langage python

Table des matières

I - Présentation du langage Python

1. Langage python

Python est un **langage** informatique **interprété** à usage généraliste , simple à apprendre et à utiliser, qui est essentiellement utilisé pour :

- **coder des petits scripts ou programmes simples**
- piloter/orchestrer des appels vers des fonctions prédéfinies efficaces (quelquefois codées en langage "C" de bas niveau et rapide)
- **piloter/orchestrer des calculs scientifiques**

Le langage python peut également être utilisé dans d'autres domaines (contrôles d'affichages graphiques, accès à des bases de données ou des fichiers, sites web, ...) sans cependant se démarquer d'autres langages informatiques (également bien adaptés pour effectuer efficacement ces tâches) .

1.1. Principales caractéristiques du langage python

simple	code facilement compréhensible et apprentissage rapide
interprété (et pas compilé)	comme les langages "basic" , "javascript" , "perl" , "ruby" , ... *avantages* : **souplesse et résultats immédiats** *inconvénients* : ***exécution pas rapide*** (si 100% python)
à usage général	on trouve des bibliothèques de fonctions prédéfinies pour presque tous les usages (calculs , accès bases de données , affichages , ...)
multi-paradigme	code procédural et séquentiel pour scripts simples code en mode "orienté objet" pour applications élaborées ...
mature et bien implanté	python existe depuis plus de 25 ans et est beaucoup utilisé dans le domaine scientifique et dans le cadre de l'administration système (ex : linux, ...)
open-source	accès et usage libre et gratuit . Seules quelques extensions "clefs en main" sont quelquefois payantes .
multi-plateformes	étant interprété, le langage python peut facilement être utilisé sur tout type de plateformes (Windows , Linux , Mac , smartphones, ...)
syntaxe avec indentations	contrairement à beaucoup d'autres langages qui délimitent des blocs de code via { et } ou via "begin" et "end" , le langage python n'utilise pas de délimiteur mais a une structure de code contrôlée par des indentations (décalages par rapport de débuts des lignes)

Bien qu'intéressant sur bien des points, le langage Python n'est pas interprété par les navigateurs "web/internet" . Les navigateurs "IE , Firefox , Chrome, ..." ont historiquement fait le choix d'utiliser le langage interprété javascript (bien aussi et un peu plus rapide que python) .

1.2. Historique et évolution

Première version publique officielle : 0.9.0 en **1991** développé par "Guido van Rossum" (Pays bas , Amsterdam).

De 1995 à 1999 : évolution du langage aux états-unis (CNRI, ...) , **version 1.6 en 1999**

A partir de **2001** et la version 1.6.1 , l'évolution du langage python est contrôlée par la "***Python Software Foundation***".

Python a longtemps existé en version 2.x (durant la décennie 2000-2010) .

Les version 3.x (à partir de 2008/2010) sont sur certains petits points en rupture avec les version 2.x. Il est donc conseillé aujourd'hui de ne plus utiliser l'ancienne version 2 (dans la mesure du possible)

La version **3.7** de python (datant de ***2018***) est une bonne version récente et stable du langage python.

1.3. Distributions de "python"

En tant que langage interprété , python peut être pris en charge par plusieurs variantes du moteur d'interprétation.

L'implémentation de référence est **CPython** (interpréteur codé en langage C) .
Il existe aussi **Jython** (basé sur une machine virtuelle java) , ...

En outre, on trouve aujourd'hui certaines distributions packagées de python incluant "moteur d'interprétation + éditeurs + ensemble de bibliothèques + ...) .

Les principales distributions (basées sur CPython) sont :

- **WinPython**
- **Annaconda**
- ...

La plupart des distributions sont à usage scientifique et elles incluent les bibliothèques de calculs ***numPy*** et ***sciPy*** .

Autrement dit, pour installer python sur un ordinateur on peut installer que python ou bien toute une distribution telle qu'anaconda .

