

CHAPITRE 3 : LA BASE DE LA PROGRAMMATION PYTHON

I. Définitions:

Algorithme : ensemble des étapes permettant d'atteindre un but en répétant un nombre fini de fois un nombre fini d'instructions.

Programme : un programme est la traduction d'un algorithme en un langage compilable ou interprétable par un ordinateur.

II. Programmation avec Python :

II-1. Historique :

Python a été développé en 1989 par **Guido van Rossum**.

En 2005, il a été engagé par Google pour ne travailler que sur Python.

Les deux versions récentes de Python : 2.x et **3.x**



II-2. Caractéristiques

- Libre et gratuit
- Interprété (pas besoin de compilation)
- peut être interactif (on peut s'en servir comme une calculatrice)
- Extensible (par exemple avec des modules graphiques)
- Typage dynamique (pas de déclaration de variables)
- Syntaxe claire

II-3. Utilisation

- **Avec l'interpréteur** : on s'en sert un peu comme une calculatrice.

```
Python (command line)
Python 3.2.2 (default, Sep  4 2011, 09:51:08) [MSC v.1500 32 bit (Intel)] on wi
32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+3
5
>>> 12/4
3.0
>>>
```

- **Avec l'éditeur** : on écrit un programme dans n'importe quel éditeur de texte, puis on l'exécute à l'aide d'un terminal :



```

cpge.py - Bureau/cpge.py
File Edit Format Run Options Windows Help
age=12
print("votre age est :", age)
Ln: 3 Col: 0

```

III. Les types de données de base sous Python :

| Les Nombres | | |
|---|--|--|
| entier : int | flottant : float | complexe : complex |
| <i>Exemple :</i> | <i>Exemple :</i> | <i>Exemple :</i> |
| <ul style="list-style-type: none"> • 12 • 130 | <ul style="list-style-type: none"> • 12.5 • 3.1e5 # 3.1 x 10⁵ | <ul style="list-style-type: none"> • 3+5j • 3+2j |

| chaîne de caractères : string |
|--|
| Trois notations disponibles : <ul style="list-style-type: none"> • Guillemets pour inclure facilement des apostrophes "C'est mon cpge" • Apostrophes pour inclure facilement des guillemets 'est "froide" !' • Triples guillemets pour conserver la mise en page (lignes multiples) : """ cpge : -Tanger -Rabat """ |

| Les booléens : bool |
|---|
| Un booléen est un type de données qui ne peut prendre que deux valeurs : True ou False . |
| <i>Exemple :</i> <ul style="list-style-type: none"> • Pour les entiers (int), la valeur 0 correspond à False et les autres valeurs à True • Pour les flottant (float), la valeur 0. correspond à False et les autres valeurs à True • Pour les chaînes (string), la valeur « » correspond à False et les autres valeurs à True |

La fonction **type()** permet de connaître le type d'une valeur ou d'une variable :

```
>>> type(12.5)
<class 'float'>
```

```
>>> type("un essai")
<class 'str'>
```

```
>>> type("3+5j")
<class 'complex'>
```

IV. Variables sous Python :

Une **variable** est un espace mémoire dans lequel il est possible de mettre une **valeur**.

**IV-1. Création de variables :**

En Python, les variables sont créées automatiquement à leur **première utilisation**. Pour créer une variable, il suffit donc de l'utiliser en l'affectant (utilisation de =) une première fois,

```
>>> x = 1          # création de la variable x de type int avec la valeur 1
>>> y = 2.5      # création la variable y de type float avec la valeur 2.5
>>> z = 'cpge'   # création la variable z de type str avec la valeur 'cpge'
```

IV-2. Nom des variables

une variable peut prendre n'importe quel nom, tant qu'elle respecte les règles suivantes :

- Son nom commence par une lettre minuscule (a à z) ou majuscule (A à Z), ou bien par le caractère souligné (_)
- Pour la suite de son nom, on peut utiliser les lettres minuscules et majuscule, le souligné et un chiffre (0 à 9)
- Son nom ne doit pas être un **mot réservé** (voir Annexe 4).
- Le nom des variables est **sensible à la casse**, ainsi Age et age ne désignent pas la même variable.

V. Les opérateurs les plus usuels :**V-1. Affectation :**

| Affectation simple | Affectation multiple | Affectation parallèle |
|--------------------|-----------------------------|---------------------------------|
| >>> x=7 | >>> x=y=7 ici x et y = 7 | >>> x,y = 7,8 ici x=7 et y=8 |

V-2. Les opérateurs mathématiques :

| Opérateur | effet | exemple |
|-----------|------------------------------|-------------------------|
| + | addition | 6+4 \Rightarrow 10 |
| - | soustraction | 6-4 \Rightarrow 2 |
| * | multiplication | 6*4 \Rightarrow 24 |
| / | division | 6/4 \Rightarrow 1.5 |
| ** | élévation à la puissance | 12**2 \Rightarrow 144 |
| // | division entière | 6//4 \Rightarrow 1 |
| % | reste de la division entière | 6%4 \Rightarrow 2 |

