

# Introduction à la programmation Python

Dr. Yousfi Souheib

10 février 2021

## Algorithmme

- Solution « informatique » relative à un problème
- Suite d'actions (instructions) appliquées sur des données
- 3 étapes principales :
  - 1 Saisie (réception) des données
  - 2 Traitements
  - 3 Restitution (application) des résultats

## Programme

- Transcription d'un algorithme avec une syntaxe prédéfinie
- Python
- Même principes fondamentaux que les autres langages objets (Java, C#)
- Python s'enrichit de bibliothèques de calcul spécialisées (mathématique, crypto)

# Mode compilé vs. mode interprété

- Langage interprété : + portabilité application ; - lenteur (R, VBA, Python...)
- Langage compilé : + rapidité ; - pas portable (solution possible : write once, compile anywhere ; ex. Lazarus(a professional open-source cross platform IDE powered by Free Pascal))
- Langage pseudo-compilé : + portabilité plate-forme ; - lenteur( ?) (principe : write once, run anywhere ; ex. Java)

## Definition

Python est interprété, il est irrémédiablement lent, mais... on peut lui associer des bibliothèques intégrant des fonctions compilées qui, elles, sont très rapides.

# Étapes de la conception d'un programme (Génie Logiciel)

- ❶ **Déterminer les besoins et fixer les objectifs** : que doit faire le logiciel, dans quel cadre va-t-il servir, quels seront les utilisateurs types ? On rédige un cahier des charges avec le commanditaire du logiciel (Remarque : commanditaire = maître d'ouvrage ; réalisateur = maître d'œuvre)
- ❷ **Conception et spécifications** : quels sont les fonctionnalités du logiciel, avec quelle interface ?
- ❸ **Programmation** : modélisation et codage
- ❹ **Tests** : obtient-on les résultats attendus, les calculs sont corrects, y a-t-il plantage et dans quelles circonstances ? (tests unitaires, tests d'intégration, etc.)
- ❺ **Déploiement** : installer le chez le client (vérification des configurations, installation de l'exécutable et des fichiers annexes, etc.)
- ❻ **Maintenance** : corrective, traquer les bugs et les corriger (patches) ; évolutive (ajouter des fonctionnalités nouvelles au logiciel : soit sur l'ergonomie, soit en ajoutant de nouvelles procédures)

## PROGRAMMER EN PYTHON

- 1 Python est un langage de programmation interprété. Il est associé à un interpréteur de commandes disponible pour différents OS (Windows, Linux, Mac OS X, etc.)
- 2 C'est un « vrai » langage c.-à-d. types de données, branchements conditionnels, boucles, organisation du code en procédures et fonctions, objets et classes, découpage en modules. Très bien structuré, facile à appréhender.
- 3 Mode d'exécution : transmettre à l'interpréteur Python le fichier script « .py »

## Remarque

Python est associé à de très nombreuses bibliothèques très performantes, notamment des bibliothèques de calcul scientifique.

```
from Crypto.Hash import SHA256  
from Crypto.Cipher import AES
```

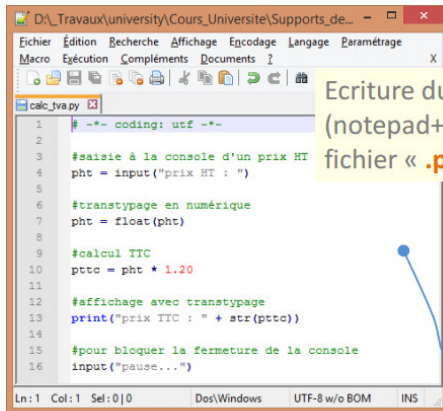
# Python propose les outils standards de la programmation I

- 1 **Données typées.** Python propose les types usuels de la programmation : entier, réels, booléens, chaîne de caractères.
- 2 **Structures avancées de données.** Gestion des collections de valeurs (énumérations, listes) et des objets structurés (dictionnaires, classes)
- 3 **Séquences d'instructions** : c'est la base même de la programmation, pouvoir écrire et exécuter une série de commandes sans avoir à intervenir entre les instructions.
- 4 **Structures algorithmiques** : les branchements conditionnels et les boucles.

# Python propose les outils standards de la programmation II

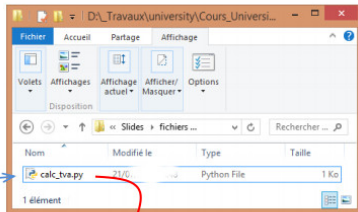
- ❶ **Les outils de la programmation structurée** : pouvoir regrouper du code dans des **procédures** et des **fonctions**. Cela permet de mieux organiser les applications.
- ❷ **Organisation du code en modules**. Fichiers « .py » que l'on peut appeler dans d'autres programmes avec la commande `import`.
- ❸ **Python est « case sensitive »**, il différencie les termes écrits en minuscule et majuscule. Des conventions de nommage existent. Mais le plus important est d'être raccord avec l'environnement de travail dans lequel vous opérez.

