

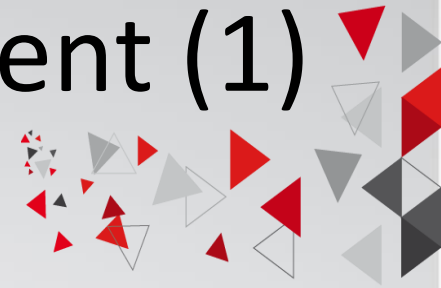


Langage Python

UP Web

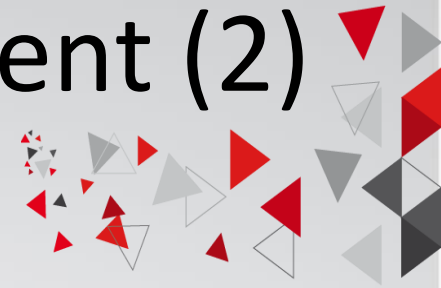
AU: 2020/2021

Installation de l'environnement (1)

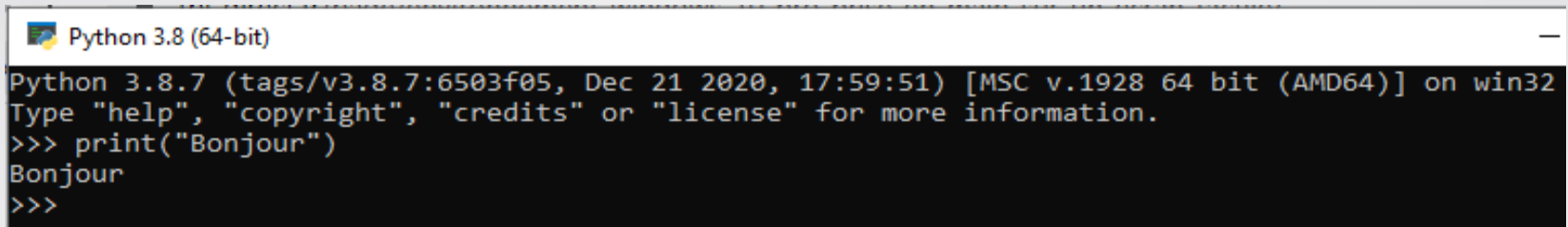


- Essayer Python sans installer de logiciel sur ton ordinateur
- **Windows:**
 - Télécharger la dernière version python compatible avec votre système d'exploitation à partir de :
www.python.org/downloads/
- **Linux:**
 - Taper la commande « `which python3` » pour vérifier si la version 3 est bien installée
 - Sinon exécuter cette commande: `sudo apt-get install python3`

Installation de l'environnement (2)



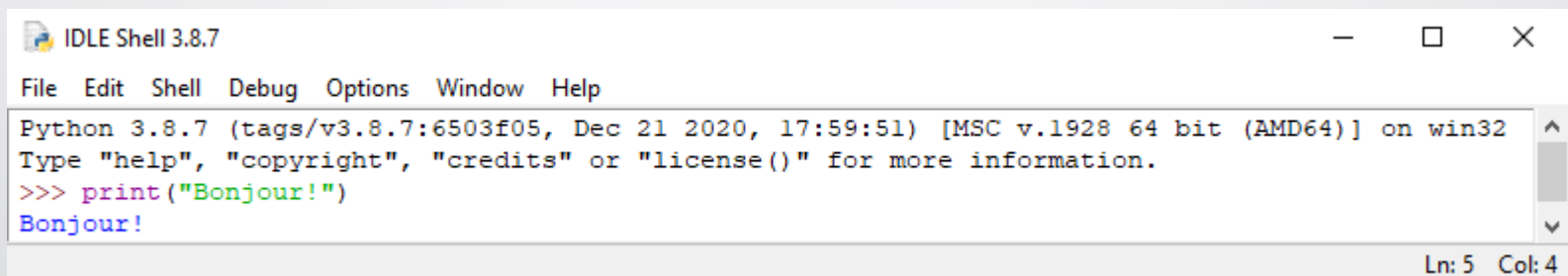
■ Environnement Interactif



```
Python 3.8 (64-bit)
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:59:51) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Bonjour")
Bonjour
>>>
```

■ IDE:

- Anaconda ->Jupyter à installer via ce lien :
www.anaconda.com/distribution/#download-section
- Utiliser l'IDE (Python IDLE)



```
IDLE Shell 3.8.7
File Edit Shell Debug Options Window Help
Python 3.8.7 (tags/v3.8.7:6503f05, Dec 21 2020, 17:59:51) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Bonjour!")
Bonjour!
Ln: 5 Col: 4
```

Opérateurs arithmétiques



- Les opérateurs arithmétiques en Python sont:

- + Pour l'addition
- - Pour la soustraction
- * Pour la multiplication
- / Pour la division

```
>>> print(2+1)
3
```

```
>>> print(1+3+5*2)
14
>>> print((1+3+5)*2)
18
```

- L'usage des parenthèses est important!

- ** est utilisé pour la puissance

```
x=2 ** 2
print(x)
```

```
4
```

- % opérateur « modulo »

```
>>> print(11%2)
1
```

- // est l'opérateur « div »

```
>>> print(17//4)
4
```

Les variables (1)



```
>>> x,y,z= 5,6,7
>>> print(x)
5
>>> print(y)
6
>>> print(z)
7
```

- Créer plusieurs variables en même temps

```
>>> x=y=3
>>> print(y)
3
```

- Assigner la valeur d'une variable à une autre variable

```
>>> x+=1
>>> x-=1
>>> x*=0.95
>>> x/=2
```

- Incrémentation
- Décrémententation
- Multiplication
- Division

```
x = 5 # 5 est assigné à x
```

- Faire un commentaire

```
"""
Ceci
est|
un commentaire
multilignes
"""

x = 5
print(x)
```

Les variables (2)



```
>>> x=5
>>> print(type(x))
<class 'int'>
>>> x=5/2
>>> print(type(x))
<class 'float'>
>>> print(int(x))
2
```