II - Prise en main de l'interpréteur Python

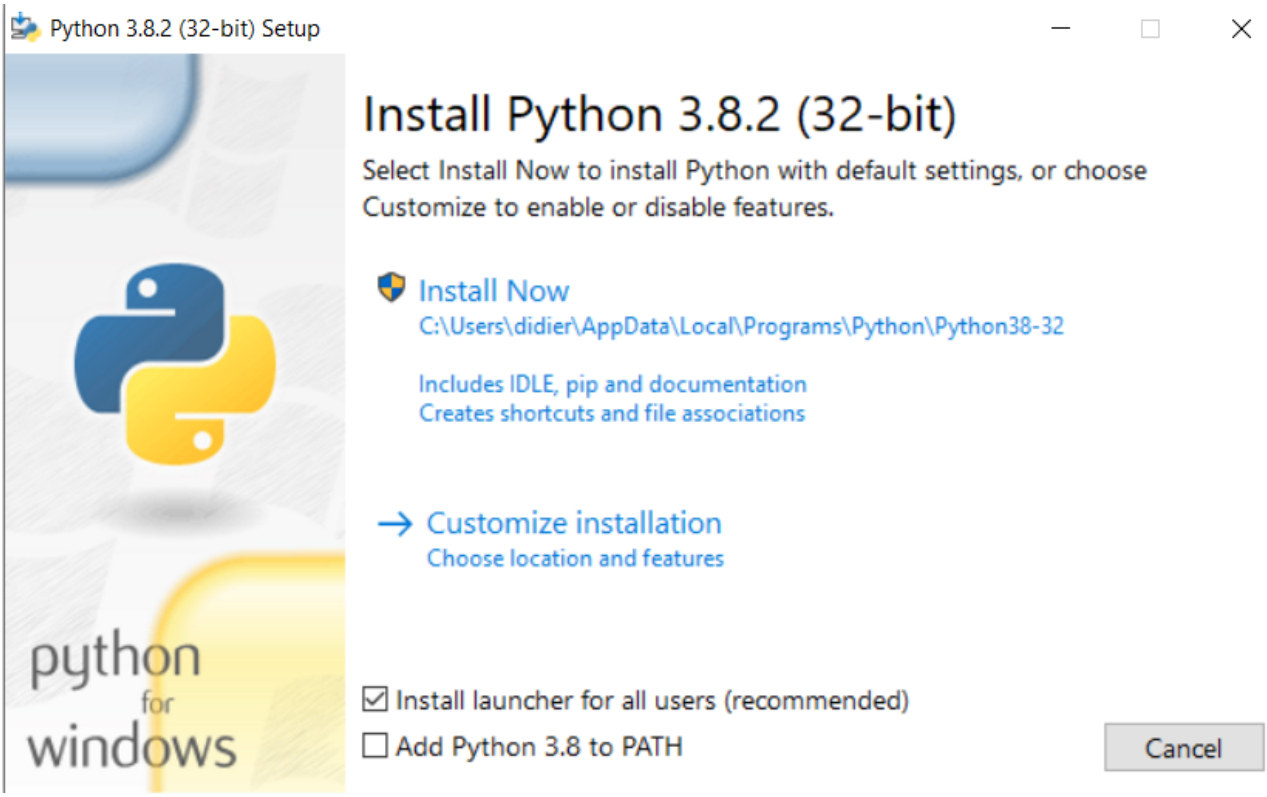
1. Installation et première utilisation de python

Avant de pouvoir utiliser le langage python il faut installer un des interpréteurs disponibles

1.1. Quelques installations possibles de python

Python est souvent installé d'office sur les distributions linux mais pas forcément dans une version récente . Par exemple sur **linux ubuntu 18.04** , le phython 3.6 pré-installé se lance avec la commande "**python3**" .

Pour installer python sur un ordinateur windows , on peut se connecter sur le site officiel **https://www.python.org/** et effectuer un téléchargement de l'installeur via l'url ***https://www.python.org/ftp/python/3.8.2/python-3.8.2.exe*** (environ 26 Mo) .



