

# Programmation avancée



## COURS 4 LE LANGAGE PYTHON

# À quoi sert PYTHON?



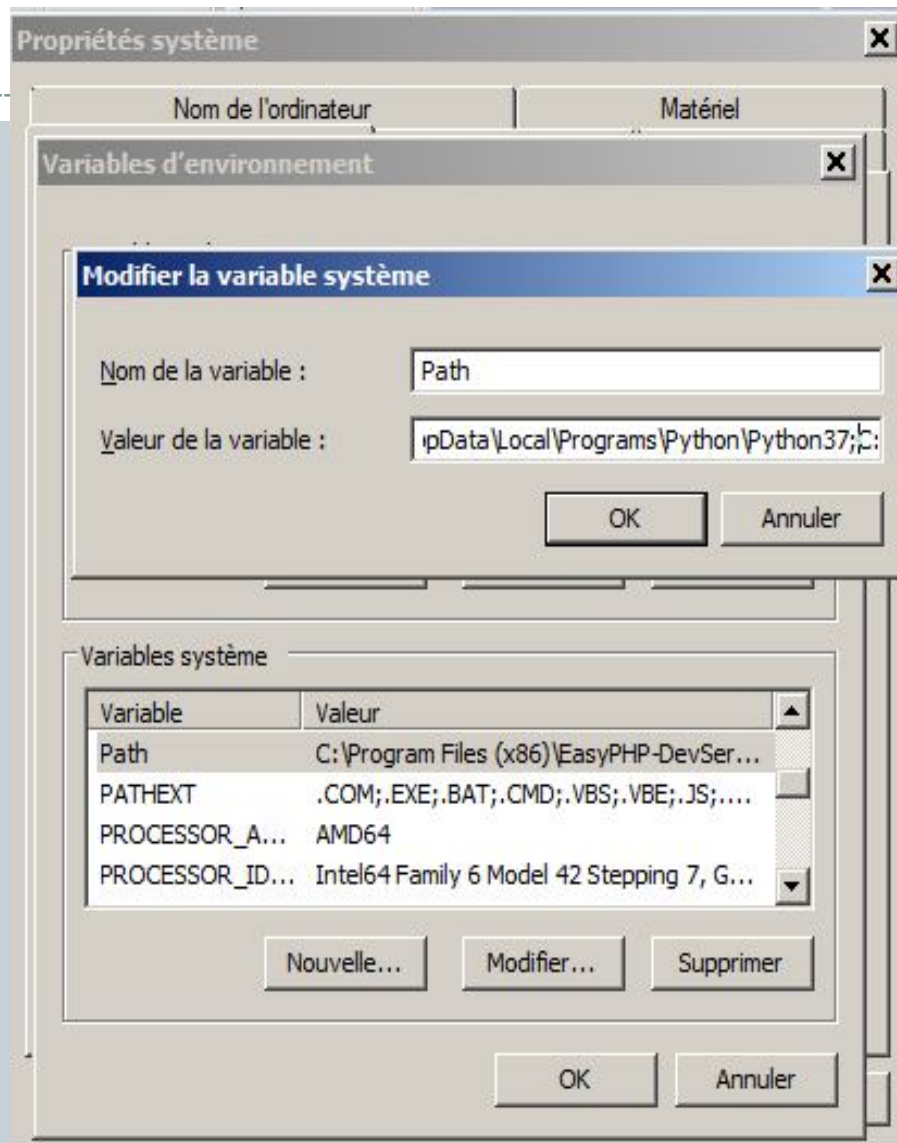
- Traitement de données textuelles.
- Manipulation de chaînes de caractères.
- Gestion de fichiers.
- Utilisation du langage des expressions régulières.
- Développement d'applications en TALN/Extraction d'information/Apprentissage automatique.
- Administration système et réseau.
- Manipulation de documents structurés (DOM, SAX).
- Développement web

# Adresse de téléchargement de PYTHON



<https://www.python.org/downloads/>

# Variable d'environnement de Python



Si vous avez le message suivant :

'python' n'est pas reconnu en tant que commande interne, vérifiez votre variable d'environnement.

**Aller dans :** Panneau de configuration\System\Paramètres Système avancés\Variables d'environnement\Variables système\Path.

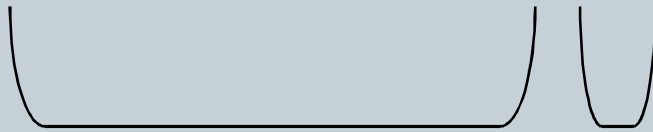
Modifiez la variable "Path", en y ajoutant le chemin de votre Python : e.g.,