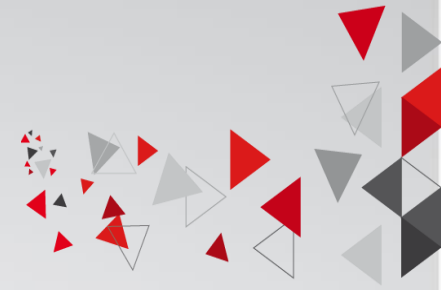
- Extraire le type d'une variable
- Conversion de type
- On perd de l'information lorsqu'on transforme des float en int → casting

```
>>> resultat=5.5
>>> resultat_int=int(5.5)
>>> print(resultat_int)
5
>>> resultat_float=float(resultat_int)
>>> print(resultat_float)
5.0
```

- Vérification du type isinstance()

```
>>> x=5
>>> print(x)
5
>>> isinstance(x,int)
True
>>> isinstance(x,float)
False
```

Application (1)



- La Valeur du volume courant d'eau dans le barrage du Bir Mcherga

reservoir_volume = 4.445e8

- La quantité de pluie → precipitation = 5e6

1. Décrémenter par 50% la quantité de la pluie
2. Ajouter le nouveau volume d'eau au barrage
3. Décrémenter par 5% le volume d'eau du barrage à cause de l'évaporation
4. afficher la nouvelle valeur du volume d'eau

Opérateurs de Comparaison



- Variables de type Booléen (bool)

```
>>> print(type(1))  
<class 'int'  
>>> print(type(True))  
<class 'bool'
```

```
if 1 == True:  
    print("1 et True sont équivalents!")  
else:  
    print("1 et True ne sont pas équivalents!")  
  
1 et True sont équivalents!
```

- On obtient un résultat de type booléen lors d'une opération de comparaison

```
>>> 2 < 5  
True  
>>> 3 > 9  
False
```

- Les opérateurs de comparaison

Inférieur strictement	<	Supérieur	>=
Supérieur strictement	>	Egale	==
Inférieur	<=	Différent	!=

Opérateurs Logiques

- L'opérateur **ET** → **and**: Il est vrai seulement quand les deux opérateurs sont vraies, sinon il est faux
- L'opérateur **OU** → **or**: Il est vrai quand l'une des expressions est vraie, et faux si les deux opérateurs sont fausses
- L'opérateur **NON** → **not**: Il évalue l'inverse de la valeur de vérité d'une expression, donc faux si l'expression est vraie, et vraie si l'expression est fausse

```
x= True and True  
y=True and False  
z=False and False  
print(x)  
print(y)  
print(z)
```

```
True  
False  
False
```

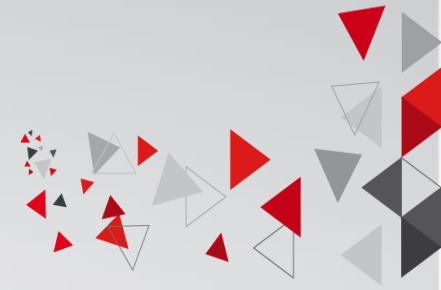
```
x= True or True  
y=True or False  
z=False or False  
print(x)  
print(y)  
print(z)
```

```
True  
True  
False
```

```
x= not True  
y=not False  
print(x)  
print(y)
```

```
False  
True
```

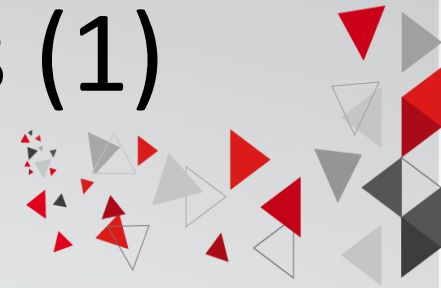
Application (2)



```
>>tomate_marche_x, pommes_de_terre_marche_x = 920, 870  
>>tomate_marche_y, pommes_de_terre_marche_y = 890, 890
```

#Ecrire Le code permettant d'afficher "True" si l'achat des légumes du
#marche_x est une bonne affaire et "False" sinon

Les chaînes de caractères (1)



- Le type **string** en python permet de manipuler les chaînes de caractère

```
x="je suis un String"
print(type(x))
```

- Accès à un string via son index ➔ random-access

```
x="je suis un String"
print(x[3])
```

s

- L'opérateur + permet de concaténer deux chaînes de caractères

```
x= " je suis "
y= "un String"
print(x+y)
```

je suis un String

- Multiplication des chaînes

```
x="abc"
print(x*3)
```

abccabccabc

- Obtenir une sous séquence de la chaîne

```
x="abcde"
print(x[0:3])
```

abc

```
x="abcde"
print(x[::2])
```

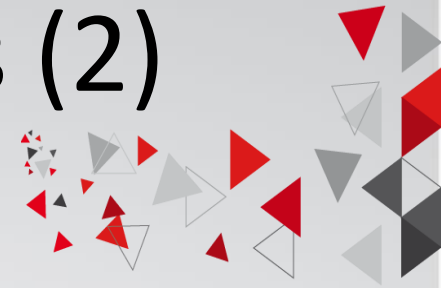
ace

- Vérifier l'existence d'une lettre dans une chaîne

```
x="abcde"
if 'a' in x:
    print("a est dans x")
```

a est dans x

Les chaînes de caractères (2)



- Chaque type de donnée a son propre symbole
 - %c pour les caractères
 - %s pour les chaînes de caractères
 - %d pour les integers
 - %f pour les floats

```
x="ceci est une %s de %s" % ("chaîne", "caractères")  
print(x)
```

```
ceci est une chaîne de caractères
```

- Une méthode prédéfinie **len** permet de retourner la longueur d'une chaîne de caractères

```
>>> name_length = len("django")  
>>> print(name_length)  
6
```

Les inputs utilisateur



```
#demander le nom d'utilisateur  
input("quel est ton nom:")
```

quel est ton nom:

- Demander une information à quelqu'un si tu ne t'en sers pas après, ce n'est finalement pas très utile

```
#on demande l'age de l'utilisateur  
age=input("Quel est ton age ?")  
#On affiche un message si il ou elle a moins de 7ans  
if age<7:  
    print("Quel beau jeune Pythoniste")
```

Ce code génère une erreur à l'exécution ! En effet, le type de la variable "age_utilisateur" est "string" et Python refuse de comparer une chaîne de caractères avec un chiffre.