On peut éventuellement ajouter Python au PATH pour pouvoir ultérieurement le lancer facilement depuis une fenêtre CMD ou autre .

On peut également installer indirectement Python en installant toute une distribution telle qu'anaconda .

https://www.anaconda.com ,
https://repo.anaconda.com/archive/Anaconda3-2020.02-Windows-x86.exe ***(environ 478 Mo)***

Répertoire d'installation par défaut: *C:\ProgramData\Anaconda3*, éventuellement ajouté au PATH.

1.2. Prise en main de l'interpréteur python

Au sein d'une fenêtre de commande (CMD ou PowerShell ou shell linux ou ...) , la commande **python -V** permet d'afficher la version de l'interpréteur python (ex : 3.7.6) .

En lançant la commande "**python**" sans argument , on peut ainsi **lancer l'interpréteur python en mode interactif** . Celui ci nous invite alors à saisir des commandes après une invite (prompt) **">>>"** .
Toute "commande / ordre / expression" saisi(e) est alors immédiatement interprété(e) et le résultat s'affiche immédiatement .

```
python
Python 3.7.6 .....
>>> 2+3
5
>>> 4*6
24
>>>
```

1.3. Ecriture et lancement d'un programme python

Un fichier de code python a par convention l'extension "**.py**" et peut être saisi avec un très grand nombre d'éditeur (nodepad++, visual studio code , ...) .

Exemple: **hello.py**

```
a=2
b=3
c=a+b
print('Hello world')
print('c=',c)
```

Pour lancer ce script (petit programme) , on peut lancer la commande

python hello.py

Ce qui provoque l'affichage suivant :

Hello world

c= 5

Et pour automatiser un lancement depuis un double click via l'explorateur de fichiers de windows on peut éventuellement écrire un fichier de lancement tel que celui ci :

lancer_prog_python.bat

```
REM avec PATH contenant le répertoire d'installation de python
python hello.py
pause
```

1.4. Instructions élémentaires du langage python .

NB : En langage python les **commentaires** sont des **fin de lignes commençant par #**

p1.py

```
a=1  # nomVariable=valeur_a_affecter
print(a) # affiche la valeur de la variable a (ici 1)
a=2
print(a) # affiche la nouvelle valeur de la variable a (ici 2)
b=a*3+4  # variableResultat = expression d'un calcul
c=a+b
print("b=",b,"c=",c)  # affiche plusieurs choses en les séparant par des espaces
                      # affiche ici b= 10 c= 12
prenom = "alex"  # une chaine de caractères est délimitée en python par des " " ou des ' '
nom = 'Therieur'
nomComplet = prenom + ' ' + nom # concaténation (ajout bout à bout)
print(nomComplet) # affiche alex Therieur

# input('texte question') demande à saisir/renseigner une valeur
age = input ("quel est ton age ? ")
print ('age renseigné:' , age);  # affichera la valeur choisie/précisée .
```

Attention: *un + entre 2 chaînes de caractères (string) déclenche une concaténation (valeurs juxtaposées bout à bout) tandis qu'un + entre 2 nombres déclenche une addition*

*La fonction **float( ) converti une chaîne de caractères en une valeur numérique** (avec potentiellement une virgule notée "." en anglais) .*

Exemple :

```
a=input('a:') # exemple a: 2 et a vu comme '2'
b=input('b:') # exemple b: 3 et b vu comme '3'
c=a+b # '2' + '3' = '23'
print('c=a+b=',c) # affiche par exemple c=a+b= 23

a=float(input('a:')) # exemple a: 2 et a vu comme 2.0
b=float(input('b:')) # exemple b: 3 et b vu comme 3.0
c=a+b # 2.0 + 3.0 = 5.0 = 5
print('c=a+b=',c) # affiche par exemple c=a+b= 5
```

Quelques exercices :

Ecrire un petit script calculant la moyenne de 2 nombres avec des valeurs à saisir :

x=*4 ou 2 ou ...*
y=*8 ou 10 ou ...*
moyenne= *6.0 ou ...*

moyenne.py

Autres essais libres. c'est en forgeant que l'on devient forgeron.

