

Programmation en python

Premier pas en python

DWFS 1

Fatiha BENDAIDA

2024/2025

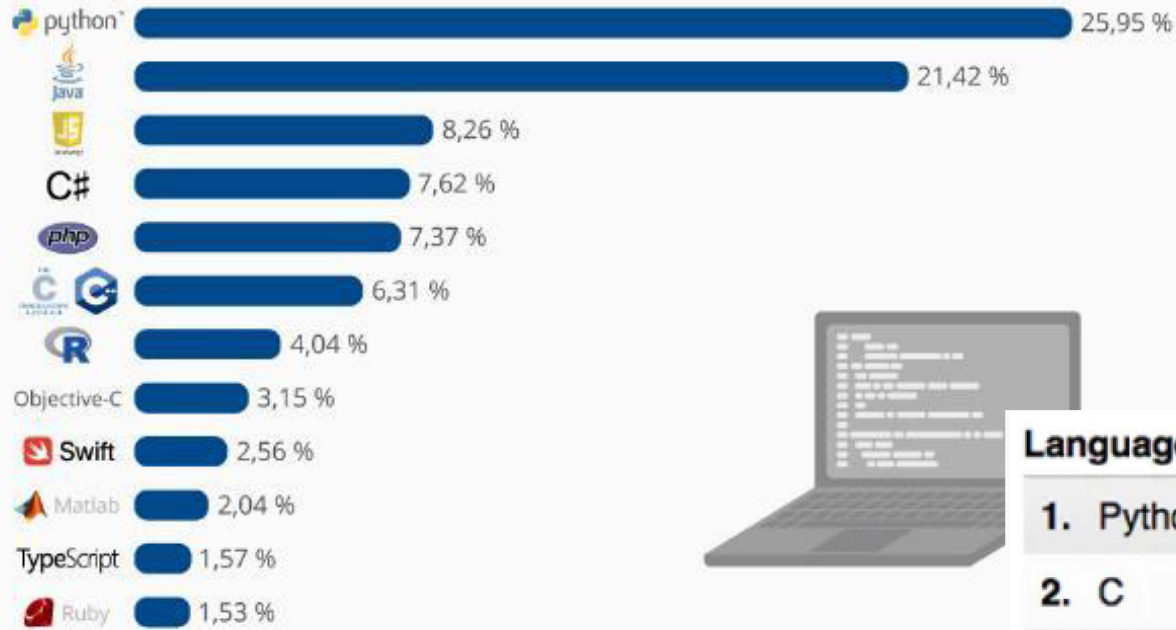


A propos?

- Pour expliquer l'origine du langage, revenons un peu en arrière. En 1989, par une froide nuit néerlandaise, un développeur du plat pays nommé *Guido van Rossum* s'ennuie. Il cherche un moyen de s'occuper pendant la période qui précède Noël car les bureaux de son entreprise sont fermés. Quand certains auraient décoré un sapin, lui se lance dans l'invention d'un langage. Etant un grand fan des *Monty Python* et d'humeur irrévérencieuse, il l'appelle Python. Voilà pourquoi les développeurs Python *ont de l'humour* et s'amuse à glisser des petites blagues dans leur code !























Les langages de programmation les plus populaires

Part des langages de programmation les plus populaires dans le monde selon le PYPL-Index *



* Basé sur une analyse de la fréquence de recherches Google des tutoriels de langage de programmation.
@Statista_FR Source : PYPL

Pourquoi Python?

Language Rank	Types	Spectrum Ranking
1. Python	 	100.0
2. C	  	99.7
3. Java	  	99.5
4. C++	  	97.1
5. C#	  	87.7
6. R		87.7
7. JavaScript	 	85.6
8. PHP		81.2
9. Go	 	75.1
10. Swift	 	73.7



Pourquoi apprendre python, le langage de programmation du futur

- Python est un langage de programmation Open Source, orienté objet, de haut niveau. Il s'agit d'un langage généraliste. Cela signifie qu'il peut être utilisé pour développer à peu près n'importe quoi, grâce à de nombreux outils et bibliothèques.
- Ce langage est particulièrement populaire pour l'analyse de données et l'intelligence artificielle, mais aussi pour le développement web backend et le computing scientifique.
- Python est aussi utilisé pour développer des outils de productivité, des jeux ou des applications. Des dizaines de milliers de sites web ont été développés avec ce langage, au même titre que plusieurs applications très connues comme *Dropbox*, *Netflix* ou *Spotify*....



Les différentes versions

- Il existe 2 versions de Python : 2.x et 3.x.
- Python 3.x n'est pas une simple amélioration ou extension de Python 2.x.
- Tant que les auteurs de bibliothèques n'auront pas effectué la migration, les deux versions devront coexister.
- Nous nous intéresserons uniquement à Python 3.x.

Python vs C: know what are the differences

Metrics	Python	C
<i>Introduction</i>	Python is an interpreted, high-level, general-purpose programming language.	C is a general-purpose, procedural computer programming language.
<i>Speed</i>	Interpreted programs execute slower as compared to compiled programs.	Compiled programs execute faster as compared to interpreted programs.
<i>Usage</i>	It is easier to write a code in Python as the number of lines is less comparatively.	Program syntax is harder than Python.
<i>Declaration of variables</i>	There is no need to declare the type of variable. Variables are untyped in Python. A given variable can be stuck on values of different types at different times during the program execution	In C, the type of a variable must be declared when it is created, and only values of that type must be assigned to it.

Python vs C: know what are the differences

<i>Error Debugging</i>	Error debugging is simple. This means it takes only one instruction at a time and compiles and executes simultaneously. Errors are shown instantly and the execution is stopped, at that instruction.	In C, error debugging is difficult as it is a compiler dependent language. This means that it takes the entire source code, compiles it and then shows all the errors.
<i>Function renaming mechanism</i>	Supports function renaming mechanism i.e, the same function can be used by two different names.	C does not support function renaming mechanism. This means the same function cannot be used by two different names.
<i>Complexity</i>	Syntax of Python programs is easy to learn, write and read.	The syntax of a C program is harder than Python.
<i>Memory-management</i>	Python uses an automatic garbage collector for memory management.	In C, the Programmer has to do memory management on their own.

