1ère partie

Initiation à la programmation

Nous avons vu, par ailleurs, qu'un ordinateur peut effectuer des opérations très variées sur des types de données très variées : des nombres, des lettres, des images, des sons, des textes, des vidéos, ... Il peut être utilisé pour retoucher une photo, la mettre sur un blog ou un site web, la conserver dans un album, ...
Un ordinateur est donc une machine complètement polyvalente : tous les automatismes peuvent être programmés sur un ordinateur. A l'inverse des machines à café ou des aspirateurs qui servent à une seule chose : faire le café, aspirer la poussière, ... Dès la sortie de l'usine, une machine à café "sait" faire le café, un aspirateur "sait" aspirer la poussière. En revanche, un ordinateur ne sait quasiment rien faire. Un ordinateur doit "être programmé" pour retoucher une photo, la mettre sur un blog ou un site web, ... C'est pour cela que les ordinateurs ont besoin de programmes.
Cette notion de programme est également ce qui distingue l'ordinateur de certaines machines comme les calculatrices, qui donnent une illusion de polyvalence dans la mesure où elles peuvent effectuer des tâches variées. Cependant, ces tâches sont fixées une fois pour toutes et on ne peut pas programmer une calculatrice pour lire une vidéo, alors qu'on peut programmer un ordinateur pour faire tout ce que fait une calculatrice.

Un programme est un texte qui décrit un algorithme que l'on souhaite faire exécuter par une machine. Ce texte est écrit dans un langage particulier, appelé *langage de programmation*. Il existe plusieurs milliers de langages de programmation, parmi lesquels Java, C, Python, Caml, Fortran, Cobol, etc. Il n'est cependant pas nécessaire d'apprendre ces langages les uns après les autres, car ils sont tous plus ou moins organisé autour des mêmes notions : déclaration, affectation, séquence, test, boucle, fonction, etc...Ici, pour apprendre, nous utiliserons notamment le langage Python.

I Arithmétique, variables, instructions

Le langage de programmation Python permet d'interagir avec la machine à l'aide d'un programme appelé *interprète Python*. On peut l'utiliser de deux façons différentes. La première méthode consiste en un dialogue avec l'interprète. C'est le *mode interactif*. La seconde consiste à écrire un programme dans un fichier source puis à la faire exécuter par l'interprète. C'est le *mode programme*.

I.1 - Mode interactif

I.1.a - Arithmétique

Exercice 1

Tester successivement les opérations suivantes et en déduire les règles de calcul utilisées par l'interprète et la signification des opérations `//` et `%`

```
>>> 2+5*(10-1/2)
>>> 2/(3-3)
>>> 1,2+3,4
>>> 7/2
>>> 7//2
>>> 7 % 2
```

I.1.b - Variables

Exercice 2

Tester successivement les opérations suivantes :

```
>>> a = 1 + 1
>>> a
>>> a * (1 + a)
>>> a = 3
>>> a * (1 + a)
>>> a = a + 1
>>> a
>>> cube = a * a * a
>>> ma_variable = 42
>>> b+1
>>> c = c + 1
```

Exercice 3

Quelle est la valeur affichée par l'interprète après la séquence d'instructions suivantes :

```
>>> a = 3
>>> a = 4
>>> a = a + 2
>>> a
```

Exercice 4

Quelle est la valeur affichée par l'interprète après la séquence d'instructions suivantes :

```
>>> a = 2
>>> b = a * a
>>> b = a * b
>>> b = b * b
>>> b
```

Exercice 5

Initialiser une variable `a` avec la valeur 2 puis répéter dix fois l'instruction `a = a * a`. Observer le résultat. Quelle puissance de 2 a-t-on ainsi calculée ?

Exercice 6

Après avoir affecté deux nombres aux variables `a` et `b` Que fait la séquence suivante :

```
>>> tmp = a
>>> a = b
>>> b = tmp
```

Exercice 7

On affecte deux entiers aux variables `a` et `b`.
On remplace le contenu de `a` par la somme de de celui de `a` et `b`. Puis on remplace le contenu de `b` par le contenu de `a` moins celui de `b`. Enfin, on remplace le contenu de `a` par son contenu moins celui de `b`. Que contiennent `a` et `b` à la fin de ces opérations.

I.2 - Mode programme

Le mode programme de Python consiste à écrire une suite d'instructions dans un fichier et à les faire exécuter par l'interprète Python. Cette suite d'instruction, s'appelle un *programme*, ou encore un *code source*. On évite ainsi de ressaisir à chaque fois les instructions dans le mode interactif.

I.2.a - Affichage

En mode programme, les résultats des expressions calculées ne sont pus affichées à l'écran. Lorsqu'on souhaite le faire, il faut utiliser une instruction spécifique d'affichage. En Python, elle s'appelle `Print`.

Exercice 8

Exécuter le programme suivant en observant les différents résultats obtenus :

```
print(3)
print(1 + 3)
print("Salut tout le monde")
print("1+3")
print(On est bien)
print("On est bien" + "a Cassin")
```

Exercice 9

Qu'affichent les instructions suivantes :

```
print("1+")
print(1+)
```

Remarque 1 (*Instruction `Print` et retour à la ligne*)

Par défaut l'instruction `Print` provoque un retour à la ligne. On peut changer ce comportement en utilisant l'option `end` de cette fonction

```
print("abc", end=" ")
print("def", end="")
print("gh")
print("ijk", end="toto")
```

I.2.b - Interagir avec l'utilisateur

Pour permettre l'interaction du programme avec l'utilisateur, par exemple la saisie de valeur d'une variable, il faut utiliser l'instruction `input`.

Exercice 10

Exécuter le programme suivant :

```
s=input()
print(s)
print(s+1)
print(type(s))
print(s+'1')
```

Les valeur récupérées par l'instruction `print` sont des *chaînes de caractères*. Si nous voulons les convertir en entier (pour les utiliser dans un calcul par exemple) nous devons utiliser l'instruction `int`.

Exercice 11

Exécuter le programme suivant :

```
s=input()
a=int(s)
print("le nombre suivant est ", a + 1)
```

Exercice 12

Que se passe-t-il quand on exécute le programme suivant ?

```
a = input("saisir un nombre :")
print("le nombre suivant est ", a+1
```

Le rectifier.

Exercice 13

Ecrire un programme demandant à l'utilisateur son année de naissance et affichant son âge.

Exercice 14

Ecrire un programme qui demande à l'utilisateur des longueurs des deux côtés d'un rectangle et affiche son aire.

Exercice 15

Ecrire un programme qui demande à l'utilisateur d'entrer une base (entre 2 et 36) et un nombre dans cette base puis qui affichera ce nombre en base 10.
La notation `int` (*chaîne, base*) permet de convertir une chaîne de caractère représentant un entier dans une base donnée en un entier Python.

Exercice 16

Ecrire un programme qui demande à l'utilisateur un nombre entier de secondes et qui l'affiche sous la forme d'heures/minutes/secondes.

Exercice 17

On souhaite écrire un programme qui demande à l'utilisateur un nombre d'oeufs et affiche le nombre de boîtes de 6 oeufs nécessaires à leur transport. On considère ce programme, qui utilise la division euclidienne.

```
n=int(input("combien d'oeufs ? "))
print(n // 6)
```

Tester ce programme sur différentes entrées.

1. Sur quelles valeurs de `n` ce programme est-il correct ?
2. Pourquoi n'est-il pas correct de remplacer `n // 6` par `n // 6 + 1` ?
3. Proposer une solution correcte.

Exercice 18

On souhaite calculer les coordonnées du point d'intersection de deux droites données sous la forme

$$y = ax + b$$
$$y = cx + d$$

On suppose ici que `a, b, c` et `d` sont des entiers. Ecrire un programme qui demande ces quatres valeurs à l'utilisateur puis affiche les coordonnées du point d'intersection. Que se passe-t-il si les droites sont parallèles ?

