LEARNING PYTHON

* Print(‘’\*’’ \* 10) : Cette ligne permet d’écrire 10 fois le symbole \*

1. Les variables

Déclaration et type de variables :

* Déclaration

En python, Nous n’avons pas besoin de préciser le type d’une variable pour lui affecter une valeur

Price=10

Nous pouvons réinitialiser la valeur d’une variable

* Types de variables
* Integer
* Float
* String
* Boolean

NB : Le Python est sensible à la casse

* La concaténation
* Pour concaténer deux chaines de caractères en python, on utilise l’opérateur (+)
* Pour afficher un message à l’écran et demander une valeur à l’utilisateur, on utilise la fonction **input()**

**NB : input() Peut être affecter à une variable, cette variable va donc contenir la valeur saisie par l’utilisateur**

Notes\_\_\_ : Il faut savoir que la fonction **input ()** renvoie une chaine de caractère… Alors si nous voulons utiliser la donnée saisie par l’utilisateur (Pour effectuer un calcul par exemple), Nous devons d’abord convertir l’entrée utilisateur en type de varia le dont nous avons besoin pour notre opération

Pour convertir les chaines de caractères, nous pouvons utiliser les fonctions suivantes :

* **Int()** Pour convertir en **INT**
* **Float()** Pour convertir en **FLOAT**
* **Bool()** Pour convertir en **Booléen**

Pour afficher le type d’une variable, on utilise la fonction **type()**

**Note\_\_\_\_ : Pour écrire un texte sur plusieurs lignes, on utilise les triples cotes comme ceci : ‘’’ ‘’’ et on peut aller simplement à la ligne et mettre notre code entre les cotes**

Les chaines de caractères sont enfaite des tableaux de caractères… Pour accéder à un caractère, on peut utiliser son index

**Note\_\_\_\_ : On peut afficher les caractères entre deux indexes, pou cela, on utilise la syntaxe suivante : course [0 :3]** Ce code veut dire qu’on veut afficher tous les caractères entre l’indexe 0 et l’indexe 3(Mais le caractère à l’indexe 3 ne sera pas inclus)

Note\_\_\_\_ : Si nous mettons plutôt un chiffre suivit des deux points et nous ne mettons riens de l’autre côté des deux points, le programme va afficher tous les caractères qui suivent le chiffre qu’on a mit de l’autre côté. **Course [1 :] (On affiche tous les caractères qui viennent après le premier)**

Note\_\_\_\_ : Inversement, si on ne met rien au début et on met plutôt un nombre de l’autre côté des deux points, on va afficher tous les caractères commençants à l’indice 0 jusqu’au caractère qu’on a précisé. **course [ :3] (On affiche tous les caractères qui viennent avant celui qui est à l’indice 3)**

Note\_\_\_\_ : Cette syntaxe est très utile pour la copie ou pour le clonage … Si on veut copier une chaine de caractère dans une autre variable, on peut simplement faire :

**another=course[ :]**

**Note\_\_\_\_ : Les derniers caractères d’une chaine ont des indices négatifs. Par exemple : course[-1] représente le dernier caractère de la chaîne**

1. UTILISATION, MANIPULATION DES VARIABLES

Pour afficher des variables dans une chaine de caractère, on peut utiliser la syntaxe suivante (Formated string):

msg=f'Hello {name}, your age is {age}'

Applications:

Note\_\_\_\_: **chaine.split()** permet de diviser une chaîne en sous chaînes en utilisant l’espace comme séparateur

On peut ensuite compter les caractères de cette chaine (Qui sont ici les mots) aven **len()**

**28/02/2024**

1. **LES CHAINES DE CARACTERE**

**Note\_\_\_\_ : Pour convertir une chaine :**

* **Chaine.upper() permet de convertir en majuscule**
* **Chaine.lower() permet de convertir en minuscule**
* **Chaine.islower()** permet de vérifier si tous les caractères du chaine sont en minuscule
* **Chaine.upper()** permet de vérifier si tous les caractères d’une chaîne sont en majuscule

Note\_\_\_\_ : Pour rechercher un caractère dans un tableau de caractères(Chaine de caractères)

