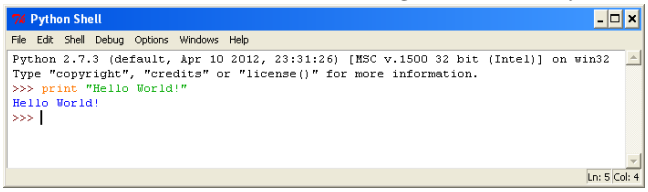
Chapitre 1 - Variables, types et opérateurs

Une variable est un espace mémoire dans lequel il est possible de stocker une valeur (une donnée). Ouvrir IDLE : Démarrer → Programmes → Python → IDLE (Python GUI)



0- Noms de variables

Le nom d'une variable s'écrit avec des lettres (non accentuées), des chiffres ou bien l'underscore _ Le nom d'une variable ne doit pas commencer par un chiffre. En langage Python, l'usage est de ne pas utiliser de lettres majuscules pour nommer les variables (celles-ci sont réservées pour nommer les classes). Exemple : `age, mon_age, temperature1` A éviter : `Age, AGE, MonAge, monAge, Temperature1`

1- Le type `int` (integer : nombres entiers)

Pour affecter (on dit aussi assigner) la valeur 17 à la variable nommée `age` :

```
>>> age = 17
```

La fonction `print` affiche la valeur de la variable :

```
>>> print(age)
17
```

La fonction `type()` retourne le type de la variable :

```
>>> print(type(age))
<type 'int'>
```

`int` est le type des nombres entiers.

```
>>> # ceci est un commentaire
>>> age = age + 1  # en plus court : age += 1
>>> print(age)
18
>>> age = age - 3  # en plus court : age -= 3
>>> print(age)
15
>>> age = age*2        # en plus court : age *= 2
>>> print(age)
30
```

```
>>> a = 6*3 - 20
>>> print(a)
```

```
-2
```

```
>>> b = 25
>>> c = a + 2*b
>>> print(b, c)          # ne pas oublier la virgule
25 48
```

L'opérateur `//` donne la division entière :

```
>>> 4//3  # 4 = 3 * 1 + reste
1
>>> tour = 450//360
>>> print(tour)
1
```

L'opérateur `%` donne le reste de la division (opération modulo) :

```
>>> angle = 450%360
>>> print(angle)
90
```

L'opérateur `**` donne la puissance :

```
>>> mo = 2**20
>>> print(mo)
1048576
>>> racine2 = 2**0.5
>>> print(racine2)
1.41421356237
```

2- Le type `float` (nombres en virgule flottante)

```
>>> b = 17.0     # le séparateur décimal est un point (et non une virgule)
>>> print(b)
17.0
>>> print(type(b))
<type 'float'>
```

```
>>> c = 14.0/3.0
>>> print(c)
4.66666666667
```

```
>>> c = 14.0//3.0    # division entière
>>> print(c)
4.0
```

Attention : avec des nombres entiers, l'opérateur `/` renvoie généralement un flottant :

```
>>> c = 14/3
>>> print(c)
```

```
4.666666666666667
```

Notation scientifique :

```
>>> a = -1.784892e4
>>> print(a)
-17848.92
```

Les fonctions mathématiques Pour utiliser les fonctions mathématiques, il faut commencer par importer le module `math` :

```
>>> import math
```

```
>>> dir(math)
['__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf',
'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum',
'gamma', 'hypot', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

Pour appeler une fonction d'un module, la syntaxe est la suivante : **module.fonction(arguments)** Pour accéder à une donnée d'un module : **module.data**

```
>>> print(math.pi)       # donnée pi du module math (nombre pi)
3.14159265359
>>> print(math.sin(math.pi/4.0)) # fonction sin() du module math (sinus)
0.707106781187
>>> print(math.sqrt(2.0))  # fonction sqrt() du module math (racine carrée)
1.41421356237
>>> print(math.sqrt(-5.0))
Traceback (most recent call last):
    print(math.sqrt(-5.0))
ValueError: math domain error
>>> print(math.exp(-3.0))  # fonction exp() du module math (exponentielle)
0.0497870683679
>>> print(math.log(math.e))    # fonction log() du module math (logarithme népérien)
1.0
```

3- Le type str (string : chaîne de caractères)