C:\xxxxxx\xxxxxx\AppData\Local\Programs\Python\Python37

# Mon premier script en Python



nom-du-script.py



1

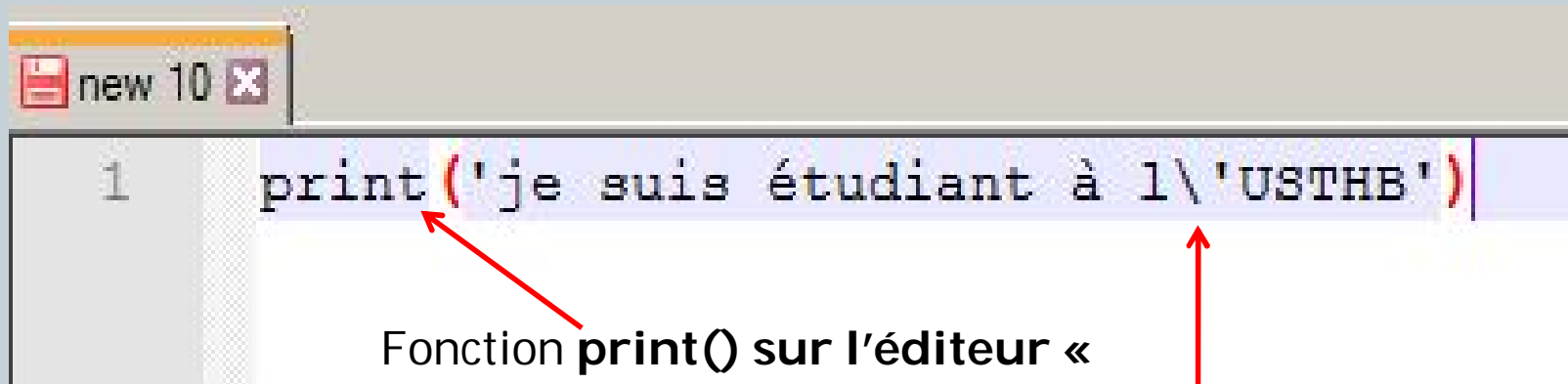
2

1- nom de votre script

2- extension de votre script en .py

# Mon premier script en Python

## « Notepad++ » - ouverture d'un nouveau fichier



The screenshot shows a Notepad++ window titled 'new 10'. The editor contains a single line of Python code: `print('je suis étudiant à l\'USTHB')`. The line is numbered '1' on the left. A red arrow points from the text 'Fonction print() sur l\'éditeur « Notepad++ »' to the `print` function. Another red arrow points from the text 'Il faut déspecialiser le caractère ' with l\'antislash' to the backslash character in the string.

Fonction **print()** sur l'éditeur «  
**Notepad++** »

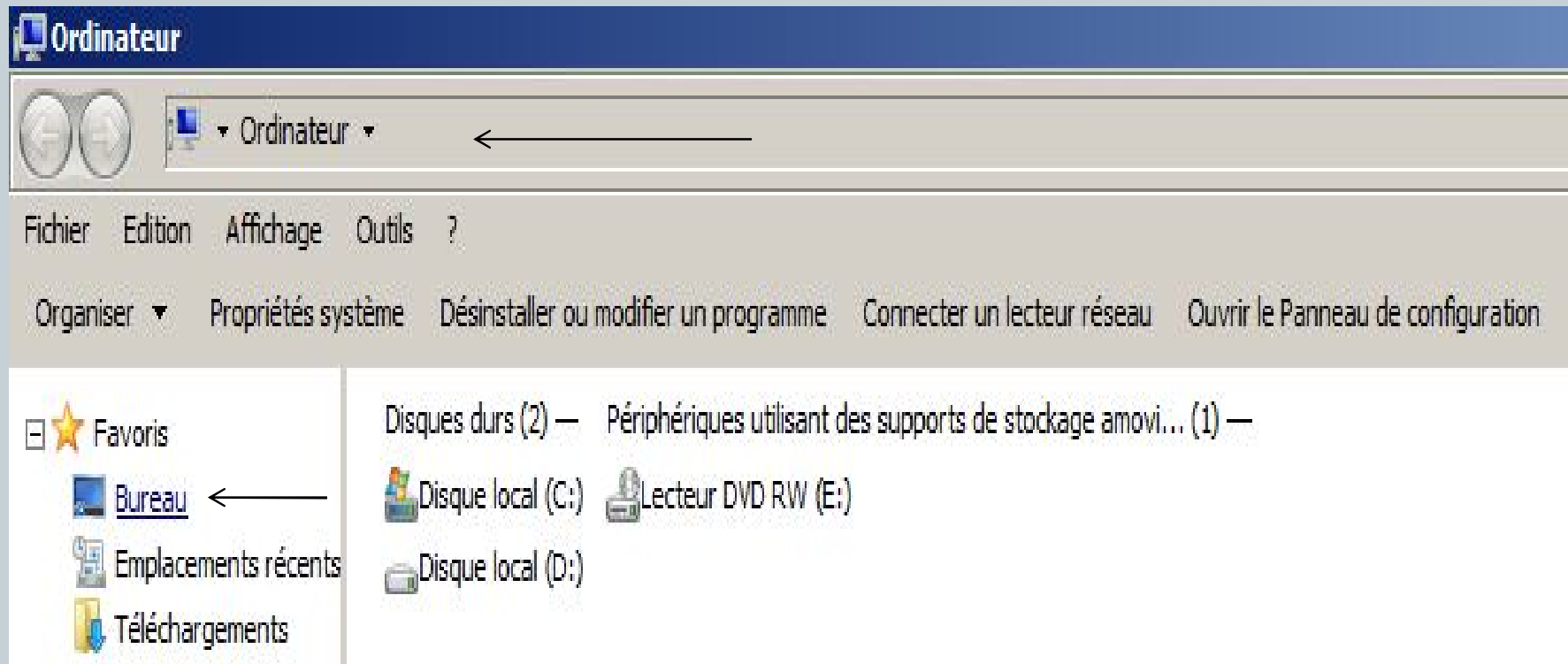
Il faut déspecialiser le caractère ' avec  
l'antislash

