Programmer en Python

Programmer en Python

Je vous recommande vivement de programmer sur le site :
France-ioi.fr
Je vous encourage à traiter

- Niveau 1 ch1 : tout sauf les challenges
- Niveau 1 ch2 : 1, 2, 3, 4, 5, 6, 11, 12 ,13
- Niveau 1 ch3 : 1, 2, 5, 7, 8, 12, 15, 16 ch4 : 1, 2, 3, 13
- Niveau 1 ch5 : 1, 2, 5, 6, 9 ch6 : 1, 3, 4
- Niveau 1 ch7 : rien ch8 : 1, 2, 3, 4
- Niveau 2 : ch2 :1, 2, 3, 4

Le site suivant est aussi très bien fait : Université de Waterloo
Sur ces deux sites, la démarche est très progressive et votre code est évalué à l'aide de tests.
Vous trouverez d'autres liens, en particulier vers des leçons videos en bas de page.
On supposera que Python3 est installé : la distribution Anaconda est recommandée.

Conseils pour l'installation

Les éléments de cours qui suivent, sont très largement issus de deux manuels existant en librairie, mais aussi disponibles en ligne.

Je me réfèrerai au premier par SWI :
Apprendre à programer en Python3 de G. Swinnen
Je me réfèrerai au second par OPC (openclassrooms) :
Apprenez à programmer en Python de V. Le Goff

Il est souvent moins coûteux d'acheter les manuels que d'imprimer de nombreuses pages !

La page sur python au format PDF

Voici les différents points abordés:

Les exercices marqués d'une étoile sont obligatoires pour tout le monde.

1) Premiers pas

Retour au plan de la page

Les premières instructions seront écrites directement dans la console python qui se trouve au bas de la page, à droite dans Spyder.

Les opérations

Tapez des opérations simples (suivies de "Enter"), les résultats s'affichent. Les règles de priorité mathématiques sur les opérations s'appliquent.
Pour écrire les puissances, on utilise **. Tapez par exemple : 3**2.
Remarque : en tapant sur la flèche vers le haut, vous remontez dans l'historique des instructions
Deux opérations spéciales, mais souvent utiles, sont à connaître :
- la division entière. Par exemple : 20//3 qui donne 6.
- le modulo qui donne le reste de la division entière. Par exemple : 20%3 qui donne 2.

Les variables

Des précisions sur ce qu'est une variable et comment l'ordinateur la voit :
OCR: les variables
SWI : les variables
Pour affecter une valeur à une variable a, on écrit simplement : a = 5.
Les noms de variable peuvent comporter plusieurs lettres simples (pas d'accents etc.), ainsi que des chiffres (sauf au début), mais pas de caractères spéciaux (sauf _).
On peut alors effectuer des opérations avec ces variables...
On peut changer l'affectation de a en écrivant par exemple a = 7 ou même a = a + 1 (ce qui augmente le contenu de la mémoire a de 1).
Attention, le nom de la variable est toujours à gauche du "="
Affectations multiples : on peut écrire "a,b = 3,4" ce qui affecte 3 à a et 4 à b, et même "a,b = b,a" ce qui échange a et b de manière très simple.
Ce mode d'utilisation (dans la console) de Python est intéressant pour tester des commandes ou des bouts de code...

2) Premiers programmes

Retour au plan de la page

Ouvrez un nouveau fichier (cela s'appelle un module) dans Spyder (si aucun n'est ouvert) et enregistrez le sous "essai.py".
Tapez des instructions analogues à celles du paragraphe précédent, en passant à la ligne pour chaque nouvelle instruction, puis cliquez sur la petite flèche verte...(qui fait exécuter le porgramme).
Que se passe-t-il ? Pas grand chose ! Contrairement à ce qui se passe dans la console, pour afficher un résultat, il faut demander à Python de l'afficher : c'est la fonction print.
Elle s'utilise comme dans beaucoup de langages sous la forme : print(a) affiche le contenu de la variable a, print("Hello") affiche ce qu'il y a entre les guillemets (c'est une chaîne de caractères) et print ("a vaut ", a) affiche la chaîne de caractères suivie de la valeur de la variable a. On peut donc afficher différents objets en les séparant par des virgules, et en mettant les chaînes de caractères entre guillemets.
Enregistrer votre programme avant de l'exécuter
Il est recommandé de mettre des commentaires (des explications) dans les programmes en commençant par un #

Exercices pour les futurs isn :
2.a) Ecrire un programme qui affiche "bonjour tout le monde"
2.b) Ecrire un programme qui affecte 1058 à la variable a et 337 à b, puis qui échange a et b, puis affiche "a vaut maintenant 337 et b vaut 1058" de telle sorte que si on change les valeurs données au départ (1058 et 337), le programme continue à retourner le bon message.
2.c) Ecrire un programme qui compte le nombre d'heures pleines contenues dans un nombre a de minutes et le nombre de minutes restantes

Réponse 2.a

Réponse 2.b

Réponse 2.c

Exercice 2.1 : Ecrire un programme qui affecte une valeur à la variable a et qui affiche son carré, avec un petit message de votre choix.
Pour chaque programme, on prendra soin de faire un jeu de tests qui recouvre tous les cas, et on choisira des noms de variables évocateurs (nb_livre pour un nombre de livres plutôt que n par exemple)

*Exercice 2.2 : Ecrire un programme qui trouve le nombre de centaines, de dizaines et d'unités d'un nombre a
On pensera à utiliser la division entière (c'est à dire //) : en effet, si on prend 637 par exemple, 637//100 donne le nombre de

centaines.

3) Les entrées et les types

Retour au plan de la page

Pour que l'on puisse entrer des données dans un programme, il existe la fonction input, qui s'utilise de la manière suivante : a=input("entrez un nombre"). Dans la parenthèse se trouve, entre guillemets, le message qui s'affichera et la donnée entrée par l'utilisateur dans la console sera stockée dans la mémoire a.
Attention, la fonction "input" retourne toujours une chaîne de caractères et jamais un nombre !
Dans notre cas, a sera donc une chaîne de caractères.
Il faut savoir que la plupart des langages ne gèrent pas de la même manière les différents types d'objets (par exemple, ils ne réservent pas la même place en mémoire selon que l'on a un entier, un décimal, etc ...). La plupart du temps, Python attribue automatiquement le type qui convient le mieux à chaque objet. Les principaux types auxquels nous allons avoir à faire sont, dans un premier temps :

- les entiers , "integer". Par exemple si on écrit a = 3, Python attribue automatiquement le type integer à a (ce qui correspond à int)
- les décimaux, "float", par exemple b = 1.36
- les chaînes de caractères, "string", par exempel c = "hello" (str en Python)
- les listes, "list", par exemple d = [3,5,7,19], qui est une liste de 4 entiers.

Pour connaître le type d'une variable on écrit : print (type(a)).
Faire des essais avec différents type de variables.
De nombreux autres types existent, et on peut même construire ses propres types !.
Pour plus de détails consulter SWI : les types de données
On peut passer d'un type à un autre : pour transformer une chaîne de caractères en entier, on écrit :
int ("32"), et pour transformer un nombre en une chaîne de caractères on écrit str(32) par exemple.
Voici un exemple :

```
1
2  a = input("entrez un entier")
3  a = int(a) #on transforme la chaîne a en un entier
4  #pour pouvoir faire des opérations
5  b = a**2
6  print("le carré de", a , "est ",b)
```

Vous noterez que l'on met usuellement un espace avant et après le signe "=".
Quand on lance ce programme, la chaîne de caractères "entrez un entier" s'affiche dans la console. Aller dans cette console et y entrer l'entier de votre choix.

Remarque : on peut composer les fonctions de Python. On peut par exemple écrire :
a = int (input("entrez un nombre")).

Pour éviter les problèmes avec la fonction "input" nous allons l'utiliser assez systématiquement avec la fonction "eval".

```
1
2  a = eval(input("entrez une donnée"))   # attention à fermer les deux parenthèses
3  print( type(a) )
4
```

Faire des essais avec différents types de variables.

Exercices pour les futurs isn :
3.a) et 3.b) Reprendre les exercices 2b) et 2c), mais c'est l'utilisateur qui entre a et b dans le premier et le nombre de minutes dans le second.

Réponse 3.a

Réponse 3.b

Réponse 3.b

Exercice 3.1 : Ecrire un programme qui demande deux entiers, puis qui donne le quotient entier et le reste de la division du premier par le deuxième.

4) Les fonctions

Retour au plan de la page

Nous avons déjà rencontré la fonction "print", mais nous allons maintenant apprendre à coder nos propres fonctions.
Voici le code d'une fonction affine f définie par f (x) = 3x - 5.

```

def f(x):
    return 3*x - 5


```

Notes :

- le symbole de multiplication est indispensable
- les ":" marquent l'entrée dans un bloc et la ligne d'après est forcément indentée (décalé) d'une tabulation.
- pour sortir d'un bloc, il suffit de se décaler à nouveau vers la gauche
- "def" est le mot clé pour définir une fonction, f est le nom de la fonction
- derrière le nom de la fonction on met toujours des parenthèses et dans les parenthèses, les variables que l'on passe en paramètres : c'est le vocabulaire qu'il faut connaître.
- ce qu'il y a derrière le "return" est ce qui est renvoyé (ou retourné) par la fonction. Ce "return" fait aussi sortir du bloc de la fonction

Si vous écrivez ce programme et le lancez, rien ne se passera. En effet, cette fonction est comme une entreprise sous-traitante à qui on aurait demandé de se préparer à fabriquer un certain objet. Encore faut-il lancer la commande, pour que l'objet soit effectivement fabriqué.
Pour lancer cette fonction il faut l'appeler dans le programme, après la fin de l'indentation :

```

def f(x):
    return 3*x - 5
a = f(3)
print(a)
print(f(7))
```

Notes :

- à la troisième ligne, on voit que l'indentation est finie, ce qui marque la fin de la fonction.
- on voit que l'on a utilisé la fonction de deux manières différentes : dans la première, on récupère ce que la fonction retourne dans une variable a, dans la deuxième, on affiche directement le résultat. Les deux sont utiles.

La fonction écrite ci-dessus ressemble beaucoup à une fonction mathématique, mais on peut aussi passer en paramètre plusieurs nombres, des chaînes de caractères, des listes ou même rien (la parenthèse est alors vide). Voici des exemples :

```

def coucou():
    print("coucou tout le monde")
def hello(nom):
    print("hello " + nom +" !")
coucou()
hello("Toto")

```

Notes :

- la première fonction ne prend aucun paramètre et ne retourne rien, mais elle exécute une tâche : elle affiche une chaîne de caractères.
- la seconde prend un paramètre appelé "nom". Elle ne retourne rien mais affichera "hello Toto !". L'opération qui correspond au "+" ici s'appelle la concaténation et a juste pour effet d'accoler les chaînes de caractères.
- si vous appelez la fonction "hello" en lui passant 2 en paramètre, vous aurez une erreur, car Python ne peut additionner un nombre et une chaîne de caractères.

A savoir :

- l'instruction "randrange" permet de choisir un entier au hasard. Elle ne fait pas partie des instructions accessibles directement en Python. Il faut charger l'instruction qui se trouve dans le module "random" en utilisant l'instruction : from random import randrange.
  randrange(3,50) choisit un entier au hasard entre 3 et 49 (attention, le dernier entier n'est pas pris).
- len(l) est la longueur d'une liste l.
  Par exemple si l = [13, 8, 9, 17], on a len(l) qui vaut 4.
- Comment atteindre les éléments d'une liste : l[0] est le premier élément de la liste, ici c'est 13, l[1] le deuxième, ici 8 et le dernier est l[3] qui vaut ici 17.
- 0 est l'index du terme 3, tandis que 1 est celui de 8 etc...

Exemple avec des listes :

```
1  from random import randrange
2
3  def choixHasard(liste):
4      index = randrange(len(liste))
5      return liste[index]
6
7  def afficheHasard (liste):
8      index = randrange(len(liste))
9      print("terme choisi : ", liste[index])
10 #---------------------------------------------------------------
11 #programme principal
12 #---------------------------------------------------------------
13 listeMois = ["jan","fev","mars","avr","mai","juin","juil","aout","sept","oct","n
14 listeAn = list(range(1950,2001))    #on transforme le range en une liste
15 mois = choixHasard (listeMois)
16 annee = choixHasard (listeAn)
17 print ( "le mois choisi au hasard entre 1950 et 2000 est ", mois  +  " "  +  str
18 afficheHasard (listeMois)
19
20
```

*Exercice 4.1 :

1)Répondre aux questions sans utiliser Python (à la main).

- Compléter les phrases suivantes :
  "def" est un mot .....servant à .....
  "choixHasard" est le .....de la fonction.
  "liste" est le .....de la fonction
- Si "index" vaut 0 quand on exécute "choixHasard", qu'est ce qui est retourné ?
- Combien de paramètres sont passés à la fonction "afficheHasard", et qu'est ce qui est retourné ?
- De quel type est la variable "mois" ? et la variable "annee" ?
- Comment s"appelle l'opération qui correspond au "+" dans la ligne avec le "print" ?
- A quoi sert le "str" ?
- Donner l'affichage obtenu si le "randrange" de "choixHasard" vaut d'abord 2 , puis 15 lors du deuxième appel de la fonction.
- Quelle est la valeur maximale que peut prendre le randrange quand on exécute choixHasard(listeMois) ?
- Que fait la fonction "afficheHasard" si le "randrange" est 4.

