



R107

Bases de la programmation

Ismail Bennis

Ismail.bennis@uha.fr

MCF, IUT de Colmar, Département Réseaux & Télécoms, bureau N°006
34 rue du Grillenbreit - 68000 Colmar Cedex



www.uha.fr



Introduction à la programmation Python

Historique de Python














- **Python** est un langage de programmation apparu officiellement en **1991**. Guido van Rossum l'a conçu à la fin des années 1980 à l'institut national de recherche en mathématique et informatique aux Pays-Bas.
- Le nom du langage étant inspiré de la comédie britannique '**Monty Python**'
- Plusieurs versions ont vu le jour; la dernière version stable est **3.9.7**



Classement de Python

- Top 10 des meilleurs langages pour le développement d'applications d'entreprise, de bureau et d'applications scientifiques

Language Ranking: IEEE Spectrum

Rank	Language	Type	Score
1	Python	  	100.0
2	Java	  	96.3
3	C	  	94.4
4	C++	  	87.5
5	R		81.5

***IEEE 2019**

Oct 2021	Oct 2020	Change	Programming Language		Ratings	Change
1	3	▲		Python	11.27%	-0.00%
2	1	▼		C	11.16%	-5.79%
3	2	▼		Java	10.46%	-2.11%
4	4			C++	7.50%	+0.57%

Baromètre TIOBE – Octobre 2021

Caractéristiques de base de Python

- Langage de programmation **interprété** : associé à un interpréteur de commandes disponible pour différents OS (Windows, Linux, Mac OS X, etc.)
- Très bien structuré, facile à appréhender, c'est un langage privilégié pour l'enseignement.
- Doté d'un **typage dynamique** : pas de déclaration explicite de variables ni de spécification de types; le type d'une variable est déterminé à l'exécution seulement de même que l'allocation de l'espace mémoire.
- Il gère lui-même **l'espace mémoire** disponible.
- Il offre des possibilités pour la programmation orientée objets.
- Python est « **case sensitive** », il différencie les termes écrits en minuscule et majuscule.
- **Universel, compact, moderne, ouvert, simple, portable, extensible.**

Installation et environnement de travail



The image shows the top section of the Python.org website. It features the Python logo and name on the left, a search bar with a magnifying glass icon and a 'GO' button in the center, and a 'Donate' button on the right. Below this is a navigation bar with links: About, Downloads, Documentation, Community, Success Stories, News, and Events. The main content area has a dark blue background with a large illustration of two parachutes carrying boxes. The text 'Download the latest version for Windows' is prominently displayed in yellow. Below it, a yellow button says 'Download Python 3.9.6'. Further down, there are links for other operating systems (Linux/UNIX, macOS, Other), pre-releases, Docker images, and a link to see specific releases for Python 2.7.

python™

Donate Search GO Socialize

About Downloads Documentation Community Success Stories News Events

Download the latest version for Windows

Download Python 3.9.6

Looking for Python with a different OS? Python for [Windows](#), [Linux/UNIX](#), [macOS](#), [Other](#)

Want to help test development versions of Python? [Prereleases](#), [Docker images](#)

Looking for Python 2.7? See below for specific releases



The image shows the 'Python 3.9.6 (64-bit) Setup' window. It has a title bar with the text 'Python 3.9.6 (64-bit) Setup'. The main content area is titled 'Install Python 3.9.6 (64-bit)' and contains the text 'Select Install Now to install Python with default settings, or choose Customize to enable or disable features.' There are two main options: 'Install Now' and 'Customize installation'. The 'Install Now' option is selected and shows the installation path 'C:\Users\IB\AppData\Local\Programs\Python\Python39', along with the text 'Includes IDLE, pip and documentation' and 'Creates shortcuts and file associations'. The 'Customize installation' option is also visible with the text 'Choose location and features'. At the bottom, there are two checked options: 'Install launcher for all users (recommended)' and 'Add Python 3.9 to PATH'. A 'Cancel' button is located at the bottom right. The left side of the window features the Python logo and the text 'python for windows'.

Python 3.9.6 (64-bit) Setup

Install Python 3.9.6 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

 **Install Now**
C:\Users\IB\AppData\Local\Programs\Python\Python39
Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ **Customize installation**
Choose location and features

☒ Install launcher for all users (recommended)
☒ Add Python 3.9 to PATH

Cancel

python
for
windows

Installation et environnement de travail

The screenshot illustrates the process of setting up the environment for Python 3.9.6 on a Windows system. It shows several overlapping windows:

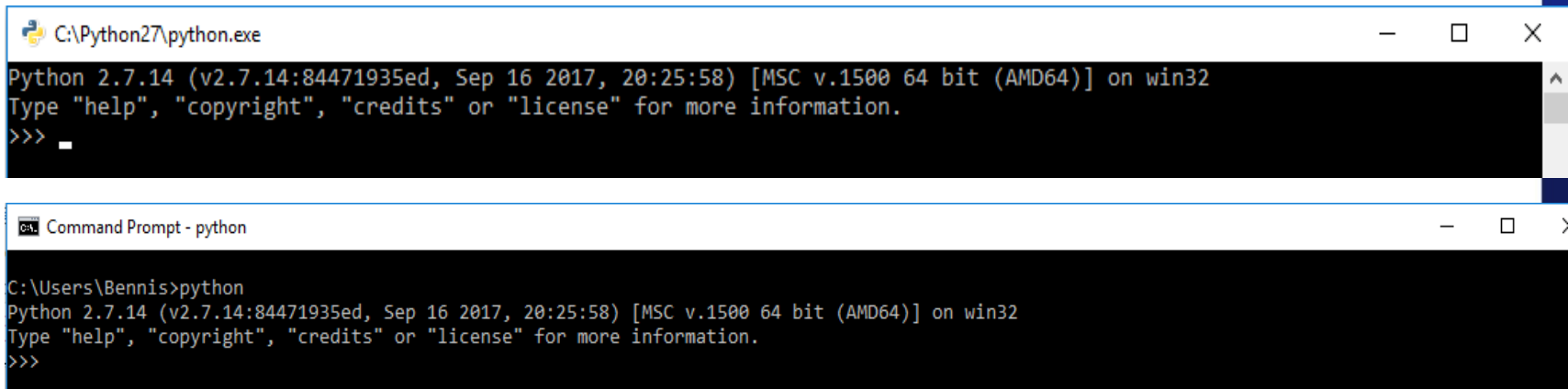
- System Properties**: The 'Advanced' tab is selected, and the 'Environment Variables...' button is highlighted with a red box.
- Environment Variables**: This window shows 'User variables for Bennis' and 'System variables'. The 'Path' variable under 'System variables' is highlighted with a red box.
- Edit environment variable**: This window shows the 'Path' variable being edited. The path 'C:\Python27' is highlighted with a red box.
- Variables d'environnement**: This window shows the 'Variables utilisateur pour IB' and 'Variables système'. The 'Path' variable is highlighted with a red box.
- Modifier la variable d'environnement**: This window shows the 'Path' variable being modified. The path 'C:\Users\IB\AppData\Local\Programs\Python\Python39\Scripts\' is highlighted with a red box.