* **Chaine.find(‘’P’’)** renvoie l’indice de la première occurrence de P dans la chaine ; nous pouvons également passer une chaine de caractères en paramètre à la fonction. Si le caractère ou la chaine de caractère n’est pas trouvée, la fonction renvoie un nombre négatif (-1)
* **Chaine.replace(‘’caractère/mot à remplacer’’,’’caractère/mot avec lequel on veut remplacer’’)**
* **Chaine.title()** Permet de mettre les premières lettres d’une chaine de caractère en majuscule tandis que les autres lettres sont converties en minuscule

Note\_\_\_\_ : Si nous voulons rechercher une séquence d’un caractère ou d’une chaine de caractère dans une autre, nous pouvons utiliser l’expression : **‘Python’ in chaine**…

Cette expression renvoie **True** si la chaine **‘Python’** se trouve dans la variable chaine et **False** sinon.

1. LES OPERATEURS ARITHMETIQUES

* **+** permet de faire l’addition
* **-** Permet de faire la soustraction
* **\*** Permet de faire la multiplication de deux nombres
* **//** Permet de faire une division entière (Le quotient sera toujours un nombre entier)
* **/** Permet de faire une division
* **\*\*** Permet de calculer la puissance d’un nombre… 10 \*\* 3 permet de calculer 10 à la puissance 3

Fonctions/Méthodes sur les nombres :

* **Abs ()** permet de calculer la valeur absolue d’un nombre
* **Round ()** Permet d’arrondir un nombre (à 0 nombre après la virgule par défaut) mais on peut aussi préciser le nombre de chiffres que l’on veut après la virgule
* **Ceil ()** Permet d’arrondir un nombre à la valeur qui lui est directement supérieure
* **Floor ()** Permet d’arrondir un nombre à la valeur qui lui est directement inférieure
* **Facrotial ()** Permet de calculer le factoriel d’un nombre
* **Comb ()** Permet de calculer le nombre de combinaisons de possible d’un nombre d’éléments (Combinaison comme en proba)
* **n = int(input().strip()):** Cette ligne demande à l'utilisateur de saisir une valeur via la fonction input(), puis la convertir en un entier en utilisant int(). La méthode strip()est utilisée pour supprimer les espaces inutiles éventuellement présents autour de la saisie.

La liste des méthodes sur les nombres est disponible sur : <https://docs.python.org/3/library/math.html>

NB : Pour utiliser des fonctions mathématiques avancées, il faut inclure la bibliothèque **math. C’est elle qui contient les fonctions mathématiques avancées**

1. **IF STATEMENT**

**Avec le if, nous pouvons avoir les opérateurs suivants :**

* **AND**
* **OR**
* **NOT**

**Note\_\_\_**

**Nous pouvons vérifier le type d’une variable en faisant :**

number = 0  
*if type*(number) *is int*:  
 *print*(number)

Ou encore

*if type*(number) == *int*:  
 *print*(number)

* **Les opérateurs de comparaison**
* **>**
* **<**
* **>=**
* **<=**
* **==**

1. **LES BOUCLES**

* **La boucle while**

**Syntaxe : while condition :**

**Instructions**

**Note\_\_\_ La boucle while fonctionne également avec un else qui permet d’exécuter une ou plusieurs instructions au cas où on n’a pas pu avoir le résultat qu’on cherchait avec la boucle (Il faut indenter ce else avec la boucle while)**

**NB : Pour s’assurer que l’entrée utilisateur sera toujours en minuscule, on fait ceci :**

**Name=input(‘’Enter your name’’).lower()**

* **LA BOUCLE FOR**

**Note\_\_\_** Nous pouvons parcourir une chaîne de caractère en **Python** de cette manière**:**

**For item in ‘Python’ :**

**Print(item)**

**Note\_\_\_** Nous pouvons également parcourir une liste de cette manière :

*for* item *in* ['Python','Java','C++','JavaScript','PHP','C#']:  
 *print*(item)