Dans ces cas là, la solution est de convertir la variable en int (à l'aide du casting int()). Ensuite, pas de problèmes pour utiliser :

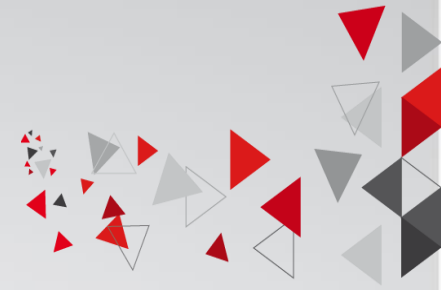
```
#on demande l'age de l'utilisateur  
age=input("Quel est ton age ?")  
  
#On convertit son age en String  
age=int(age)  
  
#On affiche un message si il ou elle a moins de 7ans  
if age<7:  
    print("Quel beau jeune Pythoniste")
```

Application (3)



- Pour deux chaînes de caractères que vous saisissez appelées part1 et part2
 - ➔ Écrire le script permettant de retourner:
 - ✓ **True** Si leur concaténation (Espace comme séparateur) est valide comme « Tweet »
 - ✓ **False** Sinon
- Notes
 - ✓ Une Tweet ne dépasse pas 140 caractères
 - ✓ La fonction `input()` permet d'entrer des caractères

Application (4)



- Déclarer les variables suivantes :
 - `depenses_lundi = "30"`
 - `depenses_mardi = "65"`
 - `depenses_jeudi = "12"`
 - `depenses_vendredi = "45"`
 - `depenses_dimanche = "20"`
- Écrire un programme qui affiche: « Durant cette semaine vous avez dépensé #X dinars »

Gérer les chaînes: upper(), lower() et title()...



- Mettre une chaîne en majuscule ou minuscule

```
message="bonjour les TWIN"  
print(message.upper())  
print(message.lower())
```

```
BONJOUR LES TWIN  
bonjour les twin
```

- Formater des noms propres

```
message="bonjour ahmed ben salem"  
print(message.title())
```

```
Bonjour Ahmed Ben Salem
```

- Calculer le nombre de caractère « x »

```
message="bonjour ahmed ben salem"  
print(message.count("m"))
```

```
2
```

- Voir « <https://docs.python.org/3/library/stdtypes.html#string-methods> »

Le formatage des chaînes

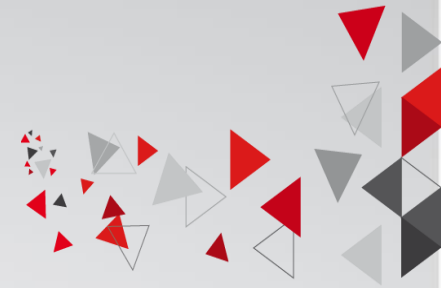


- Une méthode particulièrement utile est **.format**.
- Elle permet de construire des chaînes de caractères tout en suivant des « templates ».

- **Exemple**

```
log_server_access = "IP address {} accessed {} at {}".format(user_ip, url, now)
```

Application (5)



- Déclarer les variables suivantes
 - `city = "Tunis"`
 - `temperature_max = 37`
 - `temperature_min = 12`
 - `unit_temp = "degrees Celsius"`
 - `notif = "Les prévisions pour aujourd'hui pour" + city + ": Température entre " + str(temperature_min) + " et " + str(temperature_max) + " " + unit_temp + "`
- Simplifier cette chaîne de caractères en utilisant le formatage

Les fonctions (1)



- On définit une fonction via le mot clé **def**.

Exemple: une fonction pour le calcul du volume d'un cylindre

- Une fois la fonction est définie, on peut l'appeler

```
def cylindre_volume(height,radius):  
    pi=3.14159
```

```
    return (height*pi*radius)** 2  
cylindre_volume(4,3)
```

```
1421.2206328463997
```

- Une fonction peut ne pas avoir une valeur de retour. La valeur de retour dans ce cas est **None**.

Les fonctions (2)



Documentation

- La documentation est le texte précédé par """

```
def cylinder_volume(height, radius):  
    """ Calcule le volume d'un cylindre height: flaot.  
    Hauteur du cylindre radius: float. Rayon du cylindre """  
    pi = 3.14159  
    return height * pi * radius ** 2
```

- Référence sur les standards de définition des documentations en python
<https://www.python.org/dev/peps/pep-0257/>

Les fonctions (3)



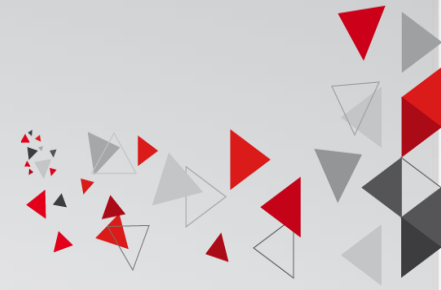
Les arguments par défaut

- Il est possible de définir une fonction avec un argument par défaut.

Exemple

```
def cylinder_volume(height, radius, unité = "m3"):
    """ Affiche le volume d'un cylindre
    L'unité de mesure par défaut est m3 """
```

Application (6)



- Définir une fonction `jours_semaine` permettant de retourner pour une valeur de jours donné une chaîne de caractères indiquant le nombre de semaines et les jours restant.

Structures de contrôle



On utilise le mot clé **if** suivi par

- la condition
- Les instructions dans le bloc « if » sont précédées par une tabulations.

```
if poids_en_kg < 25:  
    print("votre valise est acceptée")
```

Attention aux **indentations** dans vos scripts!