At the bottom left, a terminal window shows the command to verify the installed version of Python:

```
C:\Users\IB>python -V
Python 3.9.6
```

Interpréteur interactif

- L'interpréteur interactif permet d'écrire et d'exécuter du code Python à la volée, de faire des tests rapides, d'obtenir facilement des informations sur une fonction ou un module.
- La distribution standard de Python en propose 2:
 - **Shell interactif:** console Python la plus basique



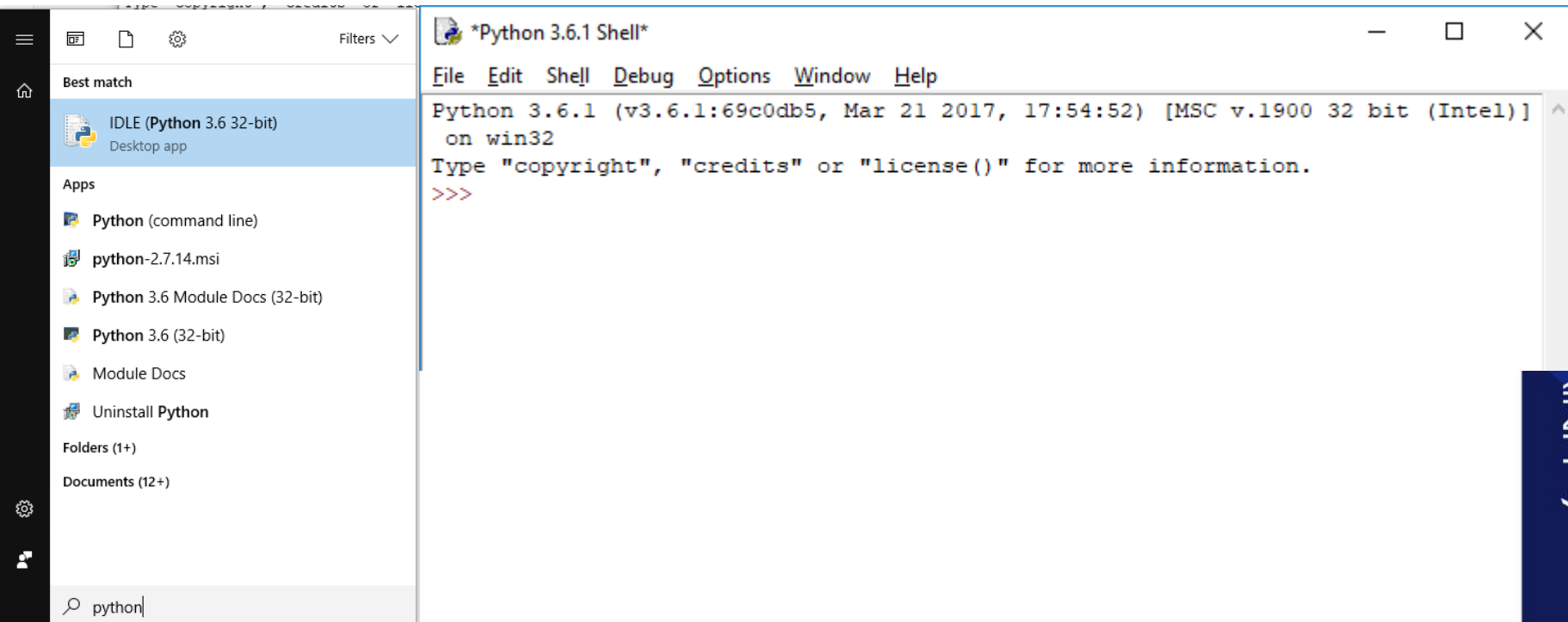
```
C:\Python27\python.exe
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

```
Command Prompt - python
C:\Users\Bennis>python
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

➔ Ce n'est pas adapté pour la programmation = enchaînement automatique d'instructions

- **IDLE:** un environnement REPL (Read-Evaluate-Print-Loop)

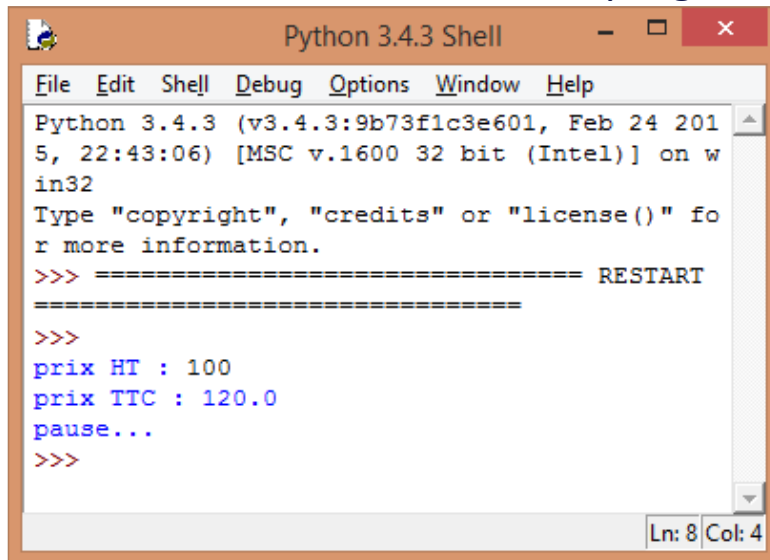
Interpréteur interactif - IDLE



- Le signe >>> est l'invite de commande
- Vous pouvez écrire et exécuter de la même façon qu'avec le shell
- Avantage : interface est un peu plus élaboré + fonctionnalités intéressantes : coloration syntaxique, l'autocomplétion (avec la touche Tabulation)