2) Copier le programme dans Spyder. Utiliser le menu "Déboguer" et choisir Déboguer. Au bas du cadre en haut à droite, cliquez sur "Explorateur de variables". Dans ce cadre s'afficheront les valeurs de toutes les variables. Dans la console, taper "n + Enter" jusqu'au bout du programme (ou utiliser le deuxième symbole bleu, en forme de flèche reliant deux points) : ceci fait avancer votre programme pas à pas. Observez attentivement ce qui se passe.

Notes :

- Notez la structure du programme qui commence par les imports, ensuite les définitions de toutes les fonctions, ensuite le programme principal : veuillez utiliser cette structure dans vos programmes.
- tout ligne qui commence par un # est un commentaire et est ignoré lors de l'exécution du programme. Cela sert à informer le lecteur du programme
- les noms des variables et des fonctions doivent faciliter la compréhension du code
- lorsque l'on utilise deux mots pour un nom, on met une majuscule au deuxième ou on utilise le souligné (underscore) entre les deux mots
- le première fonction retourne un terme, qui peut alors être utilisé dans la suite du programme. La seconde ne retourne rien, elle ne fait qu'un affichage.
- une même fonction peut être utilisée plusieurs fois avec des paramètres différents

*Exercice 4.2 :
Ecrire une fonction qui ne prend aucun paramètre, mais qui simule le lancer de deux dés et retourne la somme des nombres obtenus. Ecrire aussi un programme principal qui affiche le résultat (en dehors de la fonction).

5) Les instructions conditionnelles

Retour au plan de la page

Il s'agit des instructions "if", "elif" et "else". Voici un exemple :

```
1
2  a = eval(input("entrez un entier"))
3
4  b = a%2  #reste de la division de a par 2
5  if a == 0:
6      print (a, " est nul et pair")
7  elif b == 0:
8      print(a," est pair")
9  else:
10     print(a," est impair")
11
```

Vous remarquerez :

- que l'opérateur de comparaison s'écrit "==", le "=" est réservé à l'affectation
- que le programme est décomposé en instructions simples et en blocs : le premier bloc commence à "if"
- chaque ligne d'entête d'un bloc se finit par ":"
- la suite du bloc est indentée (cela se fait automatiquement dans Spyder)
- la fin du bloc est signalée par la fin de l'indentation (on tape sur shift et tab)

Les indentations sont très précises, si vous ajoutez un espace de plus en début de ligne, python signalera une erreur (souvent difficile à déceler) !
Les autres opérateurs de comparaison sont : "!=" (différent),"<", ">","<=" et ">=".
Il faut aussi savoir que l'on peut imbriquer plusieurs blocs...

Exercices pour les futurs isn :
5.a) Ecrire un programme dans lequel l'utilisateur entre deux nombres, et le programme détermine le plus grand des deux.
5.b) Ecrire un programme qui détermine si une année est bissextile.
On rappelle qu'une année est bissextile si elle est multiple de 4, mais que si elle est multiple de 100 elle n'est pas bissextile, sauf si elle est multiple de 400.
Indications:

- a est multiple de b si a%b est nul (quand on divise a par b, le reste est nul).
- Attention à ne pas confondre "=" et "==" !

Réponse 5.a

Réponse 5.b

*Exercice 5.1 : le temps
1) Ecrire un programme dans lequel l'utilisateur entre un nombre entier de secondes. Si cet entier est négatif afficher un message d'erreur.
2) Si l'entier est positif trouver combien d'heures pleines correspondent à ce nombre de secondes. On pensera à utiliser la division entière (//) et le modulo (%).
3) Modifier le programme pour convertir le nombre de secondes en heures, minutes et secondes.

*Exercice 5.2 : chaînes de caractères
A savoir : on peut aussi faire des tests du type 'if c in "maths" ', qui teste si un caractère c est dans la chaîne "maths" . On peut aussi tester 'if c not in "maths" ' et faire de même avec une liste.
Ecrire un programme qui détermine si un mot contient un "e" accentué ou pas.

*Exercice 5.3 : valeur absolue
Ecrire une fonction qui retourne la valeur absolue d'un nombre et la tester dans un programme. Voici le canevas d'une solution :

```
1
2  def valAbs(a):
3      """fonction qui retourne la valeur absolue du nombre passé en paramè
4      if ........:
5          return.....
6      else
7          return ......
8  #-------------------------------------------------------------------------
9  #Programme principal(pour tester)
10 #-------------------------------------------------------------------------
11 x =...........    #à entrer par l'utilisateur
12 y = valAbs(x)
13 print (y)
14
```

Note : le commentaire écrit entre les triples guillemets s'appelle un docstring. Il est recommandé d'en écrire pour chaque fonction codée, dès lors que le programme est un peu consistant.

6) Les boucles "for"

Retour au plan de la page

Lorsqu'on veut itérer une opération avec un paramètre prenant des valeurs bien définies, on utilise, en général, la boucle "for".
La syntaxe est : "for i in .....: " et on passe à la ligne en indentant pour mettre les instructions que l'on veut répéter. i s'appelle alors un compteur
A la place des points de suspension, on peut mettre de nombreux types de suites ou de listes (ou autre). Dans un premier temps, nous utiliserons : "for i in range (15)" qui signifie de 0 à 14 , ou range(2,15) qui veut dire de 2 à 14, ou range(2,15,3) qui signifie de 2 à 14 avec un pas de 3 (c'est à dire 2, 5, 8, 11, 14).
Attention, la valeur finale de "range" n'est jamais atteinte, toutefois range(10) contient bien 10 valeurs !
Exemple:

```
1
2  n = eval(input("entrez le nombre d'heures"))
3
4  for i in range (n):
5      print("coucou")
6
7  if n == 12:
8      print("il est midi ou minuit")
9
```

Ce programme modélise la fonction "coucou" d'une horloge.
Autre exemple :

```
k = eval (input("quelle table voulez vous ?"))
for i in range (1,11):
    print(i," * ",k," = ",i*k)
print("terminé")
```

Il faut bien comprendre qu'au premier passage dans la boucle, i vaut 1, le programme effectue toutes les instructions indentées, puis revient au début et incrémente i de 1 (il l'augmente de 1) sauf si on est arrivé à 11. Au deuxième passage, i vaut donc 2 etc....Quand i vaut 11, les instructions indentées ne sont pas exécutées et on éxécute les instructions qui suivent la boucle. Si on écrit :

```
k = eval(input("quelle table voulez vous ?"))
for i in range (1,11):
    print(i," * ",k," = ",i*k)
    print("terminé")
```

le "terminé" sera écrit 10 fois, une fois dans chaque boucle !

Exercice 6.1 :
Etudiez attentivement le programme ci-dessous, sans utiliser Python, pour répondre aux questions posées.

```
def prixSejour(nbJours, nbPersonnes):
    """fonction qui calcule le prix total d'un séjour et le prix par per
    prixTotal = 0
    prixJournee = 80
    for nb in range(nbJours):
        prixTotal = prixTotal + prixJournee
        prixJournee = prixJournee - 5
    prixTotal= prixTotal + 14
    return prixTotal , prixTotal/nbPersonnes

#----------------Programme principal--------------------

nbJours = eval(input("combien de jours voulez vous rester ? "))
personnes = eval(input("combien de personnes ? "))
total , parPersonne = prixSejour(nbJours,personnes)
fidelite = eval(input("Avez vous une carte de fidélité ? 1 pour Oui et 0
if fidelite == 1:
    total = total * 0.9
    parPersonne = parPersonne * 0.9
print("le prix total est ", total, "et par personne cela fait ", parPers

```

1) Compléter un tableau où l'on voit l'évolution des variables "nb", "prixTotal", et "prixJournee" si on exécute la fonction avec 4 jours et 3 personnes.
2) Que valent toutes les variables à la fin du programme si on possède une carte de fidélité.
3) Faites tourner ce programme avec Python en utilisant le mode "Déboguer" pas à pas et l'explorateur de variables, pour vérifier.

Notes :

- Pour retourner plusieurs objets, il suffit de les séparer par des virgules (en fait, pour les connaisseurs, on renvoie un tuple, mais les parenthèses sont omises)
- Pour récupérer les deux objets retournés par la fonction, on utilise aussi deux noms de variables séparés par des

virgules.

- Il faut comprendre que les variables définies dans la fonction, y compris les paramètres, n'existent que dans la fonction. Dans le programme principal, elles sont inconnues. On dit que se sont des variables locales. La variable "nbJours" existe deux fois : une fois localement, dans la fonction, une fois globalement, dans le programme principal. Ajouter, avant le "return" de la fonction "nbJours = nbJours + 1" et "print ("dans la fonction", nbJours). PUis ajouter à la fin du programme "print("dans le programme",nbJours). On aurait, de même, très bien pu choisir comme nom de variable "prixTotal" au lieu de "total" dans le programme principal et de même avec "personnes". Essayez d'ajouter "print (prixTotal)" à la fin du programme : que se passe-t-il ?
- En fait, c'est un peu plus subtil : dans une fonction vous pouvez utiliser une variable qui est définie dans le programme, mais pas la modifier. Si vous voulez la modifier, il faut indiquer au début de la fonction "global" devant le nom de la variable. Pour plus de détails, voir le paragraphe de compléments sur les fonctions.

Exercices pour les futurs isn :
6.a) Soit U la suite définie par $U_{n+1} = 3^* U_n - 1$ et $U_0 = 2$.
Ecrire un programme qui calcule U_{10}

Réponse 6.a

6.b) Reprendre le programme précédent et calculer la somme $S = U_0 + U_1 + ... + U_{10}$.
On prendra une variable S qui vaudra 0 au départ. A l'aide d'une boucle "for" on ajoutera successivement U_0,U_1 etc ...à S.

Réponse 6.b

*Exercice 6.2 : Dessin.
Info : par défaut, on passe à la ligne après chaque instruction "print". En revanche, si on écrit "print("toto",end = " ")", le prochain afichage sera dans la même ligne que "toto", séparé par un espace. Si on écrit :

```
1
2  for i in range(10):
3      print("XO",end = "")
4  #attention ci-dessus on avait écrit end=" " (un espace) et ici : end=""
5
```

on aura une ligne "XOXOXOXOXOXOXOXOXO"
Pour passer à la ligne sans rien afficher : print()
1) Ecrire une fonction qui affiche une ligne de n croix ("x" majuscule) où n est un entier passé en paramètre, et tester cette fonction.
2) Ecrire un programme qui dessine un rectangle de n lignes et p colonnes (où n et p sont entrés par l'utilisateur) du type:
XXXX
XXXX
XXXX
pour 3 lignes et 4 colonnes
On utilisera la fonction du 1).
3) Triangles
Ecrire un programme qui dessine un triangle de n (entré par l'utilisateur) lignes du type :
X
XX
XXX
XXXX
si n vaut 4.

6.3 : Exercice supplémentaire recommandé : suite
On définit la suite V par $V_1 = 2$ et $V_{n+1} = V_n + n - 10$.
1) Ecrire un programme qui calcule les termes de cette suite quand n est entre 1 et 10. Prendre un papier et un crayon et vérifier que les valeurs sont justes.
2) Compléter le programme pour qu'il détermine le minimum de cette suite quand n est entre 1 et 10. Quand ce programme fonctionne bien, changer 10 en 200.
3) Compléter le programme pour qu'il donne l'index correspondant au minimum.

Vérifier en allant d'abord seulement jusqu'à 10.

Réponse 6.3

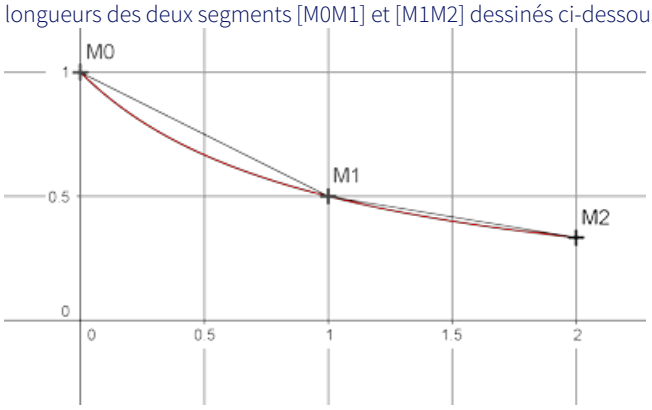
A retenir : toujours vérifier que les valeurs extrêmes sont correctes (on regarde ce qui se passe pour les premiers passages dans la boucle et pour le dernier).