# Éditeur de texte

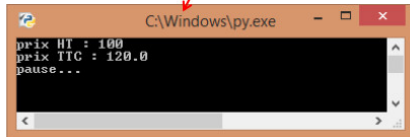


```
1 # -*- coding: utf -*-
2
3 #saisie à la console d'un prix HT
4 pht = input("prix HT : ")
5
6 #transtypage en numérique
7 pht = float(pht)
8
9 #calcul TTC
10 pttc = pht * 1.20
11
12 #affichage avec transtypage
13 print("prix TTC : " + str(pttc))
14
15 #pour bloquer la fermeture de la console
16 input("pause...")
```

Ecriture du code dans un éditeur de code (notepad++) puis l'enregistrer dans un fichier « .py »



Double cliquer le fichier « .py » pour lancer automatiquement le programme dans la console.



```
C:\Windows\py.exe
prix HT : 100
prix TTC : 120.0
pause...
```



Pour lancer le programme

C'est le mode de fonctionnement  
que nous allons privilégier !

The screenshot displays the Spyder Python IDE interface. The main window is titled 'Spyder (Python 3.4)' and contains several panes:

- Code Editor:** Displays a Python script named 'calcul TVA.py'. The code calculates the TTC (Total Tax Inclusive) price based on the HT (Total Tax Exclusive) price. The code is as follows:

```
1 # -*- coding: utf -*-
2
3 #saisie à la console d'un prix HT
4 pht = input("prix HT : ")
5
6 #transtypage en numérique
7 pht = float(pht)
8
9 #calcul TTC
10 pttc = pht * 1.20
11
12 #affichage avec transtypage
13 print("prix TTC : " + str(pttc))
14
15 #fin du programme
16 #inutile de mettre input("pause") ici
```
- Variable Explorer:** Shows the variable 'print' selected, with its definition and type (Function of builtins module).
- IPython Console:** Shows the execution of the code. The output is:

```
In [4]:
runfile('D:/Travaux/universite/CC.../calcul TVA.py',
wdir='D:/Travaux/universite/CC.../calcul TVA.py')

prix HT : 100
prix TTC : 120.0

In [5]:
```

Yellow callout boxes provide additional context:

- Pour lancer le programme:** Points to the 'Exécuter le fichier (F5)' button in the toolbar.
- C'est le mode de fonctionnement que nous allons privilégier !:** Points to the 'Exécuter le fichier (F5)' button.
- Editeur de code:** Points to the code editor pane.
- Informations, dont l'aide (CTRL+I sur les mots clés):** Points to the variable explorer pane.
- Console IPython Sorties + interaction avec l'utilisateur:** Points to the IPython console pane.

# Types de données, variables, opérations

## Affectation – Typage automatique

➤ `a = 1.2`

`a` est une variable, en interne elle a été automatiquement typée en flottant « float » parce qu'il y a un point décimal. `a` est l'identifiant de la variable (attention à ne pas utiliser les mots réservés comme identifiant), `=` est l'opérateur d'affectation

## Calcul

➤ `d = a + 3`

`d` sera un réel contenant la valeur 4.2

## Forcer le typage d'une variable (sert aussi pour le transtypage)

➤ `b = float(1)`

Même sans point décimal, `b` sera considéré comme float (`b = 1`, il aurait été int dans ce cas).

## Connaître le type d'un objet

➤ `type(nom_de_variable)`

Affiche le type interne d'une variable (ex. `type(a)` → `<class 'float'>`)

## Supprimer un objet de la mémoire

➤ `del nom_de_variable`

où `nom_de_variable` représente le nom de l'objet à supprimer.

# Types élémentaires de Python

- Numérique qui peut être **int** (entier) ou **float** (double). Les opérateurs applicables sont : `+` , `-` , `*` , `/` (division réelle) , `**` (puissance) , `%` (modulo) , `//` (division entière)
- **bool** correspond au type booléen, il prend deux valeurs possibles **True** et **False** (respecter la casse). Les opérateurs sont **not** (négation), **and** (ET logique), **or** (OU logique)

ex. `not(True)` → `False` ; `True and False` → `False` ; etc.

- **str** désigner les chaînes de caractères. Une constante chaîne de caractère doit être délimitée par des guillemets (ou des quotes)

ex. `a ← « tano »` affecte la valeur « tano » à l'objet `a` qui devient donc une variable de type chaîne de caractères. Une chaîne de caractère se comporte comme un vecteur : `len()` pour connaître sa longueur, `a[0]` → « t », `a[1:3]` → « ano », `a[2:]` → « no », etc.