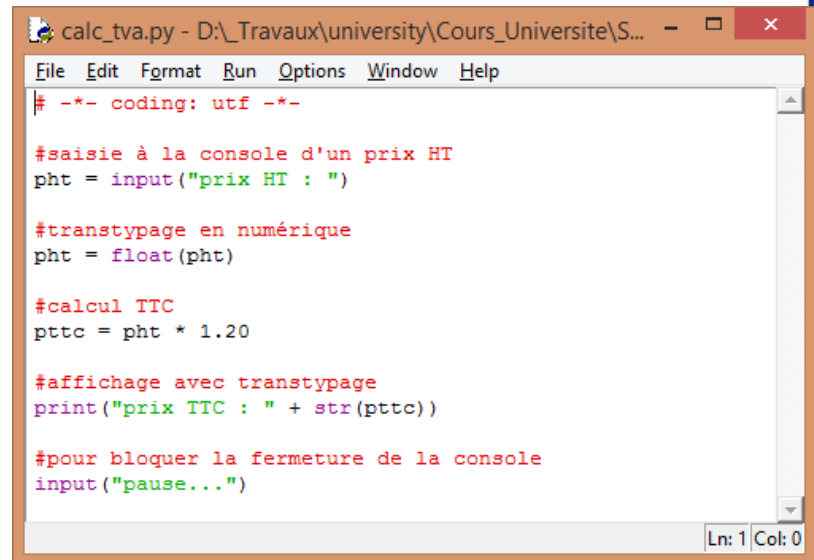
Ecrire et exécuter un script Python

Shell : fenêtre d'exécution du programme



```
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART
>>>
prix HT : 100
prix TTC : 120.0
pause...
>>>
```

Editeur de code



```
calc_tva.py - D:\Travaux\university\Cours_Universite\S...
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

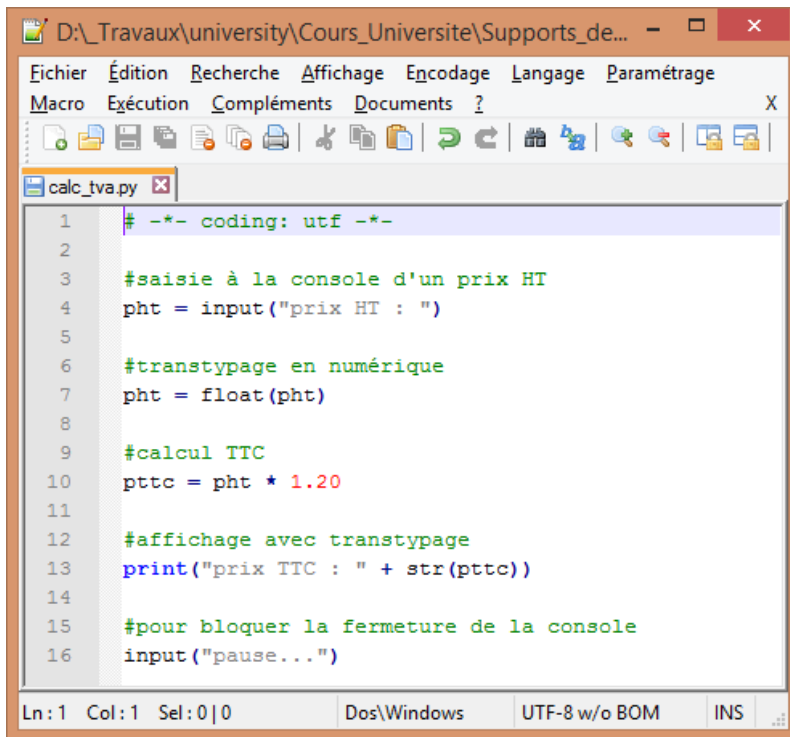
#affichage avec transtypage
print("prix TTC : " + str(pttc))

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

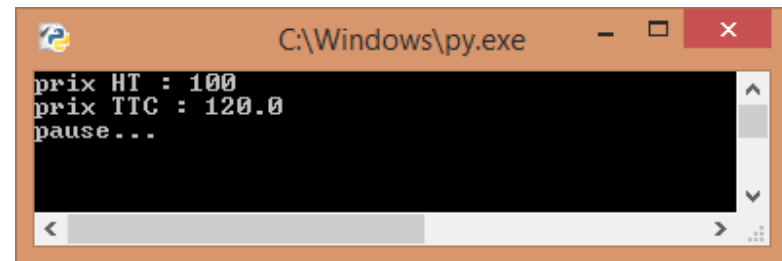
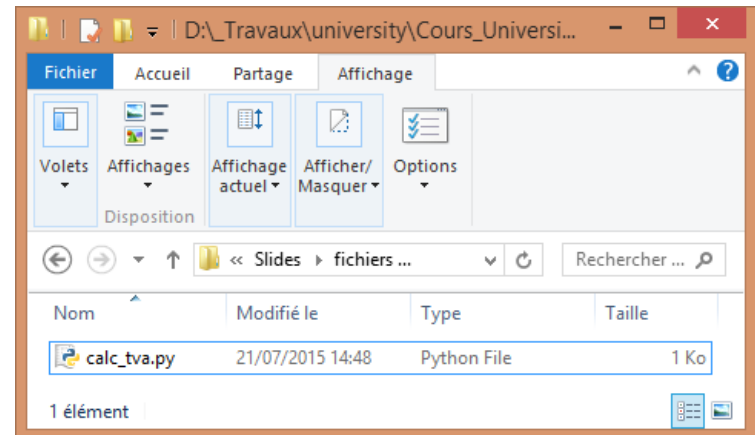
- Avec l'IDLE, aller dans le menu **File - New Window**, écrire le programme puis l'enregistrer avec l'extension **.py**
- Exécuter : Menu : RUN / RUN MODULE (ou raccourci clavier F5)
- Permet de mieux suivre l'exécution du programme. Messages d'erreur accessibles, pas comme pour l'exécution console

Ecrire et exécuter un script Python

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