**NB :** Pendant l’exécution de la boucle, item va représenter chaque élément de la liste

* La fonction **range**

**La fonction range en Python est utilisée pour générer une séquence de nombres entiers dans un interval donné.**

**Syntaxe générale :**

**Range(start,stop,step)**

* **Start(Optionnel) :La valeur de départ de la séquence, si cette valeur n’est pas spécifiée, elle est par défaut à 0**
* **Stop : Valeur de fin de séquence… La séquence s’arrête avant d’atteindre cette valeur**
* **Step (Optionelle) : L’incrément (pas) entre chaque nombre dans la séquence, si cette valeur n’est pas spécifiée elle est par défaut à 1**

**Nous pouvons indenter deux boucles for pour générer des coordonnées de point(Par exemple)**

*for* x *in range*(4):  
 *for* y *in range*(3):  
 *print*(f'({x},{y})')

**Note\_\_\_ Nous pouvons ajouter un else juste à la fin d’une boucle for… Ce else va s’exécuter juste après le for (Pour indiquer que la boucle for a bien éré exécutée)**

1. **LES LISTES**

* **Print(name[2 :])** Va afficher tous les éléments de la liste de noms mais à partir de celui qui a l’indice **2**
* **Name[-1**] permet d’accéder au dernier élément de la **liste**
* **Name[2 :4]** Permet d’accéder aux éléments de la liste qui sont entre lesindices **2 et 4 (On exclut l’élément à l’indice 4)**

1. **LES LISTES A DEUX DIMENSIONS**

**Une liste à deux dimensions ou matrice est une liste dans laquelle chaque élément représente une liste**

**Pour parcourir on utilise les boucles imbriquées :**

matrix=[  
 [1,3,8],  
 [4,0,8],  
 [5,17,7]  
]  
  
*for* row *in* matrix:  
 *for* item *in* row:  
 *print*(item)  
  
 *print*("\n")

1. **METHODES/ FONCTIONS SUR LES LISTES**

* **Append()** permet d’ajouter un élément à la fin de la liste
* **Insert(position,element)** Permet d’ajouter un élément à une position précise de la liste
* **Remove(element\_a\_retirer)** Permet d’enlever un élément de la liste
* **Clear()** permet de supprimer tous les éléments présents dans la liste
* **Pop()** Permet de supprimer le dernier élément de la liste
* **Index(element\_a\_chercher)** Permet de chercher un élément dans la liste de nombre, la fonction renvoi l’index de la première occurrence de l’élément dans la liste
* Nous pouvons aussi utiliser l’opérateur **IN** pour vérifier si un élément est dans la liste, On verra True si l’élément est dans la liste et False sinon
* **NOT IN** permet de vérifier si un élément ne se trouve pas dans la liste
* **Count(élément)** Permet de compter le nombre d’occurrences de l’élément passer en paramètre dans la liste
* **Max(liste) Permet de connaitre le plus grand élément d’une liste**
* **Nous pouvons passer plusieurs valeurs numériques à max et la fonction va renvoyer le plus grand élément parmis les valeurs passées en paramètre**
* **Min(liste) Permet de connaitre le plus petit élément d’une liste**
* **Sum(liste) Permet de faire la somme des éléments d’une liste**
* **Bin() Permet de convertir un nombre en binary string**
* **Extend() permet de fusionner deux listes**

Note\_\_\_ Pour importer toutes les fonctions mathématiques du module math, on peut faire :

*from* math *import* \*

Note\_\_\_ Nous pouvons aussi fusionner deux listes en utilisant tout simplement l’opérateur +

**Liste3=liste1+liste2**

NB : None est un Objet en Python qui représente l’absence de valeurs

* **Sort()** Permet de trier une liste dans l’ordre croissant
* **Reverse()** Permet d’afficher une liste dans l’ordre inverse

NB : avant d’appeler la méthode reverse, il faut d’abord appeler la méthode **sort()**

Nous pouvons trier une liste dans l’ordre décroissant en passant un paramètre à la méthode **sort() :**

liste = [5, 8, 45, 7, 3, 9]  
liste.sort(reverse=True)  
print(liste)