Rq : Pour les chaînes de caractères :

| Opérateur | Effet | Exemple |
|-----------|---------------|--|
| + | concaténation | 'cpge'+'Tanger' \Rightarrow 'cpgeTanger' |
| * | répétition | 'cpge'*2 \Rightarrow 'cpgecpge' |

**V-3. Les opérateurs d'affectation composée :**

| Opérateur | Exemple |
|-----------|--|
| | pour chaque exemple on prend $X=5$ |
| $+=$ | $X+=2$ équivaut à $X=X+2 \Rightarrow 7$ |
| $-=$ | $X-=2$ équivaut à $X=X-2 \Rightarrow 3$ |
| $*=$ | $X*=2$ équivaut à $X=X*2 \Rightarrow 10$ |
| $/=$ | $X/=2$ équivaut à $X=X/2 \Rightarrow 2.5$ |
| $**=$ | $X**=2$ équivaut à $X=X**2 \Rightarrow 25$ |
| $//=$ | $X//=2$ équivaut à $X=X//2 \Rightarrow 2$ |
| $\%=$ | $X\%=2$ équivaut à $X=X\%2 \Rightarrow 1$ |

V-4. Les opérateurs de comparaisons:

| Opérateur | effet |
|---------------|--|
| $>$ | Supérieur |
| $<$ | Inférieur |
| $>=$ | Supérieur ou égal |
| $<=$ | Inférieur ou égal |
| $==$ | Egal |
| $!=$ | Différent |
| is | $X \text{ is } Y \Rightarrow X$ et Y représentent le même objet |
| is not | $X \text{ is not } Y \Rightarrow X$ et Y ne représentent pas le même objet |

Rq : Il est possible d'enchaîner les opérateurs : $X < Y < Z$, dans ce cas, c'est Y qui est pris en compte pour la comparaison avec Z et non pas l'évaluation de $(X < Y)$

V-5. Les opérateurs logiques :

| Opérateur | effet |
|------------|--|
| or | $X \text{ or } Y$: OU logique, si X évalué à True, alors l'expression est True et Y n'est pas évalué. Sinon, l'expression est évaluée à la valeur booléenne de Y . |
| and | $X \text{ and } Y$: ET logique, si X est évalué à False, alors l'expression est False et Y n'est pas évalué. Sinon, l'expression est évaluée à la valeur booléenne de Y . |
| not | not X : évalué à la valeur booléenne opposée de X . |

VI. Les instructions de lecture et d'écriture :**VI-1. Affichage vers l'écran : print() :**

La fonction **print()** permet d'afficher un message et/ou la valeur d'une ou plusieurs variables.

- Afficher un texte :
`print ("texte à afficher")`
- Afficher la valeur d'une variable :
`print (nom_de_la_variable)`
- Mélange de texte et de valeurs :
`print("texte",nom_de_la_variable,"texte", nom_de_la_variable)`

**Exemple 1:**

```
print ("La surface ?")      #affiche La surface ? à l'écran
print (S)                  #affiche le contenu de la variable S à l'écran
```

Exemple 2:

```
>>> a = 3
>>> print(a)              #affiche 3 à l'écran
>>> a = a + 3
>>> b = a - 2
>>> print("a =", a, "et b =", b) #affiche a=6 et b=4 à l'écran
```

VI-2. La fonction de saisie input() :

Il s'agit de réaliser une saisie au clavier : la fonction standard **input()** interrompt le programme, affiche une éventuelle invite à l'écran et attend que l'utilisateur entre une donnée au clavier (affichée à l'écran) et la valide par la touche **Entrée**

```
variable=input()    ou    variable=input(" Message ")
```

Exemple :

```
>>> a = input("Entrez un flottant : ")
Entrez un flottant : 12.345
>>> type(a)
<class 'str'>
>>> b = input("Entrez un entier : ")
Entrez un entier : 10
>>> type(b)
<class 'str'>
```

Rq : La fonction standard **input()** effectue toujours une saisie en mode texte (la valeur retournée est une chaîne) dont on peut ensuite changer le type.

VI-3. Conversion de type :

Il existe plusieurs fonctions qui permettent de forcer le type d'une variable en un autre type.