```
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...")
```

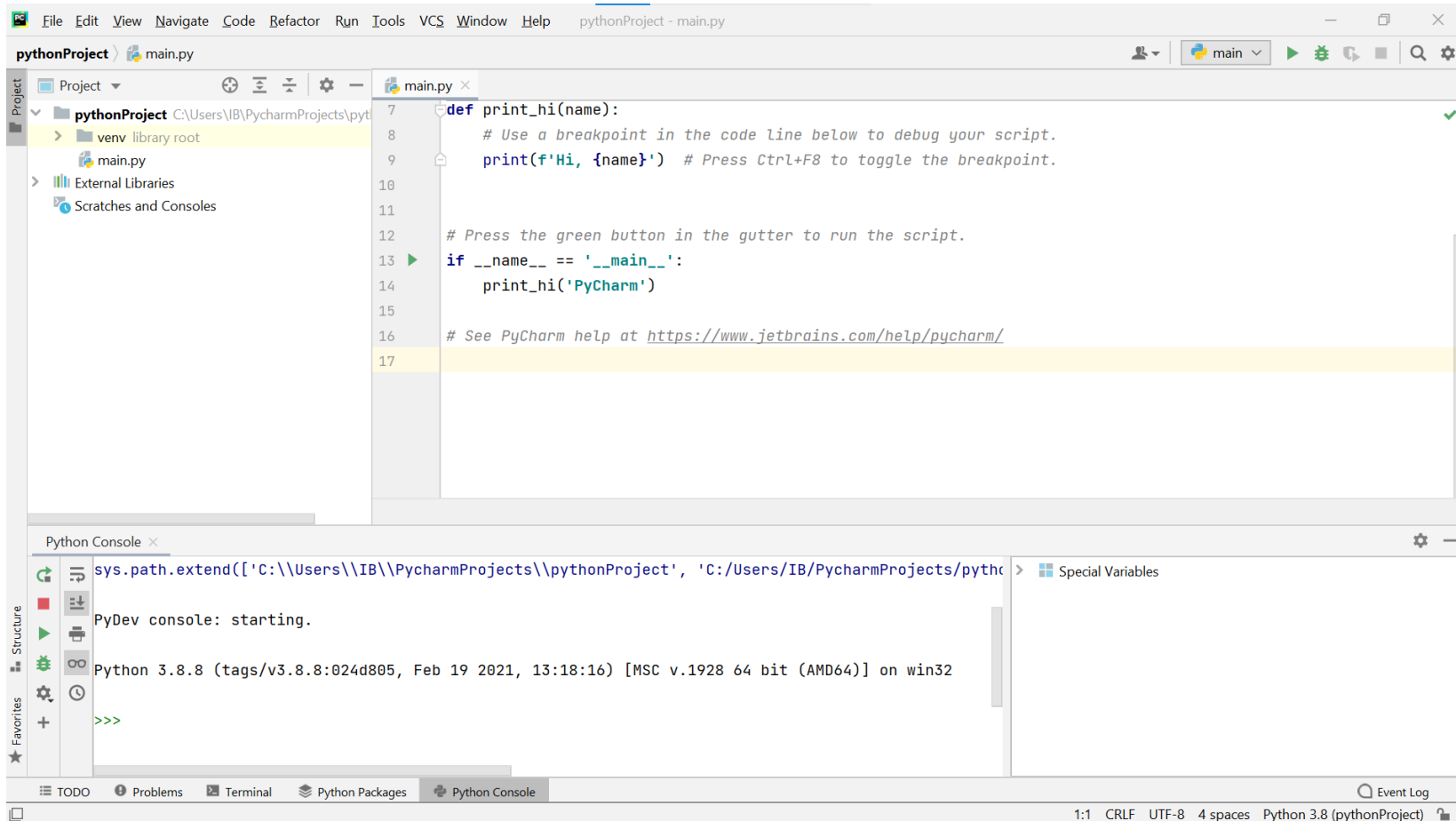


➔ Double cliquer le fichier « .py » pour lancer automatiquement le programme dans la console.

Ecrire et exécuter un script Python



- On peut utiliser un IDE spécialisé pour programmer en Python. Ex : Pycharm



Ecrire et exécuter un script Python

- Si votre fichier **.py** est associé à **python.exe**, un double-clic sur le fichier lancera le programme. En revanche, la console Python se refermera automatiquement à la fin du programme si aucune instruction ne bloque l'exécution
- On peut ajouter à la fin du programme l'instruction ***input("Appuyer sur une touche pour quitter")***, qui bloquant le programme jusqu'au moment où l'utilisateur appuie sur une touche.
- On peut également créer un fichier **.bat** dans le même répertoire que le fichier python avec le contenu suivant :
 - Nom_fichier.py
 - pause

www.uha.fr



Bases de Python : calcul, types de données, variables, opérations, affichage

Calcul arithmétique

- Une des premières fonctionnalités d'un interpréteur est de faire des calculs

```
>>> 1-10
```

```
-9
```

```
>>> 2*10
```

```
20
```

```
>>> (50-5*6) / 4
```

```
5.0
```

```
>>> 100/4
```

```
25
```

```
>>> 17//3 #On demande un quotient entier
```

```
5
```

```
>>> 10%4 #le reste de la division
```

```
2
```

```
>>> 2**3 # Puissances
```

```
8
```

Les variables

- Une **variable** est une zone de la mémoire dans laquelle une **valeur** est stockée. Elle est définie par un **nom** (qui doit respecter des règles), alors que pour l'ordinateur, il s'agit en fait d'une adresse (*i.e.* une zone particulière de la mémoire).
- En Python, la **déclaration** d'une variable et son **initialisation** se font en même temps

```
>>> x = 2
```

```
>>> x
```

2

```
>>> a=5 # a est un entier
```

```
>>> b=7.2 # b est un nombre flottant
```

```
>>> a+b
```

12.2

```
>>> a*b
```

36.0

Les variables - Nommage

- Pour nommer une variable vous devez utiliser que les lettres de l'alphabet, les chiffres et le caractère "_" et "-". N'utilisez pas les accents, ni les signes de ponctuation ou le signe @.
- De plus les chiffres ne doivent jamais se trouver en première position dans votre variable, de même pour "_"
- Vous ne pouvez pas utiliser d'espace dans un nom de variable.
- Python est sensible à la casse, ce qui signifie que les variables **Test**, **test** ou **TEST** sont différentes.
- Liste des mots réservés par python :

**print in and or if del for is raise assert elif from lambda return break else
global not try class except while continue exec import pass yield def finally**

Les affectations

- 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 le mots réservés comme identifiant), `=` est l'opérateur d'affectation

- Typage explicite 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)
```