Ou encore :

liste = [5, 8, 45, 7, 3, 9]  
sorted\_list = sorted(liste, reverse=True)  
print(sorted\_list)

* **Copy()** Permet de copier une liste dans une autre
* Exercice pour supprimer les doublons dans une liste :
* numbers=[3,0,7,3,-2,9,18,4,8,7,2,4,5,5,6]  
  *#Remove the duplicates in the list*uniques =[]  
  *for* number *in* numbers:  
   *if* number *not in* uniques:  
   uniques.append(number)  
  *print*(numbers)  
  *print*(uniques)

NB: L’opérateur not in permet de vérifier qu’un élément n’est pas dans la liste

1. **LES TUPLES**

Contrairement aux listes, les tuples sont un type de données qui ne peuvent être modifiées, tout ce qu’on peut faire, c’est compter le nombre d’occurrences d’un éléments et de trouver l’index d’un élément dans le **tuple**

numbers =(1,2,3)  
*print*(numbers.count(3))

Note\_\_\_ Nous pouvons affecter tous les éléments d’une Tuple ou d’une Liste dans des variables (Tout ceci en une ligne) en faisant :

numbers =(1,2,3)  
x,y,z=numbers  
*print*(x,y,z)

Note\_\_\_ Pour parcourir des tuples, il faut inclure le module **tuples**

**Note\_\_\_**

**Nous pouvons aussi utiliser un constructeur pour construire un tuple en python**

1. **LES DICTIONNAIRES**

Les dictionnaires sont des types de données qui permettent de stocker plusieurs éléments de type différents, ils fonctionnent avec le principe de Clé : Valeur

User={  
 "name": "AFANA JEOVANY",  
 "age": 22,  
 "email": "wilfrieddylan@gmail.com",  
 "is\_verified": *True*}

* Nous pouvons modifier les valeurs d’un dictionnaire
* Nous pouvons ajouter un nouveau couple Clé : Valeur à un dictionnaire
* User['married'] ='Yes'  
  *print*(User)

Exercice dictionnaires:

numbers={  
 0: 'zero',  
 1: 'One',  
 2: 'Two',  
 3: 'Three',  
 4: 'Four',  
 5: 'Five',  
 6: 'Six',  
 7: 'Seven',  
 8: 'Eight',  
 9: 'Nine',  
}  
  
user\_phone=*input*('Please enter your phone number \n')  
output=''  
*for* item *in* user\_phone:  
 *if int*(item) *in* numbers:  
 output+=(' '+numbers[*int*(item)])  
output+=' !'  
*print*(output)

* La méthode **get()** en Python est utilisée sur un dictionnaire pour récupérer la valeur associée à une clé donnée. Si la clé n'existe pas dans le dictionnaire, **get()** renvoie une valeur par défaut plutôt que de générer une erreur, ce qui est utile pour éviter les exceptions KeyError.

D'accord, laisse-moi clarifier ça. Imaginons un dictionnaire contenant des informations sur des fruits :

**fruits = {"pomme": 5, "banane": 3, "orange": 7}**

Si vous voulez obtenir le nombre de pommes dans ce dictionnaire, vous pouvez utiliser **get()** de cette manière :

**nombre\_de\_pommes = fruits.get("pomme")**

Cela retournera la valeur **5**, car "pomme" est une clé existante dans le dictionnaire **fruits**.

Maintenant, si vous essayez d'obtenir le nombre de poires :

**nombre\_de\_poires = fruits.get("poire")**

Au lieu de générer une erreur, **get()** retournera **None** parce que "poire" n'est pas une clé existante dans le dictionnaire. Vous pouvez également spécifier une valeur par défaut à retourner si la clé n'existe pas :

**nombre\_de\_poires = fruits.get("poire", 0)**

Cela retournera **0** si "poire" n'est pas une clé dans le dictionnaire.

