Math93.com

TD n°A1 - Algorithmes
Apprentissage de Python

Le symbole # (se lit « croisillon », « hash » en anglais , symbole proche du « dièse ») permet de faire figurer dans le corps du programme un commentaire qui ne sera pas pris en compte lors de son exécution.

Le symbole = n'est pas celui de l'égalité mathématique, il n'est d'ailleurs pas symétrique. Il s'agit d'affecter une valeur à une variable : on stocke une valeur numérique ou du texte dans une mémoire.

Première partie
Variables et affichages

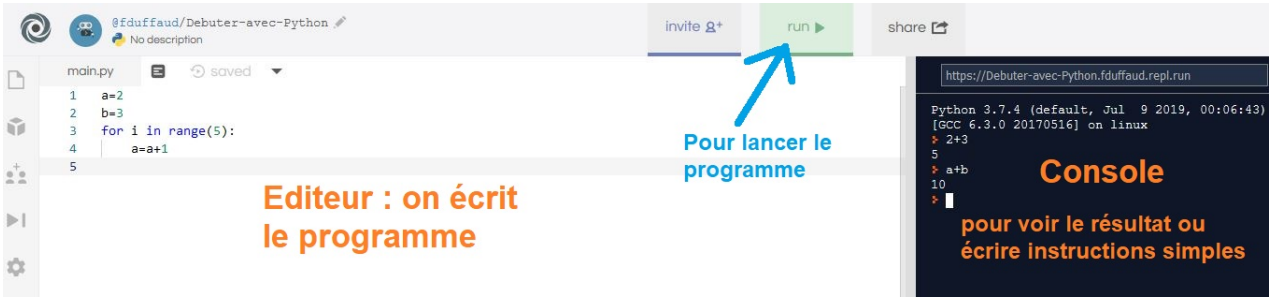
Premiers pas avec Python.

Exercice 1. Éditeur et console sous Python

1. Lancer l'éditeur Python

On ouvre un éditeur Python :

1. avec un éditeur en ligne : www.repl.it
2. avec un éditeur comme Spyder : https ://pypi.org/project/spyder/
3. avec un éditeur comme Edupython : https ://edupython.tuxfamily.org/



2. Dans la console

La console se reconnaît facilement. C'est elle qui contient le chevron > (ou le triple >>>) qui est l'invite de Python (prompt en anglais) et qui signifie que Python attend une commande.
L'esprit d'utilisation de la console est un peu le même que celui d'une calculatrice.

```
>>>  2+3
5
>>>  a=5
>>>  a-9
-4
```

Exemple.
Lancer les instructions suivantes dans la console en appuyant sur **Enter** à chaque fin de ligne et regarder le résultat :

```
>>>print ( "Hello world !")
>>>x=3
>>>print(x)
>>>4+5
>>>5/2
>>>5//2
>>>print("la valeur de x est", x)
```

3. Pour commencer un programme

- Sur repl.it : cliquez simplement sur : **+ new repl** et donner un nom à votre fichier. Il sera automatiquement enregistré après chaque **Run**.
- Sinon sur un éditeur hors ligne : Cliquer sur Fichier puis Nouveau puis sélectionner Nouveau Module Python. Enregistrer IMMEDIATEMENT votre programme dans votre répertoire de travail avec le nom Mon1erProgramme.py.

Exercice 2. Variables

```
# Dans l'éditeur PYTHON
a=2
b=-5
a,b=a+b,a-b
print("Maintenant a= ",a," et b = ",b)
```

On considère l'algorithme ci-dessus écrit sous Python. Compléter le tableau suivant et donner l'affichage de la ligne 4. Vous pourrez ensuite tapez le programme sous Python pour vérifier vos résultats.

Ligne	a	b
L1	...	...
L2	...	...
L3	...	...

```
# Dans l'éditeur PYTHON
a=2
b=-5
a=a+b
b=a-b
print("Maintenant a= ",a," et b = ",b)
```

On considère l'algorithme ci-dessus écrit sous Python. Compléter le tableau suivant et donner l'affichage de la ligne 5. Vous pourrez ensuite tapez le programme sous Python pour vérifier vos résultats.

Ligne	a	b
L1	...	...
L2	...	...
L3	...	...
L4	...	...

Remarque

Notez la différence entre les résultats.

- Dans l'exemple de gauche ci-dessus, les valeurs de a et b sont affectées simultanément en utilisant les valeurs des lignes précédentes.
- En revanche dans celui de droite, les affectations sont successives, ce qui explique les résultats différents.

Ainsi $a, b = b, a$ échange les valeurs des deux variables a et b (sans utilisation d'une variable tampon).