III - Syntaxes élémentaires , types , boucles

1. Types et syntaxes élémentaires de Python

1.1. Types de données en python 3

Comme beaucoup d'autres langages interprétés (tel que javascript) , le langage python a un ***typage dynamique*** :
Il n'est pas nécessaire de préciser le type d'une variable.
Une variable python a , à un instant donné, un type qui dépend de la valeur affectée .

Les principaux types élémentaires sont précisés dans le tableau ci-après

type	*caractéristiques*	*exemples*
int	nombre entier	**-12** **0** **234**
float	nombre à virgule flottante	**-12.5** **0.0** **234.78**
complex	nombre complexe (avec partie imaginaire)	**2+4j**
str	chaîne de caractères (string)	**'abc'** **"def"**
bool	booléen (True or False)	**True** **False**
list	liste (ou tableau redimensionnable) d'éléments	**["rouge" , "vert" , "bleu"]**

la fonction type() renvoie le type (à l'instant présent) d'une variable python .

```
a=128
type(a) # retourne int
b='bonjour'
type(a) # retourne str
```

Pour les nombres complexes , la lettre j a été choisie à la place de i car i est souvent utilisé comme indice pour les boucles.

1.2. Opérations sur chaînes de caractères (str/string)

```
nom="toto"
nom=nom.upper() # retourne la chaîne transformée avec des caractères majuscules
print(nom) # affiche TOTO
```

1.3. Bloc d'instructions et indentation

Dans beaucoup d'autres langages (ex : C/C++ , java , javascript, ...) , un bloc d'instruction est délimité par { et } (début et fin).

Le langage python a quant à lui choisi de délimiter un bloc d'instruction par un niveau d'indentation (un décalage homogène par rapport au début de ligne) : Toutes les instructions décalées via un même nombre d'espaces seront considérées comme appartenant à un même bloc . On conseil généralement 4 espaces pour différencier un sous-bloc de son bloc parent .

exemple en langage **C**	exemple en langage **python**

`int factorielle(int n)`
`{`
`if (n < 2) { return 1; }`
`else {`
`return n * factorielle(n - 1);`
`}`
`}`
	`def factorielle(n):`
	`if n < 2:`
	`return 1`
	`else:`
	`return n * factorielle(n - 1)`

NB :

- Python conseil l'utilisation de plusieurs espaces consécutifs (idéalement 4).
- Des tabulations peuvent éventuellement être utilisées à la place mais il faut choisir entre série d'espaces et tabulations : un mélange des 2 styles est normalement interdit par python3.

1.4. Tests et opérateurs logiques (comparaisons, ...)

```
if condition :
   bloc d'instruction déclenché si condition vérifiée
```

ou

```
if condition :
   bloc d'instruction déclenché si condition vérifiée
else :
   bloc d'instruction déclenché sinon (si condition non vérifiée)
```

Une condition (à vérifier) est généralement formulée comme un test de comparaison :

x == y	égal à
x != y	différent de
x > y	strictement supérieur à
x >= y	supérieur ou égal à
x < y	strictement inférieur à
x <= y	inférieur ou égal à

Exemples :

```
age=20
if age>=18:
    print('majeur pour age=',age)
    print('pas mineur')
print('suite dans tous les cas')

age=16
if age>=18:
    print('majeur pour age=',age)
else :
    print('mineur pour age=',age)
print('suite dans tous les cas')
```

==>

majeur pour age= 20
pas mineur
suite dans tous les cas
mineur pour age= 16
suite dans tous les cas

Une condition peut quelquefois être exprimée comme une combinaison logique de sous condition.