I.3 - Utilisation de bibliothèques : exemple de Turtle

L'une des grandes qualités de Python est qu'il est extrêmement facile de lui ajouter de nombreuses fonctionnalités par importation de divers modules.
Pour illustrer cela, et nous amuser un peu avec d'autres objets que des nombres, nous allons explorer un module Python qui permet de réaliser des « graphiques tortue », c'est-à-dire des dessins géométriques correspondant à la piste laissée derrière elle par une petite « tortue » virtuelle, dont nous contrôlons les déplacements sur l'écran de l'ordinateur à l'aide d'instructions simples. Activer cette tortue est un vrai jeu d'enfant. Plutôt que de vous donner de longues explications, nous vous invitons à essayer tout de suite :

```
from turtle import *
forward(60)
left(120)
forward(60)
right(90)
circle(60, 300)
forward(60)
goto(0,0)
```

Les principales fonctions mises à votre disposition dans le module turtle sont les suivantes :

reset() : On efface tout et on recommence

goto(x, y) : Aller à l'endroit de coordonnées x, y

forward(distance) : Avancer d'une distance donnée

backward(distance) : Reculer

left(angle) : Tourner à gauche d'un angle donné (exprimé en degrés)

right(angle) : Tourner à droite

circle(r, a) : Tracer un arc de cercle d'angle a et de rayon r.

dot(r) : tracer un point de rayon r.

up() : Relever le crayon (pour pouvoir avancer sans dessiner)

down() : Abaisser le crayon(pour recommencer à dessiner)

color(couleur) : peut être une chaîne prédéfinie ('red', 'blue', 'green', etc.)

width(épaisseur) : Choisir l'épaisseur du tracé

begin_fill() : activer le mode remplissage

end_fill() : désactiver le mode remplissage

fillcolor(c) : sélectionner la couleur c pour le remplissage.

write(texte) : ¡texte¿ doit être une chaîne de caractères délimitée avec des " ou des '

Remarque 2 (*Variante sur l'utilisation des bibliothèques*)

Une autre bibliothèque disponible se nomme `random`. Elle donne accès à différentes instructions produisant des nombres aléatoires. Voici 4 façons de déclarer l'utilisation d'un module puis d'en utiliser des fonctions :

- ```
  import random
  n = random.randint(1,6)
  print(n)
  ```

- On peut associer un nom plus court à la bibliothèque au moment de la charger avec le mot clé `as` :

  ```
  import random as r
  n = r.randint(1, 6)
  print(n)
  ```

- On peut charger explicitement certaines des instructions avec la combinaison `from / import` :

  ```
  from random import randint
  n = randint(1,6)
  print(n)
  ```

- On peut charger l'intégralité des instructions d'une bibliothèque en une seule fois avec une étoile `*` :

  ```
  from random import *
  n = randint(1,6)
  print(n)
  ```

  Cette solution peut paraître pratique mais elle possède un inconvénient : deux bibliothèques différentes peuvent contenir des fonctions avec le même nom qui entreraient alors en conflit. On évite donc autant que possible cette possibilité.

Voici un deuxième exemple de figure Turtle :

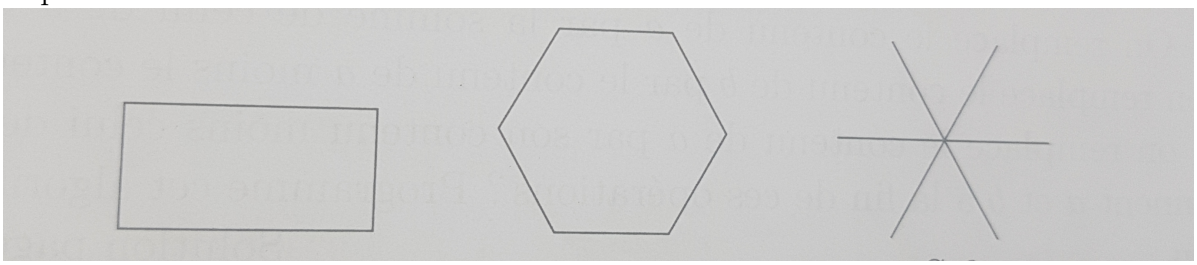
```
# Rectangle épais
width(6)
color(0.2, 0.2, 0.2)
goto(60, 0)
goto(60, 110)
goto(0, 110)
goto(0, 0)

#Déplacement
up()
goto(5,5)
down()

#sablier gris clair
width(1)
fillcolor("grey")
begin_fill()
goto(55, 5)
goto(5, 105)
goto(55, 105)
goto(5, 5)
end_fill()
```
```

Exercice 19

Reproduire avec Turtle les dessins suivants :



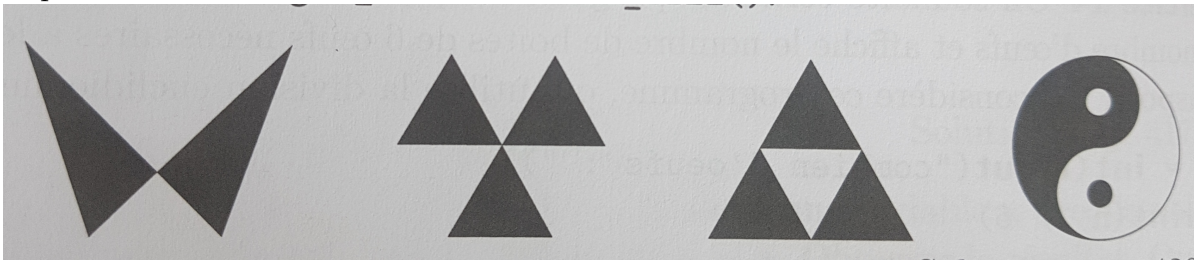
Exercice 20

Que trace le programme Turtle suivant :

```
goto(20, 0)
goto(0, 20)
goto(20, 20)
goto(0, 0)
goto(0, 20)
goto(10, 30)
goto(20, 20)
goto(20, 0)
```

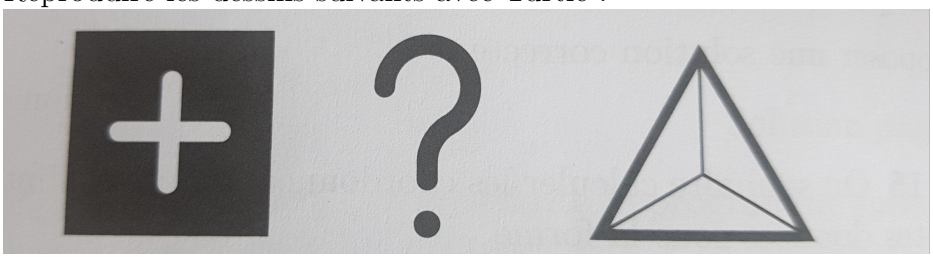
Exercice 21

Reproduire les dessins suivants avec Turtle :



Exercice 22

Reproduire les dessins suivants avec Turtle :



Exercice 23

Utiliser Turtle pour dessiner un drapeau français. Faire de même avec d'autres drapeaux.

Exercice 24

Utiliser Turtle pour dessiner un arc en ciel.

II Les boucles

II.1 - Problème : dessiner une spirale

On souhaite écrire un programme dessinant une spirale

- avec un nombre de tours définis par l'utilisateur.
- constituée de demi-cercles dont les dimensions augmentent régulièrement
- chaque demi-cercle a une épaisseur de un supérieur à celle du précédent
- son rayon est le carré de la valeur de l'épaisseur

Exercice 25

Reproduire cette spirale :

II.2 - Boucle bornée simple

Le programme suivant permet d'afficher 1000 fois la lettre A (sans retour à la ligne) :

```
for _ in range(1000):
   print('A', end='')
```

Et voici une autre utilisation d'une boucle avec cette fois le nombre de tours saisi par l'utilisateur :