## Enregistrement du fichier sur le bureau (Desktop) avec l'extension .py



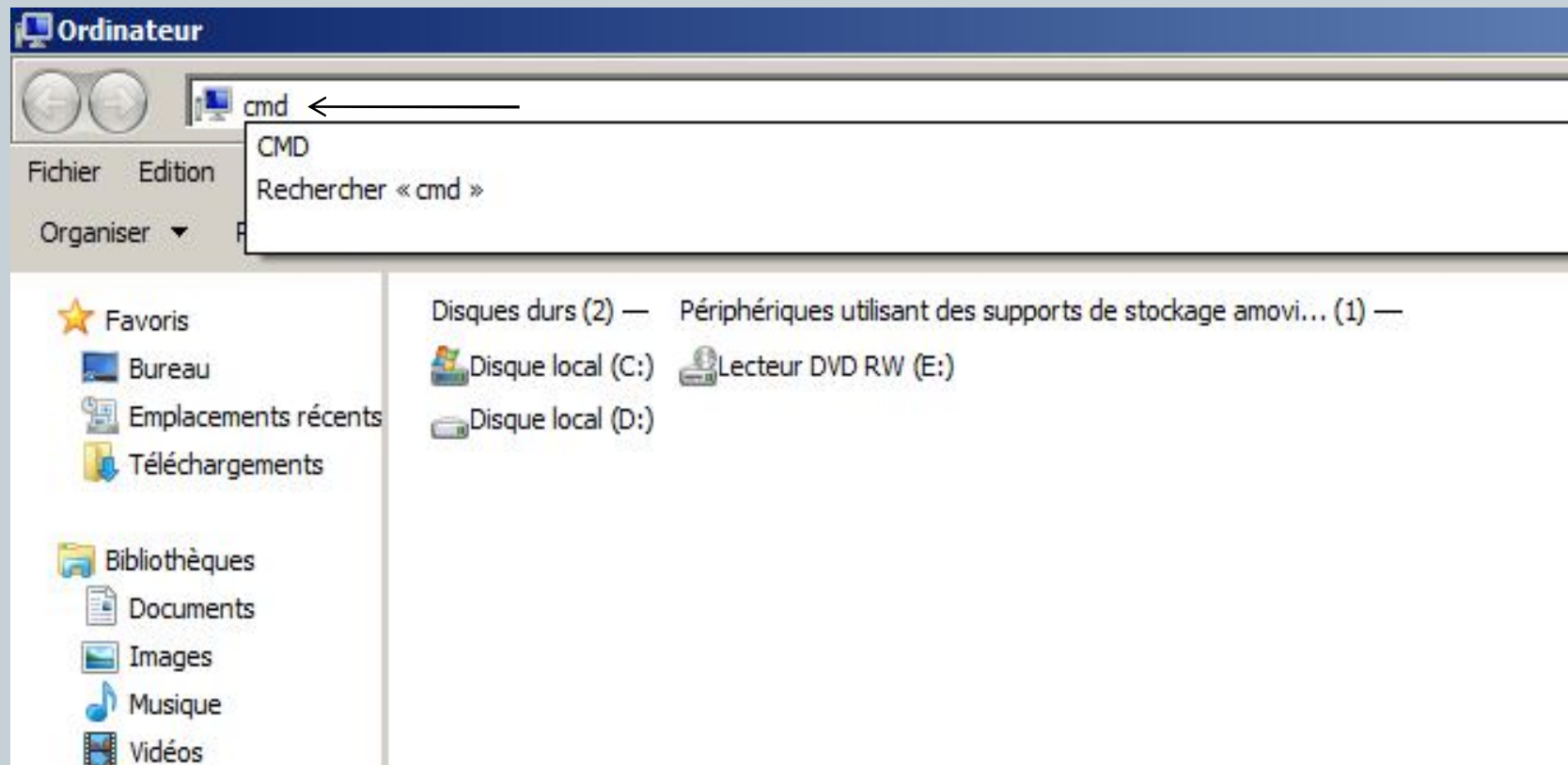
```
1 print('je suis étudiant à l\'USTHB')
```

# Ouverture d'une invite de commandes à l'aide de la barre de chemins d'accès (barre d'adresse de l'Explorateur Windows)





# Barre de chemins d'accès (ouverture d'une invite de commandes au niveau du bureau)



# Exécution d'un script Python sur une invite de commandes Windows



```
C:\Users\user\Desktop>
```

Ouvrir une invite de commandes

# Exécution d'un script Python sur une invite de commandes Windows



```
C:\Users\user\Desktop>test.py_
```

**Tapez le nom de votre script pour l'exécuter**

# Exécution d'un script Python sur une invite de commandes Windows



```
C:\Users\user\Desktop>test.py  
je suis étudiant à l'USTHB
```

# Saut de ligne avec « \n » en Python

A screenshot of the Notepad++ application window. The title bar reads "C:\Users\user\Desktop\test.py - Notepad++". The menu bar includes "Fichier", "Édition", "Recherche", "Affichage", "Encodage", "Langage", "Paramétrage", "Outils", "Macro", and "Ex". The toolbar contains various icons for file operations and editing. The tab bar shows two tabs: "new 5" and "test.py". The code editor displays a single line of Python code: 

```
1 print("je suis étudiant à l'USTHB\n")
```

 The text is in a monospaced font, with "print" in blue and the string in black. The background of the editor is white with a light gray line number margin on the left.