Python vs C: know what are the differences



Example of a C Program –

```
1  #include <stdio.h>
2  int main()
3  {
4      // printf() displays the string inside quotation
5      printf("Hello, World!");
6      return 0;
7  }
```

Example of a Python Program –

```
1  | print("Hello, World!")
```


Python vs C: know what are the differences



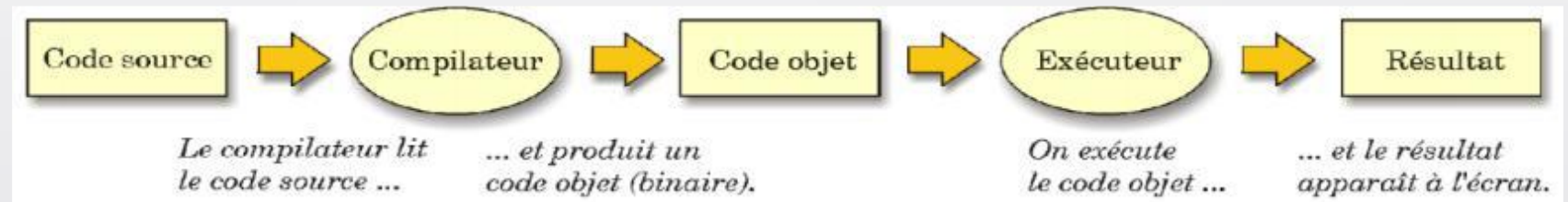
```
#include<stdio.h>
int main(){
    int year=2025;
    printf("Hello World!\n");
    printf("Welcome DWFS %d.\n",year);
    return 0;
}
```

```
Year=2025
print("Hello world!")
print("Welcome DWFS",Year)
```

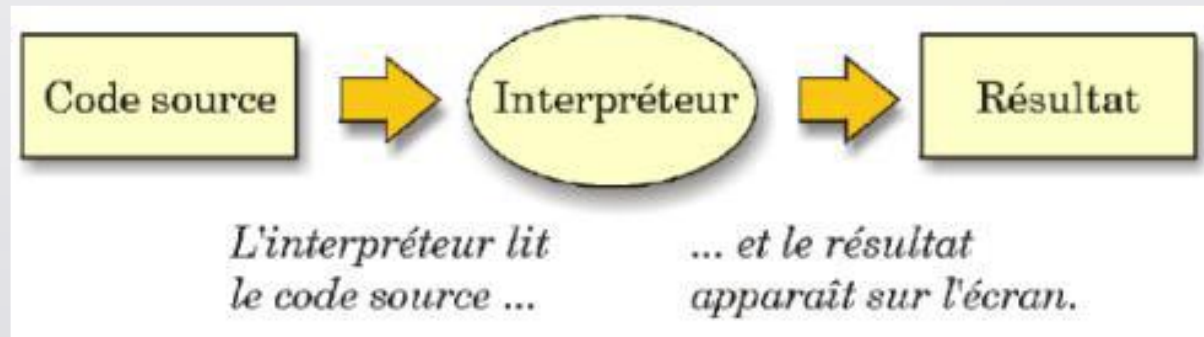
Comment faire fonctionner mon code source ?

- Il existe 2 techniques principales pour effectuer la traduction en langage machine de mon code source :

- Compilation :



- Interprétation:



Python est un langage Interprété



Avantages : interpréteur permettant de tester n'importe quel petit bout de code,...

- **Inconvénients :** peut être lent.

Comment lancer un script python

- Il y a différentes méthodes pour lancer un script Python
 - Ecrire le code dans un fichier “script.py”. Et taper dans la console la commande “*python script.py*”
 - Coder directement dans l’interpréteur(console) : instruction par instruction, un peu à la façon d’une calculatrice.
- Utiliser un IDE (eg. *Pyzo, PyCharm, Spyder, visual studio code*,...)



Utiliser *Anaconda*(choix recommandé)

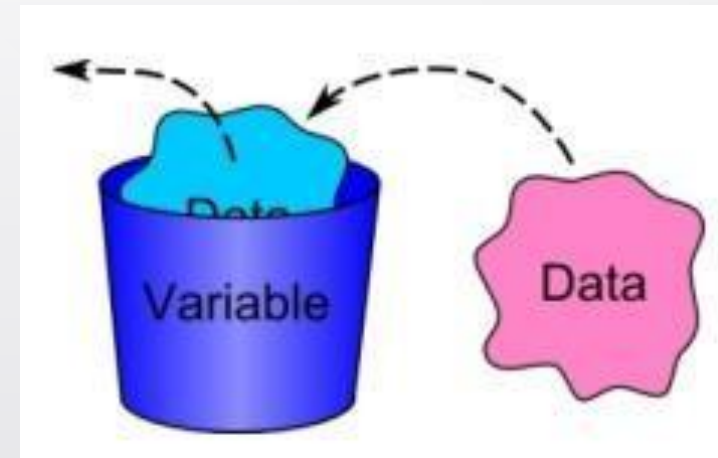




Introduction

Variable

- Conteneur d'information
- Identifié par un nom = un **identificateur**
- Ayant un « contenu »



- **Suite non vide de caractères**

- commençant par une lettre ou le caractère _
- contenant seulement des lettres, des chiffres et/ou le caractère _
- Ne peut pas être un mot réservé de Python

Identificateur en Python

➤ Exemples :

- **valides** : toto, proch_val, max1, MA_VALEUR, r2d2, bb8, _mavar
- **non valides** : 2be, C-3PO, ma var

➤ Les identificateurs sont **sensibles à la casse** :

ma_var != Ma_Var

➤ **Conventions** pour les variables en Python :

- utiliser des minuscules
- pas d'accents

Affectation

- Pour mémoriser une valeur dans une variable, on fait une affectation en utilisant le signe =

- **Exemples**

```
n = 33
```

```
a = 42 + 25
```

```
ch = "bonjour"
```

```
euro = 6.55957
```

- La première affectation d'une variable est aussi appelée **initialisation**
- En Python, le typage est **dynamique**
 - Pour connaître le type d'une variable: `type(ma_var)`

Example

```
a = 20
```

```
>>> type(a)
```

```
<class 'int'>
```

```
a = "salut"
```

```
>>> type(a)
```

```
<class 'str'>
```

```
a = 3.14
```

```
>>> type(a)
```

```
<class 'float'>
```

```
>>> type{21==7*3)
```

```
<class 'bool'>
```

Affectation vs condition en Python

- Le signe “=” sert seulement à faire une affectation.
- Pour tester l'égalité, on utilise “==”