Le mot clef **and** correspond à un "et logique" : les 2 sous conditions doivent être vraies

Le mot clef **or** correspond à un "ou logique" : au moins une des 2 sous conditions doit être vraie

Exemples :

```
age=30
if (age>=18) and (age<=42) :
        print('age entre 18 et 42 ans')
print('suite dans tous les cas')

age=16
if (age<18) or (age>=65) :
        print('enfant ou bien personne agée')
print('suite dans tous les cas')
```

1.5. Listes et fonction len() pour connaître la taille/longueur

Un ***tableau redimensionnable*** (appelé **list** en python) est ***une collection de valeurs consécutives*** dont les **positions** (appelés **indices**) vont de **0 à n-1** .

La fonction prédéfinie **len**(liste) retourne la **taille (ou longueur) d'une liste ou d'un tableau**
La syntaxe *listeXy*[*positionN*] permet d'accéder directement à un élément de la liste .

Exemples :

```
listeDeCouleurs =        [ 'rouge' , 'vert' , 'bleu' , 'noir' , 'blanc' ]
# indices ou positions :      0        1       2       3       4
print('la taille de liste de couleurs est ' , len(listeDeCouleurs) ) # affiche 5
print('la première couleur est ' , listeDeCouleurs[0] ) # affiche rouge
print('la couleurs du milieu est ' , listeDeCouleurs[2] ) # affiche bleu
print('la dernière couleur est ' , listeDeCouleurs[4] ) # affiche blanc
print('en dernier ' , listeDeCouleurs[ len(listeDeCouleurs)-1 ] ) #  blanc
```

1.6. range() , séquence et généralités sur les boucles

Une **séquence** (appelée également ***tuple***) est une **succession de valeur séparée par des virgules** et est assez souvent générée par la fonction **range**()

La fonction **list**(1 , 2 , 3) **construit une liste** (ici **[1, 2, 3]**) **à partir d'une séquence .**

La fonction prédéfinie **range**(**posDebut,posArret,pas**) ***construit la séquence***
posDebut , posDebut+pas , posDebut+2*pas , ...
qui s'arrête lorsque ***posDebut+n*pas*** n'est plus strictement inférieur à ***posArret*** .

Au sein de la fonction range(*posDebut*,**posArret**,*pas*) , le paramètre facultatif *posDebut* a la valeur par défaut *0* s'il n'est pas précisé et le paramètre facultatif *pas* vaut 1 par défaut .

Concrètement :

- La fonction prédéfinie **range**(**n**) construit la séquence **0 , 1 , 2 , ... , n-1**
- **range(3,6)** construit la séquence **3 , ... , 6-1**
- **range(3,-1,-1)** construit la séquence inversée **3 , 2 , 1 , 0**

Exemples :

```
print('range(4)=de 0 à 3<4 =', list(range(4))) # affiche [ 0,1,2, 3 ]
print('range(3,6)=de 3 à 5<6 = ', list(range(3,6))) # affiche [ 3, 4 , 5 ]
print('range(4-1,0-1,-1)= séquence inversée = ', list(range(3,-1,-1))) # affiche [ 3, 2 , 1 , 0 ]
```

En programmation, une boucle permet d'exécuter plusieurs fois un même bloc d'instructions (avec souvent une petite variante) .

En python la boucle ***for*** signifie "***pour chaque éléments de ...***" ou bien "***pour chaque indice dans un certain intervalle***"
La boucle ***while*** signifie "***tant que la condition est vraie***"

1.7. Boucle for (pour chaque élement de ...)

```
#indicesDe0a3= 0,1,2,3
indicesDe0a3= range(4) # range(4) construit 0, 1, 2, 3
for i in indicesDe0a3 :
        print("i=",i)
print('suite apres la boucle')
```

affiche

```
i= 0
i= 1
i= 2
i= 3
suite apres la boucle
```

```
print('boucle de 2 à 8<9 par pas de 2 soit de 2 en 2 :');
for i in range(2,9,2) :
        print("i=",i)
print('suite apres la boucle')
```

