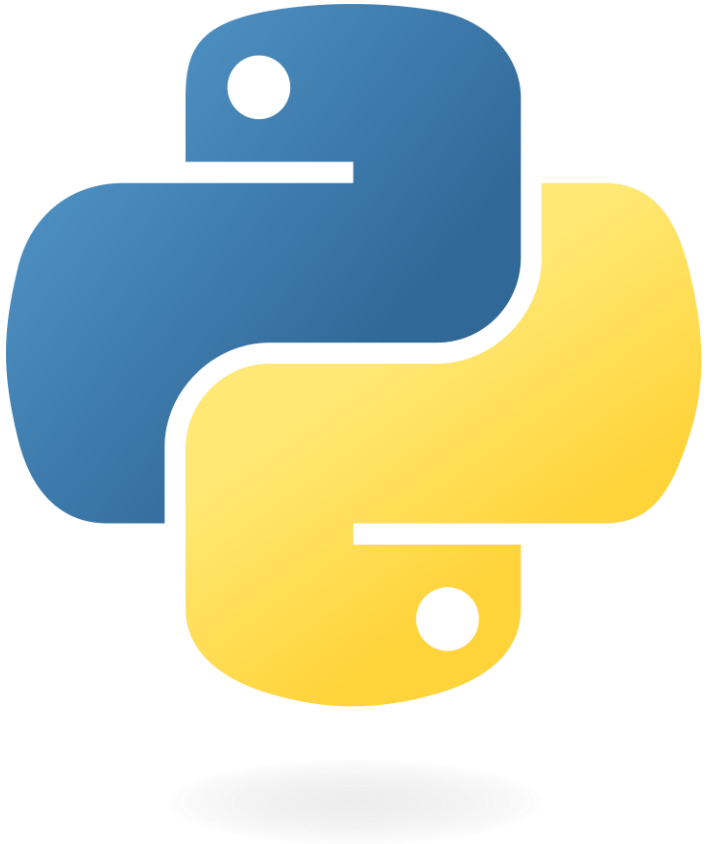
ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR

Membre de

HONORIS UNITED UNIVERSITIES

Programmation en python

2024/2025



INTRODUCTION

2024/2025

Qu'est-ce que Python ?

Python est un langage de programmation de haut niveau qui est :

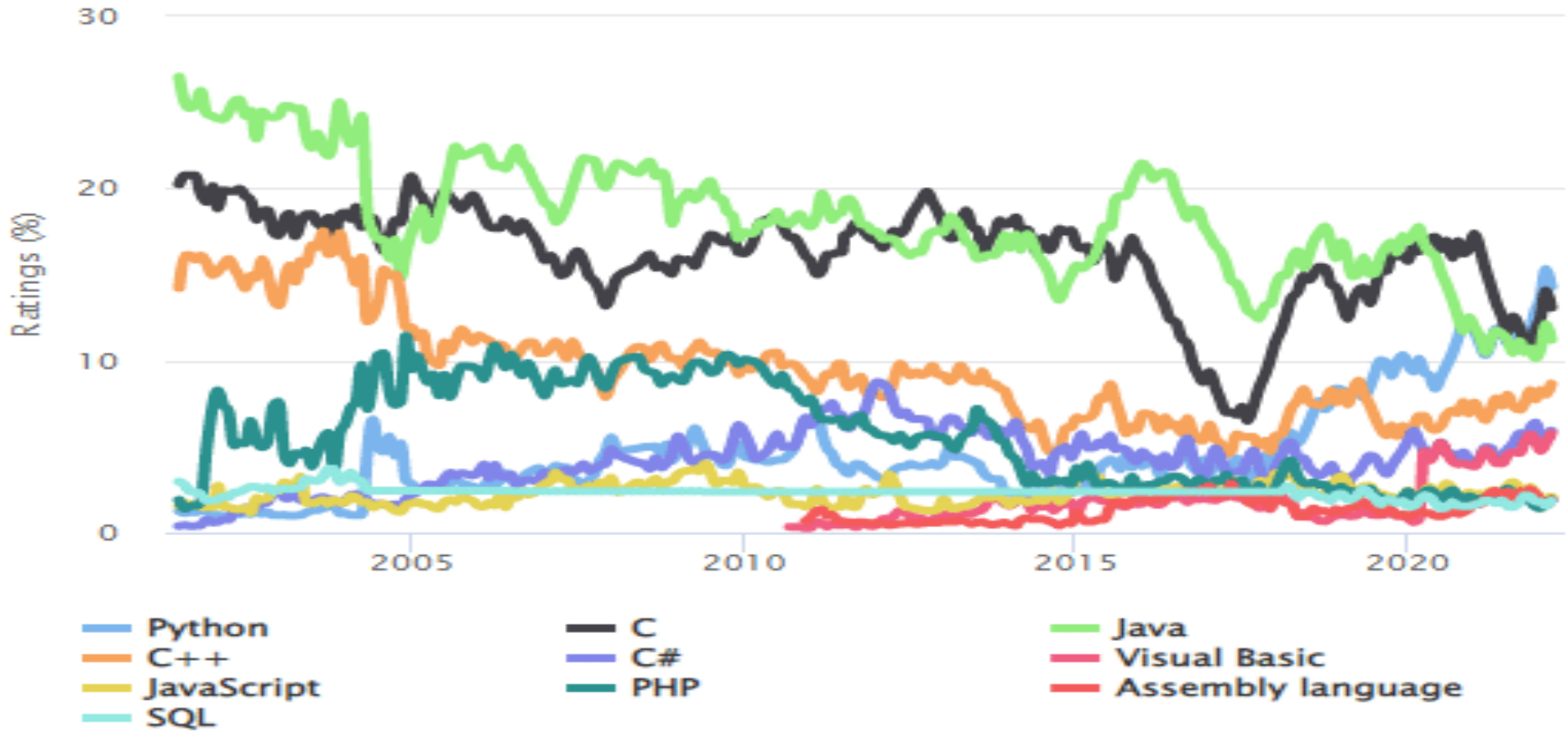
- Interprété : Python est traité à l'exécution par l'interpréteur ligne par ligne.
- Interactif : Vous pouvez utiliser une invite Python et interagir directement avec l'interpréteur pour écrire vos programmes.
- Orienté objet : Python supporte le paradigme orientée objet.