Exemples :

```
>>> a = 6
>>> a
6
>>> b = 9
>>> a == b
False
```

Expressions

- C'est une formule qui peut être évaluée

- **Exemples :**

```
42 + 2 * 5.3  
3*2.0 - 5  
"bonjour"  
20 / 3
```

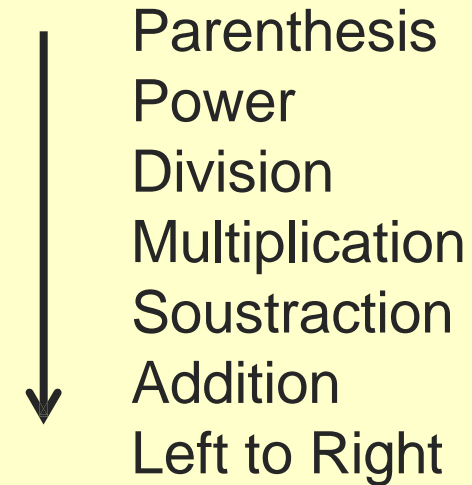
- Expression : des **opérandes** et des **opérateurs**.
- Les opérateurs que l'on peut utiliser **dépendent du type** des valeurs qu'on manipule
- Une expression qui ne peut prendre que les valeurs **True** ou **False** est appelée **expression booléenne**

Quelque Operateurs

➤ **Arithmétiques** (sur des nombres) :
+, -, *, **, /, %, //

➤ **De comparaison** (résultat booléen) :
==, !=, <, >, <=, >=

➤ **Logiques** (entre des booléens, résultat booléen) :
or, and, not



↓
Parenthesis
Power
Division
Multiplication
Soustraction
Addition
Left to Right

Exercice

- Quelle est la réponse de l'interpréteur après chaque expression ?

```
>>> 2 + 3
```

5

```
>>> 2*3
```

6

```
>>> 2**3
```

8

```
>>> 20/3
```

6.666666666666667

```
>>> 20//3
```

6

```
>>> 20%3
```

2

```
>>> 2 > 8
```

False

```
>>> (2 <= 8) and (8 < 15)
```

True

```
>>> 2 <= 8 < 15
```

True

```
>>> (x % 2 == 0) or (x >= 0)
```

NameError: name 'x' is not defined



Commentaires

En Python, une ligne d'instructions qui contient le symbole `#` (*dièse*) signifie un commentaire. Le reste de la ligne sera ignoré quand le programme sera exécuté.

```
# a=2 voici un commentaire b=5 # b=2  
print(b) # un autre commentaire @!#!@$
```

- Pour les commentaires sur plusieurs lignes en [python](#), nous aurons besoin d'utiliser le symbole (`'`) ou (`"`) **trois fois** au début et à la fin.

```
s = 1+2 '''  
print("la somme est = ")  
print () ''' print(s+1)
```



Entrées/Sorties

- On a généralement besoin de pouvoir **interagir** avec un programme :
- Pour lui fournir les données à traiter, par exemple au clavier : **entrées**
- Pour pouvoir connaître le résultat d'exécution ou pour que le programme puisse écrire ce qu'il attend de l'utilisateur, par exemple, texte écrit à l'écran : **sorties**



Entrées: la fonction input()

- A l'exécution, la fonction **input** :
 - interrompt l'exécution du programme
 - affiche éventuellement un message à l'écran
 - attend que l'utilisateur entre une donnée au clavier et appuie Entrée.
- C'est une saisie en **mode texte**
 - valeur saisie vue comme une **chaîne de caractères**
 - on peut ensuite changer le type

Entrées: la fonction input()

- **Récupérer la donnée ???**

- La fonction **input** : affiche un message (optionnel) à l'utilisateur et récupère la données saisie par l'utilisateur:

Syntaxe : **variable=input("message")**

Exemple

```
# demandez une valeur à l'utilisateur
nom = input(" Donnez votre nom : ")
prenom = input(" Donnez votre prenom : ")

# Affichage du nom complet
print( " Bienvenue :", nom, prenom)
```

Entrées: la fonction input()

- **Récupérer la donnée ???**

- Fonction **input** : retourne une valeur de type texte

- **Attention** : sous python **'3'** est différente de **3**

- Fonction **eval** : évaluer et convertir en une valeur numérique une valeur contenu dans un texte

Exemple :

▪ <code>eval("34.5")</code>	retourne	34.5
▪ <code>eval("345")</code>	retourne	345
▪ <code>eval("3 + 4")</code>	retourne	7
▪ <code>eval("51 + (54 * (3 + 2))")</code>	retourne	321

On peut aussi utiliser les fonctions : **int, float, str,...**

Entrées: la fonction input()

```
>>> texte = input()  
123  
>>> texte + 1 # provoque une erreur  
>>> val = eval(texte)  
>>> val + 1 # ok  
124
```

```
>>> x = float(input("Entrez un nombre :"))  
Entrez un nombre :  
12.3  
>>> x + 2  
14.3
```



Les sorties: la fonction print()

- affiche la **représentation textuelle** de n'importe quel nombre de valeurs fournies entre les parenthèses et séparées par des virgules
- à l'affichage, ces valeurs sont séparées par un **espace**
- l'ensemble se termine par un retour à la ligne
 - modifiable en utilisant les options **sep** et/ou **end**
- Possibilité d'insérer
 - des **sauts de ligne** en utilisant **\n** et
 - des **tabulations** avec **\t**

Les sorties: la fonction print()

```
>>> a = 20
>>> b = 13
>>> print("La somme de", a, "et", b, "vaut",
        a+b, ".")
La somme de 20 et 13 vaut 33.
>>> print(a,b,sep= ";")
20;13
>>> print("a=",a, "b=",b, sep="\n")
a=
20
b=
13
```




Les sorties: la fonction print()

- On a déjà utilisé les chaînes de caractères, notamment dans les fonctions ***print()*** et ***input()***.
- En Python, il existe 3 syntaxes pour les chaînes de caractères :
 - avec des **guillemets** :
`print("toto")`
 - avec des **apostrophes** :
`print('toto')`
 - avec des guillemets **triples** :
`print("""toto""")!`

Les sorties: la fonction print()

- On peut utiliser " dans une chaîne délimitée par ' ... '
- On peut utiliser ' dans une chaîne délimitée par "..."
- On peut utiliser " et ' dans une chaîne délimitée par """"...""""
- """"..."""" permet aussi d'écrire des chaînes de caractères sur plusieurs lignes

Les sorties: la fonction print()

```
>>> print("C'est toto")
C'est toto
>>> print('C'est toto')
SyntaxError : invalid syntax
>>> print("Il a dit "hello" !")
SyntaxError : invalid syntax
>>> print('Il a dit "hello" !')
Il a dit "hello"
>>> print("""C'est toto qui a dit "hello" !""")
C'est toto qui a dit "hello" !
>>> print("""C'est toto qui a dit "hello""""")
SyntaxError : ...
```