Mini-projet : longueur d'une rampe

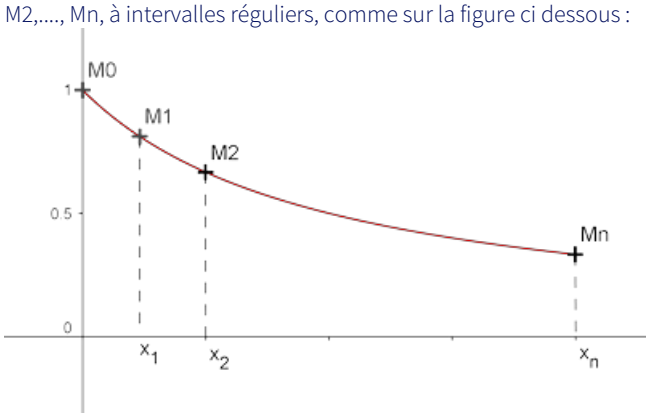
Le profil de la rampe d'une piste de skate-board peut être modélisé par le tracé de la courbe représentant la fonction définie sur l'intervalle [0 ;2] par :

$$f(x) = \frac{1}{x+1}$$

On souhaite déterminer approximativement la longueur de ce profil : pour cela on peut calculer et additionner les longueurs des deux segments [M0M1] et [M1M2] dessinés ci-dessous.



1) Ecrire une fonction longSeg qui prend pour paramètres les abscisses a et b de deux points de la rampe et qui retourne la longueur du segment correspondant. Tester cette fonction.
Note : de manière générale, on teste toutes les fonctions écrites et dans tous les cas possibles.
Utiliser cette fonction pour calculer une première approximation de la longueur de la rampe en ajoutant les longueurs des deux segments dessinés ci-dessus.
Ajouter un docstring à cette fonction, par exemple """cette fonction prend pour paramètres les abscisses a et b de deux points de la rampe et retourne la longueur du segment correspondant""".
2) Pour affiner ce calcul de longueur, on augmente le nombre de segments. Au lieu d'utiliser deux segments, on va maintenant utiliser n segments : on place donc un point M0 comme précédemment, puis n points nommés M1, M2,...., Mn, à intervalles réguliers, comme sur la figure ci dessous :



On appelle x0, x1, x2,...., xn=2 les abscisses des points M0,M1, M2,....,Mn.
a) Ecrire une fonction qui calcule une longueur approximative de la rampe en utilisant ces n segments, n étant passé en paramètre.
Note :
il ne faut pas vouloir écrire la fonction en une seule fois. Il vaut mieux commencer par lui faire afficher des résultats intermédiaires et les vérifier, avant de compléter la fonction. On pourra, par exemple, commencer par faire afficher les abscisses des points, puis les longueurs des différents segments, avant de retourner la longueur approximative de la rampe.
Test : pour n=2, il faut retrouver le résultat de la question 1).
Ajouter un docstring à la fonction.
b) Ecrire un programme utilisant cette fonction et qui affiche la longueur obtenue pour n=10, 20, 30,....,100.

7) Les itérables

Retour au plan de la page

Un itérable est un objet que l'on peut parcourir à l'aide d'une boucle "for". Outre les objets de type "range" utilisés dans le paragraphe précédent, les listes (list) et les chaînes de caractères (str) sont itérables, ainsi que bien d'autres.

```
1
2  ma_liste = [1,6,9,17,15]
3  print("les éléments de ma_liste sont")
4  for i in ma_liste:
5      print(i)
6  ma_chaine = "hello"
7  print("les caractères de ma_chaine sont")
8  for  c in ma_chaine:
9      print(c)
10 mon_range = range(1,11,2)
11 print("les éléments de mon_range sont")
12 for i in mon_range:
13     print(i)
14
```

Pour accéder à l'un des élements de ma_liste, de ma_chaine ou de mon_range, on utilise les crochets : ma_liste[0] est le premier élément, ma_liste[1] le deuxième etc... Et de même avec les listes ou les range.
0 , 1 etc...sont les index des termes (ou indices).

*Exercice 7.1 : Nombre de voyelles dans un mot
Ecrire un programme qui détermine le nombre de voyelles dans un mot sans aucun accent.
On utilisera un compteur, c'est à dire une variable c qui vaut 0 au départ et qui augmente de 1, chaque fois qu'une voyelle est rencontrée.

On est souvent amené à avoir besoin de la longueur d'une liste ou d'une chaîne de caractères.
Il suffit d'utiliser : len(ma_liste).
La grosse différence entre une liste et une chaîne de caractères, c'est qu'une liste est modifiable, alors qu'une chaîne ne l'est pas. Ceci signifie que l'on peut changer un élément d'une liste, par exemple en écrivant ma_liste[0] = 2 * ma_liste[0], mais que ceci provoque une erreur si on utilise ma_chaine.
On peut aussi facilement ajouter un élément à la fin d'une liste :
ma_liste.append(5) ajoutera 5 à la fin de ma_liste.
Attention piège : quel est l'index du dernier élément d'une liste l (on utilisera la fonction len(l) dans la réponse).
Prendre des exemples et vérifier votre résultat.

```
1
2  ma_liste = [1,6,9,17,15]
3  for i in range (len(ma_liste)):
4      ma_liste[i] = 2 * ma_liste[i]
5      print ((le nouveau terme d'index ",i, "est ",ma_liste[i])
6  print (ma_liste)
7
```

On dit alors que l'on parcourt la liste par les index.
Attention, si on écrit, dans cet exemple, ma_liste[5], on provoque une erreur "Out of range", qui signifie que le terme d'index 5 n'existe pas !

*Exercice 7.2 : liste au hasard
1) Ecrire une fonction qui retourne une liste de n entiers choisis au hasard entre 0 et maxi, où n et maxi sont des entiers passés en paramètres, et un programme pour tester cette fonction.
2) Ecrire un programme qui utilise la fonction précédente pour créer une liste de 1000 entiers entre 0 et 500, et qui détermine le nombre de multiples de 3 obtenus. Comment peut-on vérifier le résultat ?
3) Modifier le programme de telle sorte que chaque terme de la liste qui est multiple de 3, soit divisé par 3.
4) Ecrire une fonction "mini" qui détermine le minimum d'une liste passée en paramètre, la tester en utilisant des listes crées avec la fonction du 1) et comparer avec le résultat de la fonction "max" de Python.
5) Ecrire une fonction qui calcule la somme des éléments d'une liste passée en paramètre. Prenez une variable S qui vaut 0 au départ, puis à l'aide d'une boucle "for" ajoutez successivement à S le premier terme de la liste, le deuxième etc...(comparer avec le résultat de la fonction "sum" de Python).

Notez bien cette méthode car elle est très classique pour calculer une somme d'un certain nombre de termes de la même forme ou de la même liste.

*Exercices 7.3 : tri par sélection

On donne le programme suivant :

```
1
2  def f (index ,l):
3      """..................."""
4      indMin = index
5      mini = l[index]
6      print("pour l' index",index,"mini vaut au départ ",mini)
7      for i in range(index + 1,len(l)):
8          if l[i] < mini:
9              indMin = i
10             mini = l[i]
11             print ("le nouveau mini est", mini, "pour l'index ",inde
12     l[index],l[indMin] = l[indMin],l[index]
13     return l
14
15 l = [17, 85 , 3, 2]
16
17
18 for i in range(len(l) - 1):
19
20     print("pour i = ",i, "mini vaut au départ", l[i])
21
22     f(i,l)
23     print("pour i = ", i, "la liste est devenue",l)
24
```

1) Faire tourner ce programme "à la main" sans utiliser Python et indiquer tous les affichages obtenus dans un tableau avec 10 colonnes et une ligne pour i,une ligne pour l[i], une ligne pour j, une ligne pour l[j],une ligne pour mini et une ligne pour l. Décrire en quelques phrases ce que fait ce programme et compléter la docstring de "f". Donner un nom plus expressif à la fonction "f".
2) Faire tourner le programme avec Spyder et comparer. Essayer ce programme avec des listes créées grâce à la fonction de la question 1) de l'exercice 7.2.
3) Ecrire une fonction qui prend en paramètre une liste et qui retourne la liste triée et la tester. Attention au cas où la liste entrée en paramètre est vide : votre fonction ne doit pas provoquer d'erreur !
Note : l.sort() est une méthode de liste qui trie une liste l, de manière beaucoup plus rapide.

*Exercice 7.4 :

1) Ecrire une fonction qui prend en paramètre un entier positif n et retourne le binaire associé sous forme de chaîne de caractères. On pourra prendre une liste vide, puis dans un premier temps, diviser n par 2 (division entière), récupérer le reste dans la liste et remplacer n par le quotient. Il suffira ensuite de recommencer autant que nécessaire. Faire un essai sur un papier en partant de n = 197 : écrire les valeurs successives de n et de la liste. Ecrire l'algorithme en langage naturel sur votre papier avant de le programmer.
Méthodes utiles sur les listes :

- l.reverse() transforme la liste l en inversant l'ordre des termes
- "".join(l) permet de transformer la liste l en une chaîne de caractères, à condition que chaque terme soit déjà une chaîne de caractères. Par exemple "".join(['t','o','t','o']) donne 'toto'.

Attention au type des objets manipulés : on sera amené à faire du transtypage, c'est à dire, à transformer un type en un autre en utilisant les fonctions "int()" et "str".

2) Ecrire un algorithme sur papier, permettant de transformer un binaire en décimal.
Ecrire une fonction qui prend en paramètre un binaire, passé sous forme de chaîne de caractères, et qui retourne le décimal associé.
Méthode utile sur les chaînes :
list(ma_chaine) donne une liste de tous les caractères de la chaîne (qui sont de type str).

3) Ecrire une programme où l'utilisateur a le choix entre l'utilisation de ces deux fonctions.

8) Les boucles "while"

Retour au plan de la page

Ce type de boucle sert à itérer des instructions tant qu' une certaine condition est satisfaite.
Syntaxe : "while a< 100 :" est la ligne d'entête du bloc et signifie "tant que a est inférieur à 100".
Exemple :

```
s = 0 #initialisation de la somme à 0
i = 0 #initialisation du nombre i
while s < 50: #  tant que la somme est inférieure à 50
    i = i + 1      #  i augmente de 1
    s = s + i      #  on ajoute i à s
print("pour ",i, " la somme dépasse 50 et vaut ",s)
```

*Exercice 8.1
Faire tourner ce programme "à la main", sans le lancer avec Python, pour déterminer ce qu'il fait et quels sont les résultats. On fera un tableau dans lequel on donnera les valeurs des variables "s" et "i" au fur et à mesure de l'avancée du programme.

Exercices pour les futurs ISN
8.a) Ecrire un programme qui simule le lancer d'un dé jusqu'à ce qu'on obtienne 6. Le programme doit également compter combien de jets ont été nécessaires.

Réponse 8.a

8.b) Modifier le programme précédent, pour répéter 1000 fois l'expérience et compter combien il aura fallu de jets de dés en moyenne pour obtenir 6.

Réponse 8.b

8.c) On a une mare de 100 m^2 et un nénuphare de 9cm^2. La surface occupée par les nénuphares double chaque jour jusqu'à occuper toute la surface de la mare. Combien de jours sont-ils nécessaires pour cela ?

Réponse 8.c

*Exercice 8.2 : rebonds
Dans cet exercice on prendra grand soin de vérifier les résultats obtenus, en diminuant, par exemple, le nombre de boucles attendu, de manière à pouvoir contrôler grâce à un calcul à la main.
Après le premier impact au sol, une balle fait un premier rebond à 3m de haut, puis chacun des rebonds successifs a une hauteur égale à 80% du rebond précédent.
1) Calculer la hauteur du rebond n°i pour i allant de 1 à 10 (pour i=1, le rebond fait 3m), puis la distance parcourue en 10 rebonds, à partir du premier impact avec le sol.
2)a) Calculer le chemin parcouru par la balle à partir du premier impact avec le sol : on négligera tous les rebonds ne dépassant pas 2cm de haut.
2)b) Modifier le programme pour avoir également le nombre de rebonds.

Notes :

- On peut utiliser "and" et "or" dans les conditions. Par exemple : "if a == 3 or a< 0"
- On peut utiliser les instructions "break" pour sortir complètement d'une boucle et "continue" pour aller à l'itération suivante, sans exécuter d'instruction dans l'itération en cours.