Histoire de Python

- Python a été créé en 1989 par Guido Van Rossum
- La première version publique de ce langage a été publiée en 1991 au CWI aux Pays-Bas.
- Python est dérivé du langage de programmation ABC, qui a été développé au CWI
- G. Van Rossum a choisit le nom « Python », vu qu'il était un grand fan de la série Monty Python
- Puis Python 3.0 est sorti le 3 décembre 2008.

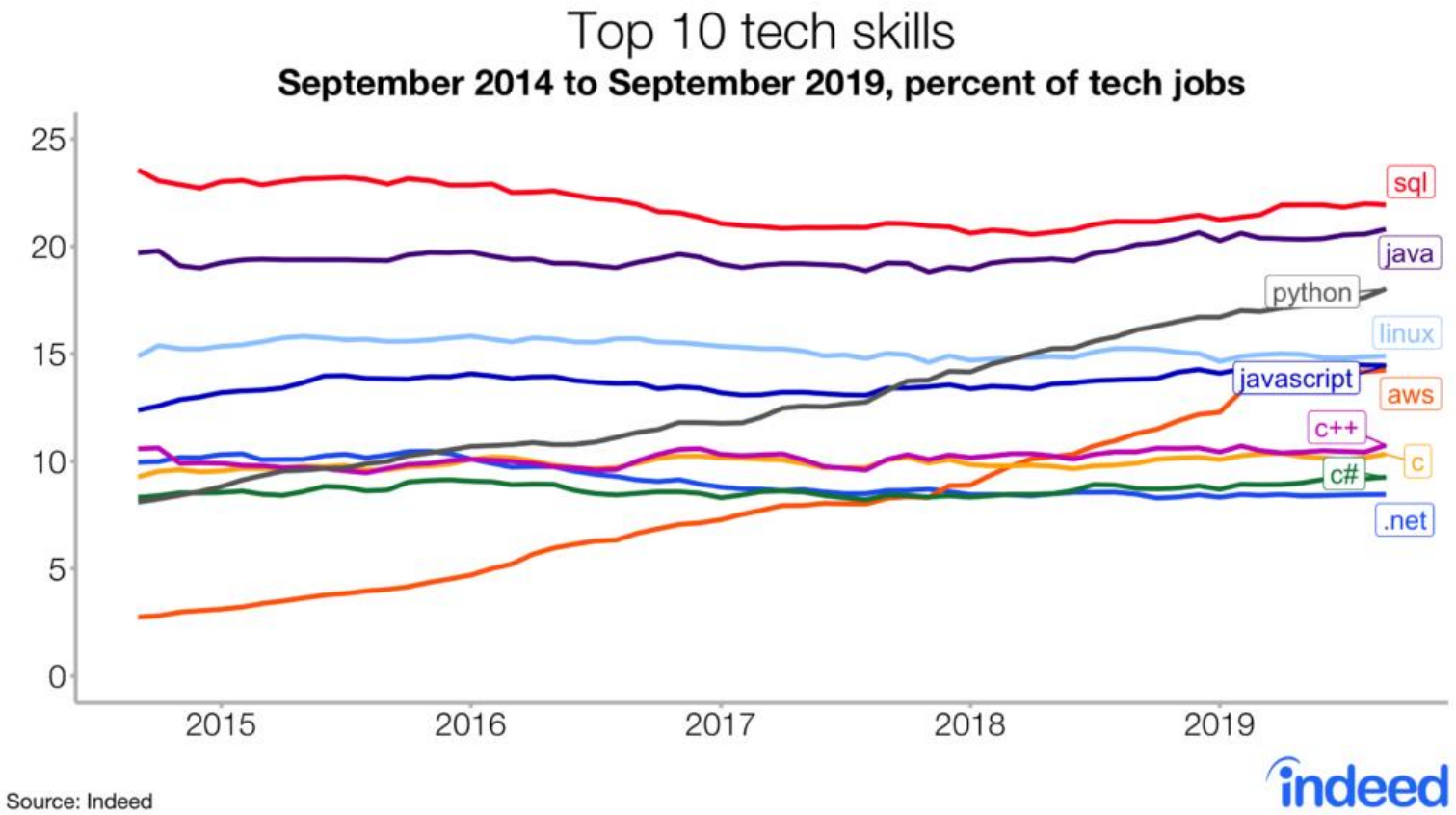
Pourquoi Python