- On utilise les opérateurs booléen **and**, **or** et **not** pour composer les conditions

```
if number % 2 == 0:  
    print("Le numéro " + str(number) + " est pair.")  
else:  
    print("Le numéro " + str(number) + " est impair.")
```

```
if poids_en_kg < 25:  
    print("votre valise est acceptée")  
elif poids >= 25 and poids < 40:  
    print("Acceptée avec des frais")  
else:  
    print("refusée" )
```

Application (7)



- Implémenter la fonction **cylinder_surface** avec la signature suivante

```
def cylinder_surface(height, radius, hasTopAndBottom):  
    """  
    Si le troisième paramètre est vrai  
    alors on doit aussi calculer les surfaces des parties supérieure et inférieure  
    """
```

- Modifier la fonction pour que la valeur par défaut de l'argument **hasTopAndBottom** soit True

Les listes (1)



- Une liste est une collection de données de différents type stockée de manière séquentielle, ce qui permet d'y accéder grâce à un index

```
Jours_semaine = ["Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "samedi", "dimanche"]
print(Jours_semaine[2])
debut_semaine=Jours_semaine[0:3]
print(debut_semaine)
```

```
Mercredi
['Lundi', 'Mardi', 'Mercredi']
```

```
ma_liste=[1,"ee",1.5,'abcde']
print(ma_liste)
```

```
[1, 'ee', 1.5, 'abcde']
```

- Stocker des structures de données dans une liste

```
ma_liste1=[1,"ee",1.5,'abcde']
ma_liste2=['a',10,ma_liste]
print(ma_liste2)
```

```
['a', 10, [1, 'ee', 1.5, 'abcde']]
```

Les listes (2)

- Les éléments d'une liste sont indexable

```
ma_liste1=[1,"ee",1.5,'abcde']  
ma_liste2=['a',10,ma_liste]  
print(ma_liste2[0:2])
```

```
['a', 10]
```

```
ma_liste1=[1,"ee",1.5,'abcde']  
ma_liste2=['a',10,ma_liste]  
print(ma_liste2[2:]) #du 2ème index jusqu'à la fin  
print(ma_liste2[0:3:2]) #identifiant[debut:fin:interval]
```

```
[[1, 'ee', 1.5, 'abcde']]  
['a', [1, 'ee', 1.5, 'abcde']]
```

- Remplacer un élément de ma liste
- On peut concaténer et multiplier des listes

```
ma_liste=[1,"ee",1.5,'abcde']  
print(ma_liste)  
ma_liste[1]='a'  
print(ma_liste)
```

```
[1, 'ee', 1.5, 'abcde']  
[1, 'a', 1.5, 'abcde']
```

```
ma_liste=[1,"ee",1.5,'abcde']  
for element in ma_liste:  
    print(element)
```

```
1  
ee  
1.5  
abcde
```

Les listes (3)



- Transformer une chaîne de caractère (string) en liste **Split**
String → Liste

```
message= "bonjour mes chers !"
print(message)
message2=message.split()
print(message2)
type(message2)
len(message2)
message3= "bonjour-mes-chers !"
message4=message3.split()
print(message4)
message5=message3.split('-')
print(message5)
message6=" ".join(message5)
print(message6)
```

```
bonjour mes chers !
['bonjour', 'mes', 'chers', '!']
['bonjour-mes-chers', '!']
['bonjour', 'mes', 'chers !']
bonjour mes chers !
```

- La méthode **join** permet de retourner une chaîne de caractères contenant les éléments de la liste de chaînes ainsi que la caractère de jointure → Transforme une **liste → String**
- Pour ajouter un élément à une liste, on utilise la méthode **append**.

```
liste_notes=["11","12.5","16","9","17"]
liste_notes2="-".join(liste_notes)
print(liste_notes)
print(liste_notes2)
liste_notes.append("19")
print(liste_notes)
```

```
['11', '12.5', '16', '9', '17']
11-12.5-16-9-17
['11', '12.5', '16', '9', '17', '19']
```

Les listes (4)



- Plusieurs fonctions nous permettent de manipuler les listes

```
ma_liste1=[1,"ee",1.5,'abcde']
ma_liste2=['a',10,ma_liste1]
ma_liste2.insert(2,"bonjour")
print(ma_liste2)
ma_liste2.remove("bonjour")
print(ma_liste2)

['a', 10, 'bonjour', [1, 'ee', 1.5, 'abcde']]
['a', 10, [1, 'ee', 1.5, 'abcde']]
```

```
>>> ma_liste=[1,2,3]
>>> ma_liste.append([1,2,3])
>>> print(ma_liste)
[1, 2, 3, [1, 2, 3]]
>>> len(ma_liste)
4
```

```
>>> ma_liste.count(1) #combien de fois un elt est présent dans une liste
1
>>> ma_liste_2=[2,3,4,5]
>>> ma_liste.extend(ma_liste_2) #ajouter une séquence d'elt
>>> print(ma_liste)
[1, 2, 3, [1, 2, 3], 2, 3, 4, 5]
>>> x=ma_liste.index(3)
>>> print("L'element 3 apparaît en premier à l'index %d" % x)
L'element 3 apparaît en premier à l'index 2
```

```
>>> liste_notes = [11, 12.5, 16,9, 17]
>>> len(liste_notes)
5
>>> max(liste_notes)
17
>>> min(liste_notes)
9
>>> sorted(liste_notes)
[9, 11, 12.5, 16, 17]
>>> sorted(liste_notes, reverse=True)
[17, 16, 12.5, 11, 9]
```

- Plus de méthodes pour la manipulation des listes:

<https://docs.python.org/3/tutorial/datastructures.html>

Les listes (5)



- Plusieurs fonctions nous permettent de manipuler les listes

```
>>> ma_liste=[1,2,3]
>>> ma_liste.append([1,2,3])
>>> print(ma_liste)
[1, 2, 3, [1, 2, 3]]
>>> len(ma_liste)
4
```

```
>>> ma_liste.count(1) #combien de fois un elt est présent dans une liste
1
>>> ma_liste_2=[2,3,4,5]
>>> ma_liste.extend(ma_liste_2) #ajouter une séquence d'elt
>>> print(ma_liste)
[1, 2, 3, [1, 2, 3], 2, 3, 4, 5]
>>> x=ma_liste.index(3)
>>> print("L'element 3 apparaît en premier à l'index %d" % x)
L'element 3 apparaît en premier à l'index 2
```

```
ma_liste1=[1,"ee",1.5,'abcde']
ma_liste2=['a',10,ma_liste1]
ma_liste2.insert(2,"bonjour")
print(ma_liste2)
ma_liste2.remove("bonjour")
print(ma_liste2)

['a', 10, 'bonjour', [1, 'ee', 1.5, 'abcde']]
['a', 10, [1, 'ee', 1.5, 'abcde']]
```

```
>>> liste_notes = [11, 12.5, 16,9, 17]
>>> len(liste_notes)
5
>>> max(liste_notes)
17
>>> min(liste_notes)
9
>>> sorted(liste_notes)
[9, 11, 12.5, 16, 17]
>>> sorted(liste_notes, reverse= True)
[17, 16, 12.5, 11, 9]
```