Exercice 3. Suite d'affectations

On considère l'algorithme suivant écrit en pseudo code :

L1	**Traitement :**	$U \leftarrow 500$
L2		$N \leftarrow 0$
L3		$U \leftarrow 0.7 \times U + 300$
L4		$N \leftarrow N + 1$

1. Compléter le tableau suivant afin de déterminer les valeurs affichées en sortie.

Ligne	U	N
L1	⋯	⋯
L2	⋯	⋯
L3	⋯	⋯
L4	⋯	⋯

2. Écrire sous Python ce programme en utilisant le moins de lignes possible.

Exercice 4. Une fonction ... d'Euler (Partie 1)

def nom_ fonction(paramètres) :

def nom_ fonction(paramètres) : définit une nouvelle fonction, les deux points entrainent une indentation délimitant la déclaration de la fonction. Le bloc peut servir à effectuer une série d'actions, mais le plus souvent il se termine par *return* pour renvoyer une ou plusieurs valeurs.

```
# Dans l'éditeur PYTHON
def f(n):
        return n**2-n+41 # n**2=n*n (notation puissance)
```

1. La fonction définie ci-dessus renvoit l'image de la variable x par la fonction f définie par $f(x) = x^2 - x + 41$.
Tester le programme avec des entiers naturels en écrivant dans la console (*à droite sur ripl.it ou votre logiciel IDLE*) directement $f(1)$ ou $f(2)$ par exemple.

```
# Dans la console PYTHON
>>> f(5)
61
>>> f(8)
97
```

2. On veut maintenant calculer les valeurs de la fonction f pour x entier variant de 0 à 19 par exemple.

range(début , fin , pas)

range(début , fin , pas) : Génère une liste d'entiers les paramètres *début* et *pas* sont optionnels.

- Dans l'intervalle [0 , fin[si un seul paramètre est renseigné.
L = range(4) va créer la liste [0 , 1 , 2 , 3] de 4 termes, le premier sera L[0]=0, le dernier L[3] = 3.
- Dans l'intervalle [début ; fin[si 2 paramètres sont renseignés.
L = range(1 , 5) va créer la liste [1 , 2 , 3 , 4] le premier terme sera L[0] = 1 et le dernier L[3] = 4.
- Dans l'intervalle [début ; fin[mais de *pas* en *pas*, si les 3 paramètres sont renseignés.
L = range(2 , 9 , 2) va créer la liste [2 , 4 , 6 , 8] .

```
# Dans l'éditeur PYTHON
def f(n):
        return n**2-n+41 # n**2=n*n (notation puissance)
valeursx=[x for x in range(20)] # range(20) itère de 0 à 19
valeursy=[f(x) for x in range(20)]
```

Pour visualiser le résultat, écrire simplement valeursx et valeursf dans la console de droite.

```
# Dans la console PYTHON
>>> valeursx
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
 18, 19]
>>> valeursy
[41, 41, 43, 47, 53, 61, 71, 83, 97, 113, 131, 151, 173, 197,
 223, 251, 281, 313, 347, 383]
>>>
```

Remarque

S'inspirant de l'écriture mathématique d'un ensemble en compréhension, par exemple

$$\left\{x \mid x \in [\![0\,;\,19]\!]\right\}$$

Python propose une syntaxe utile pour la création d'une liste en compréhension :

[x for x in range(20)]

Pour de nombreux compléments, faire le TD Fonctions 1 : Une fonction d'Euler ...

Avec le module math

Exercice 5. Périmètre et aire

Écrire une fonction de paramètre r qui renvoie le périmètre d'un cercle de rayon r puis une autre (aussi de paramètre r) qui renvoie l'aire du disque de rayon r.

Remarque

Les fonction mathématiques de base sont présentes dans le *module math* qu'il faut appeler au début du programme sous la forme **import math**.
Pour le nombre π écrire tout simplement : **math.pi**

Astuce : En écrivant **from math import *** au début on importe directement toutes les fonctions et **from math import pi** on importe le nombre pi (enfin une valeur approchée!) ce qui permet d'écrire directement pi (au lieu de math.pi)

Docstring

Les **docstrings** sont des chaines de caractères qui doivent être placées juste en dessous des définitions de fonction et entre 3 apostrophes ' ' '. On définit ainsi les variables de la fonction en entrée (IN) et celles en sortie de la fontion (OUT). Les docstrings sont récupérables dynamiquement avec la fonction **help( )**.

```
def f(x):
    ''' In : x décimal (un flottant)
        Out : image de x par f definie par f(x) = x**2 + 1'''
```

```
    return x**2+1
```

```
from math import *

def perimetre(r):
        '''IN : r le rayon du cercle
          OUT : une valeur approchée du périmètre du cercle de rayon r'''
        return ...

def aire(r):
        '''IN : r le rayon du disque
          OUT : une valeur approchée de l'aire du disque de rayon r'''
        return ...
```

