depart et une zone d'arrivee (2 carres)
Il y a un objet que l'on peut deplacer (un cercle)
Il y a des obstaces (des carres)
Les fleches permettent de se deplacer
Le 1er enfoncement d'une touche memorise l'instant de depart
Une fois dans la zone d'arrivee l'instant d'arrivee est memorise
puis la duree est determinee et un score est affiche

Le fait de toucher des obstacle ou le bord engendre des penalites.
Un bouton reset permet de recommencer
"""

from time import strftime
from Tkinter import *

# les chaines key.keysym.lower() pour les touches de deplacement
HAUT = 'up'
BAS = 'down'
GAUCHE = 'left'
DROITE = 'right'

#dimensions de la zone de dessin
LARGEUR_CANVAS = 400
HAUTEUR_CANVAS = 100

# dimensions des zones (depart et arrivee) et du rayon de l'element
DIM_ZONE = 30
RAYON = 10

# constantes de la zone de depart (coordonnées du centre et couleur)
X_D = DIM_ZONE
Y_D = HAUTEUR_CANVAS / 2
COL_D = 'green'

# constantes de la zone d'arrivee (coordonnées du centre et couleur)
X_A = LARGEUR_CANVAS - DIM_ZONE
Y_A = HAUTEUR_CANVAS / 2
COL_A = 'blue'

# les boites definissent les zones d'encombrement des elements de la zone de dessin
# on precise Xmin, Ymin, Xmax, Ymax
BOX_D = (X_D-DIM_ZONE/2, Y_D-DIM_ZONE/2, X_D+DIM_ZONE/2, Y_D+DIM_ZONE/2)
BOX_A = (X_A-DIM_ZONE/2, Y_A-DIM_ZONE/2, X_A+DIM_ZONE/2, Y_A+DIM_ZONE/2)

# variables globales
col = 'white'          #couleur de l'objet
x, y = X_D, Y_D        #coordonnees de l'objet initialise dans la zone de depart
penalites_canvas = 0

#les StringVar() qui seront utilises comme variabletext d'un label
str_var_moment_depart = ''
str_var_moment_arrivee = ''
```

```
str_var_penalites_canvas = ''
str_var_duree_parcours = ''
str_var_score = ''

def renvoie_box_cercle(x, y, rayon):
    box = (x-rayon, y-rayon, x+rayon, y+rayon)
    return box

def box_hors_canvas(box):
    hors_canvas = False
    x_min = box[0]
    y_min = box[1]
    x_max = box[2]
    y_max = box[3]
    if x_min < 0 or y_min < 0:
        hors_canvas = True
    elif x_max > LARGEUR_CANVAS:
        hors_canvas = True
    elif y_max > HAUTEUR_CANVAS:
        hors_canvas = True
    return hors_canvas

def determine_duree(string_arrivee, string_depart):
    #le format est %H:%M:%S pour les string
    h_a = int(string_arrivee[:2])
    m_a = int(string_arrivee[3:5])
    s_a = int(string_arrivee[6:])
    h_d = int(string_depart[:2])
    m_d = int(string_depart[3:5])
    s_d = int(string_depart[6:])

    s_tot_a = s_a + 60 * m_a + 3600 * h_a
    s_tot_d = s_d + 60 * m_d + 3600 * h_d
    duree_en_sec = s_tot_a - s_tot_d
    if duree_en_sec < 0 :  #cas particulier avec un changement de jour
        duree_en_sec = 60  #disons 60s dans ce cas
    return duree_en_sec

def calcule_score(duree, penalites_canvas):
    #la duree est en secondes
    score = 1000 - 4*duree - 50*penalites_canvas
    if score < 0:
        score = 0
    return score

def gestion_touche(key):
    ma_touche = key.keysym.lower()
    #print key.keysym.lower()                                    #on active pour le debugg
    #print "key.char : " + key.char + " repr(key.char) : " + repr(key.char) #on active pour le debugg
    global col, x, y
    global penalites_canvas
    x_avant_deplacement = x
    y_avant_deplacement = y

    pas_px = 5              #nombre de pixels pour chaque deplacement
```

```
    #le 1er enfoncement d'une touche est le moment de depart
    if str_var_moment_depart.get() == '':
        str_var_moment_depart.set(strftime("%H:%M:%S"))

    #la touche 'c' modifie la couleur de l'objet
    elif ma_touche == 'c':
        if col =='white':
            col = 'red'
        else:
            col = 'white'
        zone_dessin.itemconfig(balle, fill=col)

