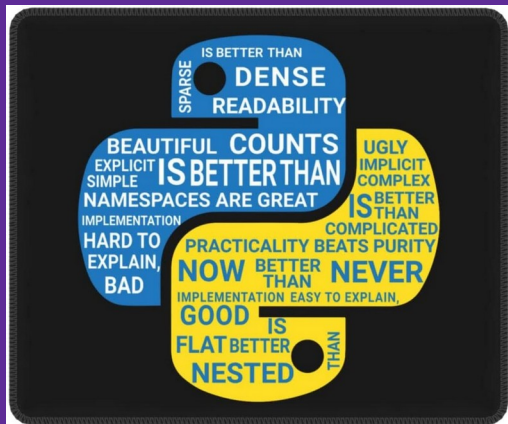


Nom intervenant.e : Charlène Benke (Elle/Elle)

Python - Les bases

Cours n° 001

Version du support : 1.0.2



Plan de cours

- **Les bases de python**
- Dossiers, Fichiers et Librairies avec python
- Bases de données SQL et data
- Python pour l'administration
- Nmaps et scapy
- Outils et Métiers de Développement

Une histoire de python

- 1980 Conçu par Guido Van Rossum
- 2000 sortie de python 2
 - (compatibilité Unicode, chaîne de caractère)
- 2008 python 3 (simplification et maturité) – (version actuelle 3.11)
- 2018 Guido quitte le projet python

Pourquoi utiliser python

- Simple à écrire (lisibilité), déployer, utiliser
 - Open-Source (forte communauté d'utilisateur)
 - Langage interprété et pseudo-compilé (performance)
 - Multiplateforme (python est partout)
 - Lisibilité (langage de haut niveau, indentation, ...), facilite son apprentissage
 - Extensibilité (très nombreuses bibliothèques 'clé en main')
-
- Idéal pour réaliser des applications locales, l'automatisation (vs internet)
 - il est aussi possible de créer des application web en python (Django)
 - Très utilisé dans le domaine de l'administration système, l'automatisation (batch)
 - Très utilisé dans le domaine de la Data (pandas, webscrapping, elk, ...)

Les bases de python

- Les Données
 - Les commentaires
 - Les Types de bases
- Structures de contrôles (boucles et conditions)
 - For, while
 - If, elif, else
- Les fonctions
 - Déclaration, utilisation, importation
- Les objets et les librairies

Les commentaires

- Toute ligne avec un #

Ceci est un commentaire

valeur = 3 # un commentaire après

"""

Il est possible de commenter
plusieurs lignes à la fois

Avec 3 double-quote en début et en fin de commentaire

"""

Les variables - *Tout est objet*

- Une variable se définit par
 - Son nom
 - Son type (numérique, chaîne, booléen, ...)
 - Des actions possibles
 - Une adresse en mémoire
- La notion de « classe d'objet » va généraliser la notion de type.
- Classe : définition de la structure d'un objet
- Objet : Nom + données (attributs) + opérations sur les données (méthodes)

Le nom d'une variable

- Un nom de variable en python peut être défini en lettres minuscules, en majuscules, avec les entiers de 0 à 9 et le caractère _ (underscore). Un nom de variable peut commencer par une lettre, par un underscore
- mais pas par un chiffre.
Variable _autreVariable __CestOkAussi 0Pasbon
- Les noms de variables sont sensibles à la « casse » id <> ID
- Il existe des conventions de nommage (PEP8)
 - lower_case_underscore est à utiliser pour les fonctions
 - « camelCase » est à utiliser pour les classes
 - Pour les variables, les deux précédentes conventions sont possibles (mais en choisir une)
- Utiliser des noms de variable EXPLICITE toto01 nbClient V3
tblQuestion
- Il est pertinent de rajouter un indicateur du type de la variable dans son nom