- Remarque : pour connaître la classe d'un objet i.e. le type associé à un objet, on utilise la fonction **`type(nom_objet)`**

ex. `type(1.2)` → renvoie la valeur 'float'

## Affectation simple

La seconde évite les ambiguïtés.

```
#typage automatique  
a = 1.0  
#typage explicite  
a = float(1)
```

## Affectations multiples

Pas fondamental

```
#même valeur pour plusieurs variables  
a = b = 2.5  
  
#affectations parallèles  
a, b = 2.5, 3.2
```

Les opérateurs de comparaison servent à comparer des valeurs de même type et renvoient un résultat de type booléen.

Sous Python, ces opérateurs sont `<`, `<=`, `>`, `>=`, `!=`, `==`

ex. `a = (12 == 13)` # a est de type bool, il a la valeur False

N.B. On utilisera principalement ces opérateurs dans les branchements conditionnels.

## Saisie

```
a = input (« Saisir votre valeur »)
a = float(a)
```

`input()` permet d'effectuer une saisie au clavier

Il est possible d'afficher un message d'invite

La fonction renvoie toujours une chaîne, il faut convertir

## Affichage

```
#Affichage explicite
print(a)
```

- Un affichage multiple est possible

Ex. `print(a,b)` #affiche **a** et **b**

- L'affichage direct du contenu d'un tableau (liste) est possible également.

# Exemple

```
calc_tva.py - D:\Travaux\university\Co...
File Edit Format Run Options Window Help
# -*- coding: utf -*-

#saisie à la console d'un prix HT
pht = input("prix HT : ")

#transtypage en numérique
pht = float(pht)

#calcul TTC
pttc = pht * 1.20

#affichage avec transtypage
print("prix TTC : " + str(pttc))

#pour bloquer la fermeture de la console
input("pause...")
Ln: 1 Col: 0
```

Pour gérer correctement  
l'affichage des caractères  
accentués

`print("« prix ttc : », pttc)`  
Fonctionne également

Concaténation de 2 chaînes de caractères

# Branchement conditionnel « if »

Condition est très souvent une opération de comparaison



```
if condition:
    bloc d'instructions
else:
    bloc d'instructions
```

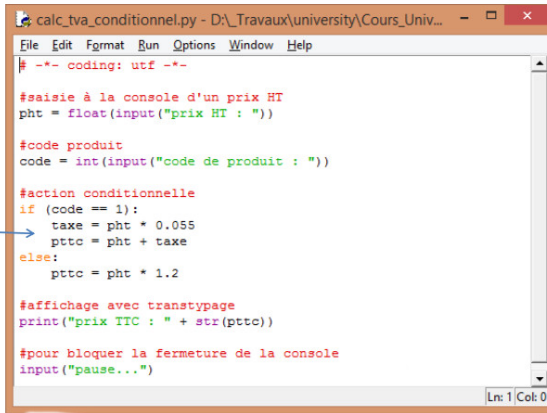
- (1) Attention au **:** qui est primordial
- (2) C'est l'**indentation** (le décalage par rapport à la marge gauche) qui délimite le bloc d'instructions
- (3) La partie **else** est facultative



# Branchement conditionnel « if » (exemple)

Noter l'imbrication des blocs.

Le code appartenant au même bloc doit être impérativement aligné sinon erreur.



```
calc_tva_conditionnel.py - D:\Travaux\university\Cours_Univ...
File Edit Format Run Options Window Help
# -*- coding: utf -*-

#saisie à la console d'un prix HT
pht = float(input("prix HT : "))

#code produit
code = int(input("code de produit : "))

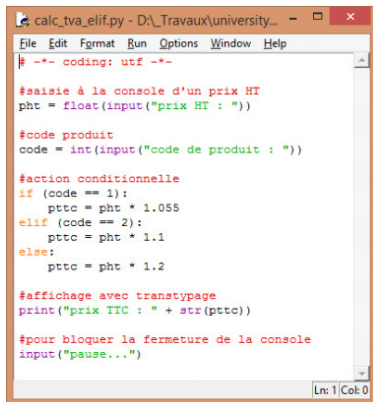
#action conditionnelle
if (code == 1):
    taxe = pht * 0.055
    pttc = pht + taxe
else:
    pttc = pht * 1.2

#affichage avec transtypage
print("prix TTC : " + str(pttc))

#pour bloquer la fermeture de la console
input("pause...")

Ln: 1 Col: 0
```