Les sorties: la fonction *format()*

La méthode *.format()* permet une meilleure organisation de l'affichage des variables.

```
1 >>> x = 32
2 >>> nom = "John"
3 >>> print("{} a {} ans".format(nom, x))
4 John a 32 ans
```

```
1 >>> x = 32
2 >>> nom = "John"
3 >>> print("{} a {} ans".format(nom, x))
4 John a 32 ans
5 >>> print("{} a {} ans".format(nom, x))
6 32 a John ans
```

```
>>> prop_GC = (4500 + 2575) / 14800
>>> print("La proportion de GC est", prop_GC)
La proportion de GC est 0.4780405405405405
```

```
1 >>> print("La proportion de GC est {:.2f}".format(prop_GC))
2 Le proportion de GC est 0.48
3 >>> print("La proportion de GC est {:.3f}".format(prop_GC))
4 La proportion de GC est 0.478
```

Les sorties: autre façons

- Un peu comme le C

```
1 | >>> x = 32
2 | >>> nom = "John"
3 | >>> print("%s a %d ans" % (nom, x))
```

- **Fstring** : Une syntaxe simple est similaire à celle qu'on a utilisée avec **str.format ()** mais moins verbeuse. Regardez à quel point c'est facile à lire:

```
>>> name = "Eric"
>>> age = 74
>>> f"Hello, {name}. You are {age}."
'Hello, Eric. You are 74.'
```

```
>>> f"{2 * 37}"
'74'
```


Exercices

- **Exercice 1 :**

Ecrire un programme qui permet d'échanger le contenu de deux variables a et b

Exécution du programme

Donnez la valeur de a : 1

Donnez la valeur de b : 5

Avant l'échange a = 1 et b = 5

Après l'échange a = 5 et b = 1

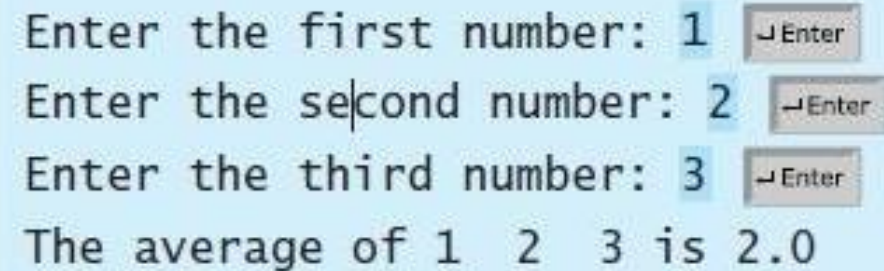
```
a = eval(input(" Donnez le premier nombre : "))
b = eval(input(" Donnez le deuxième nombre : "))
print(" avant l'échange a=",a,"b=",b)
a,b=b,a
print(" après l'échange a=",a,"b=",b)
```

Exercices

- **Exercice 2 :**

Ecrire un programme qui calcule la moyenne de trois valeurs saisies par un utilisateur et affiche le résultat:

voici l'exécution :



```
Enter the first number: 1 ↵ Enter
Enter the second number: 2 ↵ Enter
Enter the third number: 3 ↵ Enter
The average of 1 2 3 is 2.0
```

```
a = eval(input(" Donnez le premier nombre : "))
b = eval(input(" Donnez le deuxième nombre : "))
c = eval(input(" Donnez le troisième nombre : "))
moy=(a+b+c)/3
# Affichage du résultat
print(" la moyenne de ",a,b,"et",c,"est",moy)
```



Exercices

Exercice 3: Ecrire un programme qui permet d'afficher le nombre d'heure, de minutes et de secondes restantes à partir d'un nombre de seconde saisie par l'utilisateur

Exécution du programme

Donnez le nombre des secondes : 500

500 secondes vaut : 0 h 8 min 20 s

```
nbs = eval(input(" Donnez le nombre des secondes : "))
```

```
h=nbs//3600
```

```
m=(nbs%3600)//60
```

```
s=(nbs%3600)%60
```

```
print(nbs,"secondes vaut : ",h,"h",m,"min",s,"s")
```



Expressions booléennes:

Non logique « **not** »

➤ **Not**(expr) vaut vrai si expr est faux et vice versa.

➤ Exemple :

- `not(2 == 1 + 1)` renvoie **False**
- `not(3 == 1 + 1)` renvoie **True**.

Expressions booléennes:

ou logique « **or** »

- `expr1 or expr2` vaut vrai si et seulement si au moins une des deux expressions `expr1` et `expr2` est vraie.
- En Python, le « ou » est **fainéant**, c'est-à-dire que si la 1^{ère} expression vaut **vrai**, la deuxième n'est pas évaluée
 - `(2 == 1 + 1) or (x >= 5)`
- ne provoque pas d'erreur même si **x** n'existe pas, le résultat vaut vrai
 - `(3 == 1 + 1) or (x >= 5)`
- provoque une erreur si **x** n'existe pas.

Expressions booléennes: et logique « **and** »

- `expr1 and expr2` vaut vrai si et seulement si les deux expressions `expr1` et `expr2` sont vraies.
- En Python, le « et » est **fainéant**, c'est-à-dire que si la 1^{ère} expression vaut **faux**, la deuxième n'est pas évaluée
 - `(2 > 8) and (x > = 5)`
 - ne provoque pas d'erreur même si **x** n'existe pas, le résultat vaut faux
 - `(2 < 8) and (x > = 5)`
 - provoque une erreur si **x** n'existe pas.

Lois de Morgan

not(expr1 **or** expr2) = **not**(expr 1) **and** **not**(expr2)

Exemple :

not(a > 2 or b <= 4) équivaut à
(a <= 2) and (b > 4)

not(expr1 **and** expr2) = **not**(expr 1) **or** **not**(expr2)

Exemple :

not(a > 2 and b <= 4) équivaut à
(a <= 2) or (b > 4)

Exemples

Soient x , y , z et t des entiers.

➤ Les variables x , y et z sont identiques

- $x==y$ and $y==z$ $x==y==z$

➤ Les valeurs de x et y sont identiques mais différentes de celle de z .

- $x==y$ and $x!=z$ $x==y!=z$

➤ Les valeurs de x sont strictement comprises entre y et t .

- $y<x$ and $x<t$ $y<x<t$

➤ Parmi les valeurs de x , y et z deux valeurs au moins sont identiques

- $x==y$ or $y==z$ or $x==z$

➤ x et y ont la même parité.