Quelques problèmes de pourcentages

Exercice 6. TVA et fonctions

1. Chercher sur internet ce qu'est la T.V.A., le prix Hors Taxes (HT) et le prix Toutes Taxes Comprises (T.T.C) d'un article.
2. Écrire un algorithme utilisant une fonction qui calcule directement le prix TTC avec une T.V.A. à 20 % en fonction du prix HT.

```
def calc_TTC(prix_HT):
        '''IN : prix HT d'un article
           OUT : Prix TTC avec une TVA à 20%'''
        return ...
```

Exercice 7. TVA, fonction et affichage

Écrire un algorithme utilisant une fonction qui demande le prix d'un article TTC et qui calcule directement le prix HT avec une T.V.A. à 20 %.

```
def calc_HT(prix_TTC):
        '''IN : Prix TTC avec une TVA à 20%
           OUT : prix HT correspondant'''
        return ...
```

Exercice 8. Hausse et Baisse de x %

1. Écrire un algorithme avec une fonction de paramètres p et t qui calcul le prix final, après une évolution de t% d'un prix initial de p euros.

```
def f(p,t):
        '''IN : p prix initial,
            t correspond à une évolution de t%
          OUT : prix après evolution de t%'''
        return ...
```

Remarque

Les fonction mathématiques de bases sont listées sur la fiche 1 Python

Compléments

Les instructions *input* et *print* ne sont plus à mettre en avant. Il est cependant très utile pour comprendre un programme de faire des affichages de valeurs, parfois même dans une boucle afin de comprendre comment les variables sont modifiées.

Exercice 9. (Optionnel) Affichage avec print()

1. Écrire et exécuter le programme suivant :

```
# Dans l'éditeur PYTHON
# Méthode 1 : on place le texte entre ""
#             et les variables séparées par des virgules
var1=2
var2=-9
print("Méthode 1 : var1 =", var1," et var2 =", var2)

# Méthode 2 :  f (format) devant une chaine de charactères

print(f"Méthode 2 : var1 = {var1} et var2 =  {var2}")
```

On remarque vite l'intérêt de la deuxième écriture.

```
# Dans la console PYTHON
Méthode 1 : var1 = 2  et var2 = -9
Méthode 2 : var1 = 2 et var2 =  -9
```

2. Quelques compléments.

Print("Texte",var)

- *print()* affiche la valeur numérique ou le texte qui suit.
- *print(a, b)* affiche à la suite sans passer à la ligne les éléments a et b.
- *print("Texte",var)* affiche le texte suivi de la valeur de la variable *var*.
- *print(a,end="")* affiche l'élément a et ne passe pas à la ligne : le prochain affichage continuera sur cette ligne.
- \n : permet de passer à la ligne dans l'affichage d'une même instruction print().
- \ : si une ligne de script est trop longue pour notre écran, on peut la casser avec un \ sans que Python interprète cela comme un saut de ligne.

Exercice 10. (Optionnel) Input() ou float(input()

nom=input("question") et a=float(input("question"))

- *var=input("question")* :
  La fonction *input*, dont la syntaxe est : *nom=input(question)* affiche une fenêtre où figure le texte *question* et un cadre blanc dans lequel on entrera ce qui est demandé.
  La réponse est alors affectée à la variable *var* qui est alors considérée comme une chaine de caractère (un mot).
- *a=float(input("question"))* : si la valeur de a demandée est considérée comme un nombre flottant,
- *a=int(input("question"))* : pour un entier.

Par exemple :

```
# Dans l'éditeur PYTHON
nom=input("Donnez votre nom svp ")
age=int (input("Donnez votre age svp ")    )
quizz=float (input("Diviser 1 par 4 svp ")    )
print(f"Bonjour {nom}, vous avez {age} ans et avez dit que 1/4 donnait {quizz}")
```

```
# Dans la console PYTHON on obtient :
Donnez votre nom svp Evariste
Donnez votre age svp 31
Diviser 1 par 4 svp 0.25
Bonjour Evariste, vous avez 31 ans et avez dit que 1/4 donnait 0.25
```

Deuxième partie

Instructions conditionnelles

Exercice 11. Si ... alors... sinon

if *test*

effectue (une fois) les instructions indentées qui suivent lorsque le test est vérifié.