==>

```
i= 2
i= 4
i= 6
i= 8
suite apres la boucle
```

```
joursDeLaSemaine = ['lundi','mardi','mercredi','jeudi','vendredi','samedi','dimanche']
# la liste (ou tableau) joursDeLaSemaine comporte 7 éléments dont les indices vont de 0 à 6
for jour in joursDeLaSemaine :
        print(jour,'est un element de joursDeLaSemaine')
print('suite apres la boucle')
```

==>

```
lundi est un element de joursDeLaSemaine
mardi est un element de joursDeLaSemaine
mercredi est un element de joursDeLaSemaine
jeudi est un element de joursDeLaSemaine
vendredi est un element de joursDeLaSemaine
samedi est un element de joursDeLaSemaine
dimanche est un element de joursDeLaSemaine
```

```
#nbJoursDansUneSemaine=7
nbJoursDansUneSemaine=len(joursDeLaSemaine) # la fonction len() retourne la taille/longueur
                                            #(length en englais)
for i in range(nbJoursDansUneSemaine) :
        print("i=",i,"jour=",joursDeLaSemaine[i])
print('suite apres la boucle')
```

==>

```
i= 0 jour= lundi
i= 1 jour= mardi
i= 2 jour= mercredi
i= 3 jour= jeudi
i= 4 jour= vendredi
i= 5 jour= samedi
i= 6 jour= dimanche
```

En exercices :

1. coder et exécuter avec python tous les exemples ci dessus (sans les commentaires et sans copier/coller) pour se familiariser avec la syntaxe .
2. calculer la somme et la moyenne de la liste [12 , 48 , 32 , 8 , 24]

1.8. Boucle while (tant que ...)

```
x=0
while x <= 5 :
        print ('x=',x,'x*x=',x*x)
        x=x+1
print('suite apres la boucle')
```

==>

```
x= 0 x*x= 0
x= 1 x*x= 1
x= 2 x*x= 4
x= 3 x*x= 9
x= 4 x*x= 16
x= 5 x*x= 25
suite après la boucle
```

Attention: de pas oublier x=x+1 sinon boucle infinie qui ne s'arrête jamais !!!

```
listeCouleurs = [ "rouge" , "noir" ]
listeCouleurs.append("vert") # ajoute l'élément "vert" à la liste
listeCouleurs.append("bleu") # ajoute l'élément "bleu" à la liste
print("listeCouleurs=",listeCouleurs); # ['rouge', 'noir', 'vert', 'bleu']
```

```
listeValeurs = []
valeurOuFin=input("val=")
while (valeurOuFin != 'fin' ) and (valeurOuFin != '' ) :
        nouvelleValeur = float(valeurOuFin)
        listeValeurs.append(nouvelleValeur)
        valeurOuFin=input("val=")
print('listeValeurs=',listeValeurs)
```

==>

```
val=4
val=6
val=8
val=
listeValeurs= [4.0, 6.0, 8.0]
```

En exercices :

3. coder et exécuter avec python tous les exemples ci dessus (sans les commentaires et sans copier/coller) pour se familiariser avec la syntaxe .
4. calculer la somme et la moyenne de la liste [12 , 48 , 32 , 8 , 24] sans utiliser la boucle for mais en utilisant la boucle while et en initialisant une variable i=0 avant la boucle .
5. transformer ["hiver" , "printemps" , "ete" , "automne"] en liste de valeurs en majuscules via une boucle au choix (for ou while) puis afficher toute la liste transformée .

IV - fonctions python

1. Titre_Section1 du chapitre englobant

Texte ....

1.1. Titre Paragraphe

Texte ....

ANNEXES

V - Annexe – Bibliographie, Liens WEB + TP

1. Bibliographie et liens vers sites "internet"

2. TP