- $(x\%2)==(y\%2)$

➤ Parmi les valeurs de x , y et z deux valeurs au plus sont identiques

- $\text{Not}(x==y==z)$

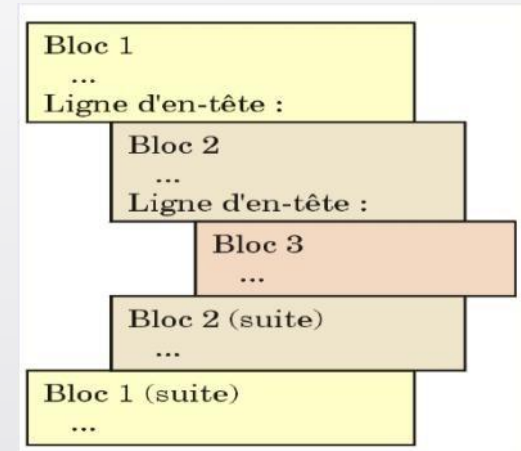


Instructions conditionnelles

If.....elif.....else

Les structures de contrôle IF

- Python n'utilise pas de { } pour entourer les blocs de code pour les if/loops/fonctions, etc. Au lieu de cela, Python utilise les deux points (:) et l'indentation/espace blanc pour regrouper les instructions.



Instructions conditionnelles : if

➤ **Objectif** : effectuer des actions seulement si une certaine condition est vérifiée

• **Syntaxe en Python** :

```
if condition :  
    <instructions à exécuter si vrai >  
    <suite du programme>
```

La condition est une **expression booléenne**

➤ **Attention à l'indentation !**

- Indique dans quel bloc se trouve une instruction.
- **obligatoire** en Python.

Instructions conditionnelles : if

- **Exemple** : calcul de la racine carré d'un nombre positif

```
# demandez une valeur à l'utilisateur
x= eval(input(" Donnez une valeur positive : "))
if (x>= 0):
    racine= x**0.5 # Calcul de la racine carré de x
    print( " la racine carré de    ", x, "est ", racine)
```

Instructions conditionnelles : if...else

➤ **Objectif** : effectuer des actions **différentes** selon qu'une certaine condition est vérifiée ou pas

Syntaxe en Python :

```
if condition :  
    <instructions à exécuter si vrai >  
else :  
    <instructions à exécuter si faux >
```

Attention : le **else** n'est pas suivi d'une condition

Instructions conditionnelles : if...else

- Exemple : Nature d'un nombre

```
x= float(input("Entrez un nombre : "))  
if (x> 0):  
    print( x," est plus grand que 0")  
    print("il est strictement positif")  
else:  
    print(x,"est négatif ou nul")  
print("Fin")
```

Instructions conditionnelles : if...elif....else

Syntaxe générale

```
if <test1>:  
    <blocs d'instructions 1>  
elif <test2>:  
    <blocs d'instructions 2>  
else:  
    <blocs d'instructions 3>
```


Instructions conditionnelles : if...elif....else

➤ **Exemple** : calculer le nombre de racines réelles d'un polynôme du second degré

$$a x^2 + b x + c = 0$$

```
a= float(input(" Donner a : "))  
  
b= float(input(" Donner b : "))  
  
c= float(input(" Donner c : "))  
D=b**2-4*a*c  
if (D> 0):  
    print(" Deux racines réelles distinctes")  
elif (D==0):  
    print("une seule racine reelle")  
else :  
    print("Aucune racine reelle")
```



Exercice 4:

Les habitants de paris paient l'impôt selon les règles suivantes :

- les hommes de plus de 20 ans paient l'impôt
- les femmes paient l'impôt si elles ont entre 18 et 35 ans
- les autres ne paient pas d'impôt

Le programme demandera donc l'âge et le sexe (*M* ou *F*) du parisien, et se prononcera donc ensuite sur le fait que l'habitant est imposable.

Exemple d'exécution :

```
>>> donner l'âge : 25  
>>> donner le sexe : M  
>>> le citoyen est imposable
```

////////////////////////////////////

Exercice 5: Ecrire le programme « Soustraction Quiz » suivant:

❖ **Etape1** : générer deux valeurs entières n1 et n2 à l'aide du module *random.randint(a,b)*

Exemple :

```
import random
```

```
a=random.randint(0,9) # a contient un entier entre 0 et 9
```


❖ **Etape2** : si $n1 < n2$ permutez n1 et n2

❖ **Etape3** : Posez la question à l'utilisateur: « Combien vaut $n1 - n2$? »

❖ **Etape4** : vérifier la réponse de l'utilisateur en affichant un message

```
Combien vaut: 7 - 2 ?  
5  
Bravo
```

```
Combien vaut : 7 - 2?  
1  
Ohhh
```



Exercice 5: solution

Soustraction Quiz

```
import random
n1=random.randint(1,20)
n2=random.randint(2,15)
if n1<n2:
    n1,n2=n2,n1
print("Combien vaut ",n1,"-",n2," ? : ")
rep=eval(input())
if rep ==n1-n2:
    print("Bravo")
else:
    print("Ohhh")
```



Structures Répétitives

Boucle while et for

Boucle while

Syntaxe générale:

```
while <test1>:  
    <blocs d'instructions 1>  
    if <test2>: break  
    if <test3>: continue  
else:  
    <blocs d'instructions 2>
```

- **break** : sort de la boucle sans passer par else,
- **continue** : remonte au début de la boucle,
- **pass** : ne fait rien,
- **else** : lancé si et seulement si la boucle se termine normalement.

Boucle while

Le mot-clé **break**

Exemple

```
i=1
while i<100:
    if i % 2 == 0 :
        print("*")
        break
    i=i+1
    print("Incrementation de i")
print("Fin")
```



Le programme s'arrête
dès que i=2

Affichage :
incrémentations de i
* Fin

Boucle while

Le mot-clé **continue**

Permet de **remonter immédiatement au début** de la boucle `while` en ignorant la suite des instructions dans la boucle.

```
i=1
while i<100:
    if i % 2 == 0 :
        print("*")
        continue
    i=i+1
    print("Incrementation de i")
print("Fin")
```



Le programme ne
s'arrête pas car
toujours `i=2`

Affichage :
incrémentation de i

*

*

...

Boucle while

Exercice 6 : Ecrire un programme qui permet d'afficher la représentation binaire d'un entier strictement positif saisi au clavier.