Indentation pour les blocs

- Dans le langage Python, on peut passer des lignes pour plus de clarté, ce qui n'est pas pris en compte lors de l'exécution du programme. Par contre, vous ne pouvez pas ajouter un espace en début de ligne comme bon vous semble, car cela a une signification.
- Indentation : On appelle *indentation* ce décalage d'un cran d'une ou de plusieurs lignes d'un programme. Elle permet de délimiter un bloc d'instructions dans une boucle ou lors d'une exécution conditionnelle.
- Deux points " :" La ligne précédent l'indentation se finit toujours par deux points. Quand vous appuyez sur la touche après avoir tapé « : », l'indentation est automatiquement effectuée en même temps que le passage à la ligne.
- On peut aussi appuyer sur la touche de tabulation *Tab*, à gauche de la touche "A" pour gérer l'indentation, mais les deux points : sont toujours nécessaires.

1. Le programme ci-dessous cherche à résoudre l'équation : $a\,x = b$.

```
def soleq(a,b):
    '''IN : a et b décimaux
      OUT : solution éventuelle de l'équation ax=b'''
    if a!=0: # si a est différent de 0
        return b/a
```

Exécutez-le dans la console de droite avec deux valeurs de votre choix. Il permet la résolution de l'équation $ax = b$ avec un test pour savoir si a est bien différent de zéro.
La condition « a différent de zéro » s'écrit « *a!=0* » . Aucun affichage n'est proposé ici.

2. Si on entre *soleq(0,2)* observer le résultat produit. Pour compléter, on va utiliser la structure « *if ... else* »

if test : ... else : ...

if test : ... else : ... : Effectue les instructions indentées lorsque le test est vérifié, sinon effectue les instructions alternatives indentées.
Remarques :

- *Le "else" est aligné avec le "if" qui lui correspond.*
- *Taper " :" puis appuyer sur la touche "entrée" pour passer à la ligne, provoque l'indentation automatique.*

```
def soleq(a,b):
    '''IN : ...
      OUT : ...'''
    if a!=0: # si a est différent de 0
        return b/a
    else:
        return "pas de solution (ou une infinité)"
```

3. On cherche maintenant à résoudre une équation de type $ax+b=c$.

3. a. Commencer par résoudre, à la main, l'équation $2x+3=4$.

3. b. Proposer une fonction nommée soleq2(a,b,c) renvoyant la solution de cette équation.

3. c. Vérifier votre programme avec l'équation de la question **3. a.** .

Les Conditions du test

- **a!=0** : La condition « *a* différent de zéro » s'écrit « *a!=0* ».
- **a==0** : La condition « *a* égal à zéro » s'écrit « *a==0* », (avec deux fois le symbole =).
- **< et >** : Ces symboles désignent les inégalités strictes habituelles.
- **<= et >=** : Ces combinaisons de symboles désignent les inégalités larges ≤ et ≥ habituelles.
- **and et or**
  - **and** : Permet d'effectuer une instruction si deux tests sont vérifiés simultanément.
  - **or** : Permet d'effectuer une instruction si au moins un test sur deux est vérifié.

Exercice 12. Si ... alors ... sinon ...

Voici un programme de calcul :

- Choisir un nombre entier ;
- Si il est pair, le diviser par 2 ;
- sinon le multiplier par 3 et ajouter 1.
- Afficher le résultat.

1. Appliquer ce programme de calcul à 8 et à 11 (à la main).

2. Écrire un algorithme correspondant à ce programme de calcul. On pourra compléter le programme ci-dessous :

Aide

a % b : Renvoie le reste de la division euclidienne de *a* par *b*.
Par exemple $12=2\times5+2$ *donc le reste de la division euclidienne de 12 par 5 est 2. Donc :*

```
12%5
 => 2
```

```
def g(n):
    '''IN : ...
       OUT : ...'''
    if ...
        return ...
    else:
        return ...
```

3. Que se passe-t-il lorsqu'on entre un decimal ? Pourquoi ?

4. Élaborer une stratégie pour éviter cette erreur et modifier le programme en conséquence.

Aide

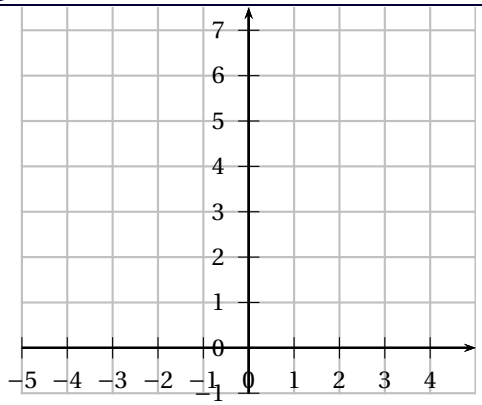
if n==int(n) : teste si la variable *n* est égale à sa partie entière, et donc si *n* est entier.

