Syntaxe de base de Python

La syntaxe du langage de programmation Python est l'ensemble des règles qui définissent la manière dont un programme Python sera écrit et interprété (à la fois par le système d'exécution et par les lecteurs humains).Le langage Python présente de nombreuses similitudes avec des langages tels que Perl, C et Java. Le langage Python présente de nombreuses similitudes avec des langages tels que Perl, C et Java. Il existe toutefois des différences marquées entre ces langages.

Philosophie de conception

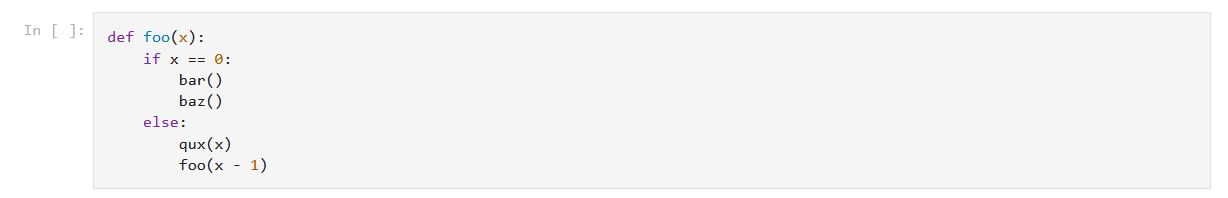
Python a été conçu pour être un langage très lisible. Sa présentation visuelle est relativement épurée et il utilise fréquemment des mots-clés anglais là où d'autres langages utilisent la ponctuation. Python vise à être simple et cohérent dans la conception de sa syntaxe, ce qui est résumé dans le mantra "Il devrait y avoir une - et de préférence une seule - façon évidente de le faire", tiré de "The Zen of Python" (Le zen de Python).

Avec Python, oubliez les points-virgules et les accolades pour délimiter les blocs. Python travaille par ligne et la hiérarchisation se fait par indentation.

Nous le verrons plus tard avec les conditions et les fonctions.

Indentation

Python utilise des espaces blancs pour délimiter les blocs de flux de contrôle (suivant la règle du hors-jeu). Python emprunte cette caractéristique à son prédécesseur ABC : au lieu de la ponctuation ou des mots-clés, il utilise l'indentation pour indiquer la fin d'un bloc.

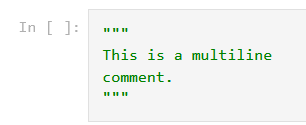
Dans les langages dits "à format libre" - qui utilisent la structure en blocs dérivée d'ALGOL - les blocs de code sont séparés par des accolades ({ }) ou des mots-clés. Dans la plupart des conventions de codage de ces langages, les programmeurs indentent conventionnellement le code à l'intérieur d'un bloc, afin de le distinguer visuellement du code environnant (prettyprinting).

Commentaires en Python

Un signe dièse (#) qui ne se trouve pas à l'intérieur d'une chaîne littérale commence un commentaire. Tous les caractères après le # et jusqu'à la fin de la ligne physique font partie du commentaire et l'interpréteur Python les ignore.



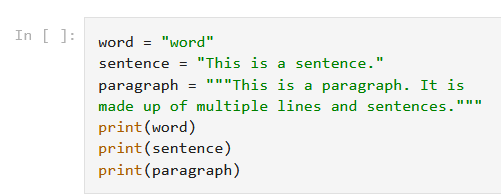
La chaîne suivante entre trois guillemets est également ignorée par l'interpréteur Python et peut être utilisée comme un commentaire sur plusieurs lignes :



Les guillemets en Python

Python accepte les guillemets simples ('), doubles (") et triples (''' ou """) pour désigner les chaînes de caractères littérales, à condition que le même type de guillemet commence et termine la chaîne.

Les guillemets triples sont utilisés pour étendre la chaîne sur plusieurs lignes. Par exemple, tous les éléments suivants sont légaux :



Mots réservés

La liste suivante présente les mots-clés Python. Ce sont des mots réservés et vous ne pouvez pas les utiliser comme constantes ou variables ou tout autre nom d'identifiant. Tous les mots-clés Python ne contiennent que des lettres minuscules.

* and
* exec
* not
* assert
* finally
* or
* break
* for
* pass
* class
* from
* print
* continue
* global
* raise
* def
* if
* return
* del
* import
* try
* elif
* in
* while
* else
* is
* with
* except
* lambda
* yield

Très bien, vous pouvez passer à autre chose !

2.Variables et types de données

Les variables sont utilisées pour stocker temporairement une valeur dans la mémoire de l'ordinateur. Nous pouvons ensuite l'utiliser plus tard, lorsque nous en avons besoin dans le programme. Nous pouvons comparer cela à une petite boîte dans laquelle nous pourrions stocker des informations.

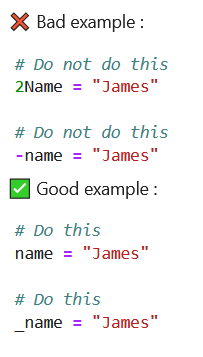
Comme Python est un langage à typage dynamique, ce sont les valeurs Python, et non les variables, qui portent le type. Cela a des implications sur de nombreux aspects du fonctionnement du langage.

Toutes les variables en Python contiennent des références à des objets, et ces références sont transmises aux fonctions ; une fonction ne peut pas modifier la valeur des références des variables dans sa fonction appelante (mais voir ci-dessous pour les exceptions). Certaines personnes (y compris Guido van Rossum lui-même) ont appelé ce système de passage de paramètres "appel par référence d'objet". Une référence d'objet signifie un nom, et la référence passée est un "alias", c'est-à-dire une copie de la référence au même objet, comme en C/C++. La valeur de l'objet peut être modifiée dans la fonction appelée avec l'"alias", par exemple :

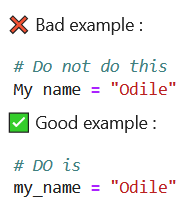
Conventions d'appellation et d'écriture du code

Comme dans de nombreux langages de programmation, certaines conventions doivent être respectées pour nommer une variable.

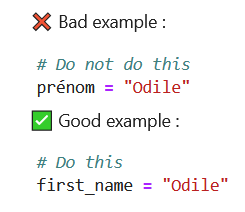
1. Le nom de la variable doit commencer par une lettre ou un trait de soulignement. La variable ne peut pas commencer par un chiffre ou un trait d'union.