* Le [dict()](https://docs.python.org/3/library/stdtypes.html#dict)constructeur construit des dictionnaires directement à partir de séquences de paires clé-valeur :

>>>

**>>>** dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])

{'sape': 4139, 'guido': 4127, 'jack': 4098}

* Lorsque les clés sont de simples chaînes, il est parfois plus simple de spécifier des paires à l'aide d'arguments mots-clés :

>>>

**>>>** dict(sape=4139, guido=4127, jack=4098)

{'sape': 4139, 'guido': 4127, 'jack': 4098}

* Lors d'une boucle dans une séquence, l'index de position et la valeur correspondante peuvent être récupérés en même temps à l'aide de la [enumerate()](https://docs.python.org/3/library/functions.html" \l "enumerate" \o "énumérer)fonction.

>>>

**>>> for** i, v **in** enumerate(['tic', 'tac', 'toe']):

**...**  print(i, v)

**...**

0 tic

1 tac

2 toe

* Pour parcourir deux séquences ou plus en même temps, les entrées peuvent être associées à la [zip()](https://docs.python.org/3/library/functions.html#zip)fonction.

>>>

**>>>** questions = ['name', 'quest', 'favorite color']

**>>>** answers = ['lancelot', 'the holy grail', 'blue']

**>>> for** q, a **in** zip(questions, answers):

**...**  print('What is your **{0}**? It is **{1}**.’. format(q, a))

**...**

What is your name? It is lancelot.

What is your quest? It is the holy grail.

What is your favorite color? It is blue.

* Pour parcourir une séquence en sens inverse, spécifiez d’abord la séquence dans le sens avant, puis appelez la [reversed()](https://docs.python.org/3/library/functions.html" \l "reversed" \o "renversé)fonction.

>>>

**>>> for** i **in** reversed(range(1, 10, 2)):

**...**  print(i)

**...**

9

7

5

3

1

* Pour parcourir une séquence dans un ordre trié, utilisez la [sorted()](https://docs.python.org/3/library/functions.html" \l "sorted" \o "trié)fonction qui renvoie une nouvelle liste triée tout en laissant la source inchangée.

>>>

**>>>** basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']

**>>> for** i **in** sorted(basket):

**...**  print(i)

**...**

apple

apple

banana

orange

orange

pear

* L'utilisation [set()](https://docs.python.org/3/library/stdtypes.html#set)sur une séquence élimine les éléments en double. L'utilisation de [sorted()](https://docs.python.org/3/library/functions.html#sorted)en combinaison avec [set()](https://docs.python.org/3/library/stdtypes.html#set)sur une séquence est une manière idiomatique de parcourir des éléments uniques de la séquence dans un ordre trié.

>>>

**>>>** basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']

**>>> for** f **in** sorted(set(basket)):

**...**  print(f)

**...**

apple

banana

orange

pear

Il est parfois tentant de modifier une liste pendant que vous la parcourez ; cependant, il est souvent plus simple et plus sûr de créer une nouvelle liste.

>>>

**>>> import** **math**

**>>>** raw\_data = [56.2, float('NaN'), 51.7, 55.3, 52.5, float('NaN'), 47.8]

**>>>** filtered\_data = []

**>>> for** value **in** raw\_data:

**...**  **if** **not** math.isnan(value):

**...**  filtered\_data.append(value)

**...**

**>>>** filtered\_data

[56.2, 51.7, 55.3, 52.5, 47.8]

1. **LES FONCTIONS**

Pour définir une fonction en Python, on utilise **def** :

NB : Ceci est une implémentation avec les positionals arguments

*def* show(number):  
 *print*("Tu t'appelles AFANA JEOVANY")  
 *print*(f'Le nombre que tu as passé en paramètre est: {number}')  
  
*print*('Voici ton message:')  
show(4)

1. **KEYWORD ARGUMENT**

Comme nous le savons, l’ordre de passage des arguments aux fonctions est important mais en python, nous pouvons passer des arguments aux fonctions sans tenir compte de l’ordre mais il faudra affecter les valeurs aux paramètres. Voici un exemple:

*def* message (nom, prenom):  
 *print* (f'Hi {nom} {prenom} how are you, my friend?')  
  
message(prenom='Jeovany',nom='Afana'.upper())

ça permet aussi d’améliorer la visibilité du code car avec ça on sait à quoi sert et quel est le rôle de chaque argument passé à la fonction.