```
n=int(input('combien de fois souhaitez-vous afficher le A'))
for _ in range(n):
    print('A', end='')
```

Mais la répétition n'est seulement limitée à une seule instruction. Elle peut concerner tout un bloc :

```
x = randint(-100, 100)
y = randint(-100, 100)
goto(x, y)
x = randint(-100, 100)
x = randint(-100, 100)
y= randint(-100, 100)
goto(x, y)
x = randint(-100, 100)
y= randint(-100, 100)
goto(x, y)
```

peut-être abrégé par :

```
for _ in range(n):
   x = randint(-100, 100)
   y= randint(-100, 100)
   goto(x, y)
```

EN Python, l'*indentation* du code, c'est à dire les passages à la ligne, les retraits de chaque ligne, jouent un rôle dans la signification du programme.

ATTENTION

II.3 - Utilisation de l'indice de boucle

Exercice 26

Que permettent d'afficher les boucles suivantes :

```
for i in range(10):
    print(i)
```

```
a=1
for _ in range(4):
    a = a + 2
print(a)
```

```
x=10
for i in range(10):
    print(i, " ",x)
    x=x-1
```

Exercice 27

Ecrire un programme permettant de dessiner une spirale telle que définie plus haut avec un nombre de tours défini par l'utilisateur.

Exercice 28 (*Calcul de la moyenne*)

Ecrire un programme demandant le nombre d'élèves de la classe, la note de chaque élève et qui permet d'afficher la moyenne de la classe.

Remarque 3

L'instruction `range` peut être enrichie. Ainsi on peut utiliser `for i in range( début, fin, pas)`.

- `range(n)` est donc équivalente à `range(0, n, 1)`
- `range(1, n+1, 1)` peut être utilisé pour avoir des tours de boucle dont les indices vont de 1 à n inclus.
- avec `range(0, n, 2)` les indice sont les entier pairs de 0 à n exclus.
- `range(n, 0, -1)` peut être utilisé pour avoir des indices allant de n à 1 dans l'ordre décroissant.

Exercice 29

Ecrire un programme qui demande un entier n à l'utilisateur puis calcule et affiche le résultat de $2 \times \cdots \times 2$ (où on a n occurrences de 2).

Exercice 30

Ecrire un programme qui calcule et affiche $1 \times 2 \times \cdots \times 100$.

Exercice 31

Ecrire un programme qui demande un entier n à l'utilisateur puis calcule et affiche $1 + 2 \cdots + n$, ainsi que $n(n+1)/2$. Qu'observe-t-on ?

Exercice 32

Ecrire un programme qui demande à l'utilisateur un montant initial `family m` déposé sur un livret, un taux d'intérêt `t` exprimé en pourcentage et un nombre d'années `n` puis affiche les intérêts perçus chaque année ainsi que le montant total présent sur le livret après n années.

Exercice 33

Ecrire un programme qui demande à l'utilisateur un nombre de notes `n` à prendre en compte, puis à tour de rôle chacune des ces `n` notes et son coefficient. Il doit ensuite en afficher sa moyenne pondérée.

Exercice 34

1. Ecrire un programme demandant à l'utilisateur un nombre de chiffres `n` puis `n` chiffres. Il doit ensuite calculer et afficher le nombre formé avec les `n` chiffres fournis dans l'ordre.
2. Ecrire une variante du même programme dans lequel les chiffres sont donnés dans l'ordre inverse.

Exercice 35

Ecrire un programme qui demande à l'utilisateur un nombre entier `n` et un nombre de chiffres `k` et qui affiche successivement les `k` derniers chiffres de `n`, en commençant par les unités.
Si `n` contient moins de `k` chiffres, il suffira d'afficher des zéros à la fin.
Indication : on rappelle que `n % 10` *renvoie le chiffre des unités de* `n`

Exercice 36

On considère à nouveau l'hexagone et le flocon à six branches vus dans l'exercice 19. Ecrire de nouvelles versions basées sur des boucles des programmes traçant ces figures avec Turtle.

Exercice 37

Reprendre l'exercice précédent en demandant à l'utilisateur un entier `n` définissant le nombre de côtés du polygone ou le nombre de branches de l'étoile.

Exercice 38

Ecrire un programme Turtle demandant à l'utilisateur un entier `n` et traçant un escalier à `n` marches.

Exercice 39

La suite de Fibonacci est la suite d'entiers (F_n) définie par :

- F_0=0, F_1=1
- pour tout entier n à partir de 2, $F_n = F_{n-1} + F_{n-2}$

Ecrire un programme qui demande à l'utilisateur un entier `n`, qu'on supposera supérieur ou égal à 1, et qui affiche la valeur de F_n.

III Structure conditionnelle - booléens

III.1 - Structures conditionnelles

L'instruction conditionnelle `if/elif/else` permet d'écrire un programme permettant la combinaison de plusieurs blocs de code de sorte que l'un ou l'autre soit exécuté en fonction des situations.
Voici 3 cas d'utilisation de cette structure :

- ```
  if n > 0 :
      print("le nombre ", n, " est positif")
  ```

- ```
  if n > 0 :
      print("Positif")
  else :
      print("Negatif ou nul")
  ```

- ```
  if n < 0 :
      print("Negatif")
  elif n == 0 :
      print("Nul")
  elif n<=2 :
      print("Petit")
  else :
      print("Grand")
  ```

Les symboles = et == sont différents.
Le premier est l'opérateur d'affectation d'un nouvelle valeur à une variable.
Le deuxième désigne l'égalité mathématique entre deux éléments.

Attention

Comme dans le cas de la boucle `for`, c'est l'indentation qui détermine quelles instructions font partie du bloc conditionnel et quelles instructions font partie de la suite du programme qui est toujours exécutée.
Devinez le résultat de chacun de ces programmes :

1. ```
   if n > 0:
       print("Strictement"}
   print("positif")
   else :
       print("Negatif")
   print("ou nul")
   ```

2. ```
   if n > 0:
       print("Strictement"}
       print("positif")
   else :
       print("Negatif")
   print("ou nul")
   ```

Indentation
   ```

III.2 - Booléens

Jusqu'à présent les conditions ont été exprimées en terme de comparaisons numériques entre deux expressions avec les symboles suivants :

- `>` : strictement plus grand que
- `<` : strictement plus petit que
- `==` : égal à
- `>=` : supérieur ou égal à
- `<=` : inférieur ou égal à
- `!=` : différent de

Les comparaisons et tests d'égalité sont des expressions Python ordinaires, qui comme les expressions arithmétiques produisent un résultat. Ce résultat est l'une des deux valeurs dites *booléennes* ; L'une représente le *vrai*, noté `True` et l'autre le *faux* noté `False`.
Trois opération booléennes permettent d'en combiner les valeurs :

- `and` : `c1 and c2` est vraie lorsque tous les deux sont vrais
- `or` : `c1 or c2` est vraie lorsqu'au moins l'un des deux est vrai.
- `not` : `not c1` est vraie lorsque `c1` est faux.

Voici en exemple un programme permmettant de comparer les coordonnées `xa, ya` et `xb, yb` de deux point x et y et affiche des informations sur leurs positions relatives :

```
if xa == xb and ya == yb :
    print("Points confondus")
elif xa == xb or ya == yb :
    print("Points alignés horizontalement ou verticalement")
else :
    print("Points indépendants")
```

Exercice 40

Ecrire un programme qui demande un entier n à l'utilisateur et affiche tous ses diviseurs.

Exercice 41

Reprendre l'exercice précédent et préciser à l'affichage si le nombre n donné est premier, c'est à dire divisible seulement par 1 et lui-même.

Exercice 42

Au jeu de mölkky, chaque joueur marque à son tour de jeu entre 0 et 12 points, qui viennent s'ajouter à son score précédent. Le premier à atteindre le score de 51 a gagné. Mais gare ! Quiconque dépasse le score cible de 51 revient immédiatement à 25 points.