- Plus de méthodes pour la manipulation des listes:

<https://docs.python.org/3/tutorial/datastructures.html>

Les listes (6)



- Parcourir les listes

```
ma_liste=[1,"ee",1.5,'abcde']  
for element in ma_liste:  
    print(element)
```

```
1  
ee  
1.5  
abcde
```

```
jours_semaine = ["Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi", "samedi", "dimanche"]  
for index,jour in enumerate(jours_semaine):  
    print(index,jour)
```

```
0 Lundi  
1 Mardi  
2 Mercredi  
3 Jeudi  
4 Vendredi  
5 samedi  
6 dimanche
```

Les boucles



- Dans le cas où le nombre d'itérations est inconnu, on peut avoir recours à la boucle **while**
- Répéter un bloc de code tant que sa condition est fausse

```
x=5
while (x!=0): #tant que x est sup à 0
    print(x)
    x=x-1 #retenir 1 à x
print("fin du programme")
```

```
5
4
3
2
1
fin du programme
```

```
>>> bonus_jeu = [1,3,5,7]
>>> score = []
>>> while sum(score) < 10:
>>>     score.append(bonus_jeu.pop())
>>> print(score)
[7, 5]
```

- **pop**: Dépile le dernier élément d'une list
- On quitte une boucle « while » par le mot clé **break**

« Mutable » vs « Non Mutable »



- Une chaîne de caractères peut être gérée comme étant une liste de caractères mais sans oublier qu'une chaîne de caractères n'est pas **mutable** pareil pour les tuples

```
tuples=(1,'a',True,1.23)
type(tuples)
#on essaie de modifier une valeur
tuples[0]=2
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-61-107df1ab95ab> in <module>
      2 type(tuples)
      3 #on essaie de modifier une valeur
----> 4 tuples[0]=2
```

```
TypeError: 'tuple' object does not support item assignment
```

```
>>> twin_groupes = ['twin51', 'twin52']
>>> twin_groupes_str = 'twin51 twin52'
>>> twin_groupes[1]
'twin52'
```

```
>>> twin_groupes_str[5]
'1'
```

```
>>> twin_groupes[1] = 'twin53'
```

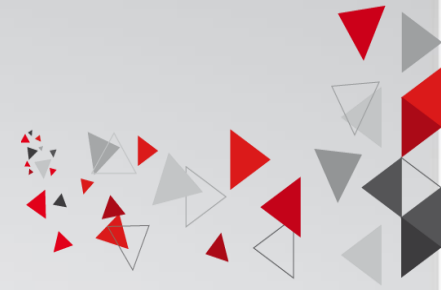
```
>>> twin_groupes_str[5] = '3'
```

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

```
File "<pyshell#5>", line 1, in <module>
    twin_groupes_str[5] = '3'
```

```
TypeError: 'str' object does not support item
assignment
```


Application (8)



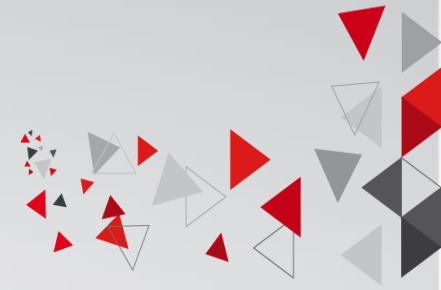
Calculer les tags XML

- Écrire une fonction « tag_counter » permettant de calculer le nombre des tags XML dans une liste.

Exemple

```
>>> liste = ['<etudiant>', 'Ahmed Ben Amine', '</etudiant>']  
>>> print(tag_counter(liste))  
2
```

Application (9)



Construire une liste

- Définir une fonction « `html_list` » qui permet de convertir une liste de valeurs en une liste HTML.
- Exemple:

```
>>> list_to_html = ['first string', 'second string']
```

```
>>> html_list(list_to_html)
```

```
<ul>
```

```
<li>first string</li>
```

```
<li>second string</li>
```

```
</ul>
```

- Indication: Pensez à utiliser la fonction **`range`**.

Application (10)



Construire une liste

- Définir la fonction `affiche_liste` dont les paramètres sont les suivants:
 - ✓ **L** : liste des valeurs
 - ✓ **Numéroté**: de type Booléen, False par défaut.

Exemple: `liste_menu_du_jours = ['Spaghetti', 'Salade César', 'Escalope panné']`

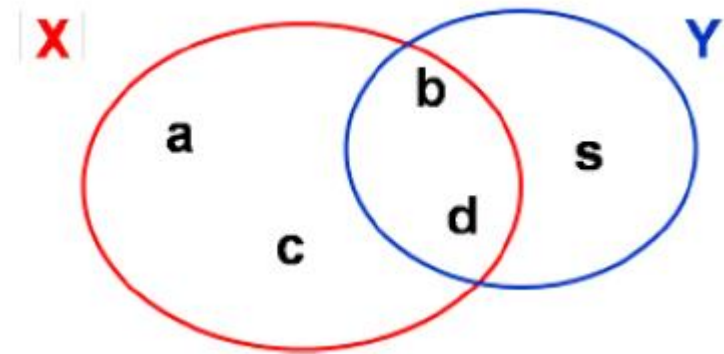
- `affiche_liste(liste_menu_du_jours , True)`
 - a. Spaghetti
 - b. Salade Cesar
 - c. Escalope panné
- `affiche_liste(liste_menu_du_jours)`
 - Spaghetti
 - Salade César
 - Escalope panné

La collection « Set »