Les types de variables

- Définie les opérations que l'on pourra réaliser sur ces données
- Types de données de base
 - Numérique **int, float et complex** avec des bases diverses (binaire, octale, décimal, hexadécimal)
 - Chaîne de caractères **str** (avec et sans limite)
 - Les booléen **bool** (Vrai, faux – true / false)
 - Dates et les heures
 - Les listes
- Types de données avancés
 - Les objets (on en reparle plus tard)
 - Les collections (tableau, dictionnaire, tuples)

Les actions / opérateurs possibles

- Selon le type de variable il existe un ensemble d'actions possible
 - Arithmétiques : Addition, Soustraction, ...
 - Affectations ou d'assignation (Variable = Valeur)
 - Chaîne de caractère : concaténation, répétition, découpage, remplacement, ...
 - Comparaison (structure de contrôle)
 - Identité, appartenance (collections)
 - Booléen : Logique booléenne
- Pour un objet, les actions possibles sont définies dans l'objet (on y revient...)
- Pour les listes (Lecture, Ajout, Modification, Suppression, Trie, Filtrage, ...)

Les variables numériques

- Principales fonctions mathématique (+, -, /, *, ...)

- $i = i + 1$ équivaut à $i += 1$

age = 55

Print ('vous avez ' + age) # erreur de conversion

Print ('vous avez : ' + str(age)) # conversion d'une valeur numérique en
chaîne de caractère

Les Listes

- A quoi cela sert?

- Manipuler un ensemble de données structurées identique de manière simple
- Améliorer les performances d'accès à ces données, pas d'accès disque (ms) mais en mémoire (ns)

- L'accès à des données s'effectue toujours au travers de 4 opérations (CRUD)

- Create : Créer, ajouter des éléments à une liste
- Read : Lire les données présentes dans une liste (3 manières)
 - Séquentielle
 - Indexés (usage d'une clé)
 - Non indexés (recherche dans les données de la liste)
- Update : modifier les données d'un des éléments de la liste
- Delete : suppression d'un élément de liste

C'est quoi une liste?

- un ensemble d'éléments à la suite
 ['eleve1', 'eleve2', 'eleve3']
- Chaque élément possède un indice avec sa position dans la liste
 Attention les liste commencent par une position à zéro
- Pour connaître la longueur d'une liste on utilise la fonction
 len(laListe)
- Les données d'une liste s'effacent à l'arrêt du programme

Création et ajout dans d'un liste

LstEleves

Index	Valeur
0	Eleve1
1	Eleve2
2	Eleve2bis
3	Eleve3
4	Eleve4
5	Eleve5
6	Eleve6

```
LstEleves = ['eleve1', 'eleve2', 'eleve3']
```

```
print (LstEleves)
```

```
['eleve1', 'eleve2', 'eleve3']
```

```
LstEleves.append('eleve4')      # ajouter à la fin d'une liste
```

```
LstEleves.extend( ['eleve5', 'eleve6'] )  # ajoute une liste en fin de liste
```

```
print (LstEleves)
```

```
['eleve1', 'eleve2', 'eleve3', 'eleve4', 'eleve5', 'eleve6']
```

```
# ajouter au milieu d'une liste
```

```
LstEleves.insert(2, 'eleve2bis')
```

```
print (LstEleves)
```

```
['eleve1', 'eleve2', 'eleve2bis', 'eleve3', 'eleve4', 'eleve5', 'eleve6']
```

Lecture dans une liste

```
print (lstEleves[0])      # c'est bien une chaine de caractères
e1 = lstEleves[0]
print (lstEleves[0:0])
e1 = lstEleves[0]         # ici on renvoie un tableau contenant la chaine
print (lstEleves[0:2])
e1 = lstEleves[0:2]
```

- Pour obtenir le dernier de la liste
print (lstEleves[len(lstEleves) - 1])
- Pour inverser l'ordre d'une liste
print (lstEleves.reverse())
- Tester l'existence d'un élément dans une liste (plus bas)
if 'e1' in lstEleves:

Modifier un élément d'une liste

```
IstEleves = ['eleve1', 'eleve2', 'eleve3']
```

Remplacer un élément de la liste

```
IstEleves[0] = 'mounia'
```

Ajouter un élément dans une liste à une position particulière

```
IstEleves.insert(2, 'sonia')
```

```
Print (IstEleves)
```

```
['mounia', 'eleve2', 'sonia', 'eleve3']
```

Trier une liste

```
liste_triee = sorted(IstEleves)    # [ 'eleve2', 'eleve3', 'mounia', 'sonia']
```

```
IstEleves.sort()
```


Supprimer un élément

si on connaît sa position

```
del lstdEleves[3]
```

```
lstdEleves.pop(3) # moins simple à retenir
```

si on connaît sa valeur

```
lstdEleves.remove('e1eve3')
```

purger toute la liste

```
lstdEleves.clear()
```

Les tuples (on en parle et on oublie)

- Les tuples sont comme les listes mais ils ne sont pas modifiables
 - `monTuple = (1, 2, 'coucou', 4, 5, [3,4],7,8)`
- Il est juste possible de lire et d'ajouter (par concaténation) des tuples
- Il est possible de modifier une liste dans un tuple
 - `monTuple[5][0]=5` `(1, 2, 'coucou', 4, 5, [5,4], 7, 8)`
- Il est possible de convertir un tuple en liste
 - `maListe = list(montuple)`
- En modifier le contenu
 - `maListe[2] = « bonjour »`
- Et de le convertir à nouveau en liste...
 - `montuple = tuple(maliste)`

Les chaines de caractères

- Une chaine de caractères est une liste de caractères
- On la manipule donc une comme liste (voir plus bas)

découpage d'une chaine de caractère en liste

```
maChaine = 'une liste de mots'
```

```
print (maChaine[4])
```

```
|
```

```
print(maChaine.split(' '))
```

```
['une', 'liste', 'de', 'mots']
```

concaténation d'une liste en chaine de caractère

```
maListe = ['une', 'autre', 'liste', 'de', 'mots']
```

```
print(maListe )
```

```
['une', 'autre', 'liste', 'de', 'mots']
```

```
print(maListe[3])
```

```
de
```

```
print(maListe[2][3])
```

```
t
```

```
print('.'.join(maListe))
```

```
'une-autre-liste-de-mots'
```

Les chaines de caractères

- ' bonjour je m\'appelle'
- " bonjour je m'appelle"
- 'ceci est une "expression" '
- " ceci est une \"expression \« "

Manipuler une liste/chaine de caractères

- Utilisation des crochets [start:stop:step] pour filtrer une chaine de caractère

- Se déplacer

- Utilisation des crochets et des : pour définir la zone que l'on souhaite récupérer
- `maChaine = 'azertyuiop'`
- `print (maChaine[:3])` `aze`
- `print (maChaine[3:])` `rtyuiop`
- `print (maChaine[:])` `azertyuiop`
- `print (maChaine[:2])` `zryip`
- `print (maChaine[:-1])` `azertyuio`
- `print (maChaine[-2:])` `op`
- `print (maChaine[-3:-2])` `i`
- `print (maChaine[::-1])` `poiuytreza`

- Ajouter des valeurs d'un tableau dans une chaine de caractères

- L'utilisation des accolades {} pour représenter dans le print là où l'on souhaite ajouter la valeur
- ```
monTableau = ['Charlène', 55, 'blonde']
print (« bonjour, je m'appelle {}, j'ai {} ans, et je suis {} ».format(monTableau))
bonjour, je m'appelle Charlène, j'ai 55 ans, et je suis blonde
```

# Exercice manipulation de chaine

```
'e'.join('nvr')
'annog'[::-1]
'forgive'[-4:]
('me', 'you')[1]
'soup'.split('o')[1]
```