1. Écrire un programme demandant un score et un nombre de points marqués. Il doit ensuite afficher le nouveau score ou signaler éventuellement la victoire.
2. Enrichir le programme avec la règle suivante : si le joueur marque un score de 0 lors de trois coups consécutifs, il faut afficher `Perdu`.
   Indication : on utilisera une variable `nb_zeros` qui enregistrera le nombre de coups à 0 points consécutifs.

Exercice 43

Au bowling, on a deux chances pour faire tomber un total de 10 quilles.

1. Écrire un programme qui demande le nombre de quilles renversées avec chacune des deux boules et qui affiche `X` si toutes les quilles sont tombées à la première boule, `/` si toutes les quilles sont tombées, et sinon le nombre de quilles renversées.
2. Reprendre le programme précédent en ne demandant les informations de la deuxième boule que si elle a besoin d'être lancée.
3. Reprendre le programme précédent en affichant `!` si les scores saisis sont impossibles.

Exercice 44

Écrire un programme demandant deux nombres à l'utilisateur et qui affiche la distance entre ces deux nombres.

Exercice 45

Ecrire un programme qui demande trois nombres `a, b` et `c` à l'utilisateur et qui affiche :

- " Plus petit" si `c` est plus petit que les deux autres.
- " Plus grand" si `c` est plus grand que les deux autres.
- " Entre les deux" si `c` est compris entre les deux autres (sans présager de qui parmi `a` ou `b` est le plus petit).

Exercice 46

Écrire un programme qui demande une année à l'utilisateur et indique s'il s'agit d'une année bissextile.
Depuis l'ajustement du calendrier grégorien, l'année n'est bissextile (comportant 366 jours)1 que dans l'un des deux cas suivants :

- si l'année est divisible par 4 et non divisible par 100 ;
- si l'année est divisible par 400

Exercice 47

Écrire un programme qui parcourt les nombres de 1 à 100. Il affichera `fizz` lorsque l'entier est un multiple de 3, `buzz` lorsque c'est un multiple de 5 (et en particulier `fizzbuzz` lorsque c'est un multiple de 15) et l'entier sinon.

Exercice 48

Écrire un programme qui demande 3 entiers à l'utilisateur et qui affiche ces trois entier dans l'ordre croissant.

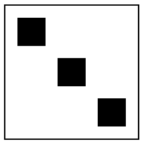
Exercice 49

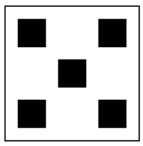
Écrire un programme Turtle demandant à l'utilisateur un entier n et traçant un damier de côté n.

IV Les fonctions

IV.1 - Introduction : dessiner une face d'un dé

On souhaite pouvoir dessiner à l'aide de la bibliothèque Turtle, les faces d'un dé. Voici par exemple la face 3 et la face 5 :





Voici un exemple de programme permettant de dessiner la face 3 :

```
# Carré extérieur
up()
goto(50, -50)
down()
for_ in range(4):
    left(90)
    forward(100)

#Point central
up()
goto(10, -10)
down()
begin_fill()
for _ in range(4):
    left(90)
    forward(20)
end_fill()

#Point inférieur droit
up()
goto(40, -40)
down()
begin_fill()
for _ in range(4):
    left(90)
    forward(20)
end_fill()

#Point supérieur gauche
up()
goto(-20, 20)
down()
begin_fill()
for _ in range(4):
    left(90)
    forward(20)
end_fill()
```

Lorsqu'on recopie ce programme, on se rend rapidement compte qu'on se retrouve avec des fragments de code très similaires les uns des autres.
Nous allons voir dans cette section comment rendre ce code plus élégant en definissant des *fonctions*, c'est à dire des fragments de codes réutilisables et modulables suivant des paramètres (comme la position, la taille, la couleur, ...).

IV.2 - Définir une fonction

La synthaxe Python pour la définition d'une fonction est la suivante :

```
def nomDeLaFonction(liste de paramètres):
...
bloc d'instructions
...
```

On peut ainsi, définir une fonction appelée `carre`, qui dessine un carré de côté 100 :

```
def carre()
    for _ in range(4):
        left(90)
        forward(100)
```

Pour exécuter les instruction de la fonction `carre()`, on appelle la fonction avec la syntaxe suivante :

```
carre()
```

IV.3 - fonctions avec paramètres

Dans notre fonction `carre`, la longueur fixée est 100. Si on veut donc pouvoir dessiner un carré de longueur quelconque on doit préférer donner un paramètre à notre fonction :

```
def carre(n):
    for _ in range(4):
        left(90)
        forward(n)
```

On peut ainsi appeler par exemple `carre(100)` ou `carre(20)`

Si on revient à notre problème initial, on remarque que les différents carrés à dessiner diffèrent seulement par leur positionnement. On peut donc créer une fonction en rajoutant ces coordonnées comme paramètres :

```
def carre(x, y, n):
    for _ in range(4):
        up()
        goto(x,y)
        down()
        left(90)
        forward(n)
```

On peut donc cette-fois appeler `carre(50,-50,100)` ou `carre(10,-10,20)`

IV.4 - Résolution du problème

```
 def carre(x, y, n):
    up()
    goto(x,y)
    down()
    for _ in range(4):
        left(90)
        forward(n)

def carre_rempli(x, y, n):
    begin_fill()
    carre(x, y, n)
    end_fill()

hideturtle()
carre(50, -50, 100)
carre_rempli(10, -10, 20)
carre_rempli(40, -40, 20)
carre_rempli(-20, 20, 20)
carre_rempli(40, 20, 20)
carre_rempli(-20, -40, 20)
```

Exercice 50

Dessiner la face de 5

IV.5 - Renvoyer un résultat

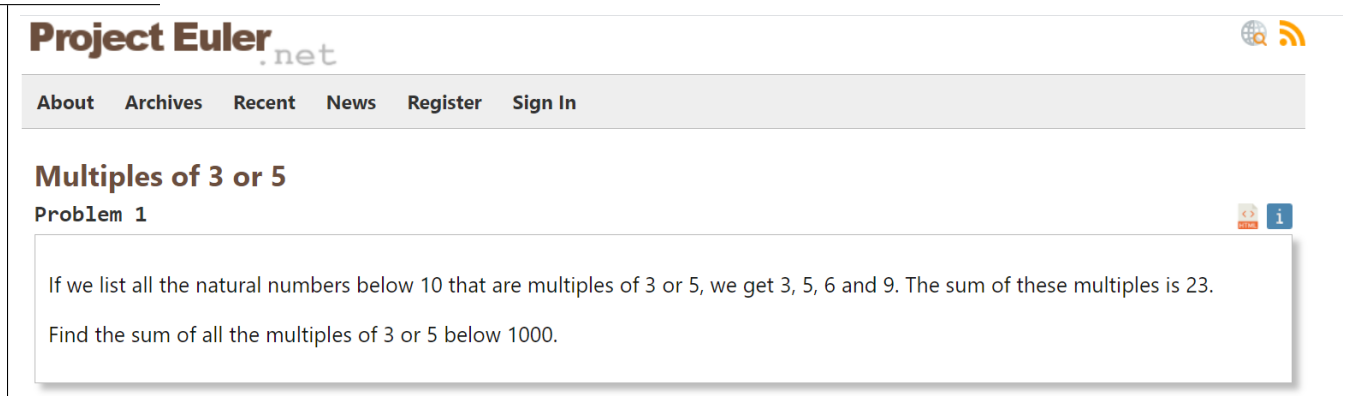
Les fonctions Python peuvent aussi représenter des fonctions mathématiques qui calculent des valeurs :

```
def f(x):
    return 3 * x + 5
```

Le mot clé `return` permet d'indiquer que le résultat de la fonction :

```
a = f(5) + 6 * f(7)
```

Exercice 51



IV.6 - Variable locale à une fonction

```
def fact(n):
    f = 1
    for i in range(2, n+1):
        f = f * i
    return f
```

Cette fonction fait intervenir une variable `f`. Cette variable est dite *locale* à la fonction.