- Affectations multiples

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

- La plus couramment utilisée : 1 instruction = 1 ligne

```
>>> a = 1  
>>> b = 5  
>>> c = a+b
```


Les affectations

- Quelques exemples d'affectation

```
>>> age = 30
>>> age = age + 10
>>> age
```

40

```
>>> age2 = age # on peut mettre une variable dans une autre
variable.
```

40

```
>>> messgae = "bonjour"
>>> messgae = message + " tout le monde"
>>> messgae
```

'bonjour tout le monde'

```
>>> msg = "Bonjour, je m'appelle \"Ismail\"" #Echapper les
quotes
```

```
>>> msg
```

'Bonjour, je m'appelle "Ismail"'

Les affectations

- Si on a $x = y - 3$, l'opération $y - 3$ est d'abord évaluée et ensuite le résultat de cette opération est affecté à la variable x .
- Raccourcis pour utiliser en 1er opérande la variable de gauche $+=$ $-=$ $*=$ $/=$

```
>>> age = 30
```

```
>>> age += 10
```

```
>>> age
```

```
40
```

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)
- **Str** désigner les chaînes de caractères. Une chaîne de caractère doit être délimitée par des guillemets (ou des quotes)
- Remarque : pour connaître la classe d'un objet i.e. le type associé à un objet, on utilise la fonction **type(nom_objet)**.

```
>>> v = 3.2
>>> type(v)
<type 'float'>
```

Opérations sur les chaînes de caractères

- L'opérateur d'addition + permet de concaténer (assembler) deux chaînes de caractères et l'opérateur de multiplication * permet de dupliquer plusieurs fois une chaîne.

```
>>> messgae = "bonjour"
>>> messgae = message + " tout le monde"
>>> messgae
'bonjour tout le monde'
>>> messgae = "Salut"
>>> message + " tout le monde"
'Salut tout le monde'
>>> message * 3
'SalutSalutSalut'
```

- Opérations illicites

```
>>> 'toto' + 2
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    'toto' + 2
TypeError: must be str, not int
```

Opérations sur les chaînes de caractères

- Séquences d'échappement : pour insérer des guillemets " à l'intérieur d'une chaîne, Python propose deux moyens :
 - commencer et terminer une chaîne avec des apostrophes. Ex. 'Bonjour " tout le monde "'
 - placer une barre oblique inversée suivie du guillemet ou de l'apostrophe (\ " or \ '). Les barres obliques inversées protègent les guillemets mais ne sont pas affichées.

```
>>> print('L\'exemple avec un apostrophe.')
```

```
L'exemple avec un apostrophe.
```

```
>>> print("Voici un \"échappement\" de guillemets")
```

```
Voici un "échappement" de guillemets
```


Transtypage

- **Conversion en numérique**
 - `a = "12"` # a est de type chaîne caractère
 - `b = float(a)` #b est de type float
 - N.B. Si la conversion n'est pas possible ex. `float("toto")` , Python renvoie une erreur
- **Conversion en logique**
 - `a = bool("TRUE")` # a est de type bool est contient la valeur True
 - `a = bool(1)` # renvoie True également
- **Conversion en chaîne de caractères**
 - `a = str(15)` # a est de type chaîne et contient «15»

Les opérateurs en Python

Principaux opérateurs

symb.	signification	symb.	signification
+	Addition pour <code>int</code> et <code>float</code> concaténation pour <code>str</code>	*	multiplication pour <code>int</code> et <code>float</code> répétition pour <code>str</code>
-	soustraction pour <code>int</code> et <code>float</code>	/	division (décimale)
%	modulo ¹	//	division entière
**	exposant ²	<code>not</code>	négation logique
<code>and</code>	« et » logique	<code>or</code>	« ou » logique

- Les opérateurs de comparaison servent à comparer des valeurs de même type et renvoient un résultat de type booléen.
 - ex. `a = (12 == 13)` # a est de type bool, il a la valeur False

priorité	opérateur	description
basse	<code>or</code>	ou logique
	<code>and</code>	et logique
	<code>not</code>	non logique
	<code><</code> , <code>></code> , <code><=</code> , <code>>=</code> , <code>!=</code> , <code><></code> , <code>==</code>	comparaison
	<code> </code>	ou bit à bit
	<code>^</code>	ou exclusif bit à bit
	<code>&</code>	et bit à bit
	<code><<</code> , <code>>></code>	décalage
	<code>+</code> , <code>-</code>	addition et soustraction
	<code>*</code> , <code>/</code> , <code>//</code> , <code>%</code>	multiplication, division, reste
	<code>+x</code> , <code>-x</code> , <code>~x</code>	signe positif, signe négatif, non bit à bit
	<code>**</code>	exposant
haute	<code>(expression)</code>	expression parenthésée

Types avancés de Python

▪ Tuples

- Ce sont des séquences qui ont des éléments hétérogènes
- ils ne peuvent pas être modifiés
 - donc, copie si modification nécessaire

```
tuple=('un',1,'deux',2)
tuple[2]='3'
```

```
-----
exceptions.TypeError
```

```
Traceback (most recent call last)
```

▪ Listes

- séquence modifiable, se crée avec des []
 - liste=[] # vide
 - liste = range(0,100) # génère 0, 1, 2, 3, ... 99
- Opérations intéressantes:
 - liste.append(element)
 - liste.extend(liste2)
 - liste.insert(position,element)
 - liste.remove(element)
 - liste.pop(position)
 - liste.index(element)
 - liste.count(element)
 - liste.sort()

Types avancés de Python

- Les **dictionnaires** ou hash arrays ou associative arrays. Ce sont des associations de **clés** et **valeurs**
 - In [23]: dict={'a':1, 'b':2, 'c':None }
 - In [24]: dict
 - Out[24]: {'a': 1, 'c': None, 'b': 2}
 - ➔ (Remarquez que l'ordre est quelconque)
- Pour le parcourir
 - In [25]: dict.**keys**()
 - Out[25]: ['a', 'c', 'b']
 - In [26]: dict.**values**()
 - Out[26]: [1, None, 2]
 - In [27]: dict['b']
 - Out[27]: 2