Python est récompensé comme le meilleur langage de programmation de l'année 2020 (Source TIOBE Programming Community)



Pourquoi Python

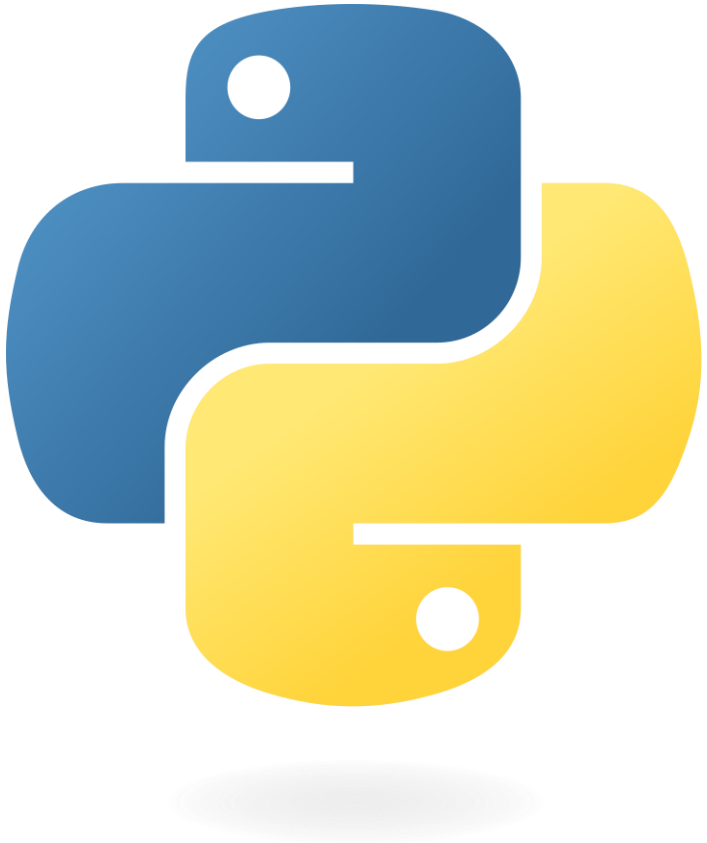
Selon le site indeed.com, la montée en croissance de Python est beaucoup plus élevé que celle des autres langues.



les caractéristiques de python

- Facile à apprendre
- Multiplateforme
- Gratuit et Open Source
- Orienté Objet
- Typiquement dynamique: nous n'avons pas besoin de spécifier le type de données lors de la déclaration.
- Large choix de bibliothèque

ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

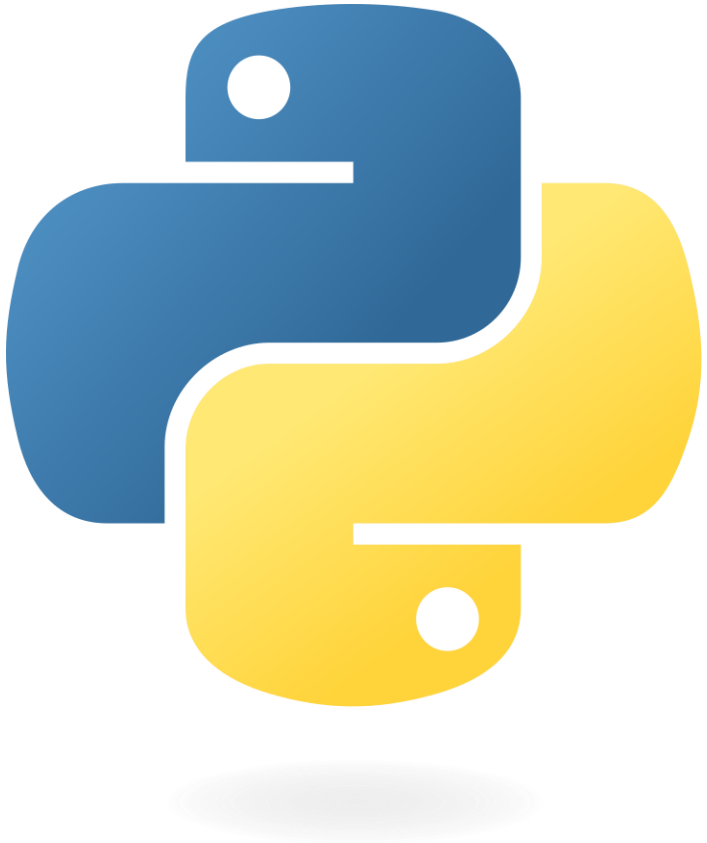


L'ENVIRONNEMENT DE TRAVAIL

ATELIER 1

2024/2025

ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES



SYNTAXE DE BASE

2024/2025

Script Python

- Les fichiers Python ont l'extension .py
- Contiennent une liste d'instructions
- Le point-virgule (;) est facultatif à la fin de l'instruction.
- Lecture des entrées clavier:

```
name = input("What is your name? ")
```

- Génère la sortie du script:

```
print("Hello " + name)
```

Commentaires


```
# C'est un commentaire
'''
Commenter plusieurs lignes
ligne 1
ligne 2
...
ligne n
'''
```

Indentation

• L'indentation est utilisée en Python pour délimiter les blocs. Le nombre d'espaces est variable, mais toutes les instructions d'un même bloc doivent être indentés de la même façon.

Variables

- Python est doté d'un typage dynamique
- La déclaration se fait automatiquement lorsque vous attribuez une valeur à une variable.

```
# integer
x= 2
# float
y= 2.0
# string
name = 'younes'
```

- Les variables peuvent changer de type, simplement en leur attribuant une nouvelle valeur d'un type différent.

```
# null
z = None
# integer
z=1
```

- Python vous permet d'assigner une seule valeur à plusieurs variables simultanément.

```
a=b=c=d=e=f=g=h=i=1
```

- Pour supprimer une variable python

```
del (x)
```

Principaux type de données

- Entier (int)
- Flottant (float)
- Texte (chaîne de caractères) (str)
- Booléen (bool):True, False
- Rien, indéfini (NoneType)

Opérateurs arithmétiques

- Addition +
- Soustraction
- Multiplication *
- Division flottante /
- Division entière //
- Reste de la division entière (modulo) %
- Puissance **

Opérateurs logiques

- et (and): x < 18 and x < 24
- ou(or): x < 18 or x < 24
- non (not): not(x < 18 or x < 24)
- **Valeurs de retour**
  - Vrai : True
  - Faux : False

Opérateurs de comparaison

Opérateur	Exemple
==	x == y
!=	x != y
>	x > y
<	x < y
>=	x >= y
<=	x <= y

Chaînes de caractères

- Première syntaxe sans retour à la ligne
- Deuxième syntaxe avec retour à la ligne
- Les indices de chaîne débutent à 0.
- Les indices commencent à -1 à la fin
- Concaténation entre str
  - Remarque: on ne peut pas concaténer un String et un Nombre
- Répétition

```
1  str1="j'aime python"
2  str2='j\'aime python'
```

```
3  str3="""
4      J'aime python
5  c'est un langage simple
6  """
```