9) Compléments sur les fonctions

Retour au plan de la page

Syntaxe d'une fonction

La syntaxe générale d'écriture d'une fonction est :

```
def nom_fonction(param1,param2,.....):
    """aide de la fonction(docstring)"""
    bloc d'instructions
    return res1,res2,.....
# ou
def nom_fonction():  #s'il n'y a aucun paramètre d'entrée
    """aide de la fonction(docstring)"""
    bloc d'instructions
    return res1,res2,.....
# ou
def nom_fonction(param1,param2,.....):
    """aide de la fonction(docstring)"""
    bloc d'instructions
    return   #s'il aucun résultat n'est retourné
```

Rappel : règles d'écriture d'un programme :

- le programme commence par les imports (from.....import etc....)
- on écrit ensuite toutes les fonctions
- pour finir, on écrit le programme principal
- chaque fonction doit avoir un nom cohérent avec ce qu'elle fait (du type prixHT, longueurSegment ...) et une documentation d'aide appelée docstring qui se place juste après la première ligne de définition entre triple guillemets. Ceci peut être omis, si ce que fait la fonction est évident.
- chaque variable doit avoir un nom indiquant ce qu'elle représente
- ne pas utiliser deux fois le même nom pour des objets différents !
- le programme doit être commenté pour qu'un lecteur extérieur comprenne ce qui se passe

Variables locales et globales

Attention : les variables définies dans une fonction sont effacées de la mémoire quand le programme sort de la fonction. Dans la première fonction ci-dessous, a et b n'existent que pendant que le pointeur du programme est dans la fonction f.
Etudiez attentivement les deux petits programmes suivants :

```
def mult (a,b):
    c = a*b
    return c

#prgm principal

mult(3,4)
print(c)
```

Le programme ci-dessus provoque une erreur. Dans le programme principal c n'existe pas. Ce n'est qu'une variable locale de la fonction mult.
En revanche, le programme suivant fonctionne :

```
def mult (a,b):
    c = a*b
    return c

#prgm principal

d = mult(3,7)
```

```
print (d)
e = 36
f = 45
print(mult(e,f))

```

Le résultat de la fonction mult est sauvegardé dasn la variable d, on aurait pu choisir n'importe quel autre nom, y compris c !
a et b n'existent que dans la fonction mult : ce sont justes des noms que l'on donne aux deux arguments de cette fonction.
Les variables définies dans le programme principal sont globales. Elles peuvent être utilisées dans une fonction, mais pas modifiées dans une fonction sans instructions supplémentaires.
Voici des exemples :

```

#Programme 1
def essai(a):
    return a + n

n = 5
print(essai(3))

#Programme 2
def autreEssai(a):
    n = 1
    return a + n

n = 5
print(autreEssai(3))
print(n)

#Programme 3
def dernierEssai(a):
    n = n + a
    return n
n=5
print(dernierEssai(3))
```

- Le premier programme affiche 8 : dans la fonction, ne trouvant pas la variable locale n, le programme cherche une variable globale n et la trouve : il l'utilise....
- Le deuxième programme affiche 4, puis 5 : dans la fonction, le programme utilise la variable locale n, alors que en dehors il utilise la variable globale n.
- Le troisième programme provoque une erreur, car il est interdit, sans instruction supplémentaires, de modifier une variable globale dans une fonction. On peut contourner ce problème de deux manières :
  - on peut passer n en paramètre et la fonction devient : "def dernierEssai(a,n)" et l'instruction de la ligne 23 : "print (dernierEssai (3,n))".
  - on peut dire à la fonction que n est une variable globale en écrivant à la ligne 20 : global n.

Des précisions supplémentaires sur les fonctions :
SWI : les fonctions
openclassroom : les fonctions

Appel d'une fonction dans une autre fonction et récursivité

Dans l'exemple suivant on voit que l'on peut, sans problème, appeler une fonction dans une autre fonction :

```

from math import pi
def cube(n) :
    """renvoie le cube d'un nombre n"""
```

```
    return n**3
def volumeSphere(r):
    """retourne le volume d'une sphère de rayon r"""
    return 4/3*cube(r)*pi
print(volumeSphere(2))
```

Dans une fonction, on peut même appeler la fonction elle-même : on dit alors que la fonction est récursive. Voici un exemple : on veut calculer n! qui s'appelle "factorielle n" et qui vaut n*(n-1)*.....*2 * 1 si n est un entier strictement positif.
On voit aussi que n! = (n-1)!* n sauf si n vaut 1 et alors n! = 1. C'est une définition récursive de n! (elle utilise la récurrence). Ceci donne la fonction suivante :

```

def factorielle(n):
    if n == 1:
        return 1
    else:
        return n*factorielle(n-1)

print(factorielle(3))
```

Cette méthode doit s'utiliser avec prudence : il doit toujours y avoir le ou les cas de base (ici le cas n==1).
Par ailleurs, ce type de fonction utilise beaucoup de mémoire, car les résultats intermédiaires doivent être stockés.

*Exercice 9.1 : somme de nombres
Ecrire une fonction récursive qui calcule la somme des éléments d'une liste d'entiers passée en paramètre.
On notera que l'instruction "del(l[i])" permet d'effacer l'élément d'index i de la liste l.

*Exercice 9.2 : tri par sélection
Reprendre le tri par sélection et le programmer de façon récursive.

Il existe de nombreuses autres méthodes de tri : vous devez connaître la méthode de tri par sélection et celle du tri par fusion dont le principe est expliqué dans le document ci dessous, à étudier.
Description du tri fusion

10) Le module "Turtle"

Retour au plan de la page

Python possède de nombreuses bibliothèques regroupant des catégories de fonctions. Elles sont appelées des modules, au même titre que les fichier ".py" que vous créez et qui deviennent vos propres modules.
Pour utiliser le module "Turtle", écrivez au début de votre programme :
from turtle import * (sans oublier "*")
Des explications sur les modules seront données au chapitre ???.
Cette tortue est une interface graphique dans laquelle vous avez un traceur que l'on va apprendre à commander.
Une documentation très complète est disponible sur : docs.python.org
Le repère de cette fenêtre graphique a pour origine son centre, un axe des abscisses horizontal orienté vers la droite et un axe des ordonnées orienté vers le haut.Au départ la tortue (le traceur) est au centre de l'écran (de coordonnées (0,0)) et orientée vers la droite.

Les déplacements :

- fd(dist) : fait avancer la tortue de dist (fd ou forward)
- bk(dist) : la même chose en reculant (ou backward)
- left(angle) : change la direction du mouvement en la faisant tourner de angle vers la gauche(en degrés par défaut)
- right(angle) : de même vers la droite
- goto(x,y) : déplace la tortue au point de coordonnées (x,y)
- circle(r) : trace un cercle de rayon r.
- undo() : annule la dernière action (peut être répété).
- reset() : remet la tortue à l'origine et efface le tracé
- home() : remet la tortue à l'origine mais sans effacer le tracé
- dot(r,"couleur") : trace un point de rayon r et de la couleur choisie (ex : "red", "blue" etc...)

Propriétés du traceur

On peut évidemment changer les paramètres du traceur :

- pu() (ou penup()) : lève le "stylo" , la tortue se déplace, mais sans tracé
- pd() (ou pendown()) : redescend le "stylo"
- pensize(nb) : fixe la largeur du "stylo" à nb
- color("coul") : fixe la couleur du stylo ("red", "blue" etc..)
- speed(n) : règle la vitesse. Si n = 0, le tracé est quasi instantané, si n = 10, il est rapide, si n = 1 il est très lent

La fenêtre

Les propriétés de la fenêtre peuvent également être modifiées :

- setup(largeur,hauteur) : règle la largeur et la hauteur (en pixels) de la fenêtre
- bye() : permet de fermer la fenêtre
  S'il n'y a aucune boucle dans votre programme, aucune pause, vous n'aurez guère le temps de voir votre tracé
- mainloop() : permet d'entrer dans une boucle d'attente
  ceci vous donne le temps de voir votre tracé, la fenêtre se ferme en cliquant sur la croix
- exitonclick() : permet de fermer la fenêtre avec un clic
- clear() : efface la fenêtre, sans bouger la tortue

Nous sommes prêts pour un exemple simple, avant les exercices

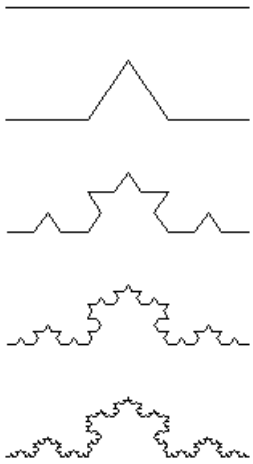
```
1
2  from turtle import *
3  try :
4      speed(5)  #d'habitude on prend speed(0) pour avoir un tracé rapide
5      setup(600,600)
6
7
8
9      for i in range(4):
10         fd(50)
11         right(90)
12
13     mainloop()
14 except:
15    #ce bloc permet de récupérer des infos en cas d'erreur
16
17    traceback.print_exc()
18 finally:
19     bye()
20
```

Dans tous les programmes utilisant une fenêtre graphique, il est recommandé d'utiliser la structure "try, except, finally" vue dans l'exemple précédent : ceci permet d'éviter les plantages et de récupérer les erreurs (grâce au except).
Ecrire "mainloop()" à la fin de votre programme pour maintenir la fenêtre graphique ouverte.
Choiisr "speed(0)" au début du programme pour avoir un tracé rapide.
Si Spyder indique une erreur, alors que le même programme avait déjà fonctionné, aller dans "Consoles" et "Redémarrer le noyau".

*Exercice 10.1 :
1)Ecrire un programme qui trace un escalier de 20 marches .
2)Ecrire une fonction récursive pour tracer un escalier de n marches.

*Exercice 10.2 : flocon de Von Koch:
Le flocon de Von Koch est une figure qui part d'un segment. Ce segment est coupé en trois parties égales, et

sur le segment du milieu on construit un triangle équilatéral. On recommence ensuite l'opération sur chacun des segments obtenus. Ceci donne un objet fractal.



1) Ecrire une phrase décrivant de manière récursive le tracé du flocon : vous décrirez donc le tracé au rang n, en utilisant le tracé au rang n-1, pour n entier strictement positif.
2) Programmer cette fonction avec la "Turtle".
Note : le "vrai" flocon de Von Koch part d'un triangle équilatéral et utilise le même principe.

11) Dichotomie et complexité

Retour au plan de la page

Merci à Guillaume Connan de l'IREM de Nantes dont l'article "La complexité c'est simple comme la dichotomie" a inspiré ce paragraphe et en particulier l'exemple du début.

*Exercice 11.1 : L'oeuf
Des élèves ont fabriqué un oeuf synthétique et veulent tester sa résistance à la chute : ils veulent savoir à partir de quel étage de leur lycée, l'oeuf se brise. Leur lycée compte 2^n étages, où n est un entier strictement supérieur à 1. On suppose que l'oeuf ne se casse pas si on le lance du rez-de-chaussée, mais qu'il se casse du dernier étage.
1) Première méthode : on part du premier étage, on essaie, si l'oeuf n'est pas cassé, on monte au deuxième etc....
Avec cette méthode combien fait-on d'essais dans le pire des cas ?
2) Deuxième méthode : la dichotomie qui vient du grec, et veut dire "couper en deux".
On essaie l'étage du milieu. Si l'oeuf survit, on essaie l'étage au quart...etc. A chaque étape, on divise par deux le nombre de solutions possibles.
Combien d'essais fait-on au maximum ?
Comparer avec le résultat de la question 1 pour différentes valeurs de n.
La complexité d'un algorithme mesure son efficacité en comptant le nombre d'opérations nécessaires, qui sera souvent une fonction d'un ou plusieurs paramètres. Ici le paramètre est le nombre d'étages.
Si on programme un algorithme correspondant à la première méthode on aura un nombre d'opérations grossièrement proportionnel à nbEtages. On dit que la complexité est d'ordre nbEtages ou qu'elle est linéaire.
Si on programme le second (voir question suivante), on aura un nombre d'opérations grossièrement proportionnel à n, où n est tel que :

$$2^n \leq \text{nbEtages} \leq 2^{n+1}.$$

Vous verrez (ou avez déjà vu) que ce nombre n vaut environ $\log_2$ (nbEtages). On dit que la complexité est d'ordre $\log_2$, ou d'ordre logarithmique.
3) Ecrire une fonction qui trouve par dichotomie le bon étage en partant d'un nombre d'étages quelconque "nbEtages". On pourra programmer cette fonction de manière itérative (avec des boucles) ou récursive.

*Exercice 11.2 : Classement
On dispose d'une liste triée l et on veut insérer un élément dans cette liste, de telle sorte que la liste reste triée.
A savoir : l'instruction "l.insert(2,p)" insère l'élément p à la place d'index 2 dans la liste l.
Ecrire une fonction qui réalise cette opération. On pourra utiliser, au départ, une petite liste écrite dans le programme, puis utiliser la fonction créée à l'exercice 7.2 et qui génère des listes aléatoires.
Noter au passage, l'intérêt d'utiliser des fonctions !

12) Utiliser des tableaux à deux dimensions

Retour au plan de la page

De nombreux programmes, en particulier de jeux, utilisent des tableaux, par exemple pour modéliser les cases d'une bataille navale ou d'un jeu de suddoku.
Voici un premier exemple :

1. Créer une liste n de trois 0. On veut créer une liste t de trois termes, chacun des termes étant égal à n, c'est-à-dire t=[[0,0,0],[0,0,0],[0,0,0]]
   On essaie le procédé suivant, qui a l'avantage de permettre de faire facilement la même chose en remplaçant 3 par 5, 15 ou n'importe quel autre entier positif.

```
1
2  n = [0]*3   #crée une liste de trois 0
3  t = [0]*3
4  for i in range(3):
5      t[i] = n   #on remplace le 0 qui est au départ dans
6      #t[i] par la liste n
7      print(t[i])
8
```

   Ceci affichera également t, une ligne en dessous de l'autre, c'est à dire sous forme de tableau.
2. Ecrire t[1][2]=5 et afficher à nouveau t.....Que se passe-t-il ?
3. Ce problème vient du fait que les listes sont manipulées par leurs adresses en mémoire. Chaque ligne de t est la même liste n : on a juste copié la référence de la liste n à un nouvel endroit.
   Pour éviter ceci on fait :

```
1
2  n = [0]*3
3  t = [0]*3
4  for i in range(3):
5      t[i] = list(n)   #ceci crée une copie de la liste n
6      # et ainsi on évitera le problème précédent.
7
```

   Reprendre la question b) et constater la différence. Où s'affiche le 5 ?

t[1][2] est donc le terme qui se trouve à la ligne d'index 1 (la deuxième ligne, car la première est d'index 0) et à la colonne d'index 2.
Dans la suite on dira que c'est la case de coordonnées (1,2).
Attention (l,c) correspond à la ligne l et à la colonne c en partant du coin en haut à gauche (et de l'index 0).