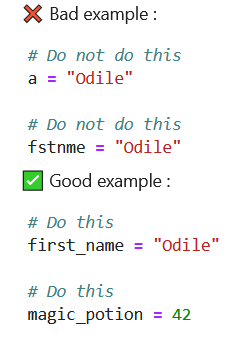
2. Ne mettez jamais d'espace entre les mots.



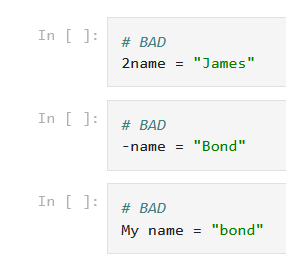
3. Pas d'accents sur les noms des variables. Utiliser uniquement l'anglais



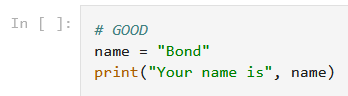
4. Donnez toujours un nom explicite à la variable.



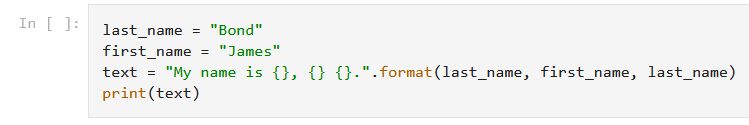
Essayez-le par vous-même en cliquant sur le bouton "Exécuter".



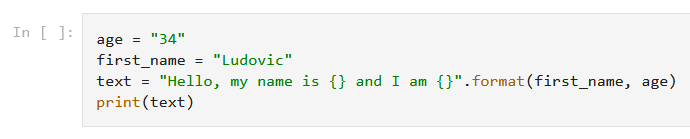
La fonction print() permet d'afficher le résultat.



Pour formater le texte, vous pouvez utiliser la méthode format() de la classe string

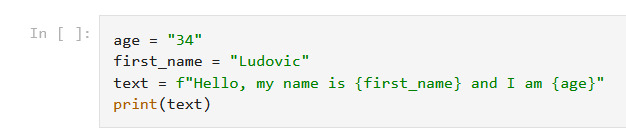


Autre exemple. Remplacez la valeur de la variable âge par votre âge et la variable prénom par votre prénom.



Il est également possible de le faire avec une syntaxe plus lisible : (C'est une fonctionnalité ajoutée dans Python 3.6, donc si vous utilisez une version plus ancienne, cela ne fonctionnera pas).

C'est une meilleure pratique de le faire de cette façon, alors faites-le toujours si vous le pouvez.



Types de données

Python étant un langage de haut niveau, il dispose d'un typage dynamique des variables.

Par dynamique, il faut comprendre que c'est l'ordinateur qui se charge de définir quel type de variable doit être utilisé.

Pour être tout à fait précis, ce n'est pas la variable qui est typée (contrairement à Java) mais son contenu.

En Java, on déclare une variable comme ceci :

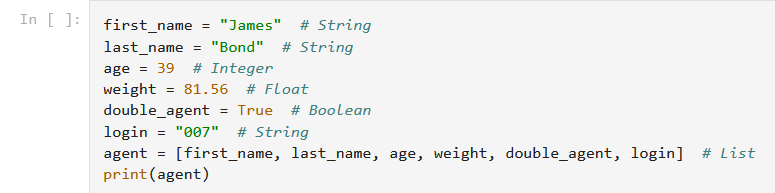
String firstName = "James"

Nous définissons nous-mêmes le type de la variable.

Avec Python, nous déclarons une variable comme ceci :

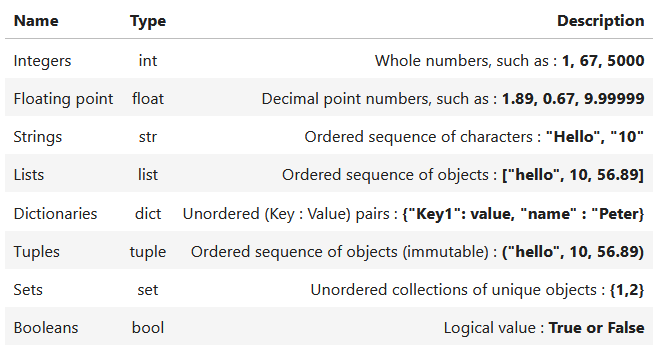
prénom = "James"

C'est donc Python qui définira le type à utiliser.

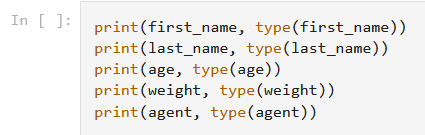


1. Notez que True et False s'écrivent avec la première lettre en majuscule.
2. login est une chaîne de caractères.
3. La liste est comme un tableau, mais vous pouvez y stocker des valeurs de différents types.

Voici une liste non exhaustive des types que nous utilisons le plus souvent. Nous verrons plus tard les tuples, les dictionnaires et les ensembles.



Il existe une fonction native de Python qui vous permet de savoir quel type de données vous avez. Il s'agit de la fonction type()



**Vous avez terminé ? Entraînez-vous avec ces exercices.**

**1. Créez un nom de variable contenant la valeur "Alan Turing".**

**2. Créez une variable age contenant la valeur 42.**

**3. Créez une variable person qui contient une liste avec les valeurs suivantes : name, age et "mathematician"**

**4. Créez une variable text qui contient "Hello, my name is Alan Turing, I am 42 years old and I am a mathematician" (Bonjour, je m'appelle Alan Turing, j'ai 42 ans et je suis mathématicien). Pour ce faire, utilisez la méthode format() et la variable person.**

**5. Créez une variable age\_type qui contient le type de la variable age.**

**Test**

**Le morceau de code suivant permet de tester que vous avez correctement effectué les exercices ci-dessus. S'il échoue, cela signifie qu'une partie de votre code n'est pas correcte. Lisez attentivement les messages d'erreur de chaque test.**

**Si la syntaxe du code ci-dessous est confuse (class,self, etc...), ne vous inquiétez pas, vous le verrez bientôt. La seule chose que vous devez savoir est que la méthode assertEqual vérifie si l'argument de droite correspond à celui de gauche, par exemple pour assertEqual(name, "Alan Turing") elle vérifie que name est égal à "Alan Turing".**

**Tout bon développeur teste son code. Nous allons donc prendre de bonnes habitudes maintenant. Ne modifiez pas le code, exécutez-le.**