# Succession de if avec elif



```
calc_tva_elif.py - D:\Travaux\university...  
File Edit Format Run Options Window Help  
# -*- coding: utf-8 -*-  
  
#saisie à la console d'un prix HT  
pht = float(input("prix HT : "))  
  
#code produit  
code = int(input("code de produit : "))  
  
#action conditionnelle  
if (code == 1):  
    pttc = pht * 1.055  
elif (code == 2):  
    pttc = pht * 1.1  
else:  
    pttc = pht * 1.2  
  
#affichage avec transtypage  
print("prix TTC : " + str(pttc))  
  
#pour bloquer la fermeture de la console  
input("pause...")  
Ln: 1 Col: 0
```

- **elif** n'est déclenché que si la (les) condition(s) précédente(s) a (ont) échoué.
- **elif** est situé au même niveau que **if** et **else**
- On peut en mettre autant que l'on veut



Il n'y a pas de `switch()` ou de `case...of` en Python

# Avant la boucle « for » : génération d'une séquence de valeurs

## Principe de la boucle for

Elle ne s'applique que sur une collection de valeurs. Ex. tuples, listes,... à voir plus tard.

## Suite arithmétique simple (séquence de valeurs entières)

On peut définir des boucles indicées en générant une collection de valeurs avec `range()`


(1) `range(4)` → 0 1 2 3

(2) `range(1, 4)` → 1 2 3

(3) `range(0, 5, 2)` → 0 2 4

# La boucle « for »

Séquence est une collection de valeurs  
Peut être générée avec range()



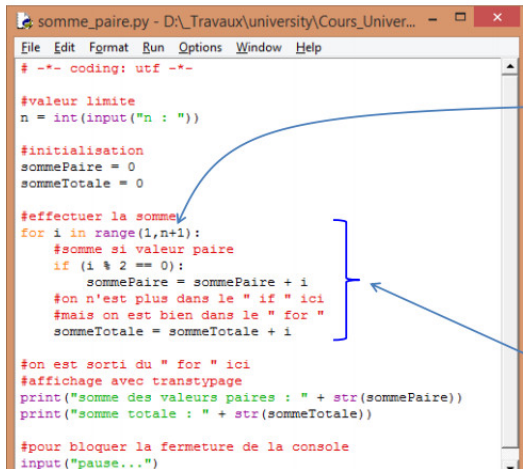
```
for indice in séquence:  
    bloc d'instructions
```

Remarque :

- Attention à l'indentation toujours
- On peut « casser » la boucle avec **break**
- On peut passer directement à l'itération suivante avec **continue**
- Des boucles imbriquées sont possibles
- Le bloc d'instructions peut contenir des conditions

# La boucle « for » (exemple)

Somme totale des valeurs comprises entre 1 et **n** (inclus) et somme des valeurs paires dans le même intervalle



```

somme_paire.py - D:\Travaux\university\Cours_Univer...
File Edit Format Run Options Window Help
# -*- coding: utf -*-

#valeur limite
n = int(input("n : "))

#initialisation
sommePaire = 0
sommeTotale = 0

#effectuer la somme
for i in range(1,n+1):
    #somme si valeur paire
    if (i % 2 == 0):
        sommePaire = sommePaire + i
    #on n'est plus dans le " if " ici
    #mais on est bien dans le " for "
    sommeTotale = sommeTotale + i

#on est sorti du " for " ici
#affichage avec transtypage
print("somme des valeurs paires : " + str(sommePaire))
print("somme totale : " + str(sommeTotale))

#pour bloquer la fermeture de la console
input("pause...")


```

Il faut mettre **n+1** dans **range()** pour que **n** soit inclus dans la somme

Observez attentivement les indentations.

# La boucle « while »

Opération de comparaison  
Attention à la boucle infinie !



```
while condition:  
    bloc d'instructions
```

Remarque :

- Attention à l'indentation toujours
- On peut « casser » la boucle avec **break**

# La boucle « while » (exemple)

```
somme_paire_while.py - D:/Travaux/university/Co... - x
File Edit Format Run Options Window Help
# -*- coding: utf -*-

#valeur limite
n = int(input("n : "))

#initialisation
sommePaire = 0
sommeTotale = 0

#effectuer la somme
i = 1
while (i <= n):
    #somme si valeur paire
    if (i % 2 == 0):
        sommePaire = sommePaire + i
    #somme toujours, paire ou pas
    sommeTotale = sommeTotale + i
    #incréméntation
    i = i + 1

#affichage avec transtypage
print("somme des valeurs paires : " + str(sommePaire))
print("somme totale : " + str(sommeTotale))

#pour bloquer la fermeture de la console
input("pause...")
```

Ne pas oublier  
l'initialisation de **i**

Observez attentivement  
les indentations.