```
>>> nom = 'Dupont'   # entre apostrophes
>>> print(nom)
Dupont
>>> print(type(nom))
<type 'str'>
>>> prenom = "Pierre"  # on peut aussi utiliser les guillemets
>>> print(prenom)
Pierre
```

```
>>> print(nom, prenom)     # ne pas oublier la virgule
Dupont Pierre
```

La concaténation désigne la mise bout à bout de plusieurs chaînes de caractères.
La concaténation utilise l'opérateur `+`

```
>>> chaine = nom + prenom    # concaténation de deux chaînes de caractères
>>> print(chaine)
DupontPierre
>>> chaine = prenom + nom  # concaténation de deux chaînes de caractères
>>> print(chaine)
PierreDupont
>>> chaine = prenom + ' ' + nom
>>> print(chaine)
Pierre Dupont
>>> chaine = chaine + ' 18 ans'    # en plus court : chaine += ' 18 ans'
>>> print(chaine)
Pierre Dupont 18 ans
```

La fonction `len()` retourne la longueur (length) de la chaîne de caractères :

```
>>> len("abc")
3
```

Indexage et slicing :

```
>>> print(chaine[0])     # premier caractère (indice 0)
P
>>> print(chaine[1])   # deuxième caractère (indice 1)
i
>>> print(chaine[1:4]) # slicing
ier
>>> print(chaine[2:])  # slicing
erre Dupont 18 ans
>>> print(chaine[-1])  # dernier caractère (indice -1)
s
>>> print(chaine[-6:]) # slicing
18 ans
```

En résumé :

```
lettre chaine          :   M   u   r   i   e   l
position depuis 0      :   0   1   2   3   4   5
position depuis la fin :  -6  -5  -4  -3  -2  -1
```

```
>>> chaine = 'Aujourd'hui'
SyntaxError: invalid syntax
>>> chaine  = 'Aujourd\'hui'       # séquence d'échappement \'
>>> print(chaine)
Aujourd'hui
```

```
>>> chaine  = "Aujourd'hui"
>>> print(chaine)
Aujourd'hui
```

La séquence d'échappement `\n` représente un saut ligne :

```
>>> chaine = 'Premiere ligne\nDeuxieme ligne'
>>> print(chaine)
Premiere ligne
Deuxieme ligne
```

Plus simplement, on peut utiliser les triples guillemets (ou les triples apostrophes) pour encadrer une chaîne définie sur plusieurs lignes :

```
>>> chaine = """Premiere ligne
Deuxième ligne"""
>>> print(chaine)
Premiere ligne
Deuxieme ligne
```

On ne peut pas mélanger les serviettes et les torchons (ici type `str` et type `int`) :

```
>>> chaine = '17.45'
>>> print(type(chaine))
<type 'str'>
>>> chaine = chaine + 2
TypeError: cannot concatenate 'str' and 'int' objects
```

La fonction `float()` permet de convertir un type `str` en type `float`

```
>>> nombre = float(chaine)
>>> print(nombre)
17.45
>>> print(type(nombre))
<type 'float'>
>>> nombre = nombre + 2        # en plus court : nombre += 2
>>> print(nombre)
19.45
```

La fonction `input()` lance une invite de commande (en anglais : prompt) pour saisir une chaîne de caractères.

```
>>> # saisir une chaîne de caractères et valider avec la touche Enter
>>> chaine = input("Entrer un nombre : ")
Entrer un nombre : 14.56
>>> print(chaine)
14.56
>>> print(type(chaine))
<type 'str'>
>>> nombre = float(chaine) # conversion de type
```

```
>>> print(nombre**2)
211.9936
```

4- Le type `list` (liste)

Une liste est une structure de données. Le premier élément d'une liste possède l'indice (l'index) 0. Dans une liste, on peut avoir des éléments de plusieurs types.