## Ajout d'un saut de ligne « \n »

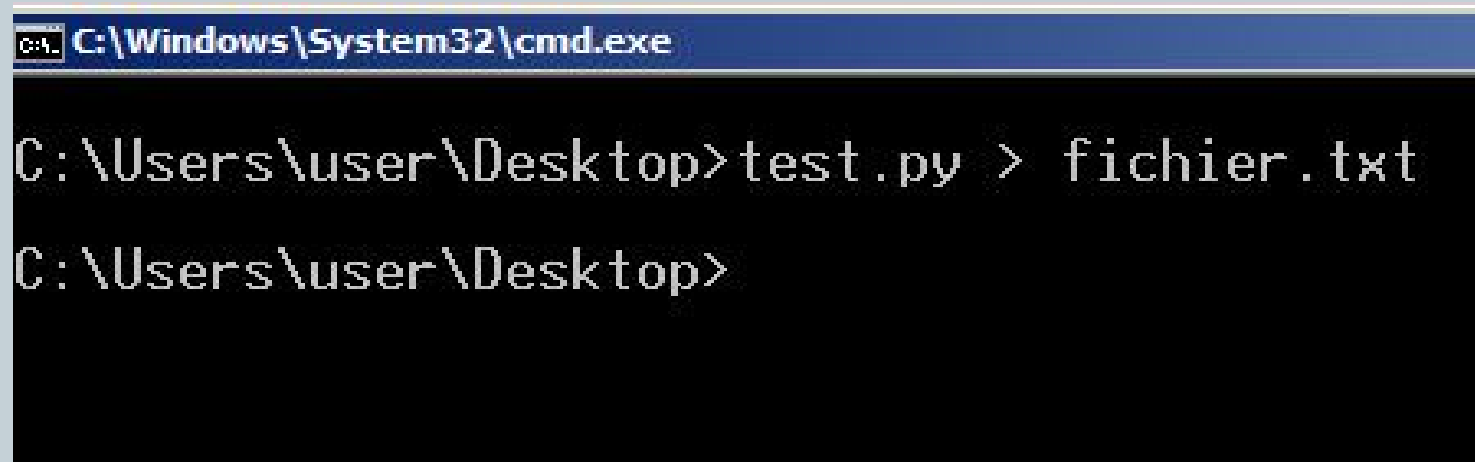


C:\Windows\System32\cmd.exe

```
C:\Users\user\Desktop>test.py  
je suis étudiant à l'USTHB
```

```
C:\Users\user\Desktop>_
```

## Rediriger le résultat de la commande print vers un fichier

A screenshot of a Windows Command Prompt window. The title bar is blue and contains the text 'C:\Windows\System32\cmd.exe'. The main area has a black background with white text. The prompt 'C:\Users\user\Desktop>' is followed by the command 'test.py > fichier.txt'. A second prompt 'C:\Users\user\Desktop>' is shown on the next line.

```
C:\Windows\System32\cmd.exe  
  
C:\Users\user\Desktop>test.py > fichier.txt  
C:\Users\user\Desktop>
```

# Résultat de la redirection

A screenshot of the Notepad++ application window. The title bar reads "C:\Users\user\Desktop\ fichier.txt - Notepad++". The menu bar includes "Fichier", "Édition", "Recherche", "Affichage", "Encodage", "Langage", and "Paramétrage". The toolbar contains various icons for file operations. The tab bar shows three open files: "new 5", "test.py", and "fichier.txt". The text area of "fichier.txt" contains the text "je suis étudiant à l'USTHB" on the first line, with line numbers 1, 2, and 3 visible on the left margin.

```
1 je suis étudiant à l'USTHB
2
3
```



# Types d'objets en Python - Classes



- **Classe str (String) : chaînes de caractères.**
- **Classe int (Integer) : nombres entiers.**
- **Classe float : nombres flottants (ou nombres réels).**
- **Remarque : Toute variable est un objet.**
- **Remarque : Toute fonction/méthode est un objet.**
- **Remarque : Un objet est une instance d'une classe.**
- **Donc, en Python, tout est objet.**

## Classes « Str », « Int » et « Float » (types de données)

- **a=5**
- **s=str(a) # fonction str(). On l'appelle aussi « constructeur ».**
- **print(s)**
- -----
- **b="5"**
- **i=int(b) #fonction int(). On l'appelle aussi « constructeur ».**
- **print(i)**
- -----
- **c=3.2**
- **f=float(c) #fonction float()#avec int() on perd la précision**
- **print(f). float() est également un « constructeur » de nombres à virgule flottante.**
- **Remarque importante : Tout type de donnée est une classe.**

# Nommage des variables (objets) en Python



- lettres (minuscules ou majuscules de préférence sans accentuation - ASCII : **American Standard Code for Information Interchange**)
- Chiffres
- \_ (tiret bas / underscore)
- Exemples :
  - maChaine
  - MaChaine2
  - Ma\_2eme\_Chaine
  - 2eme\_chaine

# Opérateur d'affectation (assignation)

- Opérateurs d'affectation simple :

- $a = b$  # assigne b à a

- $a = 6$

- $b = 2$

- $c = b$

- #-----#

- $a = \text{"il"}$

- $b = \text{"écrit"}$

- $c = b$

- Opérateurs d'affectation composés :

- $a = \text{"il "}$

- $b = \text{"écrit"}$

- **$a = a + b$  #  $a += b$**

- **$> \text{il écrit}$**

# Opérateur d'affectation - concaténation



- a="il "
- b="écrit"
- **a=a+b #a+=b**
- **> il écrit**

## Exemple d'affectation de deux variables Str



- `chaine_1="Spécialité"`
- `chaine_2="Académique"`
- `print(chaine_1+" "+chaine_2)#concaténation de chaînes de caractères avec le +`
- `> Spécialité      Académique`

# Résultat de l'affectation



C:\Windows\System32\cmd.exe

```
C:\Users\user\Desktop>test.py
```

```
Spécialité Académique
```

```
C:\Users\user\Desktop>
```

## Types d'objets en Python - suite



- **Listes - classe « list »**
- **Tuples - classe « tuple »**
- **Dictionnaires - classe « dict »**



# Les listes



Une liste est un objet pouvant contenir d'autres objets (entiers, chaînes de caractères). Vous pouvez accéder à chaque élément de cette liste à travers son indice.