Types avancés de Python - Liste

```
>>> liste = [] # déclaration d'une liste vide
>>> liste = [1,2,3] # initialisation
>>> liste
[1,2,3]
>>> liste.append(5) # ajouter la valeur 5 à la liste
>>> liste
[1,2,3,5]
>>> liste.append("ok") # ajouter la chaine 'ok'
>>> liste
[1,2,3,5,'ok']
>>> liste[4] # afficher l'item à la position 4
'ok'
>>> liste[0]= "bon" # modifier l'item à la position 0
>>> del liste[1] # supprimer l'item à la position 1
>>> liste
['bon',3,5,'ok']
```


Types avancés de Python - Liste

```
>>> liste.remove("ok") # supprime la première occurrence de 'ok'
```

```
>>> liste
```

```
['bon', 3, 5]
```

```
>>> len(liste) # Compter le nombre d'items d'une liste
```

```
3
```

```
>>> name = "Michel" # peut être utiliser pour compter la longueur d'une  
chaîne de caractères
```

```
>>> len(name)
```

```
6
```

```
>>> liste = ["a", "a", "a", "b", "c", "c"]
```

```
>>> liste.count("a") # Compter le nombre d'occurrence
```

```
3
```

```
>>> liste.index("c") # trouver l'index de la valeur
```

```
4
```

```
>>> liste2 = liste[:] # copier la liste
```

Types avancés de Python - Liste

```
>>> liste2 = "mot1:mot2:mot3"
>>> liste2.split(":") # Transformer une string en liste
['mot1', 'mot2', 'mot3']
>>> liste3 = ["mot1", "mot2", "mot3"]
>>> ":".join(liste3) # Transformer une liste en string
'mot1:mot2:mot3'
>>> liste4 = [1,3,50, 100, -4]
>>> -4 in liste4 # vérifier si un élément est dans la liste
TRUE
>>> 150 in liste4
FALSE
>>> liste3.extend(liste4) # ajouter les éléments de liste4 à
liste 3
>>> liste3
['mot1', 'mot2', 'mot3', 1,3,50, 100, -4]
```

Python – Affichage

```
>>> age = 32
>>> name = 'Jean'
>>> print(name , 'a' , age , 'ans')
```

Jean a 32 ans

- Une autre façon de faire est :

```
>>> print('{} a {} ans'.format(name, age)) # méthode format() permet une
meilleure organisation de l'affichage des variable
```

Jean a 32 ans

- Les accolades vides {} précisent l'endroit où le contenu de la variable doit être inséré.
- L'instruction .format(nom, x) indique la liste des variables à insérer

```
>>> var = (4500 + 2575) / 14800
>>> print("Le résultat du calcul est", var)
Le résultat du calcul est 0.4780405405405405
>>> print("Le résultat du calcul est {:.2f}".format(var))
Le résultat du calcul est 0.47
```

- Les deux points : indiquent que l'on veut préciser le format.
- Le formatage avec .xf (x étant un entier positif) renvoie un résultat arrondi, avec f pour float

Python – scripte - généralités

- Lorsque vous quittez et entrez à nouveau dans l'interpréteur Python, tout ce que vous avez déclaré dans la session précédente est perdu.
- Afin de rédiger des programmes plus longs, vous devez utiliser un éditeur de texte, préparer votre code dans un fichier et exécuter Python avec ce fichier en paramètre.
- Cela s'appelle créer un ***script***.
- Lorsque votre programme grandit, vous pouvez séparer votre code dans plusieurs fichiers. Ainsi, il est facile de réutiliser des fonctions écrites pour un programme dans un autre.

Python – scripte - généralités

- C'est un langage dynamique:
 - *toutes* les instructions sont exécutées au fur et à mesure, *y compris les déclarations*.

Instruction • `import sys` `# Module d'accès au systeme`
• `print(sys.path)` `monchemin='/sw/lib/'` `sys.path.append(monchemin)` `print(sys.path)`
commande •
Variable • `Commentaire`

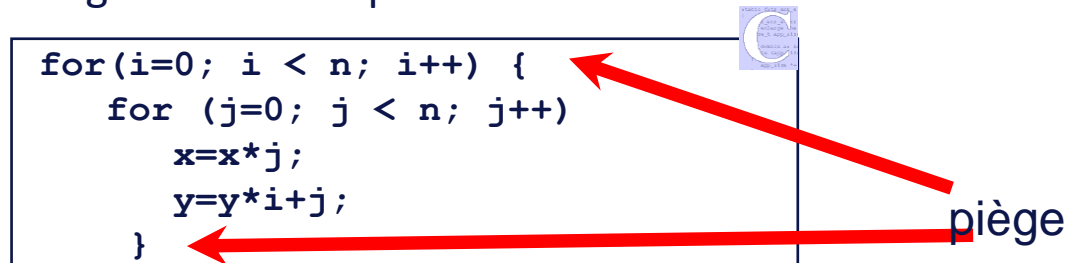
- C'est un langage où l'indentation est significative

```
import sys
    print sys.path    #---> ERREUR de syntaxe
```

Python – la forme (indentation)

- Certains langages de programmation utilisent des accolades {et} afin d'entourer les lignes de code qui vont ensemble :

```
for(i=0; i < n; i++) {  
    for (j=0; j < n; j++)  
        x=x*j;  
        y=y*i+j;  
}
```



- Python n'utilise pas les accolades, mais exige plutôt une indentation.
- Par exemple :

```
for i in range(2):  
    print("Hello")
```

 - Une deuxième ligne en retrait serait une partie de la boucle, et une deuxième ligne non indentée serait en dehors de la boucle.