# Les variables de type Date, heure et durée

- Utilisation de modules à importer **datetime**, **time** et **timedelta**
- **Attention** import datetime **est différent de** from datetime import datetime
- Construction d'une date simple

```
version import datetime # version from datetime import date
maDate = datetime.date(1968, 5, 24) maDate = date(1968, 5, 24)
dateDujour = datetime.date.today() dateDujour = date.today()
print(datetime.date.fromisoformat("2021-10-22")) print(date.fromisoformat("2021-10-22"))
```

- Affichage des dates

```
print (maDate.year, maDate.month, maDate.day) 1968 05 24
print (maDate.replace(month=07)) 1968-07-24 # on affiche la date modifiée
print(maDate.isoformat()) 1968-05-24 # mais on ne modifie pas la date initiale
maDate.strftime('%Y-%m-%d %H:%M:%S')
```

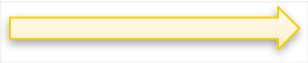
# Autres fonctionnalités sur les dates

```
maDate = datetime.date(1968, 5, 24)
dateDuJour = datetime.date.today()
dans_15_jours_moins_2_heures = dateDuJour + timedelta(days=15,
hours=-5)
age_en_jours = dateDuJour - maDate
age_en_annee = dateDuJour.year - maDate.year

maDate.weekday() # détermine le jour de l'année (0:lundi,
1:mardi, ...)
```



# Bonne pratique de codage (PEP8)

- PEP 8 (Python Extension Proposal)
  - Règles qui permettent d'homogénéiser le code et appliquer de bonnes pratiques
  - Dans le cas d'une équipe de développement, se renseigner sur les règles « tacites » de codage
- Encodage : Utiliser l'UTF-8
- Indentation : Utiliser 4 espaces pour les indentations au lieu des tabulations
- Code Layout : Nombre de caractères par ligne (79)
- Import : Les imports doivent être au début du script
- Espaces : préciser de la concision 
- Commenter en Anglais
- DRY : Don't Repeat Yourself
- KISS : Keep It Simple Stupid

```
name = 'Batman' # oui c'est
propre
name='Bruce' #non c'est moche
batmobile ['color'] = 'black' #
non
batmobile['color'] = 'black' # oui
```

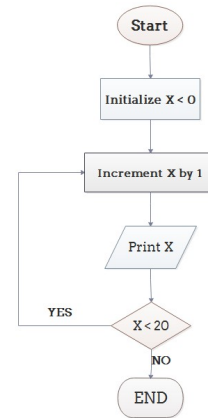
# Structures de contrôle (algorithm)

- Permet de construire un programme et résoudre des problèmes
  - Les **conditions** : permettent de diriger l'exécution du programme selon certaines conditions
  - Les **boucles** : permettent de répéter un ensemble d'opération tant qu'une condition est valide
- Condition, opération de comparaison

| Opérateur | Définition                                                            |
|-----------|-----------------------------------------------------------------------|
| ==        | Permet de tester l'égalité en valeur et en type                       |
| !=        | Permet de tester la différence en valeur ou en type                   |
| <         | Permet de tester si une valeur est strictement inférieure à une autre |
| >         | Permet de tester si une valeur est strictement supérieure à une autre |
| <=        | Permet de tester si une valeur est inférieure ou égale à une autre    |
| >=        | Permet de tester si une valeur est supérieure ou égale à une autre    |

A = 1   B = 2   C = 3

A == B        faux  
A != B        vrai  
A < B        vrai  
A > B        faux  
A + B <= C    vrai  
A + B >= C    vrai



[https://fr.wikipedia.org/wiki/Ada\\_Lovelace](https://fr.wikipedia.org/wiki/Ada_Lovelace)

Ada Lovelace 1815 – 1852  
Créatrice du premier algorithme  
(calcul des nombres de Bernoulli)