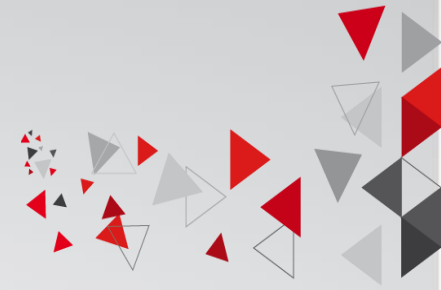


- Une autre structure de données en python qui permet d'avoir des éléments non-dupliqués est la structure « **set** ».
- On peut créer une set:
- A partir d'une liste: `my_set = set(ma_liste)`
- Vide: `my_empty_set = set()`
- On peut ajouter un élément dans une « set » par le mot clé **add**.

```
X, Y = set('abcd'), set('sbd')
print("X =", X) # X = {'a', 'c', 'b', 'd'}
print("Y =", Y) # Y = {'s', 'b', 'd'} : un seul élément 's'
print('c' in X) # True
print('a' in Y) # False
print(X - Y) # {'a', 'c'}
print(Y - X) # {'s'}
print(X | Y) # {'a', 'c', 'b', 'd', 's'}
print(X & Y) # {'b', 'd'}
```



Les dictionnaires (1)



- Les dictionnaires sont une structure de données qui sauvegardent une paire de données de la forme **clé** → **valeur**.
- Exemple:

```
elements = {'hydrogen': 1, 'helium': 2, 'carbon': 6}
```

- Pour accéder à un élément:

```
>>> print(elements['carbon'])  
6
```

- Pour ajouter un nouvel élément:

```
>>> elements['lithium'] = 3
```

- Une autre manière pour accéder à un élément est l'usage de la méthode **get**

```
>>> print(elements.get('carbon'))  
6
```

Les dictionnaires (2)



- Vider une dictionnaire

```
>>> mon_dico={0:'a',1:'b',2:'c'}
>>> print(mon_dico)
{0: 'a', 1: 'b', 2: 'c'}
>>> print(type(mon_dico))
<class 'dict'>
>>> mon_dico.clear()
>>> print(mon_dico)
{}
>>> print(type(mon_dico))
<class 'dict'>
```

- Copier les éléments d'un dictionnaire dans un autre dictionnaire

```
>>> mon_dico={0:'a',1:'b',2:'c'}
>>> mon_dico2=mon_dico.copy()
>>> print(mon_dico)
{0: 'a', 1: 'b', 2: 'c'}
>>> print(mon_dico2)
{0: 'a', 1: 'b', 2: 'c'}
>>> for x in mon_dico2:
    print(id(x))
```

```
1602082976
1602082992
1602083008
```

```
>>> for x in mon_dico:
    print(id(x))
```

```
1602082976
1602082992
1602083008
```

Les dictionnaires (3)

- Merger deux dictionnaires

```
>>> mon_dico1={0:'a',1:'b',2:'c'}
>>> mon_dico2={0:'a',1:'b',2:'c',4:'z',5:'x'}
>>> mon_dico1.update(mon_dico2)
>>> print(mon_dico1)
{0: 'a', 1: 'b', 2: 'c', 4: 'z', 5: 'x'}
```

- Retirer un élément du dictionnaire grâce à sa clef et retourner sa valeur

```
>>> mon_dico={0:'a',1:'b',2:'c'}
>>> print(mon_dico)
{0: 'a', 1: 'b', 2: 'c'}
>>> mon_dico.pop(2)
'c'
>>> print(mon_dico)
{0: 'a', 1: 'b'}
```

- Retirer un élément du dictionnaire grâce à sa clef et retourner à la fois son index et sa valeur

```
>>> mon_dico={0:'a',1:'b',2:'c'}
>>> mon_dico.popitem()
(2, 'c')
```

- Obtenir la liste des clefs du dictionnaire
- Obtenir la liste des valeurs du dictionnaire

```
>>> mon_dico={0:'a',1:'b',2:'c'}
>>> x=mon_dico.keys()
>>> print(x)
dict_keys([0, 1, 2])
```

```
>>> mon_dico={0:'a',1:'b',2:'c'}
>>> x=mon_dico.values()
>>> print(x)
dict_values(['a', 'b', 'c'])
```

Application (11)



On souhaite connaître le nombre d'utilisateurs de notre site par pays. On détient uniquement d'une liste contenant les pays des visiteurs.

1. Écrire un script permettant de:
 - ✓ Créer un dictionnaire **pays_count** de la forme {pays, nombre de visiteurs}
 - ✓ Écrire le script dans un fichier **stats.py**
 - ✓ Afficher le résultat
2. Créer le fichier python nommé **countries.py**:

```
list_countries=['ALGERIE', 'TUNISIA', 'ALGERIE', 'TUNISIA', 'EGYPT']
```

Dans **stats.py**, vous importez la liste « **country_list** » en ajoutant

```
from countries import country_list
```

3. Utiliser une boucle « for » pour afficher un message de la forme:
Les utilisateurs du pays X ont accédé Y fois.

Les structures composites



- Selon notre besoin de représentation des données, il est possible de construire des structures composites de données.
- Exemple, soit le dictionnaire suivant:

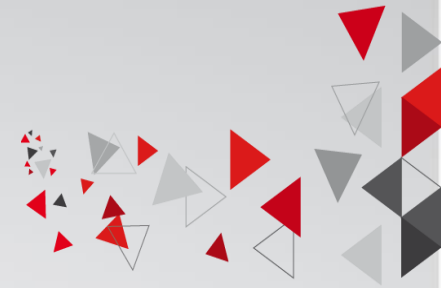
elements = {'hydrogen': {'number': 1, 'weight': 1.00794, 'symbol': 'H'},

'helium': {'number': 2, 'weight': 4.002602, 'symbol': 'He'}}

Il est possible d'accéder aux données ainsi:

```
>>> print(elements.get('hydrogen'))  
{'number': 1, 'weight': 1.00794, 'symbol': 'H'}  
>>> print(elements.get('hydrogen').get('weight'))  
1.00794  
>>> print(elements['hydrogen']['weight'])  
1.00794
```