▪ Le premier cas donne :

```
for i in range(2):  
    print("A")  
    print("B")
```

→

```
A  
B  
A  
B
```

▪ Tandis que le 2e cas donne

```
for i in range(2):  
    print("A")  
print("B")
```

→

```
A  
A  
B
```

Python – Commentaires

- Les commentaires sont ignorés dans le programme, ils sont là pour laisser des notes, et ils sont désignés par le symbole hash #.
- Les commentaires multilignes utilisent des triples guillemets comme ceci :

```
"""
```

```
This is a very simple Python program that  
prints "Hello".
```

```
That's all it does.
```

```
"""
```

```
print("Hello")
```

<https://towardsdatascience.com/single-double-and-triple-quotes-in-python-7ceea990baf>

Python – Saisie des valeurs

- Dans un programme, il est très pratique de pouvoir demander à l'utilisateur de saisir une chaîne de caractères.
- Python dispose d'une instruction : ***input()*** (version 3.X)
(l'instruction *raw_input()* est pour la version 2.7)

```
>>> mot2=input("Entrez un pays : ")
Entrez un pays : Japon
>>> print(mot2)
Japon
```

- **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 pour récupérer d'autres types

Python – Saisie des valeurs

```
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

Python – Saisie des valeurs

Exemple (Saisie de données)

```
chaine = input("Saisir une chaîne : ")
print(chaine, type(chaine))
chaine = input("Un entier : ")
print(chaine, type(chaine))
chaine = input("Une valeur flottante : ")
print(chaine, type(chaine))
```

```
toto
class'str'>
32
class'str'>
-5.14
class'str'>
```

Exemple (input avec transtypage)

```
chaine = input("Un entier : ")
nbre = int(chaine)
print(nbre, type(nbre))
chaine = input("Une valeur flottante : ")
val = float(chaine)
print(val, type(val))
```

```
32
class'int'>
-5.14
class'float'>
```

Python – Affichage (2)

```
>>> age = 32
>>> name = 'Jean'
>>> print(name , 'a' , age , 'ans')
```

Jean a 32 ans

- Une autre façon de faire est :

```
>>> print('{} a {} ans'.format(name, age)) # méthode format() permet une
meilleure organisation de l'affichage des variable
```

Jean a 32 ans

- Les accolades vides {} précisent l'endroit où le contenu de la variable doit être inséré et l'instruction .format(nom, x) indique la liste des variables à insérer
- Une autre façon de faire qui est possible à parti de la version 3.6 de Python c'est d'utiliser la syntaxe suivante avec le f-Strings : **f"phrase à afficher avec {variable}"**

```
>>> age = 20
>>> name = 'Eric'
>>> print(f"Hello, {name}. vous avez {age} ans.")
Hello, Eric. Vous avez 20 ans.
```

www.uha.fr



Bases de Python : Branchements conditionnels et boucles

Branchement conditionnel « if »

if condition:
 bloc d'instructions

else:
 bloc d'instructions

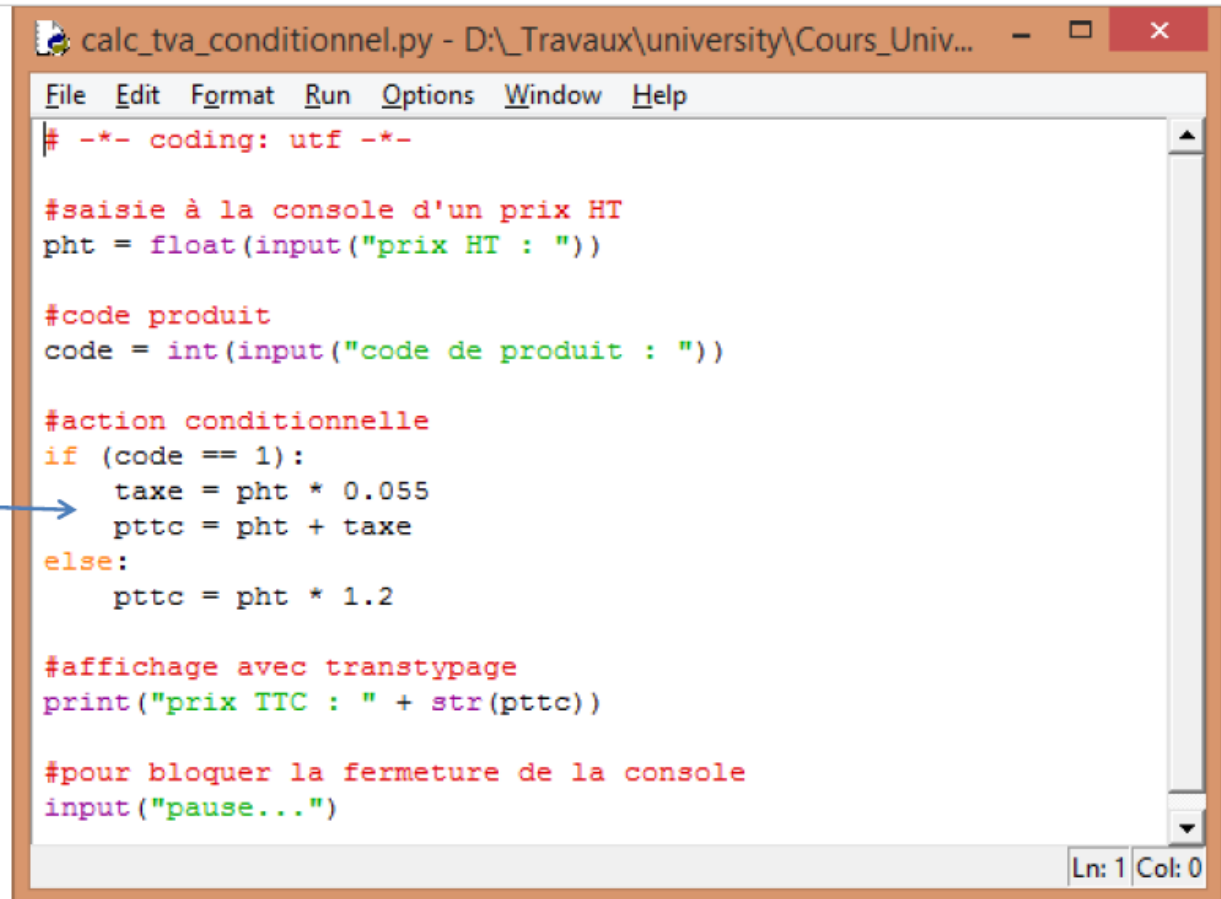
- Condition est très souvent une opération de comparaison (== ; != ; > ; >= ; < ; <=)
- Attention au **:** qui est primordial
- C'est l'**indentation** (le décalage par rapport à la marge gauche) qui délimite le bloc d'instructions
- La partie **else** est facultative
- Il est possible d'affiner une condition avec les mots clé **AND** qui signifie "ET" et **OR** qui signifie "OU".