En testant consécutivement les instructions,

```
>>> fact(4)
>>> f
>>> f = 42
>>> fact(4)
>>> f
```

On peut observer qu'il existe simultanément deux variables qui s'appellent `f` : celle définie à l'extérieur de la fonction et celle locale à la fonction. Pendant l'exécution de la fonction seule la variable locale est accessible. On dit qu'elle masque la première variable.
Voici un autre exemple illustrant le côté local d'une variable définie au sein d'une fonction :

```
def incremente(x):
    x = x +1

>>> x=1
>>> incremente(x)
>>> x
1
```

Remarque 4

Variables globales et fonctions Par opposition aux variables locales les variables définies hors de toute fonction, c'est à dire toutes les valeurs manipulées jusqu'à présent, sont appelées *variables globales*. Une fonction peut accéder à une variable globale.

```
>>> v = 42
>>> def double(): return 2*v
>>> double()
84
```

Lorsqu'une fonction accède à une variable globale, elle accède à sa valeur courante :

```
>>> v = v + 1
>>> double()
86
```

Exercice 52

Définir une fonction `test_pythagore` qui prend trois entiers a, b et c en arguments et renvoie un booléen indiquant si $a^2 + b^2 = c^2$.

Exercice 53

Définir une fonction `valeur absolue` qui prend un entier en argument et renvoie sa valeur absolue.

Exercice 54

Écrire une fonction `max2(a,b)` qui renvoie le plus grand des deux entiers a et b.

Exercice 55

En se servant de la fonction `max2` de l'exercice précédent, écrire une fonction `max3(a, b, c)` qui renvoie le plus grand des trois entiers a, b et c.

Exercice 56

Écrire une fonction `puissance(x, k)` qui renvoie x à la puissance k. On utilisera pour cela une boucle `for`.

Exercice 57

Écrire une fonction `bissextile(a)` qui renvoie un booléen indiquant si l'année a est une année bissextile. On rappelle qu'une année bissextile est une année multiple de 4 mais pas de 100, ou multiple de 400.

Exercice 58

Écrire une fonction `nbjoursannee(a)` qui renvoie le nombre de jours de l'année a. On utilisera la fonction de l'exercice précédent.

Exercice 59

Écrire une fonction `nbjoursmois(a, m)` qui renvoie le nombre de jours dans le mois m de l'année a. On suppose que le mois m est un nombre entre 1 (pour janvier) et 12 (pour décembre).

Exercice 60

En utilisant les fonctions des exercices précédents, écrire la fonction `nbjours(jn , mn, an, j, m, a)` qui renvoie le nombre de jours compris entre deux dates données (par exemple votre date de naissance et aujourd'hui).

Exercice 61

En utilisant la fonction `nbjoursmois`, écrire un programme qui demande une année à l'utilisateur et affiche le calendrier de cette année, c'est à dire la liste de tous les jours de tous les mois.

Exercice 62

Ecrire une fonction `triangle(n)` traçant avec Turtle un triangle équilatéral noir comme ci-dessous avec un côté de longueur n.

En utilisant cette fonction, reproduire les dessins suivants :

V Listes

Jusqu'à présent, on a écrit des programmes qui utilisent des variables de type **int**, **float** , **boolean** et **string**. Une boite d'un tel type contient une valeur formée d'un unique nombre, d'un unique booléen ou chaîne de caractère.
Dans de nombreuses situations nous avons besoin d'utiliser des valeurs comme les textes, les images ou les sons sont constitués de plusieurs nombres ou plusieurs booléens.
Sous Python,on peut définir une liste comme une **collection d'éléments séparés par des virgules, l'ensemble étant enfermé dans des crochets.** Par exemple :

```
>>> t = [2, 3, 5]
```

Les listes sont des séquences, c'est-à-dire des collections ordonnées d'objets. Les divers éléments qui constituent une liste sont en effet toujours disposés dans le même ordre, et l'on peut donc accéder à chacun d'entre eux individuellement si l'on connaît son index dans la liste. Il faut cependant retenir que la numérotation de ces index commence à partir de zéro, et non à partir de un.

```
>>> t[0]
2
```

La *taille* d'un tableau est son nombre de cases. On l'obtient avec la fonction `len` :

```
>>> len(t)
3
```

Les indices d'une liste t prennent donc les valeurs entre 0 et `len(t)`

V.1- Modification du contenu d'une liste

Le contenu d'une liste peut être modifié :

```
>>> t[1]=17
>>> t
[2, 17, 5]
>>> t[2]=t[2]+1
[2, 17, 6]
```

Les listes peuvent être agrandies, avec l'opération ou rétrécies du côté droit, c'est à dire du côté des indices les plus grands avec les opérations `append` ou `pop` :

```
>>> t.append(5)
>>> t
[2, 17, 6, 5]
>>> t.pop()
5
>>> t
[2, 17, 6]
```

Enfin, accéder à une liste Python avec un indice négatif ne provoque pas nécessairement une erreur. En effet, Python permet d'accéder au dernier élément de la liste avec `t[-1]`, à l'avant dernier avec `t[-2]`,...

V.2- Parcours d'une liste

Considérons la liste `t = [12, 8, 19, 9, 3, 10, 11, 5, 17, 18]` Voici un programme permettant de calculer la somme des valeurs de cette liste :

```
s=0
for i in range(len(t)):
    s=s+t[i]
```

V.3- Construire de grandes listes

Si on doit construire une liste vraiment grande, par exemple plusieurs centaines d'éléments, il devient difficile de le faire en énumérant tous ses éléments. Heureusement, il existe une opération dans le langage Python pour construire une liste de taille arbitraire :

```
>>> t = [0]*1000
>>> len(t)
1000
```

Une fois la liste ainsi construite, on peut la remplir avec les valeurs de son choix, en utilisant les affectations. Ainsi, si par exemple on veut stocker les carrés des 1000 premiers entiers :

```
for i in range (1000):
    t[i] = i * i
```

V.4- concaténation entre deux listes

Il est possible de concaténer deux listes :

```
>>> [3, 4, 9, 1] + [-1, 4]
[3, 4, 9, 1, -1, 4]
```

Remarque 5

Listes et chaînes de caractères Les chaînes de caractères et les listes ont de nombreuses similitudes :

```
>>> ch = "Cassin"
>>> len(ch)
6
>>> ch[2]
's'
>>> "On est bien à\ "+"Cassin"
On est bien àCassin
>>> "Cassin"*3
'CassinCassinCassin'
```

Cependant, contrairement aux listes, les caractères ne peuvent pas être modifiés :

```
>>> ch[3]='t'
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

V.5- Listes et variables

```
>>> x=1
>>> y=x
>>> x=2
>>> y
1
```

La modification de la variable `x` n'a aucune incidence sur la variable `y`.
Concernant les listes, la situation devient un peu plus complexe :

```
>>> s = [1, 2, 3]
>>> t = s
>>> s[2] = 4
>>> t
[1, 2, 4]
```

V.6 - listes et fonctions

Une fonction peut modifier le contenu d'une liste définie en dehors de la fonction. Examinons ci-dessous le passage d'une liste en argument :

```
x = 1
t = [1, 2, 3]

def f(a, b):
    a = a + 1
    b[a] = 7

>>> f(x, t)
>>> t
[1, 2, 7]
```

De même qu'une liste peut être passée en argument, une fonction peut renvoyer une liste :

```
def carre(n) :
    t = [0]*n
    for i in range(n):
        t[i] = i*i
    return t

>>> c = carres(4)
>>> c
[0, 1, 4, 9]
```

Exercice 63

Écrire un fonction `occurrences(v, t)` qui renvoie le nombre d'occurrences de la valeur `v` dans la liste `t`.
On doit donc avoir :