# Introduction aux fonctions

- Organiser son code : le réécrire proprement
- Eviter de refaire deux fois la même chose
- Meilleure maintenance : on ne modifie qu'une fois le code
- Une fonction se définit par :
  - Le mot def
  - Des parenthèses pouvant contenir des variables
  - Deux points : à la fin de la ligne
  - Le code de la fonction qui est indenté
- Une fonction est toujours définie avant de l'utiliser
- Les variables utilisées dans une fonction sont dites « locales »
  - Elles ne sont pas utilisées ailleurs dans le programme
  - Il est possible de rendre une variable globale aux programmes cf plus loin

# Paramètres d'une fonction

- Il est possible de transmettre des valeurs à une fonction

**def** ma\_fonction(Variable1, Variable2) :

- Les paramètres d'une fonction peuvent être définis avec une valeur par défaut.

**def** ma\_fonction(Variable1, Variable3 = 0) :

- Il est possible de récupérer des variables externes à la fonction avec la commande **global**

**def** ma\_fonction(Variable1, Variable3 = 0) :

**global** Variable2

**return** Variable1 + Variable 2 + Variable3

Variable2 = 5

Print ma\_fonction(10)   #affiche 15

Print ma\_fonction(10, 20)       # affiche 35

# Le return d'une fonction

- Une fonction retourne toujours une valeur
  - avec la clause **return**, on récupère une ou plusieurs valeurs (affectation multiple)
  - Sans la clause return, la valeur **None** est retournée

Exemple d'affectation multiple

```
def ordonner (a, b):
```

```
 if a < b:
```

```
 return(a, b)
```

```
 else:
```

```
 return(b, a)
```

```
x, y = ordonner(3, 14)
```

```
print (x, '-', y) 3 - 14
```

```
x, y = ordonner(14, 3)
```

```
print (x, '-', y) 3 - 14
```

# Les fonctions lambda

- Petite fonction ne contenant qu'une expression
  - lambda variable : expression
- Utilisé par les fonctions prenant en compte des fonctions (sorted, map, filter)
- Attention ces fonctions retournent un objet qu'il faut convertir ensuite

```
maListe = ["1","2","5","4","3","6","7","8"]
maListeEnEntier = list(map(lambda x : int(x), maListe)
Print (maListeEnEntier) [1, 2, 5, 4, 3, 6, 7, 8]
maListeFiltree = list(filter(lambda x : x > 3, maListeEnEntier)
print (maListeFiltree) [5, 4, 6, 8]
```

**Cette manière d'écrire n'est pas conseillée par le PEP8**

# Fonctions natives simples et prédéfinies

- `print` (« hello world ») # affiche « hello world » sur la console
- `range` (10) # retourne un tableau de 0 à 9 (dix valeurs)
- `range` (1,13) # retourne un tableau de 1 à 12
- `range` (2, 24, 2) # retourne le tableau 2,4,6, ... 20,22
- `val1 = input` ('Valeur 1 :') # affecte une valeur saisie (chaîne de caractères) sur l'écran à val1
- `int`(X), `str`(Y) # réalise la conversion de type
- `longueur = len`('chaîne de texte') # retourne la longueur d'un objet
- `type`(variable) # retourne le type (int, string, ...) d'une variable
- `random.random`() # retourne un nombre aléatoire entre 0 et 1
- `random.randrange`(X) # retourne un nombre aléatoire entre 0 et X
- `choice`(['win', 'lose', 'draw']) # retourne une valeur aléatoire d'un tableau
- `X.isdigit`() # retourne vrai ou faux en fonction de X numérique

# La boucle **for**

```
print (« début du traitement »)
for i in range(10) :
 print (i)
print (« fin du traitement »)
```

Début du traitement

0

1

2

...