```
>>> infoperso = ['Pierre', 'Dupont', 17, 1.75, 72.5]
>>> # la liste infoperso contient 5 éléments de types str, str, int, float et float
>>> print(type(infoperso))
<type 'list'>
>>> print(infoperso)
['Pierre', 'Dupont', 17, 1.75, 72.5]
>>> print('Prénom : ', infoperso[0])       # premier élément (indice 0)
Prénom :  Pierre
>>> print('Age : ', infoperso[2])      # le troisième élément a l'indice 2
Age :  17
>>> print('Taille : ', infoperso[3])       # le quatrième élément a l'indice 3
Taille :  1.75
```

La fonction `range()` crée une liste d'entiers régulièrement espacés :

```
>>> maliste = range(10)
>>> print(maliste)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print(type(maliste))
<type 'list'>
```

```
>>> maliste = range(1,10,2)  # range(début,fin non comprise,intervalle)
>>> print(maliste)
[1, 3, 5, 7, 9]
>>> print(maliste[2])      # le troisième élément a l'indice 2
5
```

On peut créer une liste de listes, qui s'apparente à un tableau à 2 dimensions (ligne, colonne) : 0 1 2 10 11 12 20 21 22

```
>>> maliste = [[0, 1, 2], [10, 11, 12], [20, 21, 22]]
>>> print(maliste[0])
[0, 1, 2]
>>> print(maliste[0][0])
0
>>> print(maliste[2][1])       # élément à la troisième ligne et deuxième colonne
21
>>> maliste[2][1] = 69     # nouvelle affectation
>>> print(maliste
[[0, 1, 2], [10, 11, 12], [20, 69, 22]]
```

5- Le type `bool` (booléen)

Deux valeurs sont possibles : `True` et `False`

```
>>> choix = True
>>> print(type(choix))
<type 'bool'>
```

Les opérateurs de comparaison :

Opérateur	Signification	Remarques
<	strictement inférieur	
<=	inférieur ou égal	
>	strictement supérieur	
>=	supérieur ou égal	
==	égal	Attention : deux signes `==`
!=	différent	

```
>>> b = 10
>>> print(b > 8)
True
>>> print(b == 5)
False
>>> print(b != 10)
False
>>> print(0 <= b <= 20)
True
```

Les opérateurs logiques : `and, or, not`

```
>>> note = 13.0
>>> mention_ab = note >= 12.0 and note < 14.0 # ou bien : mention_ab = 12.0 <= note < 14.0
>>> print(mention_ab)
True
>>> print(not mention_ab)
False
>>> print(note == 20.0 or note == 0.0)
False
```

L'opérateur `in` s'utilise avec des chaînes (type `str`) ou des listes (type `list`) :

```
>>> chaine = 'Bonsoir'
>>> # la sous-chaîne 'soir' fait-elle partie de la chaîne 'Bonsoir' ?
>>> resultat = 'soir' in chaine
>>> print(resultat)
True
>>> print('b' in chaine)
False
```

```
>>> maliste = [4, 8, 15]
>>> # le nombre entier 9 est-il dans la liste ?
>>> print(9 in maliste)
False
>>> print(8 in maliste)
True
>>> print(14 not in maliste)
True
```

6- Le type `dict` (dictionnaire)

Un dictionnaire stocke des données sous la forme **clé -> valeur** Une clé est unique et n'est pas nécessairement un entier (comme c'est le cas de l'indice d'une liste).

```
>>> moyennes = {'math': 12.5, 'anglais': 15.8}   # entre accolades
>>> print(type(moyennes))
<type 'dict'>
>>> print(moyennes['anglais'])     # entre crochets
15.8
>>> moyennes['anglais'] = 14.3     # nouvelle affectation
>>> print(moyennes)
{'anglais': 14.3, 'math': 12.5}
>>> moyennes['sport'] = 11.0       # nouvelle entrée
>>> print(moyennes)
{'sport': 11.0, 'anglais': 14.3, 'math': 12.5}
```

7- Autres types

Nous avons vu les types les plus courants. Il en existe bien d'autres :

- complex (nombres complexes, par exemple 1+2.5j)
- tuple (structure de données)
- set (structure de données)
- file (fichiers)
- ...

8- Programmation Orientée Objet (POO)

Python est un langage de programmation **orienté objet** (comme les langages C++, Java, PHP, Ruby...). Une variable est en fait un **objet** d'une certaine **classe**. Par exemple, la variable `amis` est un objet de la classe `list`. On dit aussi que la variable `amis` est une **instance** de la classe `list`. L'**instanciation** (action d'instancier) est la création d'un objet à partir d'une classe (syntaxe : **NouvelObjet = NomdelaClasse(arguments)**) :