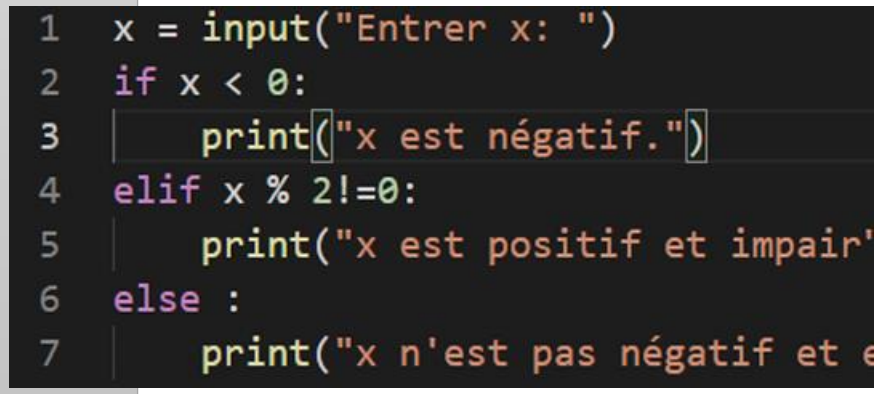
```
 9  str="python"
10  print(str[0])
11  print(str[-1])
```

```
1  str1="j'aime python"
2  str2='j\'aime python'
3  str3="""
4      J'aime python
5  c'est un langage simple
6  """
7  print(str1 +"\n"+ str2 +str3)
```

```
1  str1="j'aime python "
2  print(str1*3)
```

Instruction if

```
if condition1 :
        Bloc exécuté si condition1 est vrai.
elif condition2 :
        Bloc exécuté si condition1 est faux et
        condition2 est vrai.
else :
        Bloc exécuté si aucune condition n'est vraie.
```



```
1  x = input("Entrer x: ")
2  if x < 0:
3      print("x est négatif.")
4  elif x % 2!=0:
5      print("x est positif et impair"
6  else :
7      print("x n'est pas négatif et e
```

- Les condition sont des expressions logiques booléennes évaluées à True ou False.

List

- Une liste est une collection ordonnée et modifiable d'éléments éventuellement hétérogènes (pas nécessairement du même type).
- Les éléments de la liste sont indexés, le premier élément a un index [0].
- Il est possible de spécifier une gamme d'indices en précisant où commencer et où terminer la gamme. La valeur de retour sera une nouvelle liste avec les éléments spécifiés.
- Pour déterminer si un élément est présent dans une liste. Il faut utiliser (in)

```
1  list=['ali','casablanca',2000]
2  print(list)
3  print(list[1])
4  print(list[-1])
5  print(list[0:2])
6  print(list[:2])
7  print(list[1:])
8  prenom=input("entrer le nom:")
9  if prenom in list:
10     print("Le prénom fait partie de la liste")
11 else:
12     print("Le prénom ne fait pas partie de la liste")
```

List

- Il est possible de modifier la valeur d'un élément spécifique ou les éléments dans une plage.
- Lorsque vous insérez moins d'éléments que vous remplacez. les nouveaux éléments seront insérés à l'endroit que vous avez indiqué, et les autres éléments seront supprimés

```
14  #changer les elements de liste
15  langages=['Java','Python','C#']
16  langages[2]='J2EE'
17  print(langages)
18  langages[0:2]=['C','C++']
19  print(langages)
20  langages[1:2]=['javascript','R']
21  print(langages)
22  langages[1:]=['Python']
23  print(langages)
24  langages[1:1]=['JS']#insertion en 2 ème position
25  print(langages)
```

List

Méthodes: list= ['Java','Python','C#']

Méthode	Description
list.insert(2,"sql")	Ajoute un élément à l'index spécifié
list.append("J2ee")	Ajoute un élement à la fin de la liste
list.extend(list2)	Ajoute des éléments d'une liste (list2) à la liste actuelle (list)
list.remove("laravel")	Supprime l'élément spécifié de la liste actuelle
list.pop(1)	Supprime l'élément à l'index spécifié.
list.clear()	Vide la liste.
list.sort()	Trie la liste de manière alphanumérique, en ordre croissant, par défaut. En ordre décroissant list.sort(reverse=True)
List.reverse()	Inverse l'ordre actuel des éléments.
list2=list.copy()	Créée une copie de la liste (list) dans la liste (list2)

- Fonctions: len(moy)retourne la longueur du dictionnaire(moy)

Dictionnaires

- collection d'objets s'appuyant sur le mécanisme associatif clé:valeur .
- Les dictionnaires Python sont des objets modifiables qui peuvent changer leurs valeurs
- Pour déterminer si une clé est présente dans un dictionnaire. Il faut utiliser (in)s

```
7  moy = {'python': 17.5, 'anglais': 15.8}
8  print(moy)
9  #acces
10 print(moy['anglais'])
11 print(moy.get('anglais'))
12 #return liste des clés
13 print(moy.keys())
14 #return liste des valeurs
15 print(moy.values())
16 #return nombre d'élement
17 print(len(moy))
18 #detecter présence d'une clé
19 if 'français' in moy:
20     print("c'est une clé")
21 else:
22     print("cette clé n'existe pas")
23 #ajouter un élement
24 moy['français']=14
25 #modification
26 moy['anglais']=13.5
```