8

9

Fin du traitement

Affecte à la variable `i` une valeur de 0 à 9

Écrit sur la console la valeur de `i`

:  indique la fin de la ligne et le début de l'indentation

**indentation**  précise le code contenu dans la boucle



# La boucle **while**

```
i = 0
```

```
while i < 10 :
```

```
 print (i)
```

```
 i = i + 1 # variante i += 1
```

```
while True :
```

```
 print (« boucle infini »)
```

```
 break # sort de la boucle
```

# Boucler sur une liste

```
IstEleves = ['eleve1', 'eleve2', 'eleve3']
```

```
for eleve in IstEleves:
```

```
 print (eleve)
```

```
eleve1
```

```
eleve2
```

```
eleve3
```

# Contrôle conditionnel avec **if**

```
x = 5 # affectation de la valeur 5 à la variable x
if x == 5: # si x est égale à 5
 print ('x a pour valeur 5')
elif x > 5: # si x est strictement supérieur à 5
 print ('x est supérieur à 5 : ' + str(x))
else : # dans tous les autres cas
 print ('x est inférieur à 5')
```

**ATTENTION** **=** (affectation) est différent de **==**  
(contrôle)

**If x=5:** renvoie toujours vrai

# Tests conditionnels dans un tableau

```
lstEleves = ['eleve1', 'eleve2', 'eleve3']
```

```
if 'eleve1' in lstEleves:
```

```
 print ('Présent')
```

```
test_list = [10, 15, 20, 7, 15, 2808]
```

```
exist_count = test_list.count(15)
```

```
if exist_count > 0:
```

```
 print ('Présent {} fois'.format(exist_count))
```

présent 2 fois

# Try - except - raise - finally : Gestion des erreurs

```
X = input ('Valeur : ')
Y = input ('Diviseur : ')
try:
 # on test la possibilité de la division
 # on vérifie si la saisie n'est pas numérique
 if X.isdigit() == False:
 raise ValueError("X n'est pas un nombre") # on lève une erreur avec un texte personnalisé
 if not Y.isdigit() :
 raise ValueError# on lève une erreur générique
 result = float(X) / float(Y)

except ZeroDivisionError:
 # si une erreur de type division par zero apparait
 print("division par zero!")

except ValueError as err:
 # si on a une erreur sur la valeur
 if len(err.args) == 0:
 print('la Valeur saisie n\'est pas numérique') # on affiche l'erreur avec un texte générique
 else:
 print(err.args[0]) # on affiche l'erreur avec le texte transmis

else:
 # si il n'y a pas d'erreur
 print("résultat : ", result)

finally:
 # fin du traitement commun (avec ou sans erreur)
 print("fin du traitement")
```

# Les dictionnaires

Liste dont l'index n'est pas une suite numérique ordonnée et sur plusieurs colonnes

| Listes                                                                                                                                                       | Dictionnaire |         |   |        |   |        |   |        |                                                                                                                                                                    |       |         |     |        |     |        |     |        |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|---------|---|--------|---|--------|---|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|---------|-----|--------|-----|--------|-----|--------|
| <table><tr><th>Index</th><th>Valeurs</th></tr><tr><td>0</td><td>Eleve1</td></tr><tr><td>1</td><td>Eleve2</td></tr><tr><td>2</td><td>Eleve3</td></tr></table> | Index        | Valeurs | 0 | Eleve1 | 1 | Eleve2 | 2 | Eleve3 | <table><tr><th>Index</th><th>Valeurs</th></tr><tr><td>EL1</td><td>Eleve1</td></tr><tr><td>EL3</td><td>Eleve2</td></tr><tr><td>EL6</td><td>Eleve3</td></tr></table> | Index | Valeurs | EL1 | Eleve1 | EL3 | Eleve2 | EL6 | Eleve3 |
| Index                                                                                                                                                        | Valeurs      |         |   |        |   |        |   |        |                                                                                                                                                                    |       |         |     |        |     |        |     |        |
| 0                                                                                                                                                            | Eleve1       |         |   |        |   |        |   |        |                                                                                                                                                                    |       |         |     |        |     |        |     |        |
| 1                                                                                                                                                            | Eleve2       |         |   |        |   |        |   |        |                                                                                                                                                                    |       |         |     |        |     |        |     |        |
| 2                                                                                                                                                            | Eleve3       |         |   |        |   |        |   |        |                                                                                                                                                                    |       |         |     |        |     |        |     |        |
| Index                                                                                                                                                        | Valeurs      |         |   |        |   |        |   |        |                                                                                                                                                                    |       |         |     |        |     |        |     |        |
| EL1                                                                                                                                                          | Eleve1       |         |   |        |   |        |   |        |                                                                                                                                                                    |       |         |     |        |     |        |     |        |
| EL3                                                                                                                                                          | Eleve2       |         |   |        |   |        |   |        |                                                                                                                                                                    |       |         |     |        |     |        |     |        |
| EL6                                                                                                                                                          | Eleve3       |         |   |        |   |        |   |        |                                                                                                                                                                    |       |         |     |        |     |        |     |        |