Attention, cette méthode de copie n'est valable que pour des listes simples. Pour copier des listes de listes, il faut utiliser la fonction "deepcopy" du module copy.
Il faut bien noter que t[1] représente la ligne de numéro 1 (deuxième ligne) du tableau.

Le module numpy de python permet de traiter plus simplement les tableaux. Il faut évidemment importer les fonctions nécessaires.
En voici quelques unes :

- array([[2,4],[9,7],[3,5]) définit in extenso un tableau de trois rangées de deux éléments bien définis.
- zeros((nbLignes,nbColonnes),'type') construit un tableau avec nbLignes lignes et nbColonnes colonnes où chaque élément est initialisé à 0. Le type est : 'i'=entiers, 'f'=float32, 'd'=floatdouble, 'c'=caractères
  Exemple : a= zeros((10,10),'i') crée un tableau de 10 lignes et 10 colonnes, dont chaque élément est un entier initialisé à 0.
  a[0,2] = 5 met un "5" dans la case qui est à la ligne de numéro 0 (première ligne) et à la colonne de numéro 2.
  Si n=(3,2), a[n] correspond à a[3,2]. On dira que c'est la case de coordonnées (3,2).
  print(a) affiche le tableau ligne par ligne.

Exercice 12.1 : naviguer dans un tableau
Supposons que l'on ait un tableau de 10*5 cases. On se trouve sur la case appelée deb de coordonnées (l,c).
1) Combien ce tableau compte-t-il de lignes et combien de colonnes ? Quelles sont les valeurs possibles de l et c ?
2) Quelles sont les coordonnées de la case au dessus de deb ? Cette case existe-t-elle toujours ? Quelle sont les coordonnées de la case à droite de deb, si elle existe ?

Il faut donc faire attention dans un tableau aux "effets de bord" .....
3) Ecrire un programme qui crée un tableau "sol" de 3*3 zéros.
Choisir dans ce tableau une case au hasard, et y mettre un 1.
Afficher le tableau.
4) Créer un second tableau "essai" de même dimensions que le premier.
5) Ecrire une fonction qui ne prend aucun paramètre. Cette fonction demande à l'utilisateur d'entrer un numéro de ligne et un numéro de colonne. Si dans le tableau "sol" il y a un 1 à la case demandée, la fonction met un 1 dans la case correspondante du tableau "essai", l'affiche, affiche un message "gagné" et renvoie 1. Sinon, la fonction met un 9 dans la bonne case du tableau "essai", l'affiche, affiche un message "perdu" et renvoie 0.
6) Compléter le jeu pour que le joueur puisse jouer tant qu'il n'a pas trouvé.

Mini-projet : travail sur la modification d'images.

Nous allons utiliser la bibliothèque PILLOW.
Vous trouverez une documentation en anglais sur ce site :
http://pillow.readthedocs.org/en/latest/reference/Image.html
Votre programme devra commencer par : import PIL.Image as Image, cela permet d'écrire simplement Image, au lieu de PIL.Image dans de nombreuses instructions.
Voici les instructions dont vous aurez besoin :

- pour ouvrir une image et l'appeler im écrire : im = Image.open("nom_fichier.jpg") (attention le fichier doit être rangé dans le même dossier que votre programme)
- pour récupérer la largeur l (nombre de colonnes) et la hauteur h (nombre de lignes)de l'image : (l,h) = im.size
- pour charger les valeurs des différents pixels : pix = im.load()
  pix est un tableau qui a l colonnes et h lignes
- pour obtenir le pixel qui est à la ligne lig et la colonne col on écrit alors pix[col,lig] et on obtient un triplet (R,G,B).
  Le premier pixel est pix[0,0] et le dernier est pix[l-1,h-1]
- pour obtenir la valeur du rouge, du vert et du bleu d'un pixel : on appelle par exemple data = pix[col,lig]. data ressemble alors, par exemple,à (128,245,32) C'est une donnée de type RGB. data[0] est la composante rouge, data[1] est la verte et data[2] est la bleu.
- pour changer la valeur d'un pixel : pix[col,lig] = (R,G,B) avec les nouvelles valeurs de R, G et B
- pour enregistrer l'image im : im.save("nom_image.jpg") , attention si cette image existe déjà, elle sera écrasée...

Voici un exemple de programme qui affiche simplement la couleur des pixels d'un carré de taille 10*10 se trouvant en haut à gauche de l'image, puis les transforme en gardant la composante rouge mais en mettant les composantes vertes et bleues à 255.
Pour l'exercice, choisir une image colorée au format bmp sur internet et l'enregistrer.

```
1
2  import PIL.Image as Image
3
4  im = Image.open("mon_image.bmp") # on ouvre l'image, attention l'adresse
5  #l'image n'est pas dans le même dossier que le programme
6
7  pix = im.load()       #charge l'image de départ
8  (l,h) = im.size     #l est la largeur = le nombre de colonnes, h la haut
9
10 for colonne in range (10):      #on va parcourir 10 colonnes à partir du
11     for ligne in range(10):     #puis 10 lignes dans la colonne choisie
12         data = pix[colonne,ligne] #on stocke dans la variable data le tr
13                                   #à la ligne et la colonne données
14         print(data)
15         pix[colonne,ligne] = (data[0],255,255)  #data[0] est la composan
16 im.save("essai.jpg")        #on enregistre l'image sous un autre nom
17
```

Essayer ce programme et s'assurer de bien le comprendre avant de passer à la suite. Remarque : on peut utiliser divers formats d'images à la place des "jpg" ou "bmp".
1) Ecrire un programe qui change en rouge tous les pixels de la première ligne d'une image ?
Ecrire un programme qui change en rouge les dix premières lignes de votre image.
2) Ecrire un programme transformant une image de couleur en niveau de gris, on remplacera les composantes

2) Ecrire un programme transformant une image de couleur en niveau de gris, on remplacera les composantes rouge, verte et bleue par uen moyenne des composantes de départ.
3) Ecrire un programme transformant une image de couleur en noir et blanc, en utilisant un seuil au-delà duquel le pixel sera blanc.
4) Ecrire un programme réalisant le négatif d'une image en couleur.
5) Réaliser une autre transformation de votre choix (detecter les contours d'une image, l'aggrandir, la flouter, afficher 4 fois la même image à un format quatre fois plus petit (type photo d'identité), ...).

Mini projet complémentaire : bataille navale

Cahier des charges.
Les questions de cet exercice sont situées après ce cahier des charges.
On veut modéliser une bataille navale, dans un tableau de 5*5 cases. Le programme place les bateaux, l'utilisateur devine leurs emplacements.

- Le programme placera 3 bateaux de deux cases (ayant un côté commun) qui seront modélisés par des cases contenant 2 pour le premier bateau, 3 pour le second et 4 pour le dernier et 6 bateaux de 1 case, modélisés par des 1 dans un tableau appelé "tab"
- Le programme ne pourra pas mettre deux bateaux sur la même case
- L'utilisateur pourra proposer un n° de ligne et un n° de colonne
- Le programme affichera, à chaque coup, un deuxième tableau (appelé "trouve") de 5*5 cases dans lequel apparaîtront les cases proposées avec un "9" si la case correspondante du tableau "tab" ne contient pas de bateau, avec le n° du bateau si la case correspondante de "tab" en contient un et avec des "0" pour les cases non proposées : c'est le tableau de jeu de l'utilisateur.
- L'utilisateur pourra proposer des cases tant que tous les bateaux n'auront pas été trouvés.
- Le programme affichera un message lorsque tous les bateaux auront été trouvés.
- Prolongements possibles :
  ajouter un affichage "dans l'eau", "touché" ou "coulé" après chaque coup.
  modifier les dimensions pour avoir des tableaux 10*10 avec 4 bateaux de 2 cases et 10 bateaux de 1 case.

Note : dans tout l'exercice, une case est libre si elle correspond à un "0" dans le tableau "tab".
Partie A : jeu avec uniquement des bateaux de une case : on en mettra 12 dans cette partie

1. Créer un tableau "tab" de 5 lignes et 5 colonnes ne contenant que des zéros avec la fonction "zeros" de numpy.
2. Ecrire une fonction "place_bat1" qui ne prend aucun paramètre et qui place au hasard un "1" dans le tableau "tab" : on choisira au hasard le numéro de la ligne et le numéro de la colonne.
3. Modifier cette fonction pour quelle place 19 chiffres "1" dans le tableau "tab".
   Attention à ne pas mettre deux bateaux sur la même case. Ecrire un programme pour tester cette fonction, et le lancer 3 ou 4 fois (s'assurer que l'on a bien 19 bateaux à chaque fois !). Quand cela fonctionne, modifier votre fonction pour qu'elle ne crée plus que 12 bateaux de 1 case.
4. Créer un tableau "trouve" de 5*5 cases avec des zéros dans chaque case.
5. Ecrire une fonction "touche" qui prend en paramètre un numéro de ligne et un numéro de colonne. Plusieurs cas sont possibles :
   Si la case a déjà été demandée, la fonction retourne 0.
   Sinon:
   si il y a un bateau sur la case correspondante du tableau "tab", la fonction met un "1" dans la case correspondante du tableau "trouve" et retourne 1 (pour indiquer qu'un nouveau bateau a été trouvé).
   s'il n'y a pas de bateau dans la case, la fonction met un "9" dans la case correspondante du tableau "trouve" et retourne 0.

   Cette fonction pourra servir à compter le nombre de bateaux trouvés
   Tester cette fonction plusieurs fois et afficher les deux tableaux "tab" et "trouve" pour vérifier que le comportement est correct
6. Ecrire un programme où l'utilisateur peut proposer des cases tant que les 12 cases où il y avait des bateaux n'ont pas été trouvées en affichant le tableau "trouve" après chaque coup..
7. Terminer en faisant afficher un message quand tous les bateaux ont été trouvés.

Partie B : avec des bateaux de deux cases

1. Ecrire une fonction "case_libre" qui prend pour paramètres un numéro de ligne x et un numéro de colonne y. Cette fonction retourne une liste l contenant les coordonnées des cases disponibles autour de la case de coordonnées (x,y) (c'est à dire les cases ayant un côté commun avec la case de coordonnées (x,y) et qui sont libres). Attention aux "effets de bord " !
2. Ecrire un programme qui utilise la fonction "place_bat1" puis qui teste la fonction "case_libre" en l'appliquant à différentes valeurs correspondant à des cases qui sont au bord, qui sont dans un coin ou qui sont au centre du tableau.
3. Modifier la fonction précédente pour qu'elle place les trois bateaux de deux cases (l'un en mettant des 2, le suivant avec des 3 et le dernier avec des 4) et la tester.
4. Compléter le programme précédent pour qu'il place aussi 6 bateaux de une case et tester ce programme.
5. Modifier la fonction 'trouve" pour qu'elle affiche le numéro des bateaux trouvés.

Mini-projet difficile : sudoku

D'après une idée du HS n°30 de TANGENTE sur les algorithmes (page 63)

1. Pour représenter les nombres dans un sudoku, il faut un tableau t de 9 lignes et 9 colonnes, dans laquelle on ne met que des 0 au départ.
   t[0,3]=6 met un 6 pour le quatrième nombre de la première ligne.
   Notre sudoku sera terminé quand on n'aura plus aucun 0, c'est-à-dire quand le produit de tous les éléments de t sera non nul.

2. On va créer un deuxième tableau de même dimension appelé "pos" qui contiendra 81 fois la liste [1,2,3,4,5,6,7,8,9] : ceci représentera pour chaque case du sudoku, la liste des chiffres possibles. Au fur et à mesure de la résolution du sudoku on va retirer des chiffres de ces listes.
   Le module numpy est difficile à utiliser dans ce cadre, on va s'en passer.
   Remarque : pos[5][6] sera la liste de tous les chiffres possibles pour la case qui est à la ligne 5 et à la colonne 6 : au départ elle vaudra [1,2,3,4,5,6,7,8,9]

a. Commencer par créer une liste "essai" de 5 termes, chacun des termes étant la liste appelée chiffres = [1,2,3,4,5,6,7,8,9].
   Test : taper essai[0]=[18] et imprimer essai. Le 18 ne doit apparaître qu'une seule fois, sinon, vous devez recommencer, vous avez mis dans "essai" cinq la même liste…Vous devez mettre cinq copies de la liste chiffres. En imprimant "essai" vous devez obtenir :
   [[18],[1,2,3,4,5,6,7,8,9],[1,2,3,4,5,6,7,8,9],[1,2,3,4,5,6,7,8,9],[1,2,3,4,5,6,7,8,9]].
   Quand cela est correct, vous pouvez créer, en vous inspirant de ce que vous venez de faire, votre tableau "pos" : pos[0][0] doit être égal à la liste "chiffres" , pos[1][0] de même etc…
   Pour faire des tests on peut écrire : pos[5][6]=[1,2] ce qui signifierait que pour cette case, seuls les chiffres 1 et 2 sont possibles. Afficher "pos" et vérifier qu'une seule liste vaut [1,2] et que les 80 autres sont bien [1,2,3,4,5,6,7,8,9]. Si cela est bien le cas, reprendre pos[5][6]=chiffres, votre tableau "pos" est au point !

b. Ecrire une fonction "Possible" qui prend en paramètre un n° de ligne, un numéro de colonne et un nombre et qui retourne 1 (ou "True" ou autre chose) si le nombre fait partie des possibles pour la case correspondante et 0 sinon (ou "False" ou autre…).La tester !

c. Ecrire une fonction "Impossible" qui prend en paramètre un n° de ligne, un n° de colonne et un nombre, et qui retire le nombre de la liste des possibles de la case correspondante. La tester.