```
>>> occurences(2, [1, 2, 3, 2, 5, 7, 2])
3
```

Exercice 64

Écrire un programme qui construit une liste de 100 entiers pris au hasard entre 1 et 1000 puis l'affiche.

Exercice 65

Compléter le programme précédent pour calculer et afficher l'élément maximal de cette liste

Exercice 66

Écrire un programme qui tire au hasard mille entiers entre 1 et 10 et affiche ensuite le nombre de fois que chaque nombre a été tiré. Relancer le programme plusieurs fois.

Exercice 67

En mathématiques, la très célèbre suite de Fibonacci est définie de la façon suivante : on part des deux entiers 0 et 1 puis on construit à chaque fois l'entier suivant comme la somme des deux entiers précédents :

$$0, 1, 1, 2, 3, 5, ...$$

Écrire un programme qui construit et affiche une liste contenant les 30 premiers termes de la suite. Le dernier élément de cette liste doit être 514 229.

Exercice 68

Écrire une fonction `copie(t)` qui prend en paramètre une liste et renvoie une copie de cette liste (une copie signifie que ces deux listes sont indépendantes). Quelle expérience peut-on faire pour s'assurer qu'on ne s'est pas trompé ?

Exercice 69

Écrire une fonction `ajout(v, t)` qui crée un nouveau tableau contenant d'abord tous les éléments de `t` puis l'élément `v`

Exercice 72

Écrire une fonction `echange(t, i, j` qui échange dans la liste `t` les élément d'indice i et j.

Exercice 73

Écrire une fonction `somme(t)` qui calcule et renvoie la somme des éléments d'une liste d'entier.
En déduire une fonction `moyenne(t)` qui calcule et renvoie la moyenne des éléments de la liste `t`, supposée non vide.

Exercice 74

Écrire une fonction `produit(t)` qui calcule et renvoie le produit des éléments d'une liste d'entiers. Si la liste contient 0, la liste devra renvoyer 0 sans continuer le calcul.

Exercice 75

Écrire une fonction `miroir(t` qui reçoit une liste en argument et le modifie en échangeant le premier avec le dernier élément, le deuxième avec l'avant-dernier, etc...
On pourra se servir de la fonction `echange` de l'exercice précédent.

Exercice 76

Pour mélanger les éléments d'une liste aléatoirement, il existe un algorithme très simple : on parcourt la liste de la gauche vers la droite et, pour chaque élément à l'indice i, on l'échange avec un élément situé aléatoirement entre 0 et i (inclus).
Écrire une fonction `melange(t)` qui réalise cet algorithme (on pourra se resservir de la fonction `echange`).
Cet algorithme s'appelle le *mélange de Knuth*.

Exercice 77

Écrire une fonction `prefixe(t1, t2)` qui renvoie `True` si la liste `t1` est un préfixe de la liste `t2`, c'est à dire si la liste `t2` commence par les éléments de la liste `t1` dans le même ordre. Par exemple :

```
>>> prefixe([3, 1, 5], [3, 1, 5, 6, 7]
True
```

Exercice 78

Écrire une fonction `suffixe(t1, t2)` qui renvoie `True` si la liste `t1` est un suffixe de la liste `t2`, c'est à dire si la liste `t2` termine par les éléments de la liste `t1` dans le même ordre. Par exemple :

```
>>> suffixe([3, 1, 5], [3, 2, 3, 1, 5]
True
```

Exercice 79

Écrire une fonction `hamming(t1, t2)` qui prend en paramètre deux listes `t1` et `t2`, que l'on supposera de même taille, et qui renvoie le nombre d'indices auxquels les deux listes diffèrent.

Exercice 80

Reprendre l'exercice précédent sans supposer que les listes sont de même taille. On considérera qu'un indice pour lequel seul l'une des listes est défini compte pour une différence.

VI Boucle While

Les boucles "Tant que" ou boucles `While`, combinent le fonctionnement d'une boucle `For` avec un test qui conditionne la poursuite des itérations :

```
n = 11
i=2
while i <= n:
    print(i)
    i = 2 * i
print("Fin")
```

L'arrêt des itérations d'une boucle `While` étant conditionnée à l'échec d'un test, un nouveau phénomène devient possible : celui d'un programme dont l'exécution ne s'arrête jamais. On parle de *divergence* ou non terminaison :

```
while i!=0 :
    print(i)
    i = i - 1
```

VI.1 - Sortir d'une boucle avec l'instruction `Break`

Considérons le problème suivant : demander à l'utilisateur de saisir un nombre positif ou nul, et continuer à le lui demander tant que la donnée saisie n'est pas valide

Voici une possibilité :

```
n=int(input("Entrer un nombre positif ou nul : "))
while n<0 :
    n=int(input("Entrer un nombre positif ou nul : "))
```

Mais on peut résoudre ce problème en utilisant l'intruction `break` qui provoque l'interruption immédiate de la boucle :

```
while True :
    n = int(input("Entrer un nombre positif ou nul : ")
    if n >=0:
        break
```

VI.2 - Problème : chercher dans un tableau

On souhaite écrire une fonction `appartient` prenant en paramètre une valeur `v` et une liste `t`, renvoyant `True` si la valeur `v` apparaît au moins une fois dans la liste `t`, et `False` sinon. Voici plusieurs solutions :

1. Parcours intégral de la liste avec une boucle `For` avec variable booléenne `trouvee` indiquant si la valeur a déjà été trouvée.

```
def appartient(v, t):
    trouvee = False
    for i in range(len(t)):
        if t[i] == v:
            trouvee = True
    return trouvee
```

2. Parcours de la liste avec une boucle `while` jusqu'à ce que la valeur soit trouvée

```
def appartient(v, t):
    i = 0
    while i < lent(t) and t[i] != v:
        i = i + 1
    return i < len(t)
```

3. Parcours intégral de la liste par une boucle `While`, avec interruption si élément trouvé.

```
def appartient(v, t):
    i = 0
    while i < lent(t):
        if t[i] == v:
            break
        i = i + 1
    return i < len(t)
```

4. Parcours intégral par une boucle `For`, avec renvoi anticipé de la valeur `True` si l'élément est trouvé, et renvoi de la valeur `False` si la boucle arrive à son terme.

```
def appartient(v, t):
    for i in range(len(t)):
        if t[i] == v:
            return True
    return False
```

VI.3 - Terminaison

Considérons le programme suivant où la variable `n` contient initialement une valeur entière.

```
q = 0
while n>=3 :
    n-=3
    q+=1
print(q)
```

Il contient une boucle `While` et son exécution peut donc diverger. Cependant, on peut justifier que ce n'est pas le cas ici :
En effet, à chaque tour de boucle, la nouvelle valeur affectée à `n` est strictement inférieur à la précédente. Ainsi, à un certain moment cette valeur finira par être inférieur ou égale à 2, ce qui provoquera l'arrêt de la boucle.
Cette technique de raisonnement est appelée *technique du variant*. Elle consiste à trouver parmi les éléments du programme une quantité qui est

- entière et positive
- *strictement* décroissante

Une telle variable s'appelle *variant de boucle*

Exemple
Le variant de boucle n'est pas obligatoirement une variable du programme :