- `maliste= ["dog", "cat", "mouse" ]` #une liste avec 3 objets
- `maliste = [ 23, 45, 300, 98]`
- `maliste[2]` #imprime mouse
- `maliste[1]` #imprime 45
- `len(montab)-1` #**imprime le dernier indice du tableau avec la fonction len()**
- `maliste[-1]` #**imprime le dernier élément du tableau**

## Exemple d'affectation d'une liste



```
maList=["Licence",2]
# Ceci est un commentaire
print("Je suis étudiant en "+maList[0]+" "+str(maList[1]))
```

Commentaire en  
Python



## Résultat de l'affectation



C:\Windows\System32\cmd.exe

```
C:\Users\user\Desktop>test.py  
Je suis étudiant en Licence 2  
C:\Users\user\Desktop>
```

## Création d'une liste vide



- `maliste = list()` # création d'une liste vide grâce au « constructeur » `list()`, permettant donc de créer une liste.
- `print(type(maliste))` # la fonction `type()` donne le type de donnée
- `> <classe 'list' >`
- `maliste = []` # création d'une liste vide
- `print(type(maliste))`
- `> <classe 'list' >`

## L'opérateur « del »



- **Del** : supprime un élément d'une liste.

```
tab=[5,6,3]
```

```
del tab[2]
```

```
print (tab)
```

```
> [5,6]
```

# Les tuples



- Les tuples sont des listes immuables, qu'il est impossible de modifier.
- Pour créer un tuple, on utilise des parenthèses au lieu des crochets.
- À quoi sert un tuple ?
- Affectation multiple de variables.
- Même sans mettre de parenthèses, Python comprend qu'il doit créer un tuple pour faire l'affectation.
- Voir le slide suivant ...

# Les tuples

- Exemple :
- `a = ()` #tuple vide
- `b= (1, 2, 3)` #tuple contenant trois entiers
- `(a,b)=(3,5)` #**affectation multiple**
- `t=(a,b)`
- `print(type(t))` #tuple
- `a,b=3,5` #**affectation multiple**
- `t=(a,b)`
- `print(type(t))` #tuple
- `a=3,5`
- `print(type(a))` #tuple
- **#L'opérateur "del" ne fonctionne pas sur les tuples.**

## Méthode « split() » - listes



```
a="Bonjour j'aime l'USTHB"
```

```
mots=a.split()#l'espace est le délimiteur par défaut
```

```
print(type(mots)) #mots est une liste (classe « list »)
```

```
#-----#
```

```
x = "bleu,rouge,vert"
```

```
(a,b,c)=x.split(",")#on utilise la virgule comme délimiteur
```

```
print(a+" comme le ciel "+b+" comme la tomate "+c+" comme une  
plante")
```

```
t=(a,b,c)
```

```
print(type(t))
```

- Dans l'exemple ci-dessus, **a**, **b** ou **c** pris séparément appartiennent à la classe « str », mais **t** appartient à la classe « tuple ».



## Passage d'arguments avec la variable spéciale de type « list » `argv[x]` et le module « sys »



```
import sys #importer le module « sys »  
print("Je suis étudiant en "+sys.argv[1]+" et je m'appelle  
    "+sys.argv[2])
```

Ouvrir une ligne de commandes et tapez ceci :

```
>test.py informatique un_prenom
```

```
#-----#
```

On peut également faire :

```
from sys import argv  
print("Je suis étudiant en "+argv[1]+" et je m'appelle  
    "+argv[2])
```

```
>test.py informatique un_prenom
```

## Résultat de la récupération des deux arguments de type « Str » sur l'invite (ligne) de commandes



```
C:\Users\user\Desktop>test.py informatique omar  
Je suis étudiant en informatique et je m'appelle omar  
C:\Users\user\Desktop>_
```

## Exercice 1 - quelle est la ligne de commandes à écrire?



- `import sys`
- `mark1 = sys.argv[1]`
- `mark2 = sys.argv[2]`
- `mark3 = sys.argv[3]`
- `course1 = sys.argv[4]`
- `course2 = sys.argv[5]`
- `course3 = sys.argv[6]`
- `print("Cours1\t"+course1+" = note1\t"+mark1+"\n")`
- `print("Cours2\t"+course2+" = note2\t"+mark2+"\n")`
- `print("Cours3\t"+course3+" = note3\t"+mark3+"\n")`
- `print("-----\n")`
- `resultat=int(mark1) + int(mark2) + int(mark3)`
- `resultat=resultat/3`
- `print("moyenne\t="+str(resultat))`

## Exercice 2 - manipulation des arguments avec la méthode « split »

- Écrire un script Python permettant de diviser la phrase « **J'aime l'USTHB** » passée en argument, en utilisant comme caractère de division l'apostrophe.
- Le token « **USTHB** » doit être affiché sur votre invite de commandes.

# Types d'objets en Python - suite



- Listes - classe « list »
- Tuples - classe « tuple »
- **Dictionnaires - classe « dict »**

# Les dictionnaires



Pour instancier un dictionnaire :

- `dico = dict()` #création d'un dictionnaire vide
- `print(type(dico))`
- `><class 'dict'>`
- `#-----`  
`-----#`
- `dico = {}` #création d'un dictionnaire vide
- `print(type(dico))`
- `><class 'dict'>`

# Les dictionnaires



- Ajout de clés et de valeurs au dictionnaire :
- dico = {}
- dico["arbre"] = "vert" # [clé] = valeur
- dico["plâtre"] = "blanc"
- dico["pneu"] = 5
- Pour accéder à un élément dans un dictionnaire :
- print(dico["pneu"])
- > 5
- Pour imprimer tout le dictionnaire :
- print(dico)
- > {'plâtre': 'blanc', 'pneu': 5, 'arbre': 'vert'}
- On peut aussi écrire :
- a={'plâtre': 'blanc', 'pneu': 5, 4: 'vert'}
- print(a['plâtre'])
- print(a[4])
- > blanc
- > vert

# Les dictionnaires



- `dic={"cours1":15,"cours2":18,"cours3":11}`
- Pour imprimer les clés du dictionnaire :
- `print(dic.keys(),"sont des clés")`
- #ou bien `print(str(dic.keys())+" sont des clés")`
- **> `dict_keys(['cours2', 'cours3', 'cours1'])` sont des clés**
- Pour imprimer les valeurs du dictionnaire :
- `print(dic.values(),"sont des valeurs")`
- #ou bien `print(str(dic.values())+" sont des valeurs")`
- **> `dict_values([15, 11, 18])` sont des valeurs**