3. A faire sur un papier : si on place un 6 à la ligne 4 et la colonne 7, lister toutes les cases où le 6 devient impossible en faisant un schéma.
   De même si on place un 6 à la ligne x et la colonne y, lister toutes les cases où le 6 devient impossible. Pour les blocs on pourra utiliser la division entière par 3, par exemple si x=4 et y=7 on est dans le bloc 4//3 =1 horizontalement et 7//3=2 verticalement : on exprimera en fonction de x et de y les coordonnées de la première case de ce bloc. Le 6 est impossible dans toutes les cases de ce bloc . Exprimer en fonction de x et de y les coordonnées de ces cases.

4. Ecrire une fonction "Place" qui prend en paramètre un n°de ligne, un n° de colonne et un nombre et qui place le nombre dans la case correspondant à la ligne et à la colonne, puis qui retire le nombre de la liste des possibles pour toutes les cases concernées (ligne, colonne, bloc).

5. Ecrire une fonction "VerifCase" qui prend pour paramètre un n° de ligne et un n° de colonne et qui place un nombre dans la case correspondante (avec la fonction "place") à la condition que ce nombre soit le seul possible.

6. Ecrire une fonction "VerifColonne" qui prend pour paramètre un n° de colonne et un nombre, et qui place ce nombre dans la colonne, à condition que ce soit la seule place possible pour ce nombre dans la colonne.

7. Ecrire de même une fonction "VerifLigne" et une fonction "VerifBloc".

8. Ecrire un programme de résolution de sudoku en combinant ces différentes fonctions jusqu'à ce que la grille soit résolue….
   Comme toutes les grilles ne peuvent être résolues ainsi, faire en sorte qu'on sorte de la boucle while quand plus aucun nombre n'est trouvé .
   On pourra faire des tests avec le jeu suivant :

```
1
2
3
4  Place(0,1,6)
```

```
Place(0,3,7)
Place(0,4,2)
Place(0,5,5)
Place(0,7,9)
Place(0,8,8)
Place(1,1,7)
Place(1,3,6)
Place(1,4,4)
Place(1,7,2)
Place(2,2,9)
Place(2,4,8)
Place(3,0,9)
Place(3,4,7)
Place(3,7,6)
Place(4,0,6)
Place(4,1,1)
Place(4,7,3)
Place(4,8,7)
Place(5,1,4)
Place(5,4,3)
Place(5,8,9)
Place(6,4,5)
Place(6,6,4)
Place(7,1,5)
Place(7,4,1)
Place(7,5,2)
Place(7,7,7)
Place(8,0,7)
Place(8,1,3)
Place(8,3,4)
Place(8,4,6)
Place(8,5,8)
Place(8,7,1)

```

9. Utiliser une fenêtre graphique "Turtle" pour dessiner le sudoku, pour entrer les nombres déjà connus en cliquant sur la case correspondante, puis pour afficher la solution.

10. Prolongements possibles :

a. Améliorer le programme précédent en détectant par exemple les couples de cases d'une même ligne ne présentant que les mêmes deux nombres possibles (ces nombres étant alors à retirer des nombres possibles pour les autres cases de cette ligne). De même pour les colonnes et les blocs.

b. Utiliser les fonctions créées pour concevoir un générateur de sudoku. On pourra choisir une case au hasard, puis y mettre un nombre au hasard en utilisant la fonction Place.
On continue de la même manière à placer des nombres, en choisissant une case parmi les restantes, puis un nombre au hasard parmi les nombres possibles pour cette case.
Le sudoku est prêt quand votre programme parvient à le résoudre....
On peut ensuite chercher à ajouter ou enlever des nombres pour augmenter ou diminuer la difficulté.
Un bon indice de difficulté est vraisemblablement le nombre de boucles nécessaires au programme pour résoudre le sudoku.

13) Compléments sur les listes

Retour au plan de la page

Création d'une liste

- On peut créer une liste avec la fonction range : "liste = list(range(10))" avec la fonction list devant qui transforme le type "range" en un type "list".
  Les autres manières d'utiliser "range" sont également utilisable dans ce cadre.
- On peut aussi créer une liste de 50 zéros, par exemple, en écrivant "l = [0]*50"

- On peut aussi créer une liste de 50 zéros, par exemple, en écrivant "l = [0]*50".
- On peut créer des listes grâce à des compréhensions de listes :
  l = [i**2 for i in range(10)] donne une liste des carrés des entiers de 0 à 9.
  ou encore l = [int(c) for c in "0157841036901" if c != "0"] !!
- On peut aussi créer une liste des caractères d'une chaîne : l =list("informatique")

Plus de détails sur les compréhensions de listes :
Les compréhensions de listes

Méthodes de liste

Les listes possèdent de nombreuses méthodes qui remplacent certains des programmes que vous avez réalisés.

- "l.sort()" trie la liste appelée l
- "l.reverse()" inverse l
- "l.index(12)" retourne l'index de la première occurrence de 12
- "l.count(3)" retourne le nombre de 3 dans l
- "l.remove(3)" enlève la première occurence de 3 dans la liste l.
- "l.insert(2,'c')" insère 'c' à la place d'index 2 dans la liste l.
- On a aussi les fonctions min(l) et max(l).

Attention : de même que pour append, il ne faut pas écrire "l=l.sort()" car la méthode "append" ainsi que "sort", "remove" et "reverse", ne retournent rien ! Si vous écrivez cela, l sera devenu "rien" c'est à dire "none" dans python, l ne sera même plus du type "list" mais du type "none". Il faut écrire simplement "l.sort()".

Exercice 13.1 :
Ecrire un programme qui enlève tous les zéros d'une liste entrée par l'utilisateur (voir 9.3)
Attention, l.remove(0) n'enlève que la première occurrence de 0 !

Copie de liste

Il faut comprendre que quand on crée une liste l dans Python, le programme met en mémoire l'adresse de la liste, dans une case appelée l (cela s'appelle un pointeur).
Si vous écrivez "lbis = l", l'ordinateur recopie dans la case appelée lbis l'adresse qui était déjà dans la case l, mais vous n'avez toujours qu'une seule liste.
Si vous modifiez l[0], lbis[0] sera modifié de la même manière !
Pour copier une liste l dans la liste lbis il faut écrire "lbis=list(l)"
Attention, ceci est inefficace pour des listes de listes. Il faut alors utiliser lbis=deepcopy(l) (il faut avoir importé cette fonction depuis le module copy)

Exercice 13.2 : dérivée d'un polynôme
1) Ecrire un programme où l'utilisateur entre successivement les coefficients d'un polynôme, en commençant par la constante, puis le coefficient de x, etc...Le programme stocke ces données dans une liste. Par exemple pour $4x^2 + 5x + 2$, il stockera l = [2,5,4].
2) Ecrire une fonction qui calcule la dérivée du polynôme entré par l'utilisateur : la fonction retournera la liste des coefficients de cette dérivée.
3) Ecrire une fonction qui prend en paramètre une liste de coefficients par exemple [7,0,-8,4] et qui réalise un affichage du type :
"(4 x^3) + (- 8x^2) + (0x) + (7) "
On pourra même améliorer cet affichage pour gérer les x^1, les x^0 , les coefficients nuls et les nombres négatifs
4) Utiliser cette fonction pour afficher le polynôme de départ, puis sa dérivée.

Compléments sur les listes : Les listes sur Openclassrooms

14) Compléments sur les chaînes de caractères

Retour au plan de la page

Longueur, index, comptage et concaténation

Ces opérations s'effectuent comme pour les listes.

- len(ma_chaine) en donne la longueur,
- ma_chaine.index('j') donne l'index du premier 'j' (ici 3)
- ma_chaine.count('o') commpte le nombre de 'o'
- ma_chaine + " les gens" donne "bonjour les gens" (c'est la concaténation).

Attention : on ne peut concaténer que des chaînes avec des chaînes !

Remarque : on peut créer une chaîne vide (chaine="").
Rappel : eval("6*3") retourne 18. Evalue la valeur numérique de la chaîne.

Méthodes particulières aux chaînes

Ces méthodes sont très nombreuses. Attention, contrairement aux méthodes de listes, ces méthodes n'affectent pas la chaîne à laquelle elles sont appliquées.
En voici quelques unes.

- ma_chaine.lower() met tout en minuscule et ma_chaine.upper() tout en majuscule
  Exemple :

```
1
2 ma_chaine="Quitter"
3 chainebis=ma_chaine.lower()
4 print(ma_chaine,chainebis)
5
```

  Ceci donnera : Quitter quitter (ma_chaine n'a donc pas été modifiée).
- ma_chaine.strip() enlève les espaces au début et à la fin (on a aussi rstrip et lstrip pour les enlever à droite ou à gauche)
- ma_chaine.find("th") retourne la première position de la chaîne "th"
- ma_chaine.split(',') retourne une liste constituée des sous chaînes de ma_chaine qui sont entre deux ','.
- ma_chaine.replace(c1,c2) remplace le caractère c1 par c2.

Remarque : pour obtenir de l'aide vous pouvez taper dans la console python : help("str") par exemple, et vous trouverez toutes les méthodes sur les chaînes (en anglais et avec une syntaxe difficile à saisir, mais c'est parfois utile).
Taper 'q' pour sortir de cette fenêtre.

La selection

On peut extraire une partie d'une chaîne (par exmple chaine="bonjour") grâce aux instructions suivantes :

- chaine[i:j] donne la chaîne extraite depuis juste avant le caractère i à juste avant le caractère j
  chaine[1:3] donne "on" (le caractère 1 est pris, mais pas le caractère 3)
- chaine[:3] donne la chaine extraite à partir du début et jusqu'à juste avant le caractère 3 ("bon")
- chaine[3:] donne la chaîne extraite à partir du caractère 3 (inclus) : ici cela donne "jour".
- chaine[::-1] donne la chaine écrite dans l'ordre inverse

Comparaison

On peut comparer deux chaînes : python utilise pour cela l'encodage des caractères (nous utiliserons Utf-8). Il est donc important de n'utiliser que, soit des minuscules, soit des majuscules et de ne pas mettre de caractères accentués ou spéciaux !
on aura "canard"<"cane"par exemple.
Comme pour les listes, on peut utiliser les fonction min(chaine) et max(chaine).

Exercice 14.1 :
1) Ecrire une fonction qui "nettoie" un mot, c'est à dire, qu'elle retire tous les accents, cédilles, tirets et elle met le mot en minuscules.
2) Ecrire un programme qui classe par ordre alphabétique trois mots entrés par l'utilisateur.

Formatage

Cette méthode peut sembler bizarre au début, mais se révèle indispensable assez rapidement.
On peut vouloir inclure des variables (qui peuvent être des chaînes ou des nombres etc...) dans une chaîne préformatée.
Par exemple on peut vouloir imprimer "mon nom est .....et mon prénom est ....." avec tous les noms et prénoms d'une liste.
On va écrire chaine="mon nom est {} et j'ai {} ans".format(nom,age).
Si nom="Toto" et age =6, cette chaîne devient "mon nom est Toto et j'ai 6 ans".
Si on change le nom et l'âge, la chaîne change. Voici un exemple :

```
1
2 l=[["Toto",10],["Titi",2],["Tutu",6]]
```

```
l=[["Toto",10],["Titi",2],["Tutu",6]]
#c'est une liste de listes...
for liste in l:
    #au premier passage liste=["Toto",10]
    chaine="je m'appelle {} et j'ai {} ans".format(liste[0],liste[1])
    #au premier passage liste[0] vaut "Toto"
    print(chaine)

```

Le résultat est :
je m'appelle Toto et j'ai 10 ans
je m'appelle Titi et j'ai 2 ans
je m'appelle Tutu et j'ai 6 ans

Des compléments : les chaînes

> Exercice 14.2 : questions à la chaîne
> On dispose d'une liste de prénoms, par exemple l = ["Louise","Bob", "Clara", "Louis", "Lili"].
> Ecrire un programme qui demande successivement l'âge de Louise, de Bob...etc en utilisant une chaîne formatée dans un input et, bien sûr, une boucle.
> Le programme doit continuer à fonctionner si on change de liste l.

Codage des caractères

Nous travaillerons sur les différents codages des caractères (ASCII, Utf-8 etc...), mais pour l'instant il peut être utile de savoir obtenir le code d'un caractère quelconque :
ord('a') donne 97 qui est son numéro dans tous les codages ....ord('A') donne 65, ord('é')=233 etc..
A l'inverse, chr(97) donne 'a' etc...