N.B : Vous devez vérifier la positivité de l'entier

- Donnez un entier positif : -6
- S.v.p Donnez un entier positif : 6
- 6 en binaire 110

$$110 = 1 \times 10^2 + 1 \times 10^1 + 0 \times 10^0$$

```
1  a=int(input("Donner un nombre positif :"))
2  while(a<0):
3      a=int(input("s.v.p donner un nombre positif :"))
4  p=0;x=a;i=0
5  while(a!=0):
6      r=a%2
7      p=p+(10**i)*r
8      i+=1
9      a=a//2
10
11  print(x,'en binaire ',p)
```

Boucle while

Exercice 7 : supposons que vous avez besoin d'écrire un programme permettant de trouver le plus petit diviseur autre que 1 pour un entier n (supposons $n \geq 2$).

- Entrer un entier ≥ 2 : 35
- Le plus petit diviseur autre que 1 pour 35 est 5

```
n = eval(input("Entrer un entier  $\geq 2$ : "))
i = 2
while i <= n:
    if (n % i == 0):
        break
    i += 1
print("Le plus petit diviseur autre que 1 pour ", n, "est ", i)
```




Boucle while

break et continue

❑ Inconvénients:

- Code plus difficile à lire/analyser si plusieurs niveaux d'imbrications et/ou longues instructions dans le while
- N'a pas toujours d'équivalent dans les autres langages de programmation

➔ On essaiera tant que possible de se passer de break et continue.

Boucle while

Boucles imbriquées

- ❑ Une instruction d'une boucle **while** peut être une boucle **while**
- ❑ Ex : Quel résultat produit par ce programme ?

```
i = 1
while i <= 3 :
    j = 1
    while j <= 2 :
        print(i, ", ", j)
        j = j + 1
    i = i + 1
```



1,	1
1,	2
2,	1
2,	2
3,	1
3,	2

Boucle while

- Exemple d'application

❑ On veut écrire un programme qui affiche tous les nombres premiers entre 2 et 100.

❑ Exemple de résultat :

```
Les nombres premiers sont :  
2 , 3 , 5 , 7 , 11 , 13 , 17 , 19 , 23 , 29 , 31 , 37 , 41 , 43 , 47 , 53 , 59 , 61 , 67 , 71  
, 73 , 79 , 83 , 89 , 97 ,
```

```
print("Les nombres premiers sont :")  
n=2  
while(n<=100):  
    i=2  
    while(i<=n//2):  
        if n%i==0:  
            break  
        i+=1  
    else:  
        print(n, end=" , ")  
    n+=1
```

Boucle for

- ❑ Dans la boucle while, la condition détermine le nombre de fois que la boucle est exécutée **boucle conditionnelle**
- ❑ Si on connaît ce nombre à l'avance, on peut utiliser le **for... boucle inconditionnelle**
- ❑ **Syntaxe :**

```
for élément in séquence :  
    bloc d'instructions  
# suite du programme
```

Boucle for

- ❑ Permet de parcourir des structures :

Listes de nombres, d'objets, lettres d'un mot

```
for e in [1, 4, 5, 0, 9, 1] :  
    print(e)
```

1
4
5
0
9
1

```
for e in ["a", "e", "i", "o", "u", "y"] :  
    print(e)
```

a
e
i
o
u
y

```
for e in "python":  
    print(e)
```

p
y
t
h
o
n

e prend successivement les valeurs de la liste parcourue

Boucle for

range()

❑ **range**(*deb*, *fin*, *pas*)

- Fonction qui prend des arguments entiers
- génère une séquence d'entiers entre [***deb***, ***fin***] avec le *pas* choisi.

❑ Les paramètres *deb* et *pas* sont **optionnels**

- ❑ **range(a)** : séquence des entiers dans [0, a[, c'est-à-dire dans [0, a-1]
- ❑ **range(b,c)** : séquence des valeurs [b, c[, c'est-à-dire dans [b, c-1]
- ❑ **range(e, f, g)** : séquence des valeurs [e, f[avec un pas de g

Boucle for

range()

for var **in range**(deb, fin, pas) :
instructions

for i **in range**(1,6) :
print (i,end=",")

➔ 1, 2, 3, 4, 5,

for i **in range**(4,-1,-1) :
print (i,end=" ")

4 3 2 1 0

Boucle for

- ❑ En cas **d'incohérence**, la boucle est **ignorée** et l'on passe aux instructions suivantes :

```
for k in range(200, 210, -2) :  
    print(k)
```

```
for k in range(110, 100, -2) :  
    print(k)
```

} *ignorée*

}

110
108
106
104
102

- ❑ **Quoi qu'il arrive** dans le corps de la boucle, la variable du compteur prend la **valeur suivante** du *range* ou de la liste à chaque nouvelle étape de la boucle

Boucle for

Exercice 8: Ecrire un algorithme qui lit **n** nombres entiers et détermine quel est le maximum, minimum et la somme de ces nombres.

```
>>> Entrer n : 5
>>> Donner un nombre : 3
>>> Donner un nombre : -5
>>> Donner un nombre : 13
>>> Donner un nombre : 22
>>> Donner un nombre : 0
Max : 22  Min : -5  Somme : 33
```

```
n = int(input("Entrer n: "))
a = int(input(" Donner un nombre : "))
maxi=mini=som=a
for i in range(n-1):
    a = int(input(" Donner un nombre : "))    som+=a
    if (a>maxi):
        maxi=a
    if (a<mini):
        mini=a
print("Max :",maxi,"Mini :",mini,"Somme : ",som)
```

Boucle for

Exercice 8: Ecrire un algorithme qui lit **n** nombres entiers et détermine quel est le maximum, minimum et la somme de ces nombres.

```
>>> Entrer n : 5
>>> Donner un nombre : 3
>>> Donner un nombre : -5
>>> Donner un nombre : 13
>>> Donner un nombre : 22
>>> Donner un nombre : 0
Max : 22  Min : -5  Somme : 33
```

```
import math
n=int(input("donner un n : "))
max=-math.inf;min=math.inf;s=0
for i in range(n):
    a=int(input("donner un nombre : "))
    if a>max:
        max=a
    if a<min:
        min=a
    s+=a
print(max,min,s)
```


Boucle for

Exercice 9 :

1. On dispose d'une feuille de papier d'épaisseur 0,1 mm.
Combien sera l'épaisseur en mètre si on arrive à la plier un nombre de fois n .
2. Que remarquer si $n=42$?

```
n = int(input("Entrer le nombre de fois : "))
ep=0.1e-3
for i in range(1,n+1):
    ep=2*ep
print("l'épaisseur de la feuille en mètre est :",ep)
```

N.B : la distance Terre-Lune (384 403 km)

Boucle for

Exercice 10: Approximation de Pi par la méthode de Monte-Carlo

On cherche à calculer une approximation de la valeur de π en utilisant la méthode de Monte-Carlo.