Exercice 13. Une fonction définie par morceaux et Géogébra

1. Écrire un algorithme utilisant une fonction comme dans l'exercice 12 qui permet de calculer l'image d'une valeur demandée, par la fonction f définie sur $\mathbb{R}$ par :

$$f : x \longmapsto f(x) = \begin{cases} -2x & \text{si } x < 0 \\ x^2 & \text{si } x \geq 0 \end{cases}$$

2. Compléter alors la tableau de valeurs suivant et tracer $\mathscr{C}_f$ dans le repère ci-contre.



x	-3	-1	0	$0,5$	1	$1,5$	2	$2,5$
$f(x)$								

3. Vérifier votre graphique à l'aide du logiciel géogébra en utilisant l'instruction *Si(condition,expression,sinon condition,expression)*

Aide

Des exemples sur la page d'aide de Geogebra https://wiki.geogebra.org/fr/Fonctions.
Exemple :

$$f(x) = Si(x > 5\ ,\ x+1\ ,\ 0 < x <= 5\ ,\ x \wedge 3\ ,\ x <= 0\ ,\ 1-5x)$$

définit la fonction $f : x \longmapsto f(x) = \begin{cases} x+1 & : \ x > 5 \\ x^3 & : \ 0 < x \leq 5 \\ 1-5x & : \ x \leq 0 \end{cases}$

Exercice 14. Une autre fonction définie par morceaux

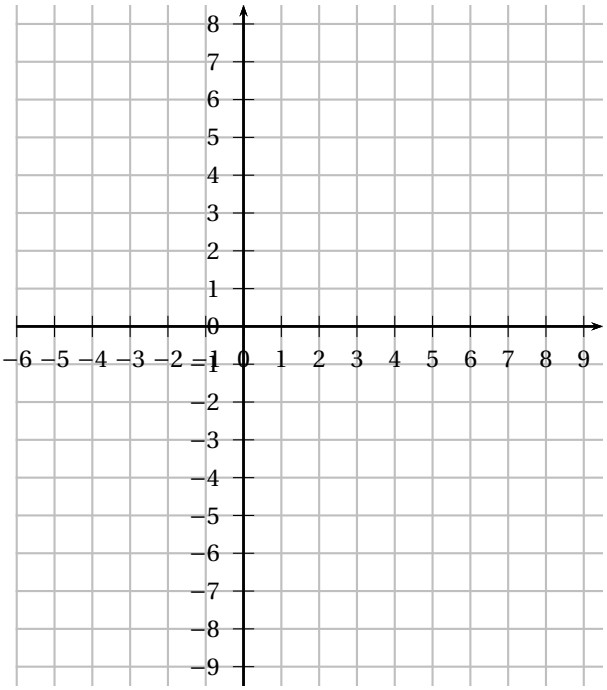
Aide

Indication : Utilisez le test : *condition 1* ***and*** *condition 2*

1. Écrire un algorithme utilisant une fonction comme dans l'exercice 12 qui permet de calculer l'image d'une valeur demandée, par la fonction f définie sur $\mathbb{R}$ par :

$$g : x \longmapsto g(x) = \begin{cases} 2x+1 & \text{si } x < 0 \\ x^2 - 1 & \text{si } 0 \leq x < 3 \\ 11 - x & \text{si } x \geq 3 \end{cases}$$

2. Compléter alors les tableaux de valeurs suivants et tracer $\mathscr{C}_g$ dans le repère ci-contre. Attention, bien identifier les fonctions de référence (fonctions affines, fonctions polynôme du second degré ...)



x	-5	-3	-2	-1	$-0,5$	-0.1	0
$g(x)$							

x	0.5	1	2	3	4	5	6
$g(x)$							

3. Vérifier votre graphique à l'aide du logiciel géogébra en utilisant l'instruction *Si[condition,fonction]*

Exercice 15. if test1 : instructions1 elif test2 : instructions2

if test1 : instructions1 elif test2 : instructions2

if test1 : instructions1 elif test2 : instructions2 : Effectue les instructions1 indentées lorsque le test1 est vérifié, sinon effectue le test2 et, si celui-ci est vérifié, effectue les instructions2 indentées.
Remarques :

- *On peut enchaîner autant de "elif" que nécessaire.*
- *On peut terminer une série de "elif" par un "else" afin d'être sûr de traiter tous les cas.*

1. On considère le programme ci-dessous qui calcule l'image d'un nombre x par une fonction h définie par morceaux sur $\mathbb{R}$. Écrire l'expression de cette fonction en fonction des valeurs de x.

$$h : x \longmapsto h(x) = \begin{cases} \cdots\cdots\cdots\cdots & \text{si } \cdots\cdots \\ \cdots\cdots\cdots\cdots & \text{si } \cdots\cdots \\ \cdots\cdots\cdots\cdots & \text{si } \cdots\cdots \end{cases}$$

```
def h(x):
    '''IN : ...
       OUT : ...'''
    if x<0:
        return 2*x+3
    elif x<2:
        return 3-x
    else:
        return x**2-3
```

2. Exécuter ce programme (dans la console) pour différentes valeurs de x puis tracer la courbe représentative de cette fonction sur l'intervalle $[-5;5]$ dans un repère de votre choix. *Vous pourrez utiliser le logiciel Geogébra.*

Exercice 16. Des photocopies

Un magasin de reprographie propose un tarif dégressif. Les 20 premières photocopies sont facturées à 10 centimes et les suivantes à 8 centimes.