Dictionnaires

- moy={'python': 17.5, 'anglais': 15.8}

Méthode	Description
moy.update({'anglais':16})	Elle met à jour le dictionnaire avec les éléments spécifié comme argument . Si l'élément n'existe pas, il sera ajouté.
moy.pop('anglais')	Supprime l'élément à la clé spécifié.
moy.popitem()	Supprime le dernier élément inséré
moy.clear()	Vide le dictionnaire.
moy2=moy.copy()	Créée une copie du dictionnaire (moy) dans le dictionnaire (moy2)

- Fonctions: len(moy)retourne la longueur du dictionnaire(moy)

Boucle: for et while

- For

```
1  list=['Java','Python','C#']
2  for item in list:
3      print(item)
```

```
5  for i in range(len(list)):
6      print(list[i])
```

- While

```
1  list=['Java','Python','C#']
2  i=0
3  while i<len(list):
4      print(list[i])
5      i+=1
```

Fonctions - Procédures

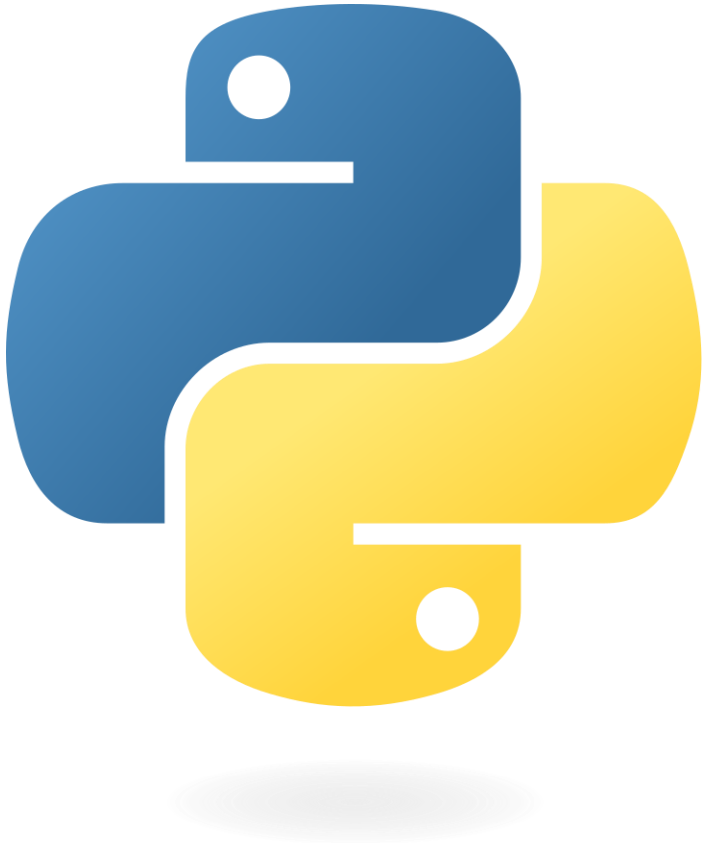
Bloc d'instructions =>attention à l'indentation

- Paramètres non typés
- Renvoie une valeur en sortie (ou plus). Ou pas

de valeurs: Procédure

```
#définition de la fonction
def parite(number):
    '''cette fonction renvoie True si le nombre et pair,
    Sinon False  '''
    pair=False
    if number%2==0:
        pair=True
    return pair
#programme
x=int(input('Entrez x: '))
#appel de la fonction et affichage
if parite(x):
    print(f'{x} est pair')
else:
    print(f'{x} est impair')
```

ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

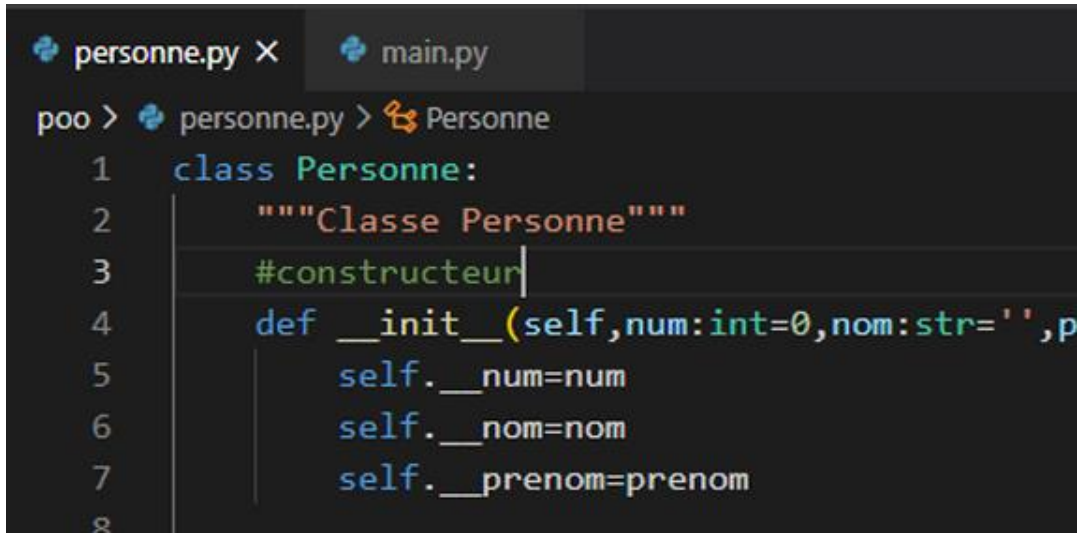


PYTHON ORIENTÉ OBJET

2024/2025

classe

- Par défaut, toute classe en Python a un constructeur par défaut sans paramètre
- Pour simplifier la création d'objets, on peut définir un nouveau constructeur qui prend en paramètre plusieurs attributs de la classe
- On le déclare avec le mot clé __init__ ()
- La déclaration d'un nouveau constructeur, le constructeur par défaut est annulé
- Python n'autorise pas la présence de plusieurs constructeurs (la surcharge)



Visibilité des attributs

Le mot clé private n'existe pas

- On préfixe les attributs par deux underscores (__), pour indiquer qu'ils sont privés
- On préfixe les attributs par un underscore (_) pour indiquer qu'ils sont protégés

__str__

- Pour afficher les détails d'un objet, il faut que la méthode __str__ (self) soit implémentée

```python
class Personne:
    """Classe Personne"""
    #constructeur

    def __init__(self,num:int=0,nom:str='',prenom:str=''):
        self.__num=num
        self.__nom=nom
        self.__prenom=prenom

    #implémentation de la méthode __str__
    def __str__(self)->str:
        return self.__prenom+" "+self.__nom+" "+str(self.__num)
```

personne.py ● main.py ×

poo > main.py > ...

```python
from personne import Personne

p= Personne(23,'alaoui','selma')
print(p)
```

Setters et getters


```python
class Personne:
    #constructeur
    def __init__(self,num:int=0,nom:str='',prenom:str=''):
        self.set_num(num)
        self.__nom=nom
        self.__prenom=prenom

    #implémentation de la méthode __str__
    def __str__(self)->str:
        return self.__prenom+" "+self.__nom+" "+str(self.__num)

    #setters et getters
    def set_num(self,num:int)->None:
        if num>0:
            self.__num=num
        else:
            self.__num=0

    def get_num(self)->int:
        return self.__num

    def set_nom(self,nom:str)->None:
        self.__nom=nom

    def get_nom(self)->str:
        return self.__nom

```

Destructeur __del__

- destructeur (__del__): exécuté à la destruction de l'objet
- Peut être implicitement (lorsque l'objet n'est plus référencée)

ou explicitement avec le mot clé del

personne.py

```python
class Personne:
    #constructeur
    def __init__(self,num:int=0,nom:str='',prenom:str=''):
        self.__num=num
        self.__nom=nom
        self.__prenom=prenom

    #destructeur
    def __del__(self):
        print("destructeur appelé")
```

main.py

```python
from personne import Personne

p= Personne(23,'alaoui','selma')
print(p)
del p
print("fin")
```

Attribut static


```python
class Personne:
    #attribut statique
    nbr_personnes=0
    #constructeur
    def __init__(self,num:int=0,nom:str='',prenom:str=''):
        self.__num=num
        self.__nom=nom
        self.__prenom=prenom
        #incrémentons le compteur
        Personne.nbr_personnes+=1
```

Méthode statique

- Une méthode static ne reçoit pas self comme paramètre
- Déclaration avec le décorateur (@staticmethod)
- On l'appel à travers le nom de la classe

```
5     def __init__(self,num:int=0,nom:str='',prenom:str=''):
6         self.__num=num
7         self.__nom=nom
8         self.__prenom=prenom
9         #incrémentons le compteur
10        Personne.increment()
11
12    #destructeur
13    def __del__(self):
14        #décrementon le compteur
15        Personne.nbr_personnes-=1
16        print("destructeur appelé")
17
18    #Méthode static
19    @staticmethod
20    def increment()->None:
21        Personne.nbr_personnes+=1
22
```