```
i=0
while i < lent(t):
    if t[i] == v:
        break
    i = i + 1
```

Ici le variant sera la quantité `lent(t) - i`

Exercice 81

Écrire une fonction qui prend un entier positif ou nul en argument et renvoie son nombre de chiffres.

Exercice 82

La suite de Syracuse d'un nombre entier $N > 0$ est définie par récurrence, de la manière suivante :

$$\begin{cases} u_0 = N \\ \\ \text{Pour tout } n \in \mathbb{N}, \quad u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair} \\ \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases} \end{cases}$$

La conjecture de Syracuse, est l'hypothèse mathématique selon laquelle la suite de Syracuse de n'importe quel entier strictement positif atteint 1.

En dépit de la simplicité de son énoncé, cette conjecture défie depuis de nombreuses années les mathématiciens.

1. Écrire un programme qui demande la valeur à donner à u_0 puis qui affiche la suite des termes jusqu'à l'apparition du premier 1.
2. Modifier le programme précédent pour qu'il affiche au final le nombre d'étapes pour arriver au nombre 1 et la nombre maximal qui a été rencontré en chemin.
   Par exemple, pour le nombre de départ 7, le nombre d'étapes est 16 et le maximum rencontré est 52.
3. Modifier encore le programme précédent pour tester tous les nombres de départ entre 1 et 20 puis afficher pour chacun le nombre d'étapes et de maximum rencontré.
   Quel est le nombre de départ donnant le plus grand nombre d'étapes ? donnant le plus grand maximum rencontré ? Tester ensuite au delà de 20.

Exercice 83

Ecrire un programme qui affiche les termes de la suite de Fibonacci inférieurs à 1000.

Exercice 84

Écrire une fonction `digramme(a, b, t)` qui teste s'il existe dans la liste `t`, une occurrence de l'élément `a` immédiatement suivie d'une occurrence de l'élément `b`.

Exercice 87

Écrire une fonction `pgs(e, t)` qui renvoie la longueur la plus grande séquence d'occurrences consécutives de l'élément `e` dans la liste `t`.

Exercice 88

Écrire une fonction `pgs(t)` qui prend en paramètre une liste `t` et renvoie la longueur de la plus grande séquence d'éléments identiques consécutifs.

Exercice 89

Écrire une fonction `pgpc(t1, t2)` qui prend en arguments deux listes et qui renvoie la longueur du plus grand préfixe commun à ces deux listes, c'est à dire une valeur `l` telle que les `l` premières valeurs de `t1` et `t2` sont égales mais pas celles d'indice `l`

Exercice 90

Écrire une fonction `ordre(m1, m2` qui prend en paramètre deux mots `m1` et `m2` sous la forme de chaînes de caractères, et qui renvoie `True` si `m1` est avant `m2` dans l'ordre du dictionnaire ou si `m1` est égale à `m2`.

Exercice 91

Dans le jeu *devine un nombre*, un joueur doit trouver un nombre mystère choisi au préalable par le maître du jeu, qui sera ici l'ordinateur. A chaque tour, le joueur propose un nombre et le maître du jeu indique si le nombre mystère est plus petit ou plus grand que le nombre proposé. Si le nombre mystère est 5, par exemple, une partie pourra se dérouler ainsi :

```
Devine le nombre !
8
plus petit
4
plus grand
6
plus petit
5
Gagné !
```

Écrire un programme permettant de jouer à ce jeu, avec un nombre à chercher compris entre 0 et 100.

Exercice 92

Le *jeu des allumettes* est un jeu à deux joueurs qui se joue ainsi. Initialement 21 allumettes sont posées sur la table. A tour de rôle, chaque joueur enlève une, deux ou trois allumettes. Celui qui enlève la dernière allumette a perdu. Nous allons écrire un programme qui permet de jouer contre l'ordinateur.

1. A l'aide d'une boucle `While` et d'une variable `allumettes` contenant le nombre courant d'allumettes, écrire un programme qui permet à l'utilisateur d'indiquer combien d'allumettes il enlève (au moins une et pas plus de trois), tant qu'il reste des allumettes. Au début de chaque tour de boucle, on affichera le nombre actuel d'allumettes. L'exécution doit donc ressembler à quelque chose comme ceci :

   ```
   il y a 21 allumettes
   combien en prenez-vous ? 3
   je prends 2 allumettes
   il y a 16 allumettes
   combien en prenez-vous ? 1
   ...
   ```

   Attention à ne pas faire retirer par l'ordinateur plus d'allumettes qu'il n'en reste.
2. Améliorer ce programme pour qu'il affiche `vous avez perdu` ou `vous avez gagné` lorsque la dernière allumette est retirée, juste avant de sortir de la boucle.
3. Améliorer encore ce programme pour qu'il assure que le nombre d'allumettes choisi par le joueur est bien au moins égal à 1, au plus égal à 3 et pas plus grand que le nombre d'allumettes restantes.
   On pourra utiliser pour cela une boucle `While` pour répéter la saisie tant que la valeur n'est pas correcte.
4. Il se trouve qu'il existe une stratégie gagnante pour le joueur qui joue en second. La trouver et la programmer, de manière à ce que l'ordianteur gagne systématiquement (et impressionner ensuite vos amis avec votre programme d'intelligence artificielle).

VII Utilisation avancée des boucles

Exercice 93 (*boucles imbriquées*)

Qu'affichent les programmes suivants :

1.
```
for i in range(3):
    for j in range(2):
        print(i,j)
```

2.
```
for i in range(2):
    for j in range(3):
        print(i, j)
```

3.
```
for i in range(1, 4):
    print("i :", i)
    for j in range(i):
        print(" |j:", j)
```

Exercice 95

Qu'affiche le programme suivant :

```
for i in range(1000):
    for j in range(1000):
        for k in range(1000):
            for l in range(1000):
                print("Mille sabords")
```

Exercice 96

Écrire un programme qui affiche les tables de multiplications sous la forme :

```
Table de 1 :
    1 x 1 = 1
    1 x 2 = 2
    ...
Table de 2 :
    ...
```

Exercice 97

Écrire un programme qui demande deux entiers h et l à l'utilisateur et affiche les symboles disposés en un rectangle de hauteur h et de largeur l

Exercice 98

Écrire des programmes qui demandent un entier n à l'utilisateur et affichent chacun l'une des 4 figures suivantes (dans un carré de côté n) :

```
#
##
###
####

####
###
##
#

####
.###
..##
...#

####
#..#
#..#
####
```

Exercice 99 (*Recherche de doublons*)

On suppose donné une liste `t` d'éléments quelconques, et on souhaite savoir si cette liste contient des doublons. Autrement dit, on souhaite savoir s'il existe deux indices i et j tels que `t[i]` et `t[j]` sont égaux.
Ecrire une fonction prenant comme argument une liste t et renvoyant `True` si cette liste contient des doublons et `False` sinon.

Exercice 100

1. En utilisant trois boucles imbriquées, afficher tous les triplets d'entiers $1 \leqslant a \leqslant b \leqslant c \leqslant 100$ tels que $a^2 + b^2 = c^2$ (dits triplets pythagoriciens).
2. Modifier le programme précédent pour qu'il renvoie le nombre de triplets (On doit trouver 52).
3. Écrire ce programme sous la forme d'une fonction qui reçoit en argument la borne supérieure des boucles et renvoie le nombre de triplets.

Exercice 101

Pour calculer les nombres premiers plus petits qu'une certaine limite N qu'on se fixe, il existe un algorithme appelé *crible d'Erathosthène* :

- on se donne une liste `t` de N booléens initialement tous égaux à `True` sauf `t[0]` et `t[1]` qui valent `False`.
- On parcourt cette liste, dans le sens des indices croissants. Pour chaque indice i, il y a deux possibilités :
  - ⋆ si `t[i]` vaut `False`, alors le nombre `i` n'est pas premier et il n'y a rien à faire.
  - ⋆ Si `t[i]` vaut `True`, alors le nombre `i` est premier et on met `False` à toutes les cases du tableau dont l'indice est un multiple de `i`, c'est à dire `2*i`, `3*i`.

Écrire un programme qui réalise cet algorithme et affiche tous les nombres premiers plus petits que 100. On doit en trouver 25.

Exercice 103

Écrire une fonction `facteur(t1, t2)` qui renvoie `True` si les éléments de la liste `t1` apparaîssent dans l'ordre et de manière consécutive dans la liste `t2`.
Proposer une version utilisant une fonction auxiliaire et une version utilisant des boucles imbriquées.

Exercice 104

Ecrire une fonction `damier(n)` qui prend en paramètre un entier n et trace avec Turtle un damier de n cases de côtés.

VIII Utilisation avancée des listes

VIII.1 - Itérer sur une liste

Lorsqu'on veut répéter une opération pour chaque élément `e` d'une liste `t`, il est possible d'utiliser l'instruction `for` directement sur la liste elle-même :