Le principe de la méthode de Monte-Carlo est de tirer au hasard des coordonnées x et y , chacune dans l'intervalle $[0;1[$.

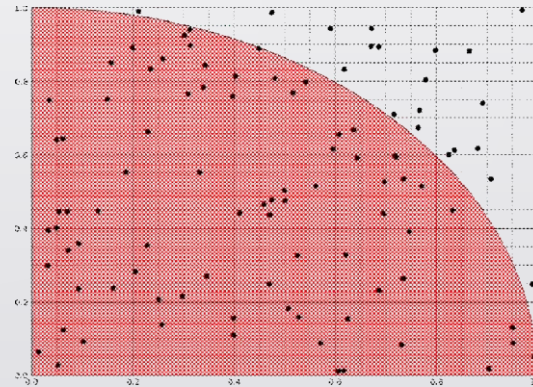
Si $\sqrt{x^2 + y^2} \leq 1$, alors le point de coordonnées (x,y) appartient au quart de disque D de centre $(0,0)$ et de rayon 1.

Si on tire au hasard n points, et soit p le nombre de points appartenant à D alors on :

$p/n = (\text{aire de } D) / (\text{aire du rectangle})$

$$\frac{p}{n} \approx \frac{\pi}{4}$$

Alors une approximation de π égal $4 * p/n$



Boucle for

Exercice 10: Approximation de Pi par la méthode de Monte-Carlo

```
1  from math import sqrt
2  from random import random
3
4  n=int(input("donner un nombre :")) # Nombre de points totale
5  p= 0      # Nombre de fois que l'on se trouve dans le quart de cercle
6
7  for i in range(n):
8      x = random()
9      y = random()
10     if sqrt(x**2+y**2)<=1:
11         p= p+ 1
12
13  Pi = 4 * p / n
14  print("Pi = %f" % Pi)
```



Les fonctions

Fonction : pourquoi ?

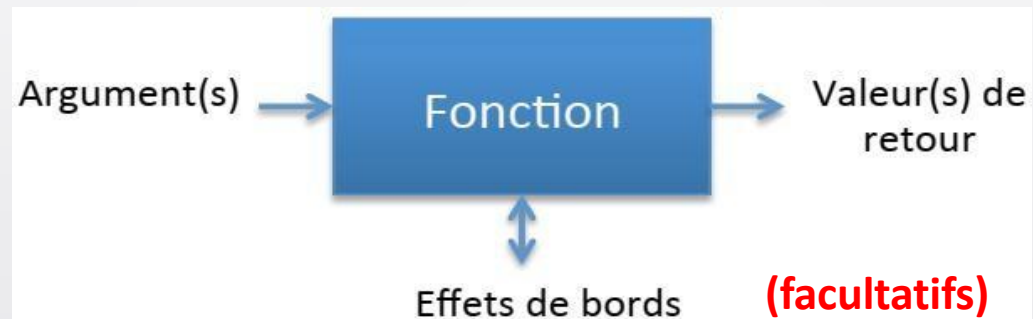
❑ **But:** **structurer** son code lorsque l'on fait plusieurs fois la même chose (ou presque)

- Pour qu'il soit plus **lisible** (plusieurs morceaux)
- Pour qu'il soit plus **facilement modifiable**
- Pour qu'il soit plus **facile à tester**



Fonction : Principe

- Une suite d'instructions encapsulées dans une « boîte »



- Qui prend **zéro**, un ou des **arguments**
- Qui retourne **zéro**, une ou **plusieurs valeurs de retour**
- Et qui contient éventuellement des "**effets de bord**" qui modifient l'environnement.

Fonction : Syntaxe

```
def nom_fonction(argument1,..., argumentN) :  
    instructions à exécuter  
    return valeur de retour
```

- **Note** : le **return** est facultatif, ainsi que les arguments (mais pas les parenthèses!)

Fonction : Exemple

- Dans un exercice de géométrie, on doit souvent calculer la distance entre deux points.

$$\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

- Arguments : xa, ya, xb, yb
- Valeur de retour : distance AB
- Effet de bord : afficher les points A et B

Fonction : Appel d'une fonction

- L'appel de la fonction prend la forme :

nomdefonction(expression₁, expression₂, ... expression_k)

- **Exemple 1:** La fonction *sommeCarre* suivante retourne la somme des carrées de deux réels x et y :

```
def sommeCarre ( x, y ) :  
    z = x**2 + y**2  
    return z
```

L'appel de la fonction *sommeCarre* peut se faire :

```
>>> print(sommeCarre(2,3)) 13
```

Note: un appel de fonction peut se faire à l'intérieur **d'une autre fonction**.



Fonction

- **Exercice 11:** Ecrire une fonction **table(n)** qui permet d'afficher la table de multiplication du nombre n:

L'appel de la fonction *table* peut se faire :

```
>>> table(7)
```

```
1 x 7 = 7
```

```
2 x 7 = 14
```

```
3 x 7 = 21
```

```
4 x 7 = 28
```

```
5 x 7 = 35
```

```
6 x 7 = 42
```

```
7 x 7 = 49
```

```
8 x 7 = 56
```

```
9 x 7 = 63
```

```
10 x 7 = 70
```

```
def table(n):  
    for i in range(1,11):  
        print(i, " x ", n, " = ", i*n)
```




Fonction

- Exercice 12: Ecrire une fonction *fact* qui retourne le factoriel d'un entier passé en paramètre.

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

Fonction

- **Exercice 13:** Servez vous de cette fonction est écrire une autre nommée **comb** qui permet de calculer coefficient binomial.

$$C_n^p = \frac{n!}{p! (n - p)!}.$$

```
def comb(n,p):  
    if p==n or p==0:  
        return 1  
    elif p==1:  
        return n  
    else :  
        return fact(n)//(fact(p)*fact(n-p))
```

Fonction: Plusieurs valeurs de retour : Un exemple

```
def division(a,b) :  
    # renvoie le quotient et le reste  
    # de la division de a par b  
    quotient=a//b  
    reste= a%b  
    return quotient, reste  
  
# programme principal  
q,r = division(22,5)  
print("q=", q, "et r=", r)
```

//////////

Fonction: Docstring: Un exemple

```
def division(a,b) :  
    """ Renvoie le quotient et le reste  
        de la division de a par b """  
    quotient=a//b  
    reste= a%b  
    return quotient, reste
```

Dans l'interpréteur (ou dans un programme):

```
>>> help(division)  
Help on function division in module __main__:  
  
division(a, b)  
    Renvoie le quotient et le reste  
    de la division de a par b
```

Définir une fonction avec des arguments optionnels: exemple

```
def affiche_pizza(saveur, taille="normale"):  
    """ Affiche saveur, taille et prix de la pizza  
    """  
    print("Pizza", saveur, "taille:", taille)  
    if taille=="normale":  
        prix=9  
    elif taille=="maxi":  
        prix=12  
    print("Prix", prix, "euros.")
```

→ taille est un argument optionnel ayant comme **valeur par défaut** "normale".