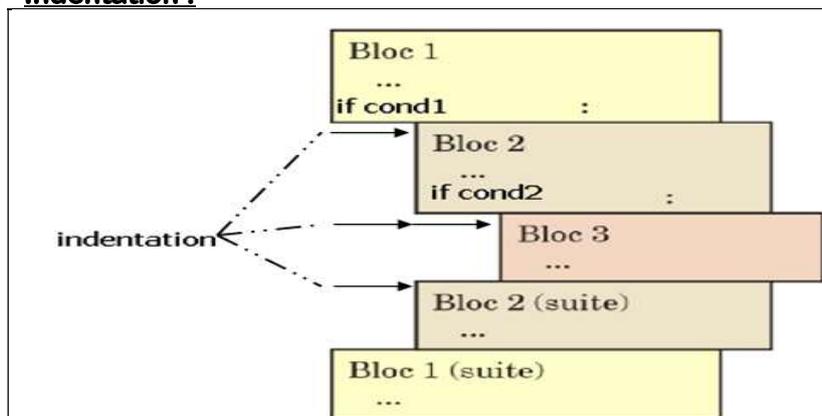
| Fonction | effet | Exemple |
|----------|--|--|
| int() | permet de modifier une variable en entier. Provoque une erreur si cela n'est pas possible. | <pre>>>> a = '30' >>> type(a) <class 'str'> >>> a = int(a) >>> type(a) <class 'int'></pre> |



| | | |
|----------------|---|---|
| float() | permet la transformation en flottant. | Exemple 1 : <pre>>>> a =input("Entrer un réel") >>> type(a) <class 'str'> >>> a = float(a) >>> type(a) <class 'float'></pre> Exemple 2 : <pre>>>> a =float(input("Entrer un réel")) >>> type(a) <class 'float'></pre> |
| str() | permet de transformer la plupart des variables d'un autre type en chaînes de caractère. | <pre>>>> a = 30 >>> type(a) <class 'int'> >>> a = str(a) >>> type(a) <class 'str'></pre> |
| eval() | évalue le contenu de son argument comme si c'était du code Python. | <pre>>>> b = 30 >>> eval('int(b)+5') 35</pre> |

VII. Les structures conditionnelles (Les structures de contrôle):

VII-1. Indentation :



**VII-2. Format général d'une structure de contrôle :**

```

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

```

Exemple 1:

```

a = -150
if a < 0:
    print (a , 'est négatif')
else :
    print (a , 'est positif')

```

le script affiche :
-150 est négatif

Exemple 2:

```

a = 10.
if a > 0:
    print (a , 'est strictement positif')
    if a >= 10:
        print (a, ' est un nombre')
    else:
        print (a, ' est un chiffre')
        a += 1
elif a != 0:
    print (a , 'est strictement négatif')
else:
    print (a, ' est nul')
print ('la valeur de a après les conditions est',a)

```

On aura l'affichage suivant :

10 est strictement positif
10 est un nombre
la valeur de a après les conditions est 11

VIII. Les structures de répétition (Boucles) :**La boucle while**

```

while test :
    blocs d'instructions

```

La boucle for

```

for var in séquence :
    blocs d'instructions

```

Pour générer une liste (séquence) d'entiers, on utilise la fonction **range** :

- **range(debut, fin, pas)** : permet d'obtenir une liste de nombre :
 - ✓ **range(4, 10, 2)** génère la liste suivante [4, 6, 8]
 - ✓ **range(4, 10)** génère la liste suivante [4, 5, 6, 7, 8, 9]
 - ✓ **range(10)** génère la liste suivante [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**Exemple 1:**

```
i = 1
while i <= 4:
    print( i)
    i = i + 1
print("Fin")
```

le script affiche :

```
1
2
3
4
Fin
```

Exemple 2:

```
sum = 0
for i in range(1, 11): :
    sum += i
print(sum)
```

le script affiche la somme des entiers de 1 à 10

Exemple 3:

```
prod = 1
for i in range(1, 11):
    prod *= i
print(prod)
```

le script affiche le produit des entiers de 1 à 10

IX. Instructions : break, continue et pass

Ces instructions permettent de modifier le comportement d'une boucle (**for** ou **while**) avec un test.

- **break** : sort de la boucle,
- **continue** : remonte au début de la boucle (saute à l'itération suivante.),
- **pass** : ne fait rien,

Exemples :

```
for i in range(0, 5):
    if i == 2:
        break
    print (i)
```

le script affiche :

```
0
1
```

```
for i in range(0, 5):
    if i == 2:
        continue
    print (i)
```

le script affiche :

```
0
1
3
4
```

```
for i in range(0, 5):
    if i == 2:
        pass
    print (i)
```

le script affiche :

```
0
1
2
3
4
```

Rq : Si dans une boucle on utilise l'instruction **break**, alors on peut ajouter **else** à la boucle **for** ou **while** :



Exemples :

| | |
|---|---------------------------------|
| <pre>for i in range(0,11): if i == 2: print(i) break else : print("la boucle est terminée normalement")</pre> | le script affiche : 2 |
|---|---------------------------------|

| | |
|--|--|
| <pre>for i in range(0,11): if i == 15: print(i) break else : print("la boucle est terminée normalement")</pre> | le script affiche : la boucle est terminée normalement |
|--|--|

| | |
|---|--|
| <pre>a=17 i = 2 while i < a: if a%i==0: print(a,"est un nombre non premier") break i = i + 1 else : print(a,"est un nombre premier")</pre> | le script affiche : 17 est un nombre premier |
|---|--|