```
for e in t:
    print(e)
```

Ce programme est équivalent à celui ci-dessous où la boucle énumèrent les indices de la liste :

```
for i in range(len(t)):
    e = t[i]
    print(e)
```

Exemple 1

Qu'affiche le programme suivant :

```
print("Directions possibles :")
for d in ["Nord", "Sud", "Est", "Ouest"]:
    print("*", d)
```

VIII.2 - Construire une liste par compréhension

Supposons que l'on veuille construire une liste de taille 100 contenant l'entier $3i + 1$ dans la case d'indice i.
Voici une première solution :

```
t=[0]*100
for i in range(100):
    t[i] = 3 * i + 1
```

Cependant Python propose une syntaxe plus compacte pour combiner l'allocation d'un tableau et son remplissage par une boucle `for` :

```
>>> [3 * i + 1 for i in range(100)]
```

Dans cette construction le parcours de la variable i n'est pas limité à un intervalle d'entier construit avec `range`. On peut ainsi parcourir une liste déjà construite :

```
>>> t = [3 * i + 1 for i in range(10)]
>>> [x * x for x in t]
[1, 16, 49, 100, 169, 256, 361, 484, 625, 784]
```

Enfin, on peut ne conserver que certaines valeurs prises par la variable, en ajoutant une condition booléenne à la compréhension :

```
>>> [i * i for i in range(30) if i%4 == 1]
[1, 25, 81, 169, 289, 441, 625, 841]
```

Remarque 6

Dans le cas particulier où l'on veut construire une liste contenant les entiers de i inclus à j exclus, on peut donc écrire :

```
>>> [v for v in range(i, j)]
```

Il existe cependant une construction encore plus simple :

```
>>> list(range(i, j))
```

VIII.3 - Listes à plusieurs dimensions

Les listes Python peuvent contenir des valeurs arbitraires, en particulier des listes :

```
>>> t = [[1, 0, 0, 0, 0], [1, 1, 0, 0, 0], [1, 2, 1, 0, 0]]
>>> t[2][1]
2
>>> t[1][2]
0
>>> t[1][3] = 7
>>> t
[[1, 0, 0, 0, 0], [1, 1, 0, 7, 0], [1, 2, 1, 0, 0]]
```

Remarque 7

On peut utiliser avantageusement la construction par compréhension pour construire de tels tableaux :

```
>>> t = [[0]*5 for i in range(3)]
>>> t
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
>>> t[0][1] = 7
>>> t
[[0, 7, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

Mais attention au type de construction ci-dessous qui pourrait ne pas correspondre à ce que l'on voulait faire :

```
>>> t = [[0]*5]*3
>>> t
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
>>> t[0][1] = 7
>>> t
[[0, 7, 0, 0, 0], [0, 7 0, 0, 0], [0, 7, 0, 0, 0]]
```

VIII.4- Parcours d'une liste à plusieurs dimensions

Pour parcourir tous les éléments d'une liste à plusieurs dimensions, on va naturellement utiliser des boucles imbriquées où chaque boucle parcourt une dimension.
Si on prend notre liste `t` de taille 3 x 5 et qu'on souhaite par exemple faire la somme de tous ses éléments :

```
for i in range(3):
    for j in range(5):
        s = s + t[i][j]
```

ou ce qui est équivalent :

```
for l in t:
    for x in l:
        s = s + x
```

Exercice 105

Que construit l'expression `i % 3 for i in range(100)` ?

Exercice 106 (*Tables de multiplication*)

Écrire un programme qui construit une liste t de taille 11 X 11 tel que `t[i][j]` contient `i x j`

Exercice 107

Écrire un programme qui transforme une liste `t` de 64 cases en une liste `m` à deux dimensions de taille 8 x 8, tel que `m[i][j] = t[8 * i + j]` pour tout $0 \leqslant i, j < 8$.
Indication : commencer par créer la liste puis la remplir avec une double boucle.

Exercice 108

1. Écrire une liste à 2 dimensions de taille 30 x 30 contenant des entier tirés au hasard entre 1 et 9999, puis l'affiche.
2. Compléter le programme précédent pour calculer et afficher l'élément maximum de cette liste.

Exercice 109

Écrire une fonction qui prend en paramètre une liste à deux dimensions de hauteur et largeur non nulles et qui renvoie le maximum parmi les minima de chaque ligne.
Indication : on pourra utiliser une fonction auxiliaire qui calcule le minimum d'une ligne.

Exercice 110

Le Puissance 4 est un jeu bien connu à deux joueurs qui se joue sur un grille verticale de six lignes et sept colonnes. A tour de rôle, chaque joueur fait tomber un pion de sa couleur dans une colonne de son choix non encore pleine. Le premier joueur qui aligne quatre pions de sa couleur, horizontalement, verticalement, ou en diagonale, gagne la partie. La partie est nulle si la grille est totalement remplie sans qu'aucun joueur ne gagne.
Modélisation :

- Pour représenter une grille de ce jeu, on utilise une liste à deux dimensions de taille 6 x 7, la première dimension représentant les lignes et la seconde les colonnes.
- Une ligne est noté `l` et prend une valeur entre 0 et 5, la ligne 0 étant située en bas.
- une colonne est notée `c` et prend une valeur entre 0 et 6, la colonne 0 étant située à gauche.

- Un joueur est noté `j` et prend la valeur 1 ou 2. Dans une grille notée `g`, la valeur 0 représente une case vide et la valeur 1 ou 2 représente un pion du joueur correspondant.

1. Écrire une fonction `grille_vide()` qui renvoie une grille vide.
2. Écrire une fonction `affiche(g)` qui affiche une grille, avec le caractère `.` pour une case vide, le caractère `X` pour le joueur 1 et le caractère `O` pour le joueur 2. On prendra soin de bien afficher les lignes de haut en bas.
3. Écrire une fonction `coup_possible(g, c)` qui renvoie un booléen indiquant s'il est possible de jouer dans la colonne `c`.
4. Écrire une fonction `jouer(g, j, c)` qui joue un coup du joueur `j` dans la colonne `c`, en supposant que la colonne `c` n'est pas pleine.
5. Écrire trois fonctions `horiz(g, j, l, c)`, `vert(g, j, l, c)` et `diag(g, j, l, c)` qui déterminent respectivement s'il y a un alignement de quatre pions du joueur `j` à partir de la case `(l, c)`.
6. Écrire une fonction `victoire(g, j)` qui renvoie un booléen indiquant si le joueur `j` a gagné.
7. Écrire une fonction `match_nul(g)` qui renvoie un booléen indiquant s'il y a match nul, c'est à dire si la grille est complétement remplie.
   Indication : on peut se contenter d'examiner la ligne du haut.
8. Écrire une fonction `coup_aleatoire(g, j)` qui joue un coup aléatoire légal pour le joueur `j`, en supposant que la grille n'est pas pleine.
   On prendra soin de ne pas favoriser de colonne particulière.
9. Écrire un programme qui fait jouer deux adversaires aléatoirement à tour de rôle, en affichant la grille après chaque coup, et qui s'arrête dès qu'un joueur gagne ou que la partie est nulle.
10. Écrire enfin un programme qui permet à l'utilisateur de jouer contre un adversaire qui joue aléatoirement, en affichant la grille après chaque coup et le résultat en fin de partie.

Exercice 111

Ecrire une fonction `dessine` qui prend en paramètre une liste de dimension 2 contenant des booléens et qui le dessine à l'aide de Turtle en représentant chaque occurrence `True` par un carré noir.
Par exemple la liste :

```
[ [True, False, True],
  [False, True, False],
  [True, False, True] ]
```

est représentée par :