> Exercice 14.3 : Codage ROT13
> 1) Ecrire une fonction qui prend pour paramètre une minuscule sans accent et qui retourne la lettre située 13 places plus loin dans l'alphabet, en revenant évidemment à a après le z (on fait une rotation de 13 places, d'où le nom).
> 2) Ecrire un programme qui code un mot, après l'avoir "nettoyé".
> 3) Comment fait-on pour décoder un message ainsi codé ?
> Exercice supplémentaire : coder des phrases.

Compléments sur les chaînes :
SWI : les données alphanumériques
SWI : les chaines
Les chaînes sur Openclassrooms

15) Importer des modules

Retour au plan de la page

Nous avons déjà rencontré à plusieurs reprise la commande "import".
On peut l'utiliser de plusieurs manières :

- import math
  Ceci importe tout ce qui est dans le module math, par exemple la fonction sqrt() (qui prend la racine carrée).
  Pour utiliser cette fonction il faut taper : "math.sqrt(251)" avec le préfixe math.

- from math import*
  Importe de même tout ce qui est dans le module math.
  Pour utilser la fonction sqrt on tape seulement : "sqrt(251)".
  Cela a l'air beaucoup plus agréable, mais pour des programmes compliqués cela peut causer des problèmes de noms...
- from math import sqrt
  N'importe que la fonction sqrt, qui s'utilise alors comme avec import*

Principaux modules utilisés :

- le module math (sqrt, tangente, sinus, cosinus etc..)
  Taper help("math") dans la console python pour voir toutes les fonctions.

- le module random (pour tout ce qui est choix au hasard)
- le module time : voir OCR: le module time
- le module turtle
- le module sys : voir OCR: programmation système
- le module os : voir Modules internes de pYthon

Et bien plus encore : Index des modules Python

16) Lecture et écriture dans un fichier

Retour au plan de la page

Dans de nombreux programmes on aimerait garder les résultats obtenus ou utiliser des informations déjà existantes.
On va donc apprendre à lire et à écrire dans des fichiers. Imaginons que vous ayez créé un fichier appelé "fich.txt" (avec le bloc note par exemple).
Pour lire ce fichier, il faut d'abord l'ouvrir. Voici la syntaxe :
fichier=open("fich.txt","r") toujours accompagné après utilisation de : fichier.close()
Trois commentaires :

- l'adresse du fichier est relative ou absolue. Si votre fichier est dans le même dossier que votre programme, le chemin donné ici est suffisant. Sinon, mettez le chemin absolu, par exemple : "C:/toto/isn/fich.txt".
- le "r" signifie "read" : votre fichier n'est ouvert qu'à la lecture.
- on peut remplacer le "r" par "rb" qui lit le fichier en mode binaire, c'est à dire octet par octet.
- Il peut y avoir toutes sortes de problèmes avec l'ouverture d'un fichier. Il est donc recommandé d'utiliser plutôt l'instruction : with open("fich.txt,'r') as fichier: et d'écrire les instructions dans le bloc qui suit. Cette instruction gère les erreurs possibles et ferme proprement fich dans tous les cas (ne nécessite donc pas l'instruction "close").

On peut ensuite lire le fichier de différentes manières :

- chaine=fichier.read() : lit le fichier en intégralité et renvoie une chaîne de caractères.
- chaine=fichier.read(10) : lit dix octets à partir de la position courante (un caractère simple est codé sur un octet). Si on répète l'opération, les 10 octets suivants seront lus. Retourne une chaîne vide quand la fin du fichier est atteinte.
- chaine=fichier.readline() : lit une seule ligne à partir de la position courante y compris le '/n' qui est le caractère de fin de ligne
  Retourne une chaîne vide quand la fin du fichier est atteinte
  Attention '/n' est généralement codé sur deux octets (un retour chariot + un passage à la ligne)
- liste=fichier.readlines() : retourne une liste de toutes les lignes du fichier.

Les deux méthodes suivantes sont utiles :

- fichier.tell() : donne le nombre d'octets parcourus depuis le début du fichier
- fichier.seek(nombre,pt_depart) : amène la position courante à nombre (positif ou négatif) d'octets du point de départ.
  Ce point de départ est 0 si c'est le début du fichier, 1 si c'est la position courante et 2 si c'est la fin du fichier).
  Pour un fichier texte, on ne peut appliquer seek qu'à partir du départ, ou fichier.seek(0,2), qui place la position courante à la fin du fichier

Pour écrire dans un fichier, il faut également l'ouvrir : il faut toutefois savoir si on veut écraser le fichier précédent ("w" comme "write") ou ajouter à la fin du fichier ("a" comme "append").
Dans les deux cas, si le fichier n'existe pas il sera créé.

Ensuite il suffit d'appliquer la méthode suivante :
fichier=open("fich.txt","w")
fichier.write("mon texte ici")
Pensez à la méthode 'format' des chaînes de caractères pour écrire des chaînes contenant des parties variables.
Exemple:
Enregistrez votre programme avant de le faire fonctionner : de cette manière le fichier habit.txt se créera dans le même dossier.

```
1
2 with open("habit.txt","w") as fich:
3     l=["Strasbourg","Colmar","Mulhouse"]
```

```
    n=[272955,68848,112260]
    for i in range(3):
        fich.write("la ville de {} compte {} habitants\n".format(l[i],n[i]))
        #revoir la méthode format


with open("habit.txt","r") as fich:
    print(fich.read())
    print("nombre d'octets du fichier ",fich.tell())
    fich.seek(0) #indispensable pour remettre la position courante au début
    for i in range(3):
        print(fich.readline())
    fich.seek(0)
    print(fich.readlines())
```

Allez voir à quoi ressemble ce fichier dans le dossier courant, puis notez bien les trois manières de lire dans ce fichier.

Il existe d'autres manières d'enregistrer des données dans un fichier :

- sous forme binaire (avec 'rb' , 'wb'...)
- en utilisant le module pickle (de l'anglais "conserver") qui enregistre les données en même temps que leur type : ceci est utile pour enregistrer des données plus complexes que du texte !
  voir : SWI : module pickle

Exercice 16.1 : fichier de meilleurs scores
Ouvrir le bloc-notes et dans un fichier nommé "score.txt" écrire :

Ali Marie Marc
1000 900 600

Ecrire un programme qui lit ce fichier
Ecrire un programme qui récupère dans une liste les trois noms, et dans une autre les trois scores (transformés en float)
Ecrire un programme qui simule le score d'un nouveau joueur, dont l'utilisateur entrera le nom et le score.
Modifier les programmes précédents pour que le score et le nom du nouveau joueur se placent dans les listes (si le score le justifie)
Ecrire un programme qui modifie alors le fichier "score.txt" avec les nouveaux meilleurs scores
Ecrire un programme qui écrit dans un fichier "scorebis.txt" des phrases du type :
"le score n°1 est de 1000 pour Ali"

Exercice 16.2 : fichier des mots du français
On recherchera sur internet un fichier de tous les mots du français

- Mettre les trois premiers mots du fichier des mots du français dans une liste
- Modifier le programme précédent pour enlever les sauts de lignes
- Ecrire un programme qui cherche le premier mot commençant par g
- Modifier le programme précédent pour déterminer à quelle ligne il se trouve et combien d'octets compte le fichier avant ce mot
- Ecrire un programme qui détermine combien de mots compte le fichier, combien de mots avec un tiret, et quel est le mot le plus long.
- Ecrire un programme qui lit le dernier mot du fichier.

17) Utilisation de pygame

Retour au plan de la page

Comme souvent, Openclassroom a mis en ligne un tutoriel très bien fait pour commencer à utiliser Pygame:

Pygame sur OCR

Voici un lien vers un autre tutoriel pour des informations complémentaires:

Wiki sur Pygame

Il est également bon de connaître la page de référence de la documentation officielle de Pygame.

Site de pygame

Ce cours vous présente quelques commandes de Pygame, permettant tout de même de réaliser de nombreux programmes.
Voici les principaux points abordés :

a. Création d'une fenêtre
b. Gestion des images
c. Gestion des évènements
d. Déplacer les objets
e. Insérer des textes
f. Insérer des formes
g. Gestion du temps
h. Créer des animations

Images à télécharger pour les exemples :
perso1.png
course0.png
course1.png
course2.png
course3.png

La page au format PDF

Remarques préliminaires :

- Pour interagir avec pygame on utilise des évènements qui peuvent être un clic de la souris, un appui sur une touche etc….Ces évènements sont stockées dans une file qui s'appelle pygame.event. Les événements sont traités dans leur ordre chronologique. On peut connaître le type de l'évènement grâce à la commande event.type. Avec des instructions conditionnelles, on peut alors gérer les actions correspondant aux différents évènements.
- Les positions d'un objet sont toujours calculées à partir du coin en haut à gauche du contenant (c'est-à-dire de l'écran pour la fenêtre, de la fenêtre pour les objets qui sont tracés dessus…etc…) et on donne toujours d'abord l'abscisse (de 0 à gauche à la largeur de la fenêtre à droite) puis l'ordonnée (de 0 en haut jusqu'à la hauteur de la fenêtre en bas : l'axe des ordonnées va donc vers le bas..).

a) Création d'une fenêtre :

Début du tutoriel pygame

Voici le schéma d'un programme créant une fenêtre Pygame :

```
1
2  import pygame  #importation de pygame
3  import os
4  import traceback  #module pour récupérer des infos sur les erreurs
5  from pygame.locals import * #on importe les constantes de pygame
6  pygame.init()  #on lance pygame
7  #toujours encadrer vos programmes pygame par try et finally ce qui permet de
8  # fermer correctement la fenêtre pygame en cas d'erreur
9  try:
10     #pour positionner la fenêtre sur l'écran à la position (400,600).
11     os.environ['SDL_VIDEO_WINDOW_POS']="400,600"
12     #création d'une fenêtre
13     fenetre=pygame.display.set_mode((640,480))#fenêtre de taille 640*480
14     continuer=1
15  #boucle perpétuelle qui permet de garder la fenêtre ouverte jusqu'à ce qu'o
16  # décide de la fermer
17     while continuer:
18         for event in pygame.event.get():
19             #pygame prend le premier évènement de la file
```

```
            if event.type==QUIT:
                #l'évènement QUIT correspond au clic sur la croix
                continuer=0  #permet de quitter la boucle
except:
   #ce bloc permet de récupérer des infos en cas d'erreur
   traceback.print_exc()
finally:
    pygame.quit()
    exit()
```

Commandes additionnelles :

- IMPORTANT : après toute modification, il faut actualiser l'affichage de la fenêtre avec la commande : pygame.display.flip()
- remplir la fenêtre (qui a pour nom "fenetre" dans notre programme) d'une certaine couleur
  fenetre.fill((R,G,B)) : R est la quantité de rouge entre 0 et 255, G (ou V) pour le vert et B pour le bleu.
  Attention il y a une parenthèse pour "fill" et une parenthèse pour la couleur (R,G,B)
  Pour tester les couleurs vous pouvez aussi utiliser le petit logiciel qui s'appelle la boîte à couleur.
- mettre un titre à la fenêtre :
  pygame.display.set_caption("Mon_titre")

Exercice 17.1 : Créer une fenêtre de taille 300*200 coloriée en bleu

b) Gestion des images.

Début du tutoriel pygame

Pour afficher une image :

- Il faut charger l'image (après avoir lancé pygame) :
  image1=pygame.image.load("mon_image.jpg").convert()
  Entre les guillemets on met l'adresse de l'image par rapport au dossier où se trouve le programme (voir les adresses dans la page sur le HTML sur le site).
  On peut utiliser toutes sortes de formats d'images (jpg, png …)
  "convert" sert à convertir l'image dans un format dont l'affichage est plus rapide.
- Il faut "coller", on dit "blitter" l'image sur l'écran :
  fenetre.blit(image1,(x,y))
  x,y sont les coordonnées du coin en haut à gauche de l'image
- Il faut rafraîchir l'affichage :
  pygame.display.flip()

Commandes complémentaires :
On peut (entre autres) :

- rendre le fond de l'image transparent de deux manières :
  si c'est une image au format png dont le fond est transparent, remplacer convert par convert_alpha
  si c'est une image autre dont le fond a une couleur (R,G,B), on utilise la commande : image1.set_colorkey((R,G,B))
  Remarque : dans la boîte à couleur on peut "récupérer" les valeurs RGB d'une couleur d'une image en utilisant la petite "pipette"
- redimensionner une image :
  image1=pygame.transform.scale(image1,(30,30)) pour avoir une image 30*30
  On peut aussi redimensionner l'image à la bonne taille avant de l'utiliser !

Exercice 17.2 :
1) Dessiner sur un papier la fenêtre, son repère, et l'image (représentée par un rectangle)
2) Afficher l'image « perso1.png » avec un fond transparent dans la fenêtre créée précédemment.
L'image devra être de taille 20*20 et être dans le coin en bas à droite.

c) Gestion des évènements .