## L'opérateur « del »



- **Del** : supprime un élément d'un dictionnaire.

```
dic={"cours1":15,"cours2":18,"cours3":11}
```

```
del dic["cours2"]
```

```
print(dic)
```

```
>{"cours1":15, "cours3":11}
```

# Opérateurs arithmétiques



- $a = (1 + 2) * 3 - 4$  # retourne 5 dans a (règle de priorité)
- $a = 1 + 2 * 3 - 4$  # retourne 3 dans a (règle de priorité)
- $a = 8 - 4 - 2$  # retourne 2 et non 6 (règle d'associativité)
- $a = 7 / 8$  # Divise 7 par 8 et met 0.875 dans a
- $a = 9 ** 2$  # 9 à la puissance 10
- $a = 5 \% 2$  # le reste de 5 divisé par 2 (modulo)

## Opérateurs de chaînes



- `print("my"+" "+"dog")` #concaténation
- `> my dog`
- `print("table"*5)` #répétition (multiplication de chaîne de caractères)
- `> tabletabletabletabletable`
- `print("ta" in "table")` # renvoie `True` si la sous-chaîne "ta" est contenue dans la chaîne "table"
- `> True`

## Opérateurs de comparaison - nombres



- **$a == b$  # égalité**
- **$a != b$  # différence**
- **$a > b$  # supériorité stricte**
- **$a < b$  # infériorité stricte**
- **$a >= b$  # supériorité**
- **$a <= b$  # infériorité**

# Opérateurs de comparaison - chaînes de caractères



- **$a == b$  # égalité**
- **$a != b$  # différence**
- **Les opérateurs ci-dessous respectent l'ordre alphabétique :**
- **$a > b$  # supériorité stricte**
- **$a < b$  # infériorité stricte**
- **$a >= b$  # supériorité**
- **$a <= b$  # infériorité**

# Opérateurs d'incrémentation et de décrémentation



- Incrémentation :
- $a=6$
- **$a+=1$  # ou bien  $a=a+1$**
- **$> 7$**
- Décrémentation :
- $a=6$
- **$a-=1$  #ou bien  $a=a-1$**
- **$> 5$**

## Opérateurs logiques - OU - or / |



OR: |

OR: **or (court-circuit)**

```
x=3
```

```
y=5
```

```
if(x==3)|(y==5):  
    print("ok")
```

```
if(x==3 or y==5):  
    print("ok")
```

```
if x==3 or y==5:  
    print("ok")
```

## Opérateurs logiques - ET - and/&



AND: **&**

AND: **and (court-circuit)**

```
x=2
```

```
if(x==3)&(x<4):
```

```
    print("ok")
```

```
if(x==3 and x<4):
```

```
    print("ok")
```

```
if x==3 and x<4:
```

```
    print("ok")
```



## Opérateurs logiques - PAS - not



**NOT**

```
x=2
```

```
if not(x==3):  
    print("ok")
```

## Structures de contrôle - conditions - if/else



```
x=2
```

```
y=8
```

```
if x == y :
```

```
    print ("x and y are equals\n")
```

```
else:
```

```
    print("x and y are different\n")
```

**Remarque : Les deux points : terminent une condition.**

## Structures de contrôle - conditions - elif

x=3

y=4

if x == y :

print("x and y are equals\n")

elif x == 3:

print("yes")

else:

print("x and y are different\n")

## Structures de contrôle - conditions – if/else



```
x="dog"
```

```
y="cat"
```

```
if x == y :
```

```
    print("x and y are equals\n")
```

```
else:
```

```
    print("x and y are different\n")
```

## Structures de contrôle - conditions – if/else



```
x="d"
```

```
y="c"
```

```
if x > y :
```

```
    print("x comes alphabetically after y")
```

```
else:
```

```
    print("x comes alphabetically before y")
```

## Structures de contrôle - boucles - while



```
i=1  
nb=7  
while i <= 10:  
    print(i, "*", nb, "=", i * nb)  
    i += 1 #ou bien i=i+1
```

Que va afficher ce script ?

## Structures de contrôle - boucles - for



- # Dans l'exemple ci-dessous, la fonction **range()** génère une liste [ ] de nombres, allant de 0 à 9 (10 itérations). Par défaut, on commence à 0.

```
for x in range(10): # ou bien range(0,10)
```

```
    print(x)
```

```
#-----range() avec trois paramètres-----#
```

```
for x in range(3,10,2):
```

```
    print(x)
```



```
#Début de liste, fin de liste, incrément.
```

## Structures de contrôle - boucles - for - parcourir une liste



```
tab=[2,6,1]
```

```
for x in tab: #ou bien for x in range(len(tab)):
```

```
    print(x) #ou bien print(tab[x])
```

```
#-----#
```

```
tab=['yellow','black','blue']
```

```
for x in range(len(tab)): #ou bien for x in tab:
```

```
    print(tab[x]) #ou bien print(x)
```

```
#-----#
```

```
tab=[('yellow',3),('black',8),('blue',7)] # 3 tuples
```

```
for x,y in tab:
```

```
    print(x,y)
```



## Structures de contrôle - boucles - for - parcourir une liste



```
a="Bonjour j'aime l'USTHB"
```

```
mots=a.split()
```

```
for x in mots:
```

```
    print(x)
```

```
#-----Ou bien-----  
-#
```

```
a="Bonjour j'aime l'USTHB"
```

```
mots=a.split()
```

```
for x in range(len(mots)):
```

```
    print(mots[x])
```

```
#la fonction range() crée une suite de 0 à 2, en s'appuyant  
sur la longueur de « mots » retournée par la fonction  
len()
```