    #on regarde les touches que si on n'est pas arrive
    elif str_var_moment_arrivee.get() =='':
        #une touche de deplacement deplace ou donne une penalite
        if ma_touche == BAS or ma_touche == HAUT or ma_touche == GAUCHE or ma_touche == DROITE:
            #on memorise l'emplacement et on prepare le deplacement
            x_avant_deplacement = x
            y_avant_deplacement = y
            if ma_touche == BAS:
                y += pas_px
            elif ma_touche == HAUT:
                y -= pas_px
            elif ma_touche == GAUCHE:
                x -= pas_px
            elif ma_touche == DROITE:
                x += pas_px

            #on cree le box qu'aurait l'objet si deplace
            box_obj = renvoie_box_cercle(x, y, RAYON)

            #on remet l'emplacement initial si on sortait du canvas
            if box_hors_canvas(box_obj):
                penalites_canvas += 1
                str_var_penalites_canvas.set(str(penalites_canvas))
                x = x_avant_deplacement
                y = y_avant_deplacement
                box_obj = renvoie_box_cercle(x, y, RAYON)

            #on met a jour l'objet
            zone_dessin.coords(balle, box_obj)

            #l'element dans la zone d'arrivee termine le labyrinthe
            if x == X_A and y == Y_A :
                str_var_moment_arrivee.set(strftime("%H:%M:%S"))
                str_dep = str_var_moment_depart.get()
                str_arr = str_var_moment_arrivee.get()
                nb_sec = determine_duree(str_arr, str_dep)
                str_var_duree_parcours.set(nb_sec)
                score = calcule_score(nb_sec, penalites_canvas)
                str_var_score.set(score)

#creation de la fenetre
my_tk = Tk()
my_tk.title("labyrinthe")
my_tk.minsize(400, 400)
```

```
#ecoute d'un enfoncement clavier
my_tk.bind("<Key>", gestion_touche)

#les lignes d'indice 0 et 1 contiennent les instants de depart et arrivee
Label(text="moment de depart ").grid(row=0, column=0, sticky=W)
Label(text="moment d'arrivee ").grid(row=0, column=2, sticky=W)
str_var_moment_depart = StringVar()
Label(textvariable = str_var_moment_depart, height = 2).grid(row=1, column=0, columnspan=2, sticky=W)
str_var_moment_depart.set('')
str_var_moment_arrivee = StringVar()
Label(textvariable = str_var_moment_arrivee, height = 2).grid(row=1, column=2, columnspan=2, sticky=W)
str_var_moment_arrivee.set('')

#la ligne d'indice 2 est la zone de dessin
zone_dessin = Canvas(my_tk, width=LARGEUR_CANVAS, height=HAUTEUR_CANVAS, bg = 'black')
zone_dessin.grid(row=2, column=0, columnspan=4)
zone_dessin.focus_set()
zone_dessin.create_rectangle(BOX_D, fill=COL_D) #depart
zone_dessin.create_rectangle(BOX_A, fill=COL_A) #arrivee

#la balle
balle = zone_dessin.create_oval(renvoie_box_cercle(x, y, RAYON), fill=col)

#les lignes d'indice 3 et 4 indiquent les penalites
Label(text="penalites contour ").grid(row=3, column=0, sticky=W)
str_var_penalites_canvas = StringVar()
Label(textvariable = str_var_penalites_canvas, height = 2).grid(row=4, column=0, sticky=W)
str_var_penalites_canvas.set(penalites_canvas)

#la ligne d'indice 5 est vide
Label(text="").grid(row=5, column=0, sticky=W) #une ligne vide

#la ligne d'indice 6 indique le temps total mis
Label(text="duree : ").grid(row=6, column=0, sticky=W)
str_var_duree_parcours = StringVar()
Label(textvariable = str_var_duree_parcours, height = 2).grid(row=6, column=1, sticky=W)
str_var_duree_parcours.set('')

#la ligne d'indice 7 indique le score
Label(text="score : ").grid(row=7, column=0, sticky=W)
str_var_score = StringVar()
Label(textvariable = str_var_score, height = 2).grid(row=7, column=1, sticky=W)
str_var_score.set('')

#la ligne d'indice 9 explique a l'utilisateur
Label(text = " Utilisez les fleches pour vous deplacer ").grid(row=9, column=0, columnspan=4)

my_tk.mainloop()
```