INITIATION PYTHON

DECLARER UNE VARIABLE

Une **variable** est composée de deux éléments : son **nom** et sa **valeur**.

- L'attribution d'une valeur à une variable s'appelle une **affectation**.
- La **valeur** d'une variable peut être **modifiée**.
- Le **type** d'une variable dépend de sa **valeur**.
- Les noms de vos variables doivent être **clairs**, **explicites** et doivent suivre une **convention typographique**.

```
>>> # val est une variable. Elle est égale à 3.
... val=3
>>> print(val)
3
>>>
```

CALCUL AVEC UNE VARIABLE :

```
(base) utilisateur@utilisateur-ThinkPad-T430:~$ python3
Python 3.7.4 (default, Aug 13 2019, 20:35:49)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> val=3
>>> val=val+6
>>> print(val)
9
>>>
```

```
Type "help", "copyright", "credits" or "license" for more information.
>>> val = 5
>>> val = val/2
>>> print(val)
2.5
>>> val = 5
>>> val = val//2
>>> print(val)
2
>>>
```

```
>>> val = 5
>>> val = val % 2
>>> print(val)
1
>>> val = 5
>>> val **2
25
```

```
>>> val = 10
>>> val += 2
>>> print (val)
12
>>> val-=2
>>> print(val)
10
>>> val*=2
>>> print(val)
20
>>> val/=2
>>> print(val)
10.0
>>>
```

```
>>> prenom ="amaria"
>>> print(prenom)
amaria
>>>
```

MANIPULATION DES TYPES DE FICHIERS

```
Type "help", "copyright", "credits" or "license" for more information.
>>> val=5
>>> val=int(5)
>>> print(val)
5
>>> val=float(val)
>>> print(val)
5.0
>>>
>>> nom="toto"
>>> nom2="pauvre"
>>> print(nom2,"",nom)
pauvre  toto
>>>
```

Il existe de nombreuses fonctions préexistantes en Python ! En plus de celles déjà vues, il y a par exemple :

- `len()` : une fonction qui renvoie la longueur d'un élément. Vous vous souvenez des chaînes de caractères ? Utiliser cette fonction sur une chaîne de caractères permet par exemple de savoir combien de caractères cette dernière contient ;
- `type()` : permet d'afficher le type d'une variable ;
- `pow(a, b)` : permet de calculer a puissance b. Elle est équivalente à l'écriture `a**b` ;
- `abs()` : retourne la valeur absolue d'un nombre.

```
>>> val="ddddddddddd"
>>> len(val)
10
>>> type(val)
<class 'str'>
```

```
>>> a = 100
>>> b = 2
>>> pow(a, b)
10000
```

LES FONCTIONS

```
>>> def afficherPerimetre(cote1, cote2, cote3):
...     perimetre = cote1 * cote2 * cote3
...     print(perimetre)
...
>>> afficherPerimetre(10, 11, 4) # => 440
440
>>> afficherPerimetre(2, 2, 3.5) # => 14
14.0
>>>
```

```
>>> def calculPerimetre(cote1, cote2, cote3):
...     perimetre = cote1 * cote2 * cote3
...     return perimetre
...
>>> perimetre1=calculPerimetre(6, 4, 3)
>>> print(perimetre1)
72
>>> perimetre2 = calculPerimetre(10, 3, 11)
>>> print("L'aire de mon premier triangle est", perimetre1, "et celle de mon sec
ond est", perimetre2)
L'aire de mon premier triangle est 72 et celle de mon second est 330
>>>
```

LA PROGRAMMATION ORIENTEE OBJET :

Python est un langage de programmation orienté objet : cela signifie qu'en Python, tout est objet !

Les classes : des modèles d'objets

Exemple : la voiture

- ses caractéristiques, appelées **attributs** : elle a forcément 4 roues, une couleur, une forme, une puissance moteur, etc. ;
- ses fonctionnalités, appelées **méthodes** : elle peut rouler, freiner, etc.
-

Ainsi, à partir de ce plan, vous pouvez créer différents modèles de voiture :

- un familiale classique, de couleur verte, de puissance moyenne (110 ch) ;
- une voiture de sport, rouge, relativement puissante (180 ch) ;
- une petite voiture citadine bleue, peu puissante (90 ch) ;
- etc.

Et peu importe le modèle de la voiture, elles sont toutes capables de rouler ou freiner, mais pas avec la même performance !
En résumé, **une classe** correspond au plan d'un objet, définissant ses **attributs** et ses **méthodes**. À partir d'une même classe, on peut donc créer plusieurs objets d'un même type, mais aux attributs différents : on appelle cela **des instances de classe**.