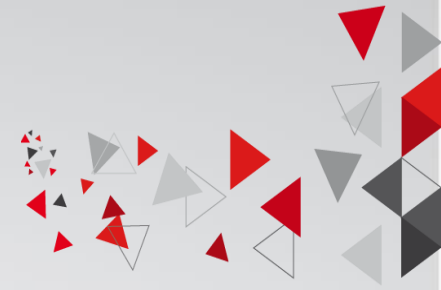
Application (12)



- Définir une fonction **calcul_annuel** qui permet de calculer les revenus annuel.
- Les données sont représentés ainsi:

```
revenus_mensuels = {'Janvier': [54, 63], 'Février': [64, 60], 'Mars': [63, 49],  
                    'Avril': [57, 42], 'Mai': [55, 37], 'Juin': [34, 32],  
                    'Juillet': [69, 41, 32], 'Aout': [40, 61, 40], 'Septembre': [51, 62],  
                    'Octobre': [34, 58, 45], 'Novembre': [67, 44], 'Décembre': [41, 58]}
```

Les tuples



- Il est parfois utile de sauvegarder un ensemble corrélé de données.
- Exemple latitude_longitude = (10.3271, 11.5473)

```
>>> latitude_longitude[0]  
10.3271  
>>> latitude_longitude[1]  
11.5473  
>>>
```

- Un « tuple » peut avoir n dimensions.
- Une fonction peut retourner un tuple de valeurs.

Bilan des types en Python



Type	Ordonné	Mutable	Parcours par indice	Parcours par éléments
str	✓	✗	✓	✓
list	✓	✓	✓	✓
set	✗	✓	✗	✓
dict	✗	✓	✗	✓
tuple	✓	✗	✓	✓

La Manipulation Des Fichiers



Fmode d'ouverture du fichier:
« w »= write (écriture)
« r »= read (lecture)
...

Fermer systématiquement après une opération

Ouvrir un fichier

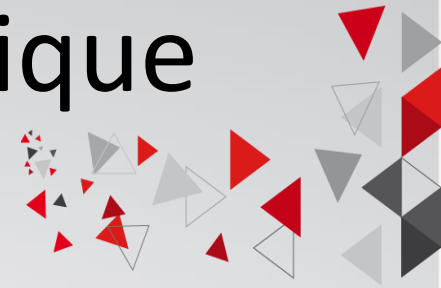
Variable qui va permettre
la manipulation du fichier

Retourner à la ligne

```
with open "C:/Users/Desktop/monFichier.txt", "w" as ff:  
    ff.write("je commence à avoir du niveau en python!\n Voici une deuxième ligne")
```

```
>>> f = open("C:/Users/RDouss/Desktop/monFichier.txt", 'r')  
>>> f.read()  
'je commence à avoir du niveau en python!\n Voici une deuxième ligne'  
>>> f.write("Bonjour")  
Traceback (most recent call last):  
  File "<pyshell#35>", line 1, in <module>  
    f.write("Bonjour")  
io.UnsupportedOperation: not writable  
>>> f.close()  
>>> f.read()  
Traceback (most recent call last):  
  File "<pyshell#37>", line 1, in <module>  
    f.read()  
ValueError: I/O operation on closed file.  
>>> |
```

Les constantes mathématique



- La constante , c'est un nombre spécial (en général un réel) qui a des propriétés utiles en mathématique, tel que:
 - Le nombre π
 - Le nombre e
 - Le nombre $\sqrt{\quad}$
 - Le nombre i

```
>>> from math import pi
>>> x=pi
>>> print(x)
3.141592653589793
```

```
>>> from math import sqrt
>>> x=sqrt(2)
>>> print("La valeur de la racine carrée de 2 est %f" %x)
La valeur de la racine carrée de 2 est 1.414214
```

La bibliothèque standard



- Python offre une bibliothèque standard « Python Standard Library » qui inclut un ensemble de bibliothèques souvent utiles.
- Importer un module est via le mot clé import.

Exemple

```
import math  
print(math.factorial(3))
```

- En parcourant la bibliothèque standard (<https://docs.python.org/3/library/>), on trouve:
 - ✓ **csv**: Lire et écrire à partir des fichiers CSV
 - ✓ **Collections**: Extension des structures de données prédéfinies.
 - ✓ **random**: Génération de nombres aléatoires
 - ✓ **re**: Manipulation des expressions régulières
 - ✓ **math**: Les opérations mathématiques
 - ✓ **os**: Intéragir avec le système d'exploitation
 - ✓ **os.path**: Un sous module de os pour la manipulation des chemins.
 - ✓ **json**: Lire et écrire à partir des fichiers JSON

Utiliser des modules



- On peut importer une classe à partir d'une librairie ainsi:

```
>>> from collections import defaultdict
```

- De même pour des importations multiples

```
>>> from collections import defaultdict,namedtuple
```

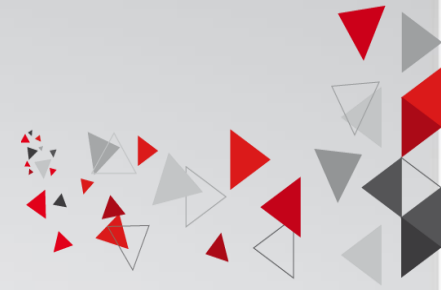
- Pour renommer un module importé: utiliser le mot clé as

```
>>> from csv import reader as csvreader
```

- A éviter

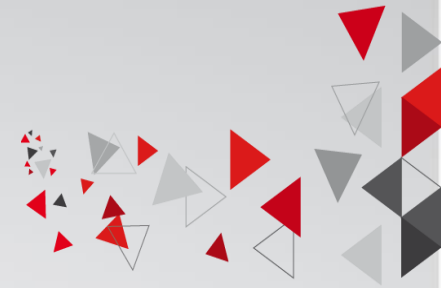
```
>>> from module_name import *
```


Application (13)



1. En utilisant la librairie « os » , écrire un script permettant d'afficher les documents dans le répertoire du script.
2. Écrire un script permettant de renommer tous les fichiers dont l'extension est «txt ».
3. En utilisant la librairie « collections » , réécrire une deuxième version du script «stats.py »

POO: Les classes (1)



Geo.py

```
class Rectangle():
    def __init__(self, largeur, longueur):
        self.largeur = largeur
        self.longueur = longueur
    def calcul_surface(self):
        return self.largeur * self.longueur
```

Test_class.py

```
from geo import Rectangle

rect = Rectangle(5,7)
print(rect.calcul_surface())
```

- Rectangle représente le **nom de la classe**.
- largeur et longueur sont les **attributs** de la classe.
- **__init__(self, largeur, longueur)** est le **constructeur** de la classe.