# Manipulation d'un dictionnaire

- Affectation (C**R**UD)

```
dicEleves = { 'eleve1' : 11, 'eleve2' : 17, 'eleve3' : 14}
```

- Lecture (C**R**UD)

```
print (dicEleves['eleve1']) # 11
for eleve in dicEleves :
 print (eleve) # eleve1 eleve2
 eleve3
for eleve in dicEleves.values() :
 print (eleve) # 11 17
 14
```

# Un dictionnaire dans un dictionnaire

```
dicEleves = {
 'eleve1' : {'note':11, 'appréciation': 'Moyenne' },
 'eleve2' : {'note':17, 'appréciation': 'Très bien' },
 'eleve3' : {'note':14, 'appréciation': 'Bonne' }
}

print (dicEleves['eleve1']['note']) # 11
for eleve in dicEleves:
 print (eleve, dicEleves[eleve]['note'], dicEleves[eleve]['appréciation'])
```

|        |    |           |
|--------|----|-----------|
| eleve1 | 11 | Moyenne   |
| eleve2 | 17 | Très bien |
| eleve3 | 14 | Bonne     |

La structuration d'un dictionnaire est au format JSON



# Modifier un dictionnaire

- Ajouter un nouvel élément

```
dicEleves['eleve4'] = { 'note' :10, 'appréciation' : 'passable' }
```

- Modifier une note d'un élève

```
dicEleves['eleve3']['note'] = 16
```

- Supprimer un élève

```
del dicEleves['eleve4']
```

- Vérifier la présence d'un élève

```
if 'eleve3' in dicEleves.keys():
```

```
print ('eleve3 présente')
```

- Vérifier la présence d'une valeur

```
if 17 in dicEleves.values():
```

```
print ('un élève a 17')
```

# Trier un dictionnaire

- Trier selon les index du dictionnaire

```
TblATrier_trie = dict(sorted(TblATrier.items()))
```

- Trier selon les valeurs du dictionnaire

```
TblATrier_trie = dict(sorted(TblATrier.items(), key=lambda item: item[1],
reverse=True))
```

# Programmation Orienté Objet - Présentation

- Concepts clés de la POO
- Avantages de la POO par rapport à la programmation procédurale
- Exemples concrets pour illustrer les concepts de la POO

# Concepts clés de la POO

- **Classes :**

- Description des caractéristiques (attributs) et des comportements (méthodes) de l'objet
- Utilise le mot clef « class »
- Utilise une fonction init() pour « construire » un objet depuis une (ou plusieurs) classe

- **Objets :** Instances d'une classe avec des caractéristiques spécifiques

- un nom de variable
- des valeurs spécifiques (attributs)
- des comportements définis (méthodes)
- Il est possible d'avoir plusieurs objets issues d'une même classe

- **Encapsulation :** Regroupement d'attribut et de méthode dans une même entité

- **Héritage :** Classe héritant des attributs et des méthodes d'une classe existante