1. Calculer à la main le coût de 15 puis de 30 photocopies.
2. Écrire un algorithme utilisant une fonction qui renvoie le montant de la facture en euros pour un nombre de photocopies donné.
3. Écrire l'expression de cette fonction en fonction des valeurs de x.

$$f : n \longmapsto f(n) = \begin{cases} \cdots\cdots\cdots\cdots & \text{si } \cdots\cdots \\ \cdots\cdots\cdots\cdots & \text{si } \cdots\cdots \end{cases}$$

Exercice 17. Une cuve

On s'intéresse au volume d'eau contenu dans une cuve parallélépipédique de base rectangulaire (2m par 3m) et de 80 cm de hauteur. Écrire un programme utilisant une fonction qui calcule le volume d'eau en fonction de la hauteur.

Aide

Pensez aux cas ou la valeur de la hauteur h d'eau est supérieure à la hauteur de la cuve, voir au cas où h est négatif!

Troisième partie

Structures itératives

Comme dans la plupart des langages, il existe en Python principalement deux manières de réaliser une boucle, c'est à dire une répétition d'un bloc d'instructions. Comme pour l'instruction « *if* » , la partie à répéter sera indentée vers la droite, ce qui permet en plus une bonne visibilité de l'algorithme.

1. Boucle dont on connait le nombre d'itérations

range(début , fin , pas)

range(début , fin , pas) : Génère une liste d'entiers les paramètres *début* et *pas* sont optionnels.

- Dans l'intervalle [0 , fin[si un seul paramètre est renseigné.
  L = range(4) va créer la liste [0 , 1 , 2 , 3] de 4 termes, le premier sera L[0]=0, le dernier L[3] = 3.
- Dans l'intervalle [début ; fin[si 2 paramètres sont renseignés.
  L = range(1 , 5) va créer la liste [1 , 2 , 3 , 4] le premier terme sera L[0] = 1 et le dernier L[3] = 4.
- Dans l'intervalle [début ; fin[mais de *pas* en *pas*, si les 3 paramètres sont renseignés.
  L = range(2 , 9 , 2) va créer la liste [2 , 4 , 6 , 8] .

for *var* in *list* L

***for** var **in** L* : Réalise une boucle en faisant parcourir à la variable *var* toute la liste *L*.

Exercice 18. Boucle dans des listes

1. On considère le programme suivant. Exécutez-le en écrivant simplement ex1() dans la console. Modifiez-le pour qu'il affiche « *Morning!* »

```
def ex1():
    liste = ['B' , 'o' , 'n' , 'j' , 'o' , 'u' , 'r' , '!']
    for i in liste :
        print (i , end="")
```

Remarque

end = "" permet de ne pas renvoyer à la ligne automatiquement après une instruction print().

2. On considère le programme suivant. Exécutez-le en écrivant simplement ex2() dans la console. Modifiez-le pour qu'il affiche les entiers de 0 à 15 .

```
def ex2():
    for n in range(10) :
        print (n)
```

3. On considère le programme suivant. Exécutez-le en écrivant simplement ex3() dans la console. Modifiez-le pour qu'il affiche les entiers de 5 à 10 .

```
def ex3():
    for n in range(7,13) :
        print (n)
```

4. Modifiez le programme suivant pour qu'il affiche les entiers impairs de 5 à 17 puis créez une fonction ex5(A,B) qui renvoie la liste des entiers de A à B de deux en deux.

```
def ex4():
    Liste = [ i for i in range( 10 , 20 , 2 ) ]
    return Liste
```

Exercice 19. Somme des entiers

1. Calculer à la mains la somme $S(10)$ des entiers de 0 à 10 puis la somme $S(15)$ des entiers de 0 à 15.

$$S(10) = 0+1+\cdots+10 = \cdots \quad \text{et} \quad S(15) = 0+1+\cdots+15 = \cdots$$

2. On cherche une fonction qui renvoie la somme des entiers de 0 à n, où n est le paramètre. Compléter le programme et vérifier que la valeur en sortie est correcte pour plusieurs valeurs de n.

```
def somme(n):
    '''IN : entier n >=0
      OUT : somme des entiers de 0 à n'''
    s = 0 # on initialise s à 0
    for i in range (...):
        s= ...
    return s
```

3. Il est souvent utile de compléter un tableau avec les valeurs des variables pour chaque itération . Faites-le ici pour :

A compléter sur cette feuille

- $n = 10$;

i	X	0	1	2	3	4	5	6	7	8	9	10
s	0											

- et $n = 15$.

i	X	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s	0																

4. Somme des impairs.

4. a. Calculer la somme des entiers impairs inférieurs à 10 :

$$I(10) = 1+3+5+7+9 = \cdots$$

4. b. Écrire un programme qui renvoie la somme des entiers impairs de 1 à n entier , où $n > 0$.