STOCKER DES VARIABLES DANS LES TABLEAUX :

```
>>> nomcli=["toto", "tata", "titi", "tutu","tete"]
>>> print(nomcli[0])
toto
>>> print(nomcli[3])
tutu
```

```
>>> nomcli=["toto", "tata", "titi", "tutu","tete"]
>>> nomcli[4]= "taitai"
>>> print(nomcli)
['toto', 'tata', 'titi', 'tutu', 'taitai']
>>>
```

```
>>> print(nomcli)
['toto', 'tata', 'titi', 'tutu', 'taitai']
>>> print(nomcli[1:3])
['tata', 'titi']
>>> print(nomcli[-1])
taitai
```

```
>>> list=[]
>>> list.append(7)
>>> list.append(5)
>>> print(list)
[7, 5]
>>> list.insert(1, 12)
>>> print(list)
[7, 12, 5]
>>> list.remove(5)
>>> print(list)
[7, 12]
```

```
>>> liste = []
>>> liste.append(7) # -> [7]
>>> liste.append(5) # -> [7, 5]
>>> liste.insert(1,12) # [7, 12, 5]
>>> liste[0] = 4 # -> [4, 12, 5]
>>> liste.remove(12) # [4, 5]
>>> liste.index(5) # affiche 1
1
>>> liste.extend([1, 2, 3]) # [4, 5, 1, 2, 3]
>>> del liste[3] # [4, 5, 1, 3]
```

```
>>> len(liste)
4
```

Les dictionnaires

Les listes et dictionnaires sont déclarés de façon similaire, à la différence qu'un dictionnaire utilise des **accolades** au lieu des crochets, et qu'il faut déclarer les associations clé-valeur :

```
Type "help", "copyright", "credits" or "license" for more information.
>>> comptes = {"Georges Dupont": 10000, "Luc Martin": 150, "Lucas Anderson": 300
, "Alexandre Petit": 1800.74}
>>> print(comptes["Luc Martin"]) # -> 150
150
>>>
```

```
>>> comptes['Georges Dupont'] -= 2000 # je soustrais 2000 au compte de Georges
>>> comptes['Cyril Andreje'] = 1000 # j'ajoute un nouvel individu dans mon dicti
onnaire
>>> print(comptes['Cyril Andreje']) # j'affiche la valeur du compte de Cyril
1000
```

```
>>> comptes.pop('Luc Martin') # supprime Luc Martin de notre dictionnaire
150
>>> print(comptes)
{'Georges Dupont': 8000, 'Lucas Anderson': 300, 'Alexandre Petit': 1800.74, 'Cyr
il Andreje': 1000}
>>>
```

```
>>> print(comptes)
{'Georges Dupont': 8000, 'Lucas Anderson': 300, 'Alexandre Petit': 1800.74, 'Cyr
il Andreje': 1000}
>>> len(comptes) # -> 4
4
```

CONTRÔLEZ VOTRE CODE GRÂCE AUX STRUCTURES CONDITIONNELLES :

```
>>> nomUtilisateur = input('Quel est votre nom, svp ?')
Quel est votre nom, svp ?Amaria
```

```
>>>
>>> nom = input('Quel est ton nom, cher(e) inconnu(e) ?')
Quel est ton nom, cher(e) inconnu(e) ?
>>> if len(nom) > 0:
...     print("Hello", nom, "!")
... else:
...     print("Hello World !")
...
Hello World !
>>>
```

```
>>> nom = input('Quel est ton nom, cher(e) inconnu(e) ?')
Quel est ton nom, cher(e) inconnu(e) ?Amaria
>>> if len(nom) > 0:
...     print("Hello", nom, "!")
... else:
...     print("Hello World !")
...
Hello Amaria !
>>>
```

```
>>>
>>> meteo = "La météo est chouette !"
>>> meteo.startswith("La météo") # -> True
True
```

Les opérateurs de comparaison :

```
>>> 2 == 2 # -> True
True
>>> 2 == 3 # -> False
False
>>> 4 != 4 # -> False
False
>>> 4!= 5 # -> True
True
>>> 1 < 2 # -> True
True
>>> 1 < 1 # -> False
False
>>> 1 <= 1 # -> True
True
>>> 3 > 4 # -> False
False
>>> 5 > 4 # -> True
True
>>> 5 >= 4 # -> True
True
>>>
```

Les opérateurs logiques

Ces opérateurs vont vous permettre de mixer plusieurs valeurs booléennes : des valeurs booléennes spécifiques ou des résultats d'expression. Il y en a 3 :