Définir une fonction avec des arguments optionnels: exemple

```
>>> affiche_pizza("4 fromages")
Pizza 4 fromages taille: normale
Prix 9 euros.

>>> affiche_pizza("4 fromages", "maxi")
Pizza 4 fromages taille: maxi
Prix 12 euros.

>>> affiche_pizza("Reine", "normale")
Pizza Reine taille: normale
Prix 9 euros.
```

Les fonctions anonymes

Le mot-clé *lambda* en Python permet la création de **fonctions anonymes** (i.e. sans nom et donc non définie par **def**).

```
f = lambda x : x*3
```

```
>>> f ( 3 )  
9
```

```
racine= lambda x : x**0.5 if x>=0 else False
```

```
>>> racine(9 )  
3
```

Les fonctions anonymes

Exercice 14: On peut approximer la dérivée d'une fonction f en un point x_0 avec la formule suivante :

$$f'(x_0) = (f(x_0+h) - f(x_0)) / h$$

Avec $0 < h < 1$ très petit.

Ecrire une fonction `derive(f,x0)` qui reçoit en paramètre une fonction f et un réel x_0 et qui permet de retourner la valeur $f'(x_0)$.

```
def derive(f,x0,h=1e-10):  
    return (f(x0+h)-f(x0))/h
```

```
>>> f=lambda x: x**2-2  
>>> x0=2  
>>> derive(f,x0)  
4.000000330961484
```

```
fp=lambda f,x0,h: (f(x0+h)-f(x0))/h
```

```
>>> f=lambda x: x**2-2  
>>> x0=2  
>>> fp(f,x0,1e-5)  
4.000010000027032
```

Le mot-clé **None**

- Il existe une valeur constante en Python qui s'appelle **None**. Cela correspond à "rien", "aucune".
- Lorsqu'une fonction n'a pas d'instruction return, elle renvoie la valeur **None**.

```
def dit_bonjour():  
    print("Bonjour!")  
    print("Bienvenue")  
    # pas de return  
  
# prog. Principal  
test=dit_bonjour()  
print("Test vaut", test)
```

Affichage :

Bonjour!
Bienvenue. Test vaut
None

Déclaration d'une fonction sans connaître ses paramètres

```
1  def f(*args, **kwargs):  
2      print(args)  
3      print(kwargs)  
4  
5  f(1, 3, 'b', j = 1)  
6
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Johri> & python c:/Users/Johri/Desktop/test.py  
(1, 3, 'b')  
{'j': 1}  
PS C:\Users\Johri> 
```




Les fonctions

Passage des paramètres en Python :

Intéressons-nous au problème suivant :

Si on modifie une variable en paramètre d'une fonction à l'intérieure, est-elle réellement modifiée après l'appel ?

Exemple 1 :

Considérons la fonction **ajoute** suivante :

```
def ajoute( a ) :
```

```
    a = a + 1
```

appel

```
>>> b = 5
```

```
>>> ajoute(b)
```

```
>>> print(b)
```

```
5
```

a est un arguments de types **immuables**

Les fonctions

Passage des paramètres en Python :

En Python, le passage des paramètres est comparable à une affectation. Le fil d'exécution ressemble donc à ceci :

```
>>> b = 5
# exécution d'ajout ( b )
a = b
a = a + 1 destruction de a
# retour au programme principal
print(b)
5
```

```
Exemple 1 :
def ajoute( a ) :
    a = a + 1
# appel
>>> b = 5
>>> ajoute(b)
>>> print(b)
5
```

*Tout se passe comme si le paramètre était **passé par valeur** : **une copie du contenu de la variable est recopiée de b vers a et c'est la copie qui est modifiée, pas l'original.***

Les fonctions

Variables globales et locales

En Python, on distingue deux sortes de variables : les **globales** et les **locales**.

Par exemple, dans le programme suivant, **x** est une variable **globale** :

```
>>> x = 7
>>> print(x)
```

À l'inverse, la variable **y** dans la fonction **f** suivante est **locale** :

```
def f( ):
    y = 8
    return y
```

Après l'appel de la fonction **f**, la variable locale **y** disparaît.

En particulier, l'instruction suivante échoue en indiquant que la variable **y** n'est pas définie :

```
>>> print(y)
Error
```

Les fonctions

Variables globales et locales

Remarque

- Si l'on veut accéder à une variable **globale** à l'intérieur d'une fonction, on utilise le mot-clé **global** en Python.

Exemple 1

Par exemple pour écrire une fonction qui réinitialise la variable globale **x** à 0, alors il ne faut pas écrire :

```
x = 7
def reinitialise(x) :
    global x
    x=0
```

Exemple 2

```
def f( ) :
    global a
    a = a + 1
    c = 2 * a
    return a + c
```

Les fonctions

Exercice 15: Le code d'une photocopieuse est un numéro N composé de 4 chiffres. Les codes corrects ont le chiffre le plus à droite égal au reste de la division par 7 de la somme des trois autres chiffres. Ainsi, le code 5733 est incorrect car $5+7+3 = 15$ et $15 \bmod 7 = 1 \neq 3$ tandis que 5731 est correct. Le but de cet exercice est de créer une fonction qui prend en entrée le code et qui renvoie "VALIDE" ou "NON VALIDE".

```
1  def code(N):
2      s=0
3      for i in range(3,0,-1):
4          r=N//10**i
5          N=N%10**i
6          s+=r
7      if s%7==N:
8          return "VALIDE"
9      return "INVALIDE"
10
```




Les fonctions

Exercice 16: (Code César)

Le **codage de César** est une manière de crypter un message de manière simple : on choisit un nombre n (appelé clé de codage) et on décale toutes les lettres de notre message du nombre choisi. Exemple avec $K = 2$: la lettre “A” deviendra “C”, le “B” deviendra “D” . . . et le “Z” deviendra “B”. Ainsi, le mot “MATHS” deviendra, une fois codé, “OCVJU” (pour décoder, il suffit d’appliquer le même algorithme avec $K = -2$).