Exercice 20. Somme de carrés

1. Calculer à la mains la somme C_1 des carrés entiers de 0 à 5 puis la somme C_2 des carrés des entiers de 0 à 10.

$$C(5) = 0^2 + 1^2 + 2^2 + \cdots + 5^2 = \cdots \quad \text{et} \quad C(10) = 0^2 + 1^2 + 2^2 + \cdots + 10^2 = \cdots$$

2. On cherche une fonction qui renvoie la somme des carrés entiers de 0 à n, où n est le paramètre. Compléter le programme et vérifier que la valeur en sortie est correcte pour plusieurs valeurs de n.

```
def sommecarres(n):
    '''IN : entier n >=0
      OUT : somme des carrés des entiers de 0 à n'''
    s = 0 # on initialise s à 0
    for i in range (...):
            s= ...
    return s
```

3. Il est souvent utile de compléter un tableau avec les valeurs des variables pour chaque itération . Faites-le ici pour :

A compléter sur cette feuille

- $n = 10$.

i	X	0	1	2	3	4	5	6	7	8	9	10
s	0											

Exercice 21. Somme des inverses des carrés

1. Calculer la somme des inverses des carrés de 1 à 4 :

$$I(4) = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} = \cdots$$

2. Écrire un programme qui calcule la somme des inverses des carrés de 1 à n. Tester votre programme avec le calcul précédent.

3. Calculer des valeurs pour n très grand et conjecturer la limite de cette somme.

   Comparer votre résultat au nombre $\frac{\pi^2}{6} \approx 1,64493416$.

   Pour calculer une valeur approchée de ce nombre, n'oubliez-pas d'importer le module math.

Remarque historique

C'est le génial mathématicien suisse Leonhard Euler (1707-1783) qui démontre le premier que cette somme converge (on parle de série convergente). Il prouve ce merveilleux résultat :

$$\frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots = \frac{\pi^2}{6}$$

En 1731 Euler trouve 6 décimales, c'est déjà un exploit car la série converge très lentement (avec 1000 termes, on n'obtient que 2 décimales). En 1733 il calcule 20 décimales!

Question Bonus

Écrire une fonction **f(n)** qui donne le nombre de termes de cette somme nécessaires à une approximation à 10^{-n} de la limite $\frac{\pi^2}{6}$ ainsi que l'approximation obtenue .

Exercice 22. Moyenne d'une liste de valeurs

Une liste : L

Une liste est une suite d'éléments numérotés de même type dont le premier indice est 0. En Python, une liste s'écrit entre crochets [... , ..., ..., ...] avec les éléments séparés par des virgules.

- Le premier élément de la liste est L[0], le 2e est L[1], ...
- Une liste peut être écrite de manière explicite : L = ["Lundi", "Mardi","Mercredi"]
- Sa longueur est donnée par $len(L)$.
- Si les éléments de la liste sont comparables, le max. est donné par *max(L)*, le min. par *min(L)*
- L=[] permet de définir une liste vide.
- Si L est une liste, l'instruction *L.append(x)* va ajouter l'élémentt [x] à la liste L.

1. Définissez deux listes de nombres L1 et L2 ainsi :

```
L1 =  [-1,3,5,7,10]  # Liste 1 pour les exemples
L2 =  [5,12,15,7,10,19]  # Liste 2 pour
```

2. Calculer la moyenne de ces deux séries de valeurs.

3. On veut écrire une fonction qui renvoie la la moyenne des termes d'une liste. Compléter-le et tester-le sur L1 et L2 .

```
def moyenne(L):
    '''IN : L une liste de nombres
      OUT : la moyenne des nombres de la liste'''
    s = 0 # on initialise s à 0
    for x in L:
            s= ...
    return s
```

4. Modifier-le pour que la fonction renvoie aussi la longueur de L, la valeur maximale et la minimale de la liste.

max(L) et min(L) :

max(L) et min(L) renvoie les valeurs maximale et minimale des éléments de la liste L.