- Rect = Rectangle(5,7) est l'**instanciation** d'un objet de type Rectangle
- On accède à un attribut via **nom_objet.nom_attribut**

POO: Les classes (2)



- `rect = Rectangle(5,7)`
- `print(rect.largeur)`

- Qu'est-ce qu'il y'a derrière?
 1. `__init__` est appelée
 2. `self` pointe sur l'objet `rect`
 3. `largeur` → 5
 4. `longueur` → 7
- Un objet en mémoire est créé avec les attributs **largeur** et **longueur** qui sont des **attributs d'instance**

PОО: Les variables de classe



```
class Rectangle():
    UNITS = ["mm2", "cm2", "m2"]
    def __init__(self, largeur, longueur):
        self.largeur = largeur
        self.longueur = longueur
    def calcul_surface(self):
        return self.largeur * self.longueur
```

```
from geo import Rectangle
print(Rectangle.UNITS)
```

- UNITS est une **variable de classe**
- On accède une variable de classe via:
NomClasse.NOMVARIABLE

POO: Les variables prédéfinies



```
class Rectangle():  
    """Ceci est une documentation"""
```

```
from geo import Rectangle  
print(Rectangle.__doc__)
```

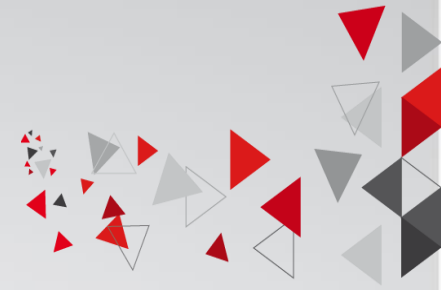
- On définit la documentation avec """
- On peut accéder à la documentation de la classe en utilisant la variable prédéfinie **__doc__**
- Il existe d'autres méthodes prédéfinies en python (**__dict__** , **__name__** , **__module__**)

POO: Les méthodes prédéfinies



- Python détient des méthodes prédéfinies pour la manipulation des classes, exemples:
 - ✓ **__getattribut__** : Accéder à un attribut par son nom
 - ✓ **__setattr__** : Modifier la valeur d'un attribut
 - ✓ **__str__** : Retourne une représentation de l'objet sous forme de chaîne de caractères
 - ✓ **__del__** : Supprimer une instance
 - ✓ ...
- Il est possible de surcharger l'opérateur en définissant une méthode **__add__**

Application (14)



1. Implémentez la classe rectangle défini par les attributs largeur et longueur.
2. Surcharger l'opérateur d'addition.
3. Instanciez deux objets de type rectangle.
4. Affichez les propriétés relatives à ces instances.
5. Appliquez l'opérateur d'addition.

POO: Héritage



- Il est possible de construire une nouvelle classe en se basant sur une classe existante: c'est le principe de **l'héritage**.

Exemple

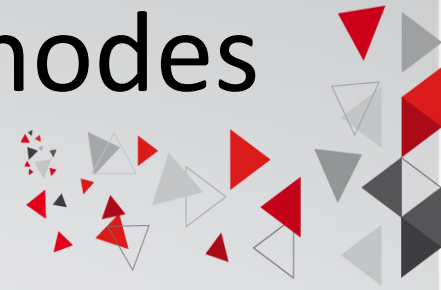
```
>>> class RectangleCouleur(Rectangle):  
    def __init__(self, largeur, longueur, couleur):  
        Rectangle.__init__(self, largeur, longueur)  
        self.couleur = couleur
```

```
>>> class Rectangle():  
    def __init__(self, largeur, longueur):  
        self.largeur = largeur  
        self.longueur = longueur
```

```
from geo import Rectangle, RectangleCouleur  
  
rect = Rectangle(5, 7)  
rect_couleur = RectangleCouleur(5, 7, "Vert")  
print(rect)  
print(rect_couleur)
```

On a le même affichage! ➔ Il faut redéfinir la méthode **__str__** dans RectangleCouleur

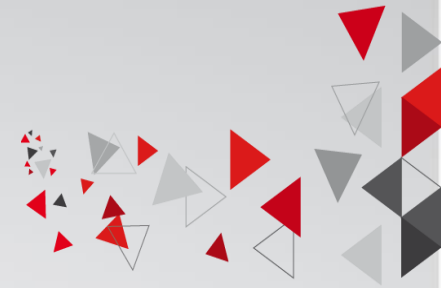
POO: Redéfinition des méthodes



```
class Rectangle():
    def __init__(self, largeur, longueur):
        self.largeur = largeur
        self.longueur = longueur
    def __str__(self):
        return "Je suis un rectangle de largeur {} et de longueur {}".format(self.largeur, self.
        longueur)

class RectangleColore(Rectangle):
    def __init__(self, largeur, longueur, couleur):
        Rectangle.__init__(self, largeur, longueur)
        self.couleur = couleur
    def __str__(self):
        return "Je suis un rectangle de largeur {} et de longueur {} et de couleur {}".format(
        self.largeur, self.longueur, self.couleur)
```

Application (15)



1. Définir une classe Time caractérisée par:
 - ✓ Secondes
 - ✓ Minutes
 - ✓ Heures
1. Redéfinir la méthode `__str__`
2. Surcharger l'opérateur d'addition
3. Définir une méthode lancer heure permettant d'augmenter l'heure chaque seconde
4. Définir une classe fille AngloTime qui prend en charge les heures sous forme AM et PM.

Références



- <https://docs.python.org/3/using/windows.html#excursus-setting-environment-variables>
- <https://google.github.io/styleguide/pyguide.html>
- <https://docs.python.org/3/reference/datamodel.html#specialnames>
- <https://readthedocs.org>
- <https://doughellmann.com/blog/>