1. **RETURN STATEMENT**

Par défaut, les fonctions en python retournent **None** qui s’signifie **ABSENCE DE VALEUR** alors si on n’indique pas une valeur de retour à une fonction, elle retourne automatiquement **None** par défaut.

1. **EXCEPTIONS**

Il peut arriver lorsqu’on demande à l’utilisateur d’entrée une valeur (Entière par exemple) que celui-ci n’entre pas correctement la valeur que nous attendions (Il peut entrer une chaîne de caractère), et ceci peut provoquer une erreur pendant l’exécution du programme.

Pour éviter ça, nous devons le prévoir et pour ça, nous allons utiliser les **exceptions**

*try*:  
 age=*int*(*input*("Please enter your age \n"))  
 *print*(f'You are {age} years old')  
*except ValueError*:  
 *print*('Invalid age')

NB : Cet exemple est fait avec une exception de type : ValueError

Note\_\_\_ Nous pouvons aussi créer une exception sans toute fois préciser dans le **except** le type d’exception (Si on ne connait pas à l’avance)

*try*:  
 number = *int*(*input*("Enter a number: "))  
 *print*(number)  
*except*:  
 *print*('Something went wrong')

Note\_\_\_ Nous pouvons ajouter un **else** après un **except (Ce qui va se passer au cas ou aucune exception n’est levée)**

*try*:  
 number = *int*(*input*("Enter a number: "))  
 *print*(number)  
*except*:  
 *print*('Something went wrong')  
*else*:  
 *print*('You entered a number')

* Quelques types d’exceptions en python :
* **ValueError** : Signifie que le type de la valeur reçue ne correspond pas au type qui était attendu
* **ZeroDivisionError** : Signifie qu’on ne peut pas diviser un nombre par 0
* **AttributeError**: Signifie que l’attribut que nous voulons utiliser ne se trouve pas dans la classe

Au cas où nous avons plusieurs choses à vérifier, par exemple nous voulons nous rassures que le nombre entrer est bien un entier et que le nombre entrer n’est pas 0, nous pouvons utiliser les deux **exceptions :**

*import* math  
*try*:  
 age=*int*(*input*("Please enter your age \n"))  
 number=200  
 division=number/age  
 *print*(f'The age: {*round*(division,2)}')  
*except ZeroDivisionError*:  
 *print*('You can\'t divide by zero. Please enter your age')  
*except ValueError*:  
 *print*('Invalid age')

1. **CLASSES**

* Syntaxe de création d’une classe en Python :

*class* Point:  
 *def* presentation(*self*):  
 *print*('I\'m a Point')  
 *def* draw(*self*):  
 *print*('Draw')

* Création d’une instance de la classe :
* point1=Point()
* Nous pouvons ajouter des attributs à une instance d’une classe :

point1.x=10  
point1.y=10

* Dans une classe, il y’a ce qu’on appelle les attributs et les méthodes de classe

1. Les attributs de classe sont des attributs qui sont déclarer en dehors de toutes les méthodes de la classe donc toutes les instances de la classe auront la même valeur de cet attribut… Modifier la valeur de cet attribut va modifier la valeur dans toutes les instances de la classe.
2. Les méthodes de classe sont des méthodes qui peuvent être accéder sans avoir besoin d’instancier la classe… Elles ne peuvent pas utiliser les attributs des instances mais peuvent utiliser les attributs de classe

NB : En Python, une méthode de classe prend comme premier paramètre **cls** et au-dessus le décorateur **@classmethod**

NB : Les méthodes et les attributs de classe peuvent être utiles dans les cas suivants :

* Partage de données entre instances, pour compter le nombre d’instances créer par exemple (Attributs de classe)
* Les méthodes de classe peuvent être utilisées pour regrouper des méthodes utilitaires qui ne nécessitent pas d’instances pour fonctionner (Méthode qui permet d’afficher la date et heure par exemple)
* Les attributs de classe peuvent être utilisés pour stocker des configurations ou des paramètres globaux que toutes les instances partagent (Configuration globale)
* Les méthodes de classe peuvent être utilisées pour fournir des moyens alternatifs de créer des instances de classe. Par exemple pour créer une instance à partir de différente types de données d’entrée