## Structures de contrôle - boucles - for - parcourir un dictionnaire

- #-----Afficher les clés-----#
- dic={"cours1":15,"cours2":18,"cours3":11}
- for i in dic.keys(): # ou bien for i in dic:
- print(i," est une clé") # ou bien + au lieu de ,
- #-----Afficher les valeurs-----#
- dic={"cours1":15,"cours2":18,"cours3":11}
- for i in dic.values():
- print(i," est une valeur") # ou bien + avec la fonction str() au lieu de ,

## Structures de contrôle - boucles - for - parcourir un dictionnaire



```
cours2 est une clé  
cours1 est une clé  
cours3 est une clé  
18 est une valeur  
15 est une valeur  
11 est une valeur
```

#En Python (comme dans d'autres langages, e.g., Perl), l'affichage est aléatoire.

## Structures de contrôle imbriquées



```
animal="cat"
num=['i','ii','iii','iv','v']
if animal == "cat":
    for x in num:
        print("We hate cat", " ", x, "\n")
else:
    print("We prefer dog")
#Ou bien + au lieu de , dans le print()
```

## Les fonctions/méthodes prédéfinies - type()



- **type** : cette fonction retourne le type de donnée.
- liste=(4,5,9)
- print(**type**(liste))
- > <class 'tuple'>

## Les fonctions/méthodes prédéfinies - reverse() / reversed()

- **Reverse** : retourne les éléments d'une liste inversés.
- `a = [7,9,8]`
- `a.reverse()`
- `print(a)`
- `> [8,9,7]`
- **Reversed** :
- `a = [7,9,8]`
- `print(list(reversed(a)))`
- `> [8,9,7]`

## Les fonctions/méthodes prédéfinies - sort()



- `sort()` : permet de trier une liste.
- `texte=["pomme","pain","pomme","orange","pomme","pain"]`
- `texte.sort()`
- `print(texte)`
- `>['orange', 'pain', 'pain', 'pomme', 'pomme', 'pomme']`

## Les fonctions/méthodes prédéfinies - sorted()

- **sorted()** : permet de trier une liste ou un dictionnaire.
- `texte=["pomme","pain","pomme","orange","pomme","pain"]`
- `print(sorted(texte))`
- `> ['orange', 'pain', 'pain', 'pomme', 'pomme', 'pomme']`
- -----  
-----
- `dic={'976':'Mongolia','52':'Mexico','212':'Morocco','64':'New Zealand','33':'France'}`
- `print(sorted(dic.keys()))` #ou bien `print(sorted(dic))`
- `> ['212', '33', '52', '64', '976']`



## Les fonctions/méthodes prédéfinies - sorted()

- #-----Trier les clés-----#
- dic={"cours1":15,"cours2":18,"cours3":11}
- for i in **sorted**(dic.keys()):#ou bien for i in **sorted**(dic)
- print(i)
- #-----Trier les valeurs-----#
- dic={"cours1":15,"cours2":18,"cours3":11}
- for i in **sorted**(dic.values()):
- print(i)

## Les fonctions/méthodes prédéfinies - sorted()



```
C:\Users\user\Desktop>test.py  
cours1  
cours2  
cours3  
11  
15  
18
```

## Les fonctions/méthodes prédéfinies - get()



- `dic={"cours1":15,"cours2":18,"cours3":11}`
- `for i in sorted(dic.keys()):#ou bien for i in sorted(dic)`
- `print(i,dic.get(i))`
- #La méthode `get(key)` prend comme paramètre la clé et retourne sa valeur.
- #On trie les clés du dictionnaire et pour chaque clé on affiche sa valeur.

## Les fonctions/méthodes prédéfinies - get()



```
C:\Users\user\Desktop>test.py  
cours1 15  
cours2 18  
cours3 11  
  
C:\Users\user\Desktop>_
```

## Les fonctions/méthodes prédéfinies - avec un seul get()



- `texte=["pomme","pain","pomme","orange","pomme","pain"]`
- `lettres = {}`
- `for c in texte:`
- `if c in lettres:`
- `lettres[c] = lettres[c] + 1#ou lettres[c]+=1`
- `else:`
- `lettres[c] = 1`
- `for i in sorted(lettres):`
- `print(i,lettres.get(i))`
- `#Ce script remplit le dictionnaire «lettres» avec le contenu de la liste «texte». On utilise une condition pour éviter l'erreur KeyError: clé, signifiant que la clé n'existe pas encore.`

## Les fonctions/méthodes prédéfinies - avec un seul get()

- `texte=["pomme","pain","pomme","orange","pomme","pain"]`
- `lettres = {}`
- `for c in texte:`
- `if c not in lettres:`
- `lettres[c] = 1`
- `else:`
- `lettres[c] = lettres[c] + 1#ou lettres[c]+=1`
- `for i in sorted(lettres):`
- `print(i,lettres.get(i))`
- `#Ce script remplit le dictionnaire «lettres» avec le contenu de la liste «texte». On utilise une condition pour éviter l'erreur KeyError:clé, signifiant que la clé n'existe pas encore.`

## Les fonctions/méthodes prédéfinies - avec deux get()



- > orange 1
- pain 3
- pomme 2

## Les fonctions/méthodes prédéfinies - items()



- `dic={"cours1":15,"cours2":18,"cours3":11}`
- `print(dic.items())`
- `>dict_items([('cours3', 11), ('cours1', 15), ('cours2', 18)])`
- #Cette méthode prédéfinie retourne les couples (clé,valeur) d'un dictionnaire.