Début du tutoriel pygame

Exemple :
On reprend le programme du début et on insère après le bloc "if event.type==QUIT", d'autres blocs, par exemple :

```
```

```
if event.type==KEYDOWN:   #appui sur une touche
 if event.key==K_UP:   #la touche est la flèche vers le haut
  déplacer le personnage  d'une case vers le haut…(voir plus loin)
 if event.key==K_DOWN:
  déplacer le personage d'une case vers le bas
```

Les différents types d'évènements :

- event.type==QUIT (c'est l'une des "constantes importées de pygame .locals)
  c'est le fait d'appuyer sur la croix : vous pouvez décider d'associer ceci à la fermeture de la fenêtre (c'est ce qui est fait en général) ou autre chose !
- event.type==KEYDOWN
  correspond à l'enfoncement d'une touche (il y a aussi KEYUP)
  On peut alors récupérer la touche appuyée avec la commande :
  event.key qui peut prendre différentes valeurs.
  Pour avoir une liste complète:

  glossaire des keycodes

  Attention, pygame voit le clavier comme un clavier QWERTY....
  K_q pour le q du clavier Qwerty c'est à dire le a du clavier AZERTY(et pareil pour le reste de l'alphabet). On peut remplacer "if event.key==K_q" par "if event.dict['unicode'] =="a" et on récupère alors les caractères correspondant au clavier. Ceci peut aussi servir pour récupérer des données entrées par l'utilisateur.
  K_F1 pour la touche F1
  K_SPACE pour la touche espace
  K_RETURN pour la touche ENTER
  K_ESCAPE pour la touche Echap
  K_UP pour la flèche vers le haut (DOWN, LEFT, RIGHT)
  K_KP0 pour le 0 du pavé numérique

  Si on maintient la touche enfoncée on peut répéter l'évènement avec la commande :
  pygame.key.set_repeat(400,30) 400 est le délai (en millisecondes) avant que la répétition ne commence et à partir de là, un nouvel évènement est généré toutes les 30 ms. Si votre touche provoque un déplacement et que vous appuyez 550ms sur la touche, le déplacement va se faire une première fois au début, il y aura 400ms d'attente , puis il y aura un déplacement à 400ms, à 430 ms, à 460 ms….Le délai est là pour qu'il n'y ait pas de répétitions indésirables.
  On peut récupérer les caractères tapés (par exemple si le joueur tape son nom et qu'on veut le récupérer) , après un if event.type==KEYDOWN:
  carac=event.dict['unicode'] si on a tapé la touche K_a alors carac='a'
- event.type==MOUSEBUTTONDOWN
  quand on clique avec la souris . Pour récupérer les coordonnées x et y de la position de la souris on écrit (x,y)=event.pos
- event.type==MOUSEMOTION
  quand on déplace la souris, par exemple si on veut déplacer un personnage avec la souris (on récupère la position comme ci-dessus). Attention ce type d'évènements peut rapidement remplir la file des évènements….

Pour attendre les évènements :

- pygame.event.wait() :
  attend le prochain évènement et le programme se met en pause en attendant
- pygame.event.poll():
  fait la même chose, mais le programme continue à tourner.

Exercice 17.3 :
Créer une fenêtre de taille 600*400 et écrire un programme qui affiche « perso1.png » à l'endroit où on a cliqué avec la souris. Comment faire pour qu'un seul personnage soit affiché à chaque fois ?

d) Déplacer des objets.

Début du tutoriel pygame

Il est recommandé d'utiliser des formes rectangles (appelées "rect") qui contiennent les objets, ce qui permet de les déplacer facilement et de mieux gérer l'affichage, les effets de bord et les collisions.

- pour récupérer le rectangle d'un objet appelé objet :
  objet_rect=objet.get_rect()
  objet_rect vaut alors (x,y,largeur,hauteur) où (x,y) est la position de l'objet

objet_rect vaut alors (x,y,largeur,hauteur) où (x,y) est la position de l'objet
et largeur et hauteur sont les dimensions du rectangle qui entoure l'image.

- pour créer un rectangle :
  mon_rect=pygame.Rect(x,y,largeur, hauteur) où x,y correspond à la position du rectangle
- pour déplacer un rect (et donc l'objet qui est dedans) :
  mon_rect.move(x,y) (x est le déplacement horizontal, y vertical)
  Attention, il faut blitter l'objet et remettre le "fond" à l'ancienne place de l'objet.
  fenetre.blit(objet,mon_rect) et pour le fond cela dépend des situations (voir l'exemple qui suit) et ne pas oublier de rafraîchir la fenêtre fenetre.display.flip()

Voici les attributs d'un rect appelé r :

- r.left (ou right, bottom, top)
- r.center
- r.centerx (ou y)
- r.size
- r.width (ou height)
- r.topleft (ou right)
- r.midtop (ou midbottom ou midright ou midleft)

Il est facile de gérer les collisions avec des rectangles. Si on a deux rectangles r1 et r2, pour savoir s'ils ont une intersection , on peut utiliser la commande : r1.colliderect(r2) qui retourne un booléen (True ou False).

Exercice 17.4 :
Afficher « perso1.png » , récupérer son « rect » et l'afficher dans la console .
Recommencer, mais choisir vous-même le « rect » pour que le personnage soit au centre de l'écran.

Exemple de déplacement d'un personnage:

```
import pygame
from pygame.locals import *
import traceback
pygame.init()
try:
    fenetre=pygame.display.set_mode((640,480))
    fenetre.fill((255,255,255)) #on remplit la fenêtre de blanc
    image1=pygame.image.load("perso1.png").convert_alpha()
    #convertit une image au fond transparent, en un format adapté à pygame
    image1_rect=image1.get_rect() #on crée un rectangle entourant l'image
    fenetre.blit(image1,image1_rect) #on blitte l'image dans ce rectangle
    pygame.key.set_repeat(400,30)  #on active la répétition des touches
    pygame.display.flip()
    continuer=1
    while continuer:
        for event in pygame.event.get():
 #attention, toujours prévoir un moyen de sortir de la boucle
            if event.type==QUIT:
                continuer=0
            elif event.type==KEYDOWN:
                if event.key==K_LEFT:
                    image1_rect=image1_rect.move(-5,0)
  #on déplace le rectangle de l'image de 5 pixel vers la gauche
                    if image1_rect.left< 0:
  #si le bord gauche de l'image sort du cadre,
        #on remet l'image à droite
                        image1_rect.left=610
                if event.key==K_RIGHT:
                    image1_rect=image1_rect.move(5,0)
                    if image1_rect.right> 640:
                        image1_rect.right=30
                #si on ne re-remplit pas le fond,
                #on verra l'objet aux deux positions
                fenetre.fill((255,255,255))
                fenetre.blit(image1,image1_rect)
```

```
                fenetre.blit(image1,image1_rect)
                pygame.display.flip()
except :
    traceback.print_exc()
finally:
    pygame.quit()
    exit()
```

Exercice 17.5 :
Compléter le programme ci-dessus en ajoutant des déplacements vers le haut et vers le bas.
Ajouter des instructions de telle sorte que le personnage soit remis au centre quand on appuie sur « Espace ».

e) Insérer des textes

Début du tutoriel pygame

Voici comment afficher une chaîne de caratères (str).

- chaine="ma chaine sur une seule ligne" (penser à la méthode format pour afficher une chaîne contenant des données variables, par exemple le nom du joueur ou le score)
- font=pygame.font.SysFont("broadway",24,bold=False,italic=False) pour choisir la police broadway , en taille 24, pas de gras, pas d'italique.
- text=font.render(chaine,1,(R,G,B)) pour créer l'objet texte
- fenetre.blit(text,(30,30)) où (30,30) correspond à la position du texte
- fenetre.display.flip() pour rafraîchir l'affichage

Exercice 17.6 :
Compléter le programme précédent pour que l'appui sur la touche « Echap » provoque l'affichage du texte suivant : « Je me suis déplacé n fois » si le personnage a effectué n mouvements .

f) Insérer des formes

Début du tutoriel pygame

On peut dessiner des formes dans pygame :

- un rectangle : mon_rectangle=pygame.draw.rect(surface,color,rect,épaisseur) avec surface est la surface sur laquelle on veut dessiner le rectangle (la fenêtre ou autre chose), color c'est un triplet (R,G,B), rect correspond à la position : ce peut être un pygame.Rect ou (positionx, positiony, largeur, hauteur) ,épaisseur est l'épaisseur du trait (0 donne un rectangle plein)
- un cercle : mon_cercle=pygame.draw.circle(surface,color,pos_centre,rayon,épaisseur) pos_centre est du type (x,y)
- une ligne : ma_ligne=pygame.draw(surface,color,position_départ,position_arrivée,épaisseur)

Inutile de blitter, mais il faut rafraîchir l'écran.
On peut créer des surfaces (par exemple pour avoir deux parties dans une fenêtre) :

- ma_surface=pygame.Surface((34,34)) crée un rectangle noir de taille 34*34
- On peut le remplir entièrement ou partiellement :
  ma_surface.fill((R,G,B) le remplira entièrement
  ma_surface.fill((R,G,B), rect) remplira le rect seulement où rect est un pygame.Rect ou (x,y,largeur,hauteur)
  Ceci s'applique aussi au remplissage de la fenêtre.
- Les différents objets peuvent alors être blittés sur cette surface : la position s'entend alors par rapport au coin supérieur gauche de ma_surface.
- Il faut évidemment blitter ma_surface et rafraîchir l'écran.

Exercice 17.7 :
Ecrire un programme qui trace les segments entre les points où on aura cliqué successivement avec la souris.

g) Gestion du temps

Début du tutoriel pygame

- pygame.time.delay(100) crée une attente de 100ms
- pygame.time.get_ticks() compte le temps en ms depuis pygame.init()
- limiter le nombre d'images à 60 images par seconde :
  clock=pygame.time.Clock
  clock.tick(60)
- Si on veut répéter une action à intervalles de temps réguliers on peut suivre un schéma analogue au suivant en utilisant un timer:

utilisant un timer:

```
import pygame
import os
import traceback
from pygame.locals import *
pygame.init()
try:
    fenetre=pygame.display.set_mode((640,480))
    fenetre.fill((255,255,255))
    image1=pygame.image.load("perso1.png").convert_alpha()
    image1_rect=image1.get_rect()
    fenetre.blit(image1,image1_rect)
    pygame.display.flip()
    #on crée un évènement qui n'est pas un évènement prédéfini par le
    #il faut lui donner un numéro pour que pygame le gère
    depla=USEREVENT+1 #on donne un numéro à l'évènement entre USEREVEN
    #et NUMEVENTS (qui sont des constantes de Pygame :sur mon ordinate
    #cela se situe entre 25 et 31. On peut les faire afficher dans la
    #on peut aussi utiliser directement un numéro à la place de "depla
    pygame.time.set_timer(depla,150)
    #l'évènement va se mettre dans la file des évènements toutes les 1
    continuer=1
    while continuer:
        for event in pygame.event.get():
            if event.type==QUIT:
                continuer=0
            elif event.type==depla:  #gestion de l'évènement répétitif
 #on fait ce que l'on veut, ici on déplace le personnage en diagonale
                image1_rect=image1_rect.move(3,3)
                fenetre.fill((255,255,255))
                fenetre.blit(image1,image1_rect)
                pygame.display.flip()
except :
    traceback.print_exc()
finally:
    pygame.quit()
    exit()

```

h) Céer des animation

Début du tutoriel pygame

Pour animer un objet le principe est simple et est le même que dans les dessins animés : on affiche successivement des images de l'objet dans différentes positions après avoir « effacé » le précédent.
Voici un exemple :

```
import sys,pygame
from pygame.locals import *
import traceback
blue=145,197,235
clock=pygame.time.Clock()
 #voir à la fin, on veut choisir le nombre d'images par secondes
def image(chaine):
    """fonctions qui charge les sprites et rend le fond transparent etc."""
```

```
    im=pygame.image.load(chaine)
    im=im.convert_alpha(im)
    return im
try:
    screen=pygame.display.set_mode((400,151))

    b=[0,0,0,0]
    for i in range (4):
        b[i]=image("course{}".format(i)) #voir la méthode "format" sur les c
    #on fait une liste des images du personnage dans différentes positions
    continuer=1
    i=0   #index de l'image qu'on va afficher
    while continuer:
        for event in pygame.event.get():
            if event.type in (QUIT, KEYDOWN):#pour quitter
                continuer=0
        screen.fill(blue)
        #on va afficher successivement les images de la liste en revenant au
        #quand on est à la fin (avec i%4)
        screen.blit(b[i%4],(180,54))   #(180,54) est la position où on blitt
        i=i+1   #pour afficher ensuite l'image suivante
        clock.tick(10) #on limite le nombre d'images à  10 images par second
#sinon on ne voit rien, c'est trop rapide
        pygame.display.flip()
except :
    traceback.print_exc()
finally:
    pygame.quit()
```

Beaucoup d'autres instructions existent et sont décrites dans la documentation de pygame (en anglais) :

site de pygame

Complément : transparence dans GIMP :

Il n'est pas du tout certain que la méthode décrite pour rendre un fond transparent soit optimale...

- avec les sprites :
- sélectionner l'un des sprite
- sélectionner rect
- édition copier
- édition coller comme……image
- Dans la nouvelle image :
- sélection par couleur
- cliquer sur le fond
- puis
- sélection inverser
- puis et copier
- coller comme…..image
- exporter……..choisir png

Code academy : avec evaluation de votre code

Apprendre avec les videos de Lucas

Université de Waterloo : avec évaluation de votre code

Cours de Gérard Swinnen

France-ioi.fr