- `and` : l'opérateur **ET**.
  Le résultat final est vrai seulement lorsque toutes les expressions/valeurs sont vraies. Par exemple : le résultat de `expression1 and expression2` sera à True seulement si `expression1` est vraie **ET** `expression2` est également vraie ;
- `or` : l'opérateur **OU**.
  Le résultat final est vrai lorsqu'au moins une des expressions/valeur est vraie. Par exemple : le résultat de `expression1 or expression2` sera à True si `expression1` est vraie **OU** `expression2` est vraie ;
- `not` : l'opérateur **N'EST PAS**.
  Cela inverse simplement le résultat de l'expression donnée. Par exemple, le résultat de `not(expression)` est vrai lorsque `expression` est faux.

```
>>>
>>> True and True # True
True
>>> True and False # False
False
>>> False and False # False
False
>>> True or False # True
True
>>> True or True # True
True
>>> False or False # False
False
>>> not(True) # False
False
>>> not(False) # True
True
>>>
```

```
>>> True and True and True # True
True
>>> True and True and False # False
False
>>> True or False or False # True
True
>>> False or False or False # False
False
>>>
```

Comme avec les opérations numériques, les opérateurs logiques respectent les priorités d'opérations : l'opérateur `not` est réalisé en premier, ensuite l'opérateur `and` puis l'opérateur `or`. Par exemple :

```
>>> False or True and True
True
>>> False or False or False # False
False
>>>
```

L'opérateur "in » :

```
>>>
>>> maListe = [4, 2, 3, 2, 10]
>>> maListeDeString = ["a", "b", "c", "d"]
>>> monString = "La météo est vraiment bien aujourd'hui !"
>>>
>>> 4 in maListe # True
True
>>> 0 in maListe # False
False
>>> 0 in maListeDeString # False
False
>>> "c" in maListeDeString # True
True
>>> "e" in maListeDeString # False
False
>>> "météo" in monString # True
True
>>> "vraiment" in monString # True
True
>>> "pluie ?" in monString # False
False
>>>
```

Gérez un enchaînement de conditions

Pour accorder un prêt, une banque se base (entre autres) sur l'état des comptes de ses utilisateurs. Par exemple, une règle de décision naïve pourrait être :

- si le client a plus de 10 000 € sur son compte, on lui attribue son prêt d'office ;
- s'il a entre 100 € et 10 000 €, on fait une étude de son dossier ;
- sinon on lui refuse.

```
>>> client = input("Quel est le montant de votre compte ?")
Quel est le montant de votre compte ?15000
>>> print(client)
15000
>>> type(client)
<class 'str'>
>>> # 15000 est considéré comme une chaine de caractère : il faut le transforme
r en integer
...
>>> client=int(client)
>>> type(client)
<class 'int'>
```

```
4 if compte >= 10000:
5     print("Prêt accordé !")
6 elif compte >= 100 and compte < 10000:
7     print("Prêt en cours de validation : à l'étude")
8 else:
9     print("Prêt refusé")
```

La boucle FOR :

Les boucles **for** vont être utilisées lorsque l'on sait par avance le nombre de fois où une action va être répétée.

```
>>> maliste = [7, 2, 4 , 10]
>>> for elt in maliste:
...     print(elt)
...
7
2
4
10
>>> # elt va prendre les valeurs successives des éléments de ma_liste
...
```

```
>>>
>>> monString = "Eléments"
>>>
>>> for elt in monString:
...     print(elt)
...
E
l
é
m
e
n
t
s
>>>
>>>
```

```
>>>
>>> for i in range(0, 5, 1):
...     print(i) # -> affiche de 0 à 4 par pas de 1 (fin - 1)
...
0
1
2
3
4
>>>
>>>
>>>
>>>
>>> for i in range(0, 5):
...     print(i) # -> affiche de 0 à 4 également (le pas par défaut est 1)
...
0
1
2
3
4
>>>
```

```
>>>
>>> for i in range(0, 5):
...     print(i) # -> affiche de 0 à 4 également (le pas par défaut est 1)
...
0
1
2
3
4
>>>
>>>
>>> for i in range(5):
...     print(i) # -> affiche de 0 à 4 également (le début par défaut est 0)
...
0
1
2
3
4
>>>
```

```
>>>
>>> for i in range(0, 5, 2):
...     print(i) # -> affiche 0, 2 puis 4
...
0
2
4
>>>
```

La boucle while (tant que) :

```
>>> nombreArbres = 0
>>>
>>> while nombreArbres < 10:
...     nombreArbres += 1
...     print("J'ai planté", nombreArbres, "arbres")
...
J'ai planté 1 arbres
J'ai planté 2 arbres
J'ai planté 3 arbres
J'ai planté 4 arbres
J'ai planté 5 arbres
J'ai planté 6 arbres
J'ai planté 7 arbres
J'ai planté 8 arbres
J'ai planté 9 arbres
J'ai planté 10 arbres
>>>
>>> print("J'ai une chouette forêt !")
J'ai une chouette forêt !
>>>
```

```
>>>
>>> panier = ["pomme", "orange", "banane"]
>>>
>>> for fruit in panier:
...     if fruit == "orange":
...         print("J'ai une", fruit, "!")
...         break
...
J'ai une orange !
>>>
```