- **Polymorphisme :** Capacité de traiter les objets de différentes classes de manière interchangeable

# Avantages de la POO

- Approche moins algorithmique de la programmation
- Représentation formelle des choses sous formes d'objets (abstraction)
- Simplification de la maintenance :
  - Réutilisation du code grâce à l'héritage et à la création de classes spécialisées
  - Modularité : Les classes permettent de diviser le programme en éléments autonomes plus simple à à développer et maintenir
  - Abstraction : La POO permet d'abstraire les détails internes des objets pour simplifier le développement, tout est objet (voiture, maison, requête SQL, ...)
  - Testabilité : il est plus simple de tester un objet qu'un programme

# Programmation Orienté Objet - Les classes

- Une classe se définit par son nom, des attributs et des méthodes

| Classe    | Attributs<br>Variable spécifique | Méthodes<br>Fonction spécifique |
|-----------|----------------------------------|---------------------------------|
| Voiture   | Couleur<br>Vitesse               | rouler()<br>freiner()           |
| Veste     | Couleur<br>Taille                | setPrix()<br>getPrix()          |
| Téléphone | Marque<br>Taille                 | téléphoner()<br>répondre()      |

# Cas d'utilisation courants

- Modélisation d'objets réels : voitures, employés, produits, etc.
- Développement d'applications avec une architecture modulaire et évolutive
- Utilisation de bibliothèques et de frameworks basés sur la POO

# Syntaxe et exemples en Python

- Déclaration d'une classe avec ses attributs et méthodes
- Création d'objets à partir d'une classe
- Utilisation de l'encapsulation pour protéger les données et contrôler leur accès
- Exemples d'héritage pour illustrer la spécialisation des classes



# Définition d'une classe

```
class Veste : # première lettre en majuscule par convention pour les classes
 """ ceci est un commentaire de la classe """
 maCouleur = 'blanche'
 maTaille = ''
 monPrix = 50
 marque = ''
 def __init__(self, taille, prix, couleur = None, marque='jules') : # constructeur de la classe
 """ definition du constructeur de la classe """
 self.maTaille = taille
 self.maCouleur = couleur
 self.monPrix = prix
 self.saMarque = marque
 def __str__(self):
 return 'veste couleur :'+ self.maCouleur+ ' Prix : ' + str(self.monPrix)
 def set_couleur(self,couleur) : # setter
 self.maCouleur = couleur
 def get_prix(self, quantite=1) : # getter
 return self.monPrix * quantite
```

# Définition d'un objet en utilisant une classe

```
vesteCharlene = Veste(42, 'rouge', 60)
```

```
print (vesteCharlene.taille) => 42
```

```
print (vesteCharlene.marque) => « jules »
```

```
vesteMarie = Veste(36)
```

```
vestePierre = Veste(38, none, 23)
```

```
print(vestePierre.get_prix(3)) # renvoie la valeur 3*23 = 69
```

# Héritage

- Principe : mutualiser des attributs et des méthodes communes à plusieurs objets

Classe Veste:

marque  
couleur  
prix  
taille  
doublure

Classe Chaussure:

marque  
couleur  
prix  
pointure  
talon

Classe Pantalon:

marque  
couleur  
prix  
taille  
braguette

Classe Vetement:

marque  
couleur  
prix

- Il convient de définir un « lien de parenté »

```
class Veste(Vetement): # on définit la classe veste basée sur la classe vetement
```

```
def __init__(marque, couleur, prix, taille, doublure):
```

```
 super().__init__(marque, couleur, prix) # on affecte les données au parent
```

- Il est possible d'hériter de plusieurs parents :

```
class classFille(classParent1, classParent2)
```

# Définir une librairie à partir d'une classe

- Création d'un fichier veste.py
- Copier la définition de la classe dans ce nouveau fichier
- Importer la classe avec la commande

```
from veste import Veste
import veste
```