***NB : Ces attributs appartiennent uniquement à l’instance à laquelle ils ont été ajouté ils ne seront donc pas visibles ni utilisables dans les autres instances de la classe.***

***Note\_\_\_ Nous pouvons supprimer un objet avec la commande suivante :***

***del*** *nom\_objet*

1. **CONSTRUCTOR**

Le constructeur est une méthode qui est appelée à chaque fois qu’une nouvelle instance d’une classe est crée

* Implémentation d’un constructeur et initialisation des attributs :
* *class* Point:  
   *def \_\_init\_\_*(*self*, x, y):  
   *self*.x = x  
   *self*.y = y

NB : **self** ici représente l’objet actuel, c’est un peu comme **this** dans les autres langages

1. **INHERITANCE (Héritage)**

Pour éviter la redondance et éviter de créer plusieurs classes alors que celles-ci devraient avoir les mêmes attributs ou les **mêmes méthodes**, on peut simplement créer une classe mère et faire **hériter les classes** **filles** des méthodes et attributs de la **classe mère :**

*class* Animal:  
 *def* walk(*self*):  
 *print*('Walk !')  
  
  
*class* dog(Animal):  
 *pass*

*class cat(Animal) :*

*pass*

**Note\_\_\_** Si nous créons une classe fille qui hérite des attributs et méthodes de la classe mère et que nous n’avons rien à mettre dans cette classe fille, nous pouvons y mettre le mot clé **pass** afin d’éviter que Python ne nous génère une erreur car le langage n’aime pas les classes vides.

***NB : Chaque classe fille peut avoir ses propres attributs***

1. **MODULES**

Nous avons parfois besoin de séparer notre programme en plusieurs fichiers pour le rendre plus lisible et facilité le débogage… Pour se faire, nous allons créer un autre fichier et y écrire nos fonction, nous allons ensuite importer ce fichier dans notre programme principal :

*import* functions  
  
functions.show(70)

* Dans le programme principal, le fichier importé est considéré comme un objet alors pour accéder aux fonctions qui y sont, on utilise le point.

Note\_\_\_ Nous pouvons également importer une fonction particulière d’un fichier, nous pourrons alors l’utiliser comme si nous l’avions créer dans notre fichier principal

*from* functions *import* show  
show(70)

1. **PACKAGES**

Les packages en Python sont essentiellement des dossiers (ou répertoires) contenant un ensemble de modules Python. Ils sont utilisés pour organiser et structurer des bibliothèques ou des collections de modules connexes.

*import* ecommerce.shipping  
*from* ecommerce.shipping *import* shipping  
shipping()

Voici quelques points clés à retenir sur les packages en Python :

1. **Structure de base** : Un package est simplement un dossier qui contient un fichier spécial nommé **\_\_init\_\_.py**. Ce fichier peut être vide, mais sa présence indique à Python que le dossier est un package Python.
2. **Hiérarchie** : Les packages peuvent être hiérarchiques, ce qui signifie qu'ils peuvent contenir d'autres packages ainsi que des modules. Cette structure en arborescence permet une organisation logique des bibliothèques et des fonctionnalités.
3. **Importation** : Pour importer des modules ou des sous-packages à partir d'un package, vous pouvez utiliser l'instruction **import** suivie du chemin relatif du module ou du sous-package à partir du package racine. Par exemple, si vous avez un package appeler **mypackage** contenant un module **mymodule.py**, vous pouvez l'importer dans un script en utilisant **import mypackage.mymodule**.
4. **Utilisation de \_\_init\_\_.py** : Le fichier **\_\_init\_\_.py** peut contenir du code d'initialisation pour le package. Cela peut inclure l'importation de certains modules, l'initialisation de variables ou d'autres tâches d'initialisation nécessaires pour utiliser le package.
5. **Gestion des chemins** : L'utilisation de packages permet de mieux gérer les chemins d'accès aux modules et facilite la réutilisation du code. Ils sont couramment utilisés pour organiser des projets Python de grande taille ou des bibliothèques qui contiennent plusieurs fonctionnalités distinctes.
6. **GENERATING RANDOM VALUES**

Pour générer des valeurs de manière aléatoire, on va tout d’abord inclure le module **random**

*import* random

Le module random vient avec un objet (random), c’est lui que nous allons utiliser pour générer des éléments de manière aléatoire grâce à ses méthodes

* La méthode random() permet de générer un nombre aléatoire entre 0 et 1
* *import* random  
  *import* math  
  *for* i *in range*(3):  
   *print*(random.random())
* La méthode **randint()** permet de générer un nombre aléatoire dans **un intervalle donné** (Entre paramètres)

*import* random  
*import* math  
*for* i *in range*(3):  
 *print*(random.randint(10,20))

Ce code va générer 3 nombres compris entre 10 et 20 de manière aléatoire

* La méthode **choice()** permet de choisir un élément de manière aléatoire dans une liste, la méthode renvoie l’élément choisie de manière aléatoire.
* *import* random  
  *import* math  
  liste=['AFANA','John','Mary','Kevin']  
    
  *print*(random.choice(liste))

1. **PATHLIB MODULE**

Le module **pathlib** permet de communiquer avec les **fichers/dossiers** de l’ordinateur en utilisant les chemins absolus/relatifs

*from* pathlib *import* Path

Pour ça, on utilise principalement le Path (Qui est un objet)

Note\_\_\_ Si on créer une occurrence de l’objet **Path** et qu’on ne passe rien en paramètre au constructeur, cela voudra dire que notre variable représente le document actuel où on se trouve.

path = Path()

Note\_\_\_ Par contre, nous pouvons aussi créer une occurrence de l’objet Path en passant en paramètre le nom du fichier/dossier que nous voulons utiliser.

path = Path("classes.py")

* La méthode **exists()** permet de vérifier si le fichier/dossier que nous avons passé en paramètre existe vraiment dans le répertoire où on se trouve
* path = Path("classes.py")  
    
  *print*(path.exists())
* La méthode **mkdir()** permet de créer un nouveau répertoire dans le répertoire actuel

path = Path("Joe")  
  
*print*(path.mkdir())

* La méthode **rmdir()** permet de supprimer un répertoire
* path = Path("Joe")  
    
  *print*(path.rmdir())
* La méthode **glob()** permet d’accéder à plusieurs éléments d’un répertoire (Fichiers et dossiers)… Cette méthode renvoie une liste d’éléments que nous pouvons parcourir avec une boucle for.
* Utilisation de la méthode **glob()**
  1. **Avec ‘’\*’’**

Ceci va permettre de récupérer tous les fichiers/dossiers contenus dans le répertoire courant

* 1. **Avec ‘’\*.py’’**

Ceci va permettre de récupérer tous les fichiers avec l’extension .py (On peut le faire également avec les autres extensions).

* 1. **Avec ‘’\*.\*’’**

Ceci permet d’accéder à tous les fichiers contenus dans le répertoire courant.

1. Reading and Write in file

Pour ouvrir un fichier, nous utilisons la fonction **open()**

**Pour fermet le fichier, nous faisons : nom\_fichier.close()**

* R : Lecture
* W : écriture
* A : pour ajouter des choses au fichier
* Pour vérifier si le fichier passer en paramètre à la fonction open a bien été ouvert et qu’il est possible de le lire, on utilise la fonction **readable()**
* file = *open*('text.txt', 'r')  
  *print*(file.readable())
* Pour lire et afficher les lignes d’un fichier, nous utilisons la méthode : **readlines()**
* file = *open*('text.txt', 'r')  
  *print*(file.readable())  
  *print*(file.readlines())

Ceci va afficher toutes les lignes du fichier dans une liste… Nous pouvons accéder à la première ligne du fichier en faisant :

*print*(file.readlines()[0])

* Nous pouvons créer un fichier si celui-ci n’existe pas encore, il suffit de l’ouvrir en écriture et y mettre quelque chose
* Nous pouvons aussi créer et écrire du code dans un autre fichier python depuis notre programme