```
>>> # instanciation de l'objet amis de la classe list
>>> amis = ['Nicolas', 'Julie']    # ou bien : amis = list(['Nicolas', 'Julie'])
```

```
>>> print(type(amis))
<type 'list'>
```

Une classe possède des fonctions que l'on appelle **méthodes** et des données que l'on appelle **attributs**. La méthode `append()` de la classe `list` ajoute un nouvel élément en fin de liste :

```
>>> # instanciation d'une liste vide
>>> amis = []          # ou bien : amis = list()
>>> amis.append('Nicolas') # synthase générale : objet.méthode(arguments)
>>> print(amis)
['Nicolas']
>>> amis.append('Julie')   # ou bien : amis = amis + ['Julie']
>>> print(amis)
['Nicolas', 'Julie']
>>> amis.append('Pauline')
>>> print(amis)
['Nicolas', 'Julie', 'Pauline']

>>> amis.sort()         # la méthode sort() trie les éléments
>>> print(amis)
['Julie', 'Nicolas', 'Pauline']

>>> amis.reverse()       # la méthode reverse() inverse la liste des éléments
>>> print(amis)
['Pauline', 'Nicolas', 'Julie']
```

La méthode `lower()` de la classe `str` retourne la chaîne de caractères en casse minuscule :

```
>>> # la variable chaine est une instance de la classe str
>>> chaine = "BONJOUR"     # ou bien : chaine = str("BONJOUR")
>>> chaine2 = chaine.lower()   # on applique la méthode lower() à l'objet chaine
>>> print(chaine2)
bonjour
>>> print(chaine)
BONJOUR
```

La méthode `pop()` de la classe `dict` supprime une clé :

```
>>> # instanciation de l'objet moyennes de la classe dict
>>> moyennes = {'sport': 11.0, 'anglais': 14.3, 'math': 12.5}
>>> # ou : moyennes = dict({'sport': 11.0, 'anglais': 14.3, 'math': 12.5})
>>> moyennes.pop('anglais')
14.3
>>> print(moyennes)
{'sport': 11.0, 'math': 12.5}

>>> print(moyennes.keys())   # la méthode keys() retourne la liste des clés
['sport', 'math']
```

```
>>> print(moyennes.values())     # la méthode values() retourne la liste des valeurs
[11.0, 12.5]
```

Exercices

Exercice 1.1 * Afficher la taille en octets et en bits d'un fichier de 536 ko. On donne : 1 ko (1 kilooctet) = 2^{10} octets !!! 1 octet = 1 byte = 8 bits

Exercice 1.2 * Le numéro de sécurité sociale est constitué de 13 chiffres auquel s'ajoute la clé de contrôle (2 chiffres). Exemple : 1 89 11 26 108 268 91 La clé de contrôle est calculée par la formule : 97 - (numéro de sécurité sociale modulo 97) Retrouver la clé de contrôle de votre numéro de sécurité sociale. Quel est l'intérêt de la clé de contrôle ?

Exercice 1.3 * Afficher la valeur numérique de $\sqrt{}$(4,6^3 - 15/16) Comparer avec votre calculette.

Exercice 1.4 * A partir des deux variables `prenom` et `nom`, afficher les initiales (par exemple LM pour Léa Martin).

Exercice 1.5 ** L'identifiant d'accès au réseau du lycée est construit de la manière suivante : initiale du prénom puis les 8 premiers caractères du nom (le tout en minuscule). Exemple : Alexandre Lecouturier → alecoutur A partir des deux variables `prenom` et `nom`, construire l'identifiant.

Exercice 1.6 ** On donne un extrait des logins d'accès au réseau du lycée :

```
alecoutur  Huz4
lmartin    monty
fsincere   gnugpl
```

1) Créer une variable de type `dict` qui contient les couples identifiant - mot de passe ci-dessus. 2) La saisie du login fournit deux variables `identifiant` et `motdepasse` : une pour l'identifiant et l'autre pour le mot de passe. Construire une variable booléenne qui donne `True` en cas d'identification correcte, et `False` dans le cas contraire : lmartin monty → True alecoutur fqsdf → False martin monty → False (ce cas est plus compliqué à traiter)

QCM

QCM sur les types int, float et str Source : Fabrice Sincère -