## Les fonctions/méthodes prédéfinies – keys() / values()



- `dic={"cours1":15,"cours2":18,"cours3":11}`
- `print (dic.keys())` #affichage des clés d'un dictionnaire
- `print (dic.values())` #affichage des valeurs d'un dictionnaire

## Les fonctions/méthodes prédéfinies - append()



- **append** : insère un élément à la fin (à droite) d'une liste.

```
t= [1,2,3,4]
```

```
t.append(8)
```

```
print(t)
```

```
> [1,2,3,4,8]
```

## Les fonctions/méthodes prédéfinies - pop()

- **pop** : supprime un élément à la fin (à droite) d'une liste et le retourne.
- `t = [1,2,3,4]`
- `x = t.pop()`
- `print(x)`
- `> 4`
- **pop** peut prendre comme paramètre l'indice de l'élément à supprimer.

```
t = [1,2,3,4]
```

```
x = t.pop(2)
```

```
print(x)
```

```
> 3
```

## Les fonctions/méthodes prédéfinies - appendleft()



- **appendleft** : insère un élément au début (à gauche) d'une liste.

```
from collections import deque
```

```
t=deque([1,2,3,4]) # création d'une « file à double  
entrée »
```

```
t.appendleft(8)
```

```
print(t)
```

```
> deque([8, 1, 2, 3, 4])
```

```
#Possibilité d'utiliser append() sur une deque().
```

## Les fonctions/méthodes prédéfinies - popleft()

- **popleft** : supprime un élément au début (à gauche) d'une liste et le retourne.

```
from collections import deque
```

```
t=deque([1,2,3,4]) # création d'une « file à double  
entrée »
```

```
x=t.popleft()
```

```
print(x)
```

```
> 1
```

#Avec **popleft()**, pas de possibilité de spécifier un indice comme avec **pop()**. On peut utiliser **pop()** mais sans indice.

## Les fonctions/méthodes prédéfinies - clear()

- **clear()** : cette méthode permet de vider une liste.

```
st= [123, 'abc', 'efg', 'abc', 123]
```

```
st.clear()
```

```
print(st)
```

```
> [ ]
```

## Les fonctions/méthodes prédéfinies - split()



- **Split** : découpe une chaîne de caractères selon un séparateur et met les éléments dans une liste.

```
inf = "Premier:Ministre:Acteur:14, rue Saint Honoré"
```

```
a=inf.split(":")
```

```
print(a[3])
```

```
> 14, rue Saint Honoré
```

- **Join** : renvoie une chaîne de caractères contenant les éléments d'une liste, concaténés et séparés par un délimiteur.

```
sep=","
```

```
myNames = ["Samir", "Meriem", "Salim"]
```

```
print(sep.join(myNames)) # ou bien print(",".join(myNames))
```

```
➤ Samir,Meriem,Salim
```

## Les fonctions/méthodes prédéfinies - len() / index()

- **Len** : renvoie le nombre d'éléments d'une liste.

```
tab=[3,6,9]
```

```
print (len(tab))
```

```
> 3
```

- **Index**: détermine la position d'une lettre dans une chaîne de caractères.

```
string = "perlmememe.org"
```

```
print(string.index('o'))
```

Si le caractère n'existe pas dans la chaîne, le message «ValueError : substring not found » est affiché.



## Les fonctions/méthodes prédéfinies - len() / rstrip()

- **Len** : retourne la longueur en caractères de la valeur d'une variable scalaire.

```
tab="Une pomme"  
print(len(tab))  
> 9
```

- **Rstrip()** : supprime les sauts de ligne.

```
var = "Je suis un étudiant en informatique\n"  
print(var.rstrip())
```

## Les fonctions/méthodes prédéfinies - upper() / lower()



- **Upper()** : retourne la chaîne en majuscules.

```
a = "table"
```

```
print(a.upper())
```

```
> TABLE
```

- **Lower()** : retourne la chaîne en minuscules.

```
a = "TABLE"
```

```
print(a.lower())
```

```
> table
```

## Les fonctions/méthodes prédéfinies - count()



- **count()** : compte le nombre d'occurrences d'une valeur dans une liste.

```
st= [123, 'abc', 'efg', 'abc', 123]
```

```
print (st.count('efg'))
```

```
print (st.count(123))
```

```
> 1
```

```
2
```

## Les fonctions/méthodes prédéfinies - count()

- **count()** : retourne le nombre d'occurrences d'une sous-chaîne dans une chaîne.

```
ch = "Python est un langage de programmation."
```

```
s = "on"
```

```
print(ch.count(s))
```

```
> 2
```

On peut préciser l'indice de début et l'indice de fin de la recherche.

```
ch = "Python est un langage de programmation."
```

```
s = "o"
```

```
print(ch.count(s,4,36))
```

```
> 3
```

## Les fonctions/méthodes prédéfinies - replace()



```
a="Je suis étudiant à l'USTHB"  
print(a.replace('USTHB','ESI'))  
> Je suis étudiant à l'ESI
```

## Création d'une fonction - def



- `def f(p1,p2):` #en-tête de la fonction
- `print(p1+p2)` #corps de la fonction
- `f(5,6)` # appel de la fonction
- `> 11`
- `#-----#`
- `def date(jour, mois,an):`
- `return str(jour) + " " + mois + " " + str(an)`
- `print(date(5,"janvier",2000))`
- `> 5 janvier 2000`

Définition  
de la  
fonction

## Création d'une fonction - def



- `t=["5","janvier","2000"]`
- `def date(tab):`
- `return tab[0] + " " + tab[1] + " " + tab[2]`
- `print(date(t))`
- `> 5 janvier 2000`
- `#-----`  
`-----#`
- `dic={'976':'Mongolia','52':'Mexico','212':'Morocco','64':'New Zealand','33':'France'}`
- `def date(tab):`
- `return tab.keys()`
- `print("Résultat ",date(dic))`
- `> Résultat dict_keys(['64', '33', '976', '52', '212'])`