Héritage

- Quand?– Lorsque deux ou plusieurs classes partagent plusieurs attributs (et méthodes)– Lorsqu'une Classe1 est une [sorte de] Classe2
- Syntaxe générale

```
1  class ClassFille(CalsseMère):
2      #Code
```

Héritage

- Exemple

– Un enseignant a un numéro, un nom, un prénom, et un salaire
– Un étudiant a aussi un numéro, un nom, un prénom, et un niveau
– Sémantiquement, enseignant et étudiant sont une sorte de personne
– En plus, les deux partagent plusieurs attributs tels que numéro, nom, prénom
– Donc, on peut utiliser la classe Personne puisqu'elle contient tous les attributs numéro, nom, prénom
– Les classes Etudiant et Enseignant hériteront donc de la classe Personne

Héritage

personne.py | etudiant.py | enseignant.py | main.py

poo > héritage > etudiant.py > Etudiant > __str__

```python
from personne import Personne
class Etudiant(Personne):

    def __init__(self,num:int=0,nom:str='',prenom:str='',niveau:str=''):
        #appel méthode de la classe mère
        super().__init__(num,nom,prenom)
        self.__niveau=niveau

    def __str__(self):
        return super().__str__()+" "+self.__niveau
```

poo > héritage > enseignant.py > Enseignant > __str__

```python
from personne import Personne
class Enseignant(Personne):

    def __init__(self,num:int=0,nom:str='',prenom:str='',salaire:float=0):
        #appel méthode de la classe mère
        super().__init__(num,nom,prenom)
        self.__salaire=salaire

    def __str__(self):
        return super().__str__()+" "+str(self.__salaire)
```

Héritage multiple

```python
from personne import Personne
from enseignant import Enseignant
from etudiant import Etudiant

class Doctorant(Etudiant,Enseignant):

    def __init__(self,num:int=0,nom:str='',prenom:str='',niveau:str='',salaire:float=0,annee:str=''):
        Etudiant.__init__(self,num,nom,prenom,niveau)
        Enseignant.__init__(self,num,nom,prenom,salaire)
        self.__annee=annee

    def __str__(self) -> str:
        return Personne.__str__(self) + " " + str(self.salaire) + " "+ str(self.niveau) + " " + str(self.__annee)
```

poo > héritage > main3.py > ...

```python
from doctorant import Doctorant
doctorant = Doctorant (300, 'cooper', 'austin',1700, 'doctorat', '1 ère année')
print(doctorant)

```

Enoncé:Héritage

On modélise une application devant servir à l'inventaire d'une bibliothèque. Elle devra traiter des documents de nature diverse : des livres, des dictionnaires

1) Tous les documents possèdent un titre. Quand un document est créé:
    - son titre est donné à la création
    - On veut attribuer un numéro d'enregistrement unique dès que l'on créé un objet Document: On veut que le premier document créé ait le numéro 0, et que ce numéro s'incrémente de 1 à chaque création de document.
2) A chaque livre est associé, en plus, un auteur et un nombre de pages,
3) les dictionnaires ont, eux, pour attributs supplémentaires une langue et un nombre de tomes.

Enoncé

- La classe document est une classe non instanciable
- Définir une méthode availabeTome(), qui vérifie si le dictionnaire a plus qu'un Tome ou non

Class abstraite

- hérite de la classe ABC (Abstract Base Class) du module abc
- ne peut être instanciée si elle contient une ou plusieurs méthodes abstraites

```
1   from abc import ABC,abstractmethod
2
3   class Personne(ABC):
```

méthode abstraite

- C'est une méthode non implémentée (sans code)
- Une méthode abstraite doit être déclarée dans une classe abstraite
- Une méthode abstraite doit être implémentée par les classes filles de la classe abstraite

Ici on a déclaré afficherDetail comme méthode abstraite (son code en utilisant pass)

Sans oublier d'importer abstractmethode

```python
from abc import ABC,abstractmethod

class Personne(ABC):
    #attribut statique
    nbr_personnes=0
    #constructeur
    def __init__(self,num:int=0,nom:str='',prenom:str=''):
        self._num=num
        self.__nom=nom
        self.__prenom=prenom
        #incrémentons le compteur
        Personne.increment()

    @abstractmethod
    def afficherDetail(self):
        pass
```

TypeScript: isinstance()

- Pour connaître la classe d'un objet, on peut utiliser la méthode isinstance()

```
8  print(isinstance(etudiant,Etudiant))
9  print(isinstance(enseignant,Enseignant))
```

Renvoie True si l'objet correspond à l'instance de la classe définit comme deuxième paramètre. Elle retourne False dans le cas contraire