Question Bonus

4. a. Écrire une fonction **maxi(liste)** qui donne le maximum de la liste, sans utiliser la fonction max.

4. b. Écrire une fonction **mini(liste)** qui donne le minimum de la liste, sans utiliser la fonction min.

2. Boucle dont on ne connait pas le nombre d'itérations

Dans la pratique, on ne connaît que rarement le nombre d'itérations pour arriver au résultat (d'où l'intérêt d'un programme). On peut alors utiliser des boucles de type TANT QUE .... FAIRE : ...

while condition :

***while** condition* : Exécute une instruction ou un bloc d'instructions tant que la condition est vérifiée. (La boucle peut donc ne jamais être exécutée si, d'entrée la condition n'est pas remplie).

Exercice 23. La population mondiale

En 2018 la population mondiale est estimée à 7 577 millions (environ 7,6 milliards) . Le taux annuel de la croissance démographique de la population mondiale est d'environ 1,2 %.

Aide

Un milliard se note : $1\,000\,000\,000 = 10^9$, et s'écrit sous Python : $10 * *9$.
Dix milliards se note : $10 \times 10^9 = 10^{10}$, et s'écrit sous Python : $10 * 10 * *9$ ou $10 * *10$.

1. Le programme suivant cherche à déterminer en quelle année, si cette évolution se poursuit, la population mondiale dépassera 10 milliards et quelle sera cette population . Compléter puis exécuter cet algorithme dans la console afin d'obtenir la réponse cherchée .

```
def f(seuil):
    '''IN : le seuil
      OUT : l'année et la population qui
      dépasse le seuil'''
    population=7 577 000 000
    annee=2018
    while .... :
            annee=...
            population=...
    return (annee,population)
```

Pseudo Code

Fonction f(seuil)
  population,annee ← 7577000000, 2018
  Tant que ····· Faire
    annee ← ·····
    population ← ·····
  Fin Tant que
  Renvoyer (annee , population)

2. On cherche un affichage différent. Compléter le programme ci-dessous et lancer-le pour déterminer quand la population mondiale dépassera les 20 milliards et qu'elle sera cette population.

```
(a,b)=f(...)
print("La population sera de ",....," milliards en  ",...)
```

3. Modifier l'algorithme afin de renvoyer une valeurs arrondie au centième de la population, exprimée en milliards.

round(*b* , n)

round(b , n) va renvoyer l'arrondie de b à 10^{-n} près.
Par exemple : round(2.2563 , 2) => 2.26

3. Applications

Exercice 24. De la somme au produit

Voici un programme de calcul écrit en pseudo-code :

Initialisation :	$S \leftarrow 0$
Traitement :	Pour k de 1 jusque n Faire
	$\quad S \leftarrow S + k$
	Fin de boucle.

1. Écrire cet algorithme en Python en utilisant une fonction (du nom de votre choix) de paramètre n.
2. Faire tourner le programme ci-dessus pour $n = 4$, puis $n = 5$.

A compléter sur cette feuille

- et $n = 4$.

k	X	1	2	3	4
S	0				

- et $n = 5$.

k	X	1	2	3	4	5
S	0					

3. Que calcule-t-il?
4. Écrire une fonction qui calcule, pour un entier n ($n \geq 1$), le produit :

$$1 \times 2 \times 3 \times \ldots \times n$$

5. **La fonction factorielle**.
En fait, il existe une fonction mathématique correspondant à ce produit, qui se nomme factorielle et se note pour tout entier n, avec un point d'exclamation : $n!$.
Le produit précédent ($1 \times 2 \times 3 \times \ldots \times n$) n'est licite que pour n entier strictement positif mais on convient que factorielle de 0 vaut 1 soit $0! = 1$.
Cette fonction est introduite par un mathématicien alsacien, Christian KRAMP (1760-1826) en 1808 dans *Éléments d'arithmétique universelle*.
Pour tout entier naturel n on a :

$$\begin{cases} n! = 1 \times 2 \times 3 \times \ldots \times n \quad (\text{si } n \geq 1) \\ 0! = 1 \end{cases}$$

   5. a. Écrire en python une fonction qui renvoie la valeur $n!$ pour n entier naturel.

   5. b. Avec le module math.

Aide

Factorielle : $n!$ = $1 \times 2 \times 3 \times \ldots \times n$.
Ce produit ce nomme factoriel n et se note $n!$.
Avec le *module math*, il existe un fonction en python qui le calcul directement : *math.factorial(n)*.
Pour l'appeler, il faut charger le *module math* au début du programme (ligne 1),
la syntaxe est *import math*.

Chargez le module math en ligne 1 en écrivant *import math* puis vérifiez que *math.factorial(n)* donne bien le même résultat que votre fonction pour quelques valeurs de n.

Fin du devoir