```
>>> v = 7
>>> v > 5 and v < 10
True
```

Branchement conditionnel « if »

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

- **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

```
#action conditionnelle
if (code == 1):
    pttc = pht * 1.055
elif (code == 2):
    pttc = pht * 1.1
else:
    pttc = pht * 1.2
```

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

la boucle « for »

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

- Séquence est une collection de valeurs qui peut être générée avec **range()**
 - ★ `range(4)` → 0 1 2 3
 - ★ `range(1,4)` → 1 2 3
 - ★ `range(0,5,2)` → 0 2 4 # pas de 2
- La boucle for ne s'applique que sur une collection de valeurs. Ex. tuples, listes,...
- Attention à l'**indentation** toujours
- On peut arrêter 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

```
#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
```

- Il faut mettre **n+1** dans range() pour que **n** soit inclus dans la somme
- Observez attentivement les indentations.

Boucle « while »

while condition:
bloc d'instructions

- Condition : opération de comparaison attention à la boucle infinie !

```
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
```

- Attention à l'indentation toujours
- On peut arrêter la boucle avec break

Instructions break et continue

- Ces deux instructions permet de modifier le comportement d'une boucle (for ou while) avec un test (if).
- L'instruction **break** stoppe la boucle.
- L'instruction **continue** saute à l'itération suivante.

```
>>> v = 7
>>> for i in range(5):
...     if i > 2:
...         Break
...     print(i)
...
0
1
2
```

```
>>> v = 7
>>> for i in range(5):
...     if i == 2:
...         continue
...     print(i)
...
0
1
3
4
```

Itérations

- Certains types de données sont **itérable**, ce qui signifie que vous pouvez boucler sur les valeurs qu'ils contiennent. Par exemple une liste:

```
numbers = [1, 2, 3]
for number in numbers:
    print(number)
```

- Cela prend chaque élément dans les numéros de liste et imprime l'élément :

1
2
3

- D'autres types de données sont aussi itérable, par exemple les chaînes de caractère :

```
dog_name = "MAX"
for char in dog_name:
    print(char)
```

- la structure **for** boucle sur chaque caractère et l'affiche :

M
A
X

www.uha.fr



Notions avancées

Fonctions / modules

Les fonctions

- Fonctions fournies par Python :
 - fonctions prédéfinies (par ex. print, input, len. . .)
 - modules (bibliothèques) thématiques

module	rôle	exemples
–	fonctions prédéfinies	int, float, abs, input, print
math	calculs mathématiques	cos, sin, exp
random	valeurs aléatoires	rand, random
os, sys	fonctions système	chdir, getlogin, getpid
re	expressions régulières	match, compile
socket	manipulation de sockets TCP	socket, create_connection, gethostname

- Une fonction :
 - possède un nom définie par le mot réservé **def**
 - appartient éventuellement à un module
 - Regroupe un bloc d'instructions
 - Peut dépendre de paramètres/arguments séparés par une virgule
 - Pas de type à préciser car typage dynamique
 - Attention à la cohérence entre type dynamique des arguments et les opérateurs utilisés dans les instructions
 - Produit éventuellement un résultat, appelé aussi valeur de retour renvoyée par le mot réservé **return**

Les fonctions : exemples

Exemples

► `sqrt(x)`

```
import math  
  
res = math.sqrt(3)  
print(res)
```

```
1.7320508075688772
```

► `max(a, b)`

```
res = max(1, 10)  
print(res)
```

```
10
```

► `getlogin()`

```
import os  
  
res = getlogin()  
print(res)
```

```
wurbel.e
```

```
>>> def carree(x):  
...     return x**2  
...  
>>> carree(3)  
9
```

```
>>> def somme(a,b):  
...     return a+b  
...  
>>> carree(3,4)  
7
```

Les modules

- Python permet de placer des définitions dans un fichier et de les utiliser dans un script ou une session interactive.
- Un tel fichier est appelé un **module** et les définitions d'un module peuvent être importées dans un autre module ou dans le module **main**
- Le module main c'est le module initial que vous lancer en premier (contient vos variables et définitions lors de l'exécution d'un script au niveau le plus haut).
- Le nom de fichier est le nom du module suffixé de .py. À l'intérieur d'un module, son propre nom est accessible par la variable **__name__**
- Un module peut contenir aussi bien des instructions que des déclarations de fonctions.
- On peut importer un module ou des fonctions de ce module avec le mot clé **import**

```
import my_module
import my_module as new_mod
from my_module import f1, f2
from my_module import *
from my_module import f1 as new_f
```

Les modules

- Un appel direct du module fait exécuter immédiatement ses instructions !
- Pour éviter l'exécution d'une ou l'ensemble des instructions, on les fait précéder par cette condition : **if __name__ == '__main__'**
- Cela rend le fichier utilisable comme script et aussi comme un module importable.
- Les instructions du modules ne sont lancées que si le module est exécuté comme fichier « main »
- Exemples :

```
# fichier my_module.py
def my_fct():
    print("bonjour")
```

```
my_fct()
```

```
>>> import my_module
bonjour
```

```
# fichier my_module.py
def my_fct():
    print("bonjour")
```

```
if __name__ == '__main__':
    my_fct()
```

```
>>> import my_module
>>>
```

Les modules – liste d'arguments

- Python permet de lancer des programmes et des scripts en ligne de commande. On peut alors transmettre aux scripts ou programmes des informations complémentaires à l'aide d'arguments spécifiques
- Le module standard « **sys** » offre le moyen d'accéder aux arguments passés à l'appel en ligne de commande du script dans la variable « **sys.argv** » qui est elle-même une liste contenant les arguments de la ligne de commande
- Exemple :

```
# script test.py
import sys
if len(sys.argv < 3));
    print("Usage: %s <n> <n>\n" % sys.argv[0])
    sys.exit()
for i in range(1, int(argv[1]))
    for j in range(1, int(argv[2]))
        sys.stdout.write( "%4d" % (i*j))
```

- Pour appeler le script : `test.py nbr1 nbr2`

Les modules – sys

- le module **sys**, est présent dans tous les interpréteurs Python
- On plus de la liste des arguments (argv), le module **sys** contient une liste qui détermine les chemins de recherche de modules pour l'interpréteur **sys.path**
- Cette liste est modifiable en utilisant les opérations habituelles des listes

```
>>> import sys
```

```
>>> sys.path.append('/ufs/guido/lib/python')
```