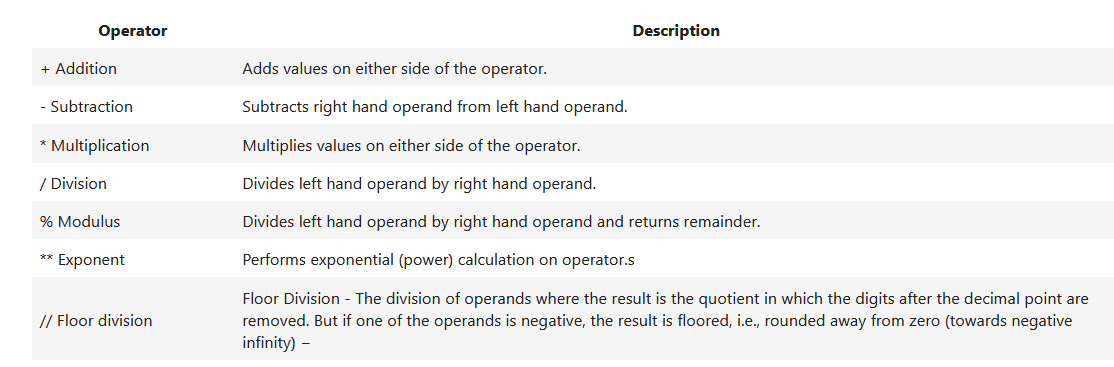
****

**Tout va bien ? Vous pouvez passer à autre chose.**

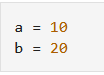
3. Opérateurs de base

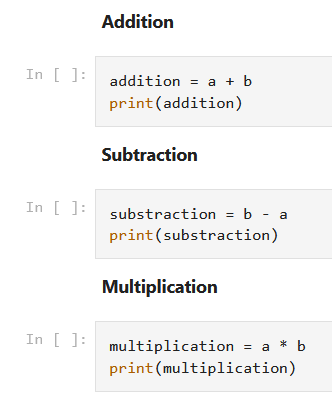
**Opérateurs arithmétiques**

**Les opérateurs arithmétiques effectuent divers calculs arithmétiques tels que l'addition, la soustraction, la multiplication, la division, le module, l'exposant, etc. Il existe plusieurs méthodes de calcul arithmétique en Python : vous pouvez utiliser la fonction eval(), déclarer des variables et effectuer le calcul, ou appeler des fonctions.**

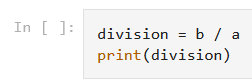
****

**Essayons-le. Nous créons deux variables et leur attribuons deux valeurs.**

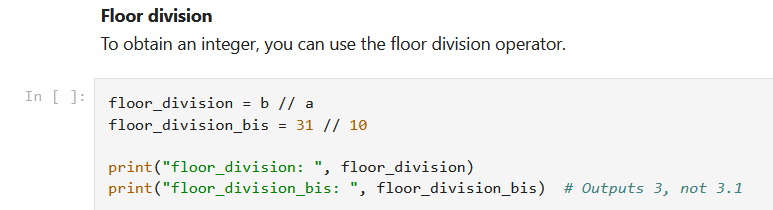
****

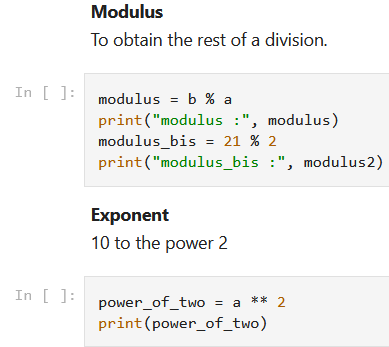
****

**Notez que dans ce cas, python renvoie un flottant.**

****

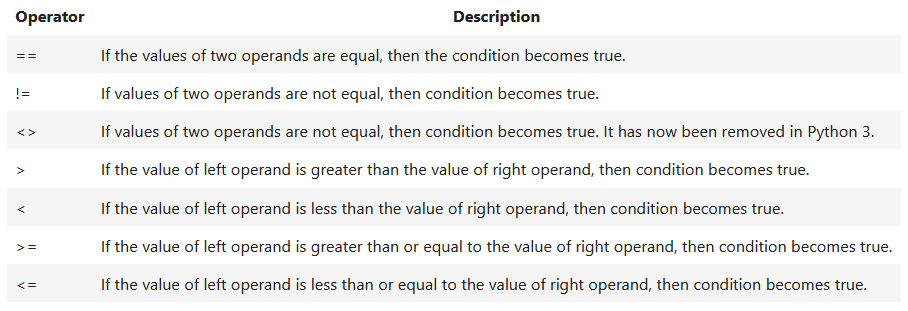
**Pour obtenir un nombre entier, vous pouvez utiliser l'opérateur de division floor.**

****

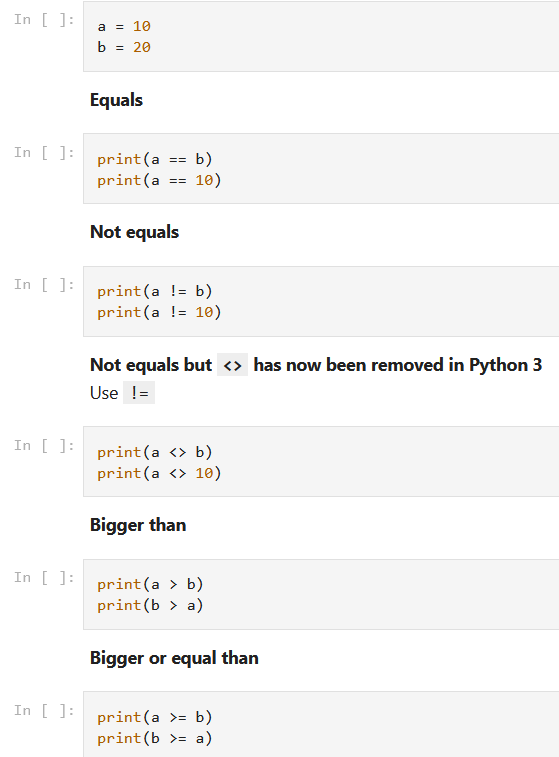
****

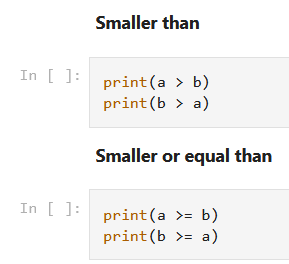
**Opérateurs de comparaison**

**Ces opérateurs comparent les valeurs de part et d'autre des opérandes et déterminent la relation entre elles. Ils sont également appelés opérateurs relationnels. Les différents opérateurs de comparaison sont ( ==, != , <>, >,<=, etc).**

****

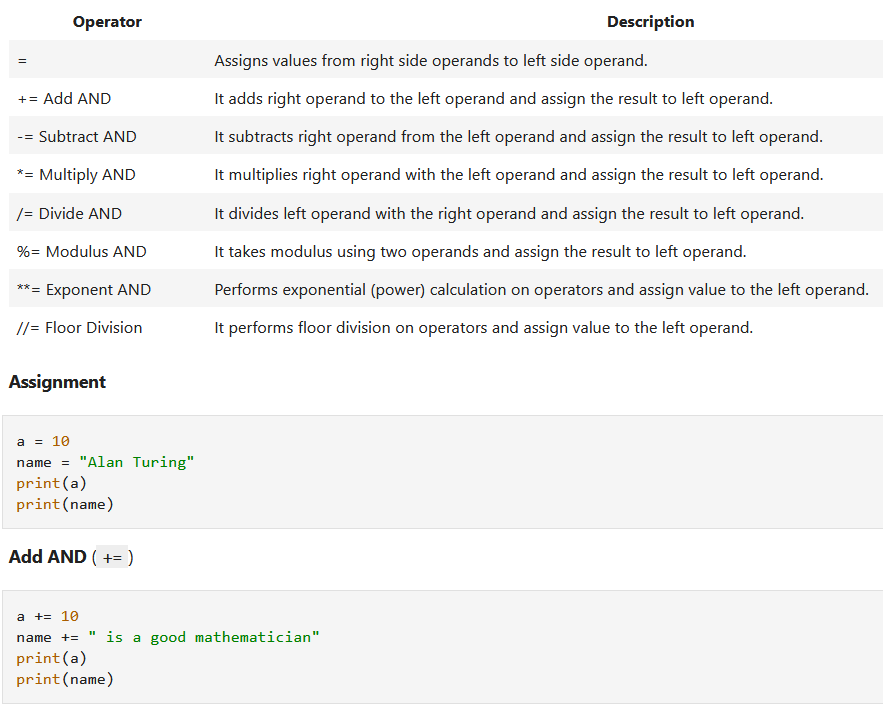
**Essayons-le. Nous créons deux variables et leur attribuons deux valeurs.**

****

****

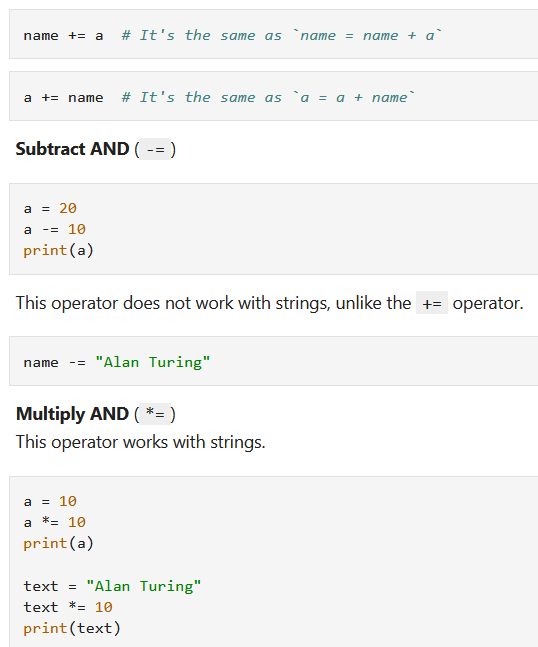
**Opérateurs d'affectation**

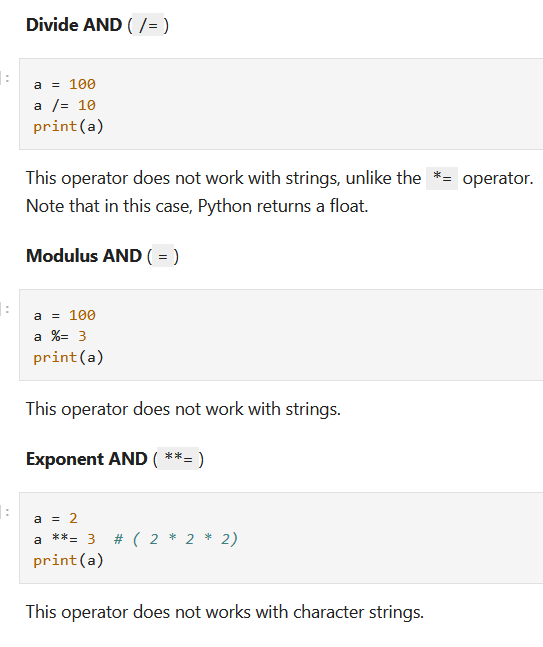
**Les opérateurs d'affectation de Python sont utilisés pour affecter la valeur de l'opérande de droite à l'opérande de gauche. Les différents opérateurs d'affectation utilisés en Python sont (+=, - = , \*=, /= , etc.).**

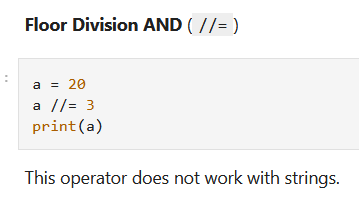
****

**❗ Mais attention, on ne peut ajouter que des variables de même type**

**La variable a est de type entier et la variable nom est de type chaîne de caractères.**

****

****

****

4.Conditions

**Cette notion est l'une des plus importantes en programmation. Elle introduit la logique et permet de créer des algorithmes.**

**L'idée est d'exprimer ceci :**

**- Si la variable est égale à quelque chose alors :**

**- Faire une action**

**- Si ce n'est pas le cas, alors :**

**- Effectuer une autre action**

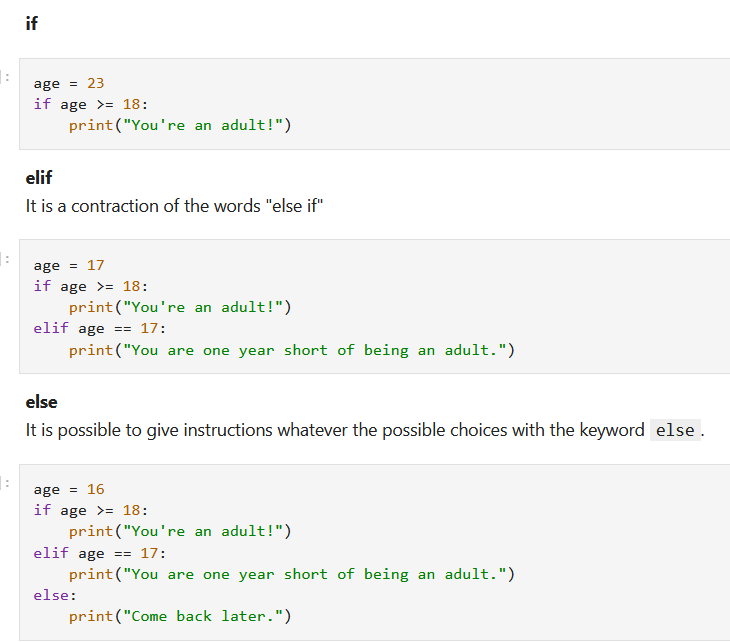
**Python utilise des variables booléennes pour évaluer les conditions. Les valeurs booléennes True et False sont renvoyées lorsqu'une expression est comparée ou évaluée. Par exemple :**

****

**Notez que l'affectation de variables se fait à l'aide d'un seul opérateur égal =, tandis que la comparaison entre deux variables se fait à l'aide du double opérateur égal ==. L'opérateur non égal est défini comme !=.**

**Python utilise l'indentation pour définir les blocs de code, au lieu des crochets. L'indentation standard de Python est de 4 espaces, bien que les tabulations et toute autre taille d'espace puissent être utilisées, à condition qu'elles soient cohérentes. Notez que les blocs de code n'ont pas besoin de terminaison.**

**Voici un exemple d'utilisation de l'instruction if de Python à l'aide de blocs de code :**

****

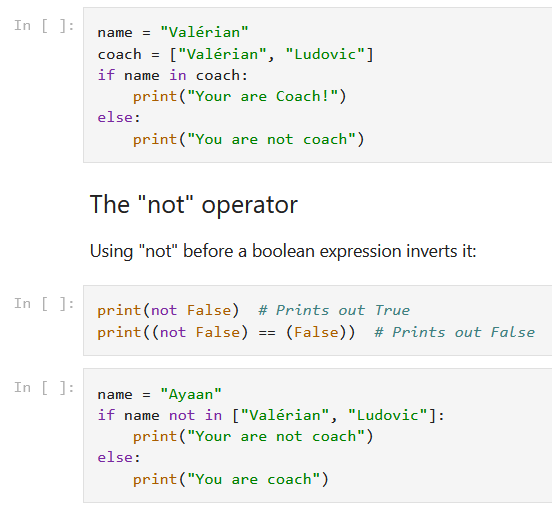
**Opérateurs logiques**

**Les opérateurs logiques en Python qui sont utilisés pour les déclarations conditionnelles sont vrai ou faux. Les opérateurs logiques en Python sont AND, OR et NOT. Les opérateurs logiques sont soumis aux conditions suivantes. Les opérateurs booléens "and" et "or" permettent de construire des expressions booléennes complexes, par exemple :**

****

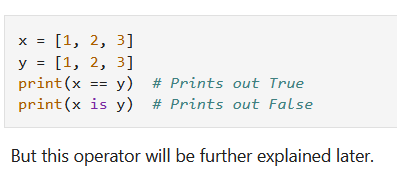
**L'opérateur "in**

**L'opérateur "in" peut être utilisé pour vérifier si un objet spécifié existe dans un conteneur d'objets itérables, tel qu'une liste :**

****

**L'opérateur "is**

**Contrairement à l'opérateur d'égalité double "==", l'opérateur "is" ne fait pas correspondre les valeurs des variables, mais les instances elles-mêmes. Par exemple, l'opérateur "is" ne correspond pas aux valeurs des variables, mais aux instances elles-mêmes :**

****

**Exercice : Les bases de la syntaxe python**

**Dans certains de ces exercices, vous aurez à utiliser des méthodes que vous n'avez pas encore vues. Dans ce cas, cherchez sur google un exemple d'utilisation. Prenez les bonnes habitudes pour ne pas être bloqué**

**Exécutez toutes les cellules avant de commencer les tests.**

**1. Créez une variable âge qui contient la valeur 32**

**2. Ajouter 10 à la variable age**

**3. Créer une variable divAge et lui attribuer la valeur de l'âge divisé par 7.**

**Attention, il doit s'agir d'un entier.**

**4. Créez une variable textDiv qui contient la chaîne de caractères " 42 divisé par 7 égale 6 ".**

**Vous devez utiliser les variables age et divAge**

**5. Créez une variable restDiv qui contient le reste de la variable age divisé par 7.**

**6. Créez une variable expDiv qui contient la valeur de restDiv à la puissance 3.**

**7. Ecrivez un programme qui entre un entier et affiche ensuite la valeur entrée et son type.**

**8. Utilisez des variables pour représenter le prix des matériaux.**

**2 bouteilles de lait à 0,45 € chacune**

**3 bouteilles de cidre brut à 3,85 € chacune**

**1 sac de farine à 0,9**

**1 paquet de beurre à 0,77 €.**

**1 pot de nutella à 1,87 €.**

**Calculez la somme de tous les articles et stockez-la dans une variable orderPrice.**

**Créez une variable allowanceMoney d'une valeur de 20, puis créez un algorithme qui calcule l'argent disponible en soustrayant le prix de la commande.**

**S'il y a suffisamment d'argent, enregistrez la phrase suivante dans la variable message et soustrayez la dépense de l'argent de poche :**

**message = "Vous avez dépensé" + prix de la commande + "il vous reste" + argent de poche**

**S'il n'y a pas assez d'argent, enregistrez la phrase suivante dans la variable message :**

**message= "Sorry you're missing amountMissing euros"**

**S'il ne reste rien, enregistrer la phrase suivante dans la variable message :**

**message = "Vous êtes fauché !"**

**message = ""**

**# Déclarer et assigner la variable `allowanceMoney`**

**# Alors**

**9. Ecrivez un programme qui vous demande d'entrer 2 valeurs et qui affiche la plus petite de ces 2 valeurs**

**10. Ecrivez un script qui vous demande d'entrer 2 chaînes de caractères et qui affiche la plus grande des 2 chaînes (celle qui a le plus de caractères).**

**11. Ecrivez un script qui convertit des euros en dollars.**

**Le programme commencera par demander à l'utilisateur d'indiquer par un caractère 'E' ou '$' selon la devise du montant qu'il introduit.**

**Ensuite, le programme vous demandera d'entrer le montant et affichera la conversion.**

**12. Vérifiez si le nom de la variable se trouve dans la liste des étudiantsTuring. (Sans faire de boucle)**

**Si le nom est dans la liste, afficher "Vous êtes chez Turing".**

**Sinon, afficher : "Vous ne faites pas partie des Turing".**

**étudiantsTuring = ["Redouane", "Justine", "Ruben", "Edouard"]**

**nom = "Julie"**

**# Ajoutez votre code ici**

**13. Calculez le volume d'une sphère en utilisant la formule (4π/3) x R³. Le rayon est de 10.**

**Enregistrez le résultat dans une variable "volume".**

5.Fonctions intégrées

**L'interpréteur Python intègre un certain nombre de fonctions et de types qui sont toujours disponibles.**

**Nous les avons déjà utilisés. Souvenez-vous de la fonction print() qui permet d'afficher du texte.**

**print("Hello World")**

**La méthode input() permet à l'utilisateur de saisir du texte.**

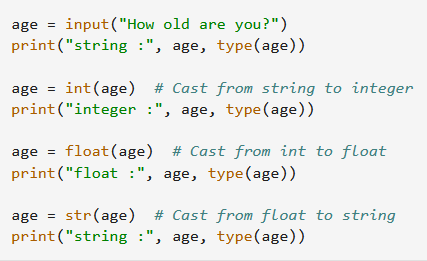
**age = input("Quel âge avez-vous ?")**

**print("Votre âge", age)**

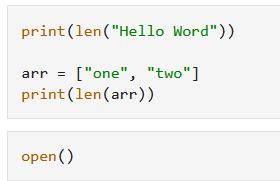
**La valeur d'entrée sera toujours une chaîne de caractères. Nous utiliserons une autre fonction native pour le vérifier.**

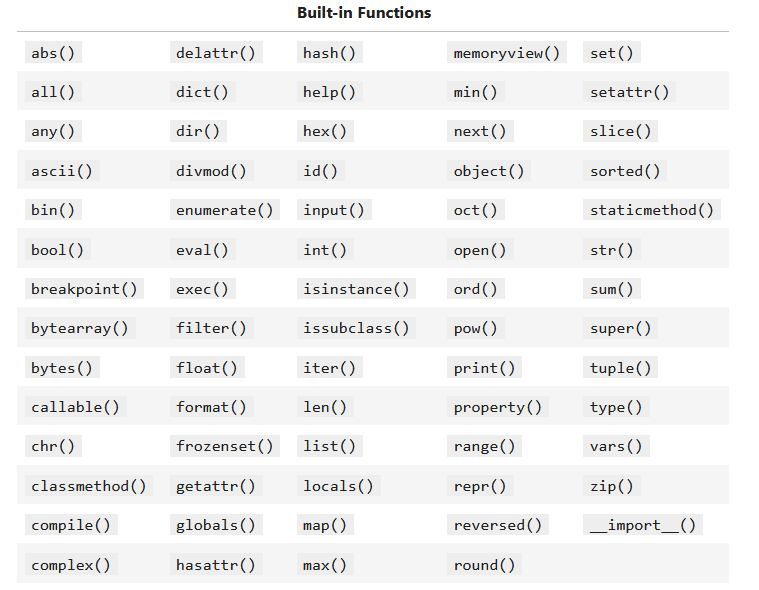
**print(type(age))**

**Fonctions qui permettent le cast : str(), int (), float(), etc…**

****

**La fonction len() permet de connaître le nombre d'éléments d'une liste ou d'une chaîne de caractères**

**.**

****

**Exercice : fonctions intégrées**

**Utilisez les fonctions natives de python pour faire ces exercices.**

**1. Créez une variable count\_alpha qui contient le nombre de caractères contenus dans la chaîne "Hello world !".**

**2. Créer une variable count\_float et convertir la variable count\_alpha en float.**

**3. Arrondissez la valeur de la variable pi à 2 décimales (0,01) et enregistrez-la dans une variable round\_pi. Vous pouvez obtenir pi à partir de la bibliothèque mathématique.**

**En Python, il existe plusieurs façons d'arrondir une valeur à un certain nombre de décimales. Vous pouvez utiliser la fonction round(), la fonction format() ou le module Decimal.**

**La fonction round() prend deux arguments : le nombre que vous voulez arrondir et le nombre de décimales à garder. Par exemple, si vous voulez arrondir la valeur de la variable pi à deux décimales, vous pouvez utiliser le code suivant :**

**import math**

**pi = math.pi**

**roundedPi = round(pi, 2)**

**print(roundedPi)**

**La fonction format() peut également être utilisée pour arrondir un nombre à un certain nombre de décimales. Elle retourne une chaîne de caractères qui représente le nombre arrondi. Voici comment vous pouvez l'utiliser pour arrondir la valeur de pi à deux décimales :**

**import math**

**pi = math.pi**

**formattedPi = format(pi, ".2f")**

**print(formattedPi)**

**Le module Decimal permet d'augmenter la précision des nombres à virgule flottante. Pour utiliser ce module, nous devons l'importer. Nous pouvons utiliser value.quantize(decimal.Decimal('0.00')) pour arrondir jusqu'à 2 chiffres après la virgule décimale. Voici un exemple de son utilisation :**

**from decimal import Decimal**

**import math**

**pi = Decimal(str(math.pi))**

**roundedPi = pi.quantize(Decimal('0.00'))**

**print(roundedPi)**

**Veuillez noter que la fonction round() peut parfois donner des résultats surprenants avec les nombres à virgule flottante. Par exemple, round(2.675, 2) donne 2.67 au lieu du 2.68 attendu. Cela est dû à la méthode d'arrondissement utilisée par Python, appelée arrondissement bancaire****.**

**4. Créez une variable reversed\_text qui contient la chaîne de caractères "Hello world !" à l'envers.**

**Le résultat doit être un objet liste.**

**5. Créez une variable age et demandez à l'utilisateur d'entrer son âge. Puis l'afficher et afficher son type.**

**6. Créez une variable sorted\_num qui contient la liste num triée.**

**num = [2, 8, 1, 4, 6, 3, 7]**

**En Python, vous pouvez créer une variable `sorted\_num` qui contient la liste `num` triée en utilisant la fonction `sorted()`. Cette fonction retourne une nouvelle liste qui est une version triée de la liste d'entrée. Voici comment vous pouvez faire cela :**

**```python**

**num = [2, 8, 1, 4, 6, 3, 7]**

**sorted\_num = sorted(num)**

**print(sorted\_num)**

**```**

**Dans ce code, `sorted()` prend la liste `num` comme argument et renvoie une nouvelle liste qui est une version triée de `num`. Cette nouvelle liste est ensuite assignée à la variable `sorted\_num`. Enfin, nous imprimons `sorted\_num` pour vérifier que la liste a été correctement triée [Source 0](https://realpython.com/python-sort/), [Source 1](https://docs.python.org/3/howto/sorting.html).**

**Notez que la fonction `sorted()` ne modifie pas la liste originale. Si vous voulez trier la liste en place (c'est-à-dire modifier la liste originale sans créer une nouvelle liste), vous pouvez utiliser la méthode `sort()` de la liste. Voici comment vous pouvez le faire :**

**```python**

**num = [2, 8, 1, 4, 6, 3, 7]**

**num.sort()**

**print(num)**

**```**

**Dans ce code, `num.sort()` trie la liste `num` en place, ce qui signifie que l'ordre des éléments dans `num` est modifié directement**

**7. Créez une variable sum\_of\_list qui contient la somme de tous les éléments de la liste num.**

**num = [2, 8, 1, 4, 6, 3, 7]**

**sum\_of\_list = sum(num)**

**8. Créez une variable min\_value qui contient la valeur minimale de la liste num.**

**num = [2, 8, 1, 4, 6, 3, 7]**

**La fonction min() en Python est utilisée pour trouver l'élément avec la valeur la plus petite parmi ses arguments ou ses éléments. Elle peut prendre n'importe quel nombre d'arguments numériques et renvoie l'argument qui a la valeur minimale. Si vous passez une séquence (par exemple, une liste ou un tuple), elle renverra l'élément de la séquence qui a la valeur minimale**

**9. Créez une variable max\_value qui contient la valeur maximale de la liste num.**

**num = [2, 8, 1, 4, 6, 3, 7]**

num = [2, 8, 1, 4, 6, 3, 7]

max\_value = max(num)

**10. Trouvez une fonction qui interprétera la chaîne de la variable calc**

**Enregistrez le résultat dans une variable nommée string\_interpret.**

**calc = "1 + 2"**

**La fonction eval() en Python permet d'évaluer des expressions arbitraires à partir d'une entrée basée sur une chaîne ou un objet de code compilé. Cette fonction peut être utile lorsque vous essayez d'évaluer des expressions Python à la volée** [**1**](https://realpython.com/python-eval-function/)**.**

**Voici comment cela fonctionne. Si vous passez une chaîne à eval(), alors la fonction la parse, la compile en bytecode, et l'évalue comme une expression Python. Mais si vous appelez eval() avec un objet de code compilé, alors la fonction ne fait que l'étape d'évaluation, ce qui est assez pratique si vous appelez eval() plusieurs fois avec la même entrée** [**1**](https://realpython.com/python-eval-function/)**.**

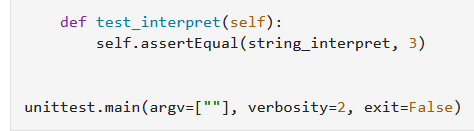
**Test**

**Le morceau de code suivant sert à tester que vous avez correctement effectué les exercices ci-dessus. S'il échoue, cela signifie qu'une partie de votre code n'est pas correcte. Lisez attentivement les messages d'erreur de chaque test.**

**Si la syntaxe du code ci-dessous est confuse (class,self, etc...), ne vous inquiétez pas, vous le verrez bientôt. La seule chose que vous devez savoir est que la méthode assertEqual vérifie si l'argument de droite correspond à celui de gauche, par exemple pour assertEqual(count\_alpha, 12) il vérifie que count\_alpha est égal à 12.**

**Ne modifiez pas le code, il vous suffit de l'exécuter.**

****

****

6.Listes, tuples et dictionnaires

**Les listes**

**Les listes sont des séquences, des collections ordonnées d'objets séparés par des virgules. Elles sont déclarées avec l'opérateur d'indication [].**

**ma\_liste = ["a", "b", "c", "d"]**

**Il est possible d'accéder à une valeur en indiquant l'indice.**

**print(ma\_liste[0])**

**Nous pouvons spécifier que nous ne voulons que les deux premiers éléments de la liste.**

**print(ma\_liste[:2])**

**Il est également possible d'inverser la liste.**

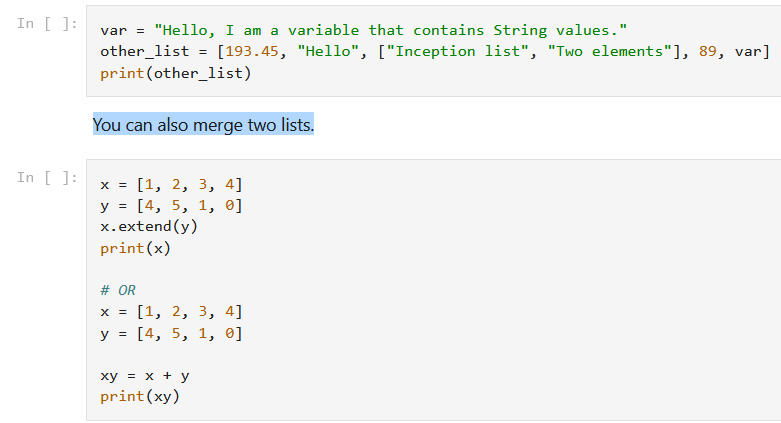
**print(ma\_liste[::-1])**

**Attention, les index commencent toujours à 0. C'est un peu dérangeant au début, mais il faut s'y habituer. Autres exemples de ce que l'on peut faire avec les listes.**

****

**Vous pouvez y placer n'importe quel type de données.**

**Vous pouvez également fusionner deux listes.**

****

**Il existe de nombreuses méthodes, je vous invite à les découvrir en faisant une recherche sur google.**

**Tuples**

**Python propose un type de données appelé tuple (terme signifiant "Table UPLEt"), qui est assez similaire à une liste mais ne peut pas être modifié. Les tuples sont donc préférables aux listes lorsque l'on veut s'assurer que les données transmises ne sont pas modifiées par erreur au sein d'un programme. En outre, les tuples sont moins "gourmands" en ressources système (ils occupent moins d'espace mémoire). Les tuples sont des objets immuables (nous verrons ce que cela signifie un peu plus loin).**

**exemple\_tuple = ("Moriarty", "Sherlock", "Watson")**

**exemple\_de\_liste = ["Moriarty", "Sherlock", "Watson"]**

**tuple VS liste**

**Comme nous pouvons le voir, lorsque nous essayons de modifier le contenu d'un objet Tuple, nous obtenons une erreur. En revanche, pas de problème pour la liste.**

**tuple\_exemple.append("Lestrade")**

**liste\_exemple.append("Lestrade")**

**print(liste\_exemple)**

**Et comme nous l'avons vu précédemment, même si nous sauvegardons la variable list\_example dans une autre variable et que nous effectuons une modification sur l'une des listes, les deux listes seront impactées.**

**deuxième\_liste\_exemple = liste\_exemple**

**deuxième\_liste\_exemple.append("Hudson")**

**print(deuxième\_liste\_exemple)**

**print(liste\_exemple)**

**Par conséquent, si vous devez traiter une collection qui ne changera pas, utilisez des tuples. En revanche, si vous savez que cette collection devra être modifiée, utilisez les listes.**

**Dictionnaires**

**Les dictionnaires ou "tableaux associatifs" ne sont pas des séquences, mais un autre type composite. Ils ressemblent aux listes dans une certaine mesure (ils sont modifiables comme elles), mais les éléments que nous allons enregistrer ne seront pas disposés dans un ordre immuable. En revanche, nous pourrons accéder à n'importe lequel d'entre eux à l'aide d'un index spécifique appelé clé, qui peut être alphabétique, numérique ou même de type composite sous certaines conditions. Remarque : comme dans une liste, les éléments stockés dans un dictionnaire peuvent être de n'importe quel type (valeurs numériques, chaînes de caractères, listes, etc.)**

**Dans les dictionnaires, les index seront des chaînes de caractères, contrairement aux listes.**

**Le type dictionnaire étant un type modifiable, on peut commencer par créer un dictionnaire vide et le remplir petit à petit. D'un point de vue syntaxique, une structure de données de type dictionnaire se reconnaît au fait que ses éléments sont entourés d'une paire d'accolades. Un dictionnaire vide s'écrira donc {}**

**heroes = {}**

**heroes["Batman"] = "Bruce Wayne"**

**heroes["Superman"] = "Clark Kent"**

**print(heroes)**

**Comme vous pouvez le voir dans la ligne ci-dessus, un dictionnaire se présente comme une série d'éléments séparés par des virgules (tous placés entre deux accolades). Chacun de ces éléments consiste en une paire d'objets : un index et une valeur, séparés par deux points.**

**Dans un dictionnaire, les index sont appelés clés, et les éléments peuvent donc être appelés paires clé-valeur. Vous remarquerez peut-être que l'ordre dans lequel les éléments apparaissent dans la dernière ligne ne correspond pas à l'ordre dans lequel nous les avons fournis. Cela n'a absolument aucune importance : nous n'essaierons jamais d'extraire une valeur d'un dictionnaire à l'aide d'un index numérique. Nous utiliserons plutôt les clés :**

**print(heroes["Batman"])**

**print(heroes["Superman"])**

**Ici, Batman et Superman sont les clés et Bruce Wayne et Clark Kent sont les valeurs.**

**Contrairement aux listes, il n'est pas nécessaire d'utiliser une méthode particulière pour ajouter de nouveaux éléments à un dictionnaire : il suffit de créer une nouvelle paire clé-valeur.**

**heroes["Spiderman"] = "Peter Parker"**

**print(heroes)**

**Questionnaire**

**Perceuse : Listes, Tuples, Dictionnaires**

**1. Choisissez 5 mots de la langue anglaise et créez des dictionnaires qui associent chacun de ces mots à sa traduction française.**

dictionnaire = {

"hello": "bonjour",

"please": "s'il vous plaît",

"thank you": "merci",

"yes": "oui",

"no": "non"

}

**2. Ajoutez une entrée au dictionnaire de la question précédente (un nouveau mot et sa définition).**

Pour ajouter une nouvelle entrée à votre dictionnaire, vous pouvez simplement assigner une nouvelle valeur à une nouvelle clé. Par exemple, si vous voulez ajouter le mot "goodbye" avec sa traduction française "au revoir", vous pouvez faire comme suit :

```python

dictionnaire["goodbye"] = "au revoir"

```

Cela ajoutera une nouvelle entrée à votre dictionnaire où la clé est "goodbye" et la valeur est "au revoir"

Vous pouvez également utiliser la méthode `update()` pour ajouter plusieurs nouvelles entrées à la fois. Par exemple, si vous voulez ajouter les mots "good morning", "good night", "excuse me" et leur traduction française, vous pouvez faire comme suit :

```python

dictionnaire.update({

"good morning": "bonne matinée",

"good night": "bonne nuit",

"excuse me": "excusez-moi"

})

```

Cela ajoutera trois nouvelles entrées à votre dictionnaire

Il est important de noter que si vous essayez d'ajouter une entrée avec une clé qui existe déjà dans le dictionnaire, la valeur existante sera remplacée par la nouvelle valeur.

**3. Comment couper la chaîne suivante à chaque espace et la mettre dans une liste ?**

phrase = "Je suis le maître du monde"

En Python, vous pouvez utiliser la méthode `split()` pour diviser une chaîne de caractères en une liste de sous-chaînes. Par défaut, cette méthode utilise les espaces blancs comme séparateurs. Donc, si vous avez une phrase et que vous voulez la diviser en mots, vous pouvez faire comme suit :

```python

phrase = "Je suis le maître du monde"

mots = phrase.split()

print(mots)

```

Dans cet exemple, `split()` divise la chaîne de caractères `phrase` en une liste de mots à chaque espace blanc. Le résultat sera : `['Je', 'suis', 'le', 'maître', 'du', 'monde']` [Source 0](https://www.w3schools.com/python/ref\_string\_split.asp).

La méthode `split()` est très utile lorsque vous travaillez avec des chaînes de texte et que vous voulez les diviser en morceaux plus petits pour les analyser ou les manipuler plus facilement [Source 3](https://www.freecodecamp.org/news/python-split-string-how-to-split-a-string-into-a-list-or-array-in-python/).

**4. Transformez cette chaîne "Le\_nombre\_universel\_est\_42" en supprimant les underscores : "Le nombre universel est 42"**

**nombre\_universel = "Le\_nombre\_universel\_est\_42"**

Pour remplacer tous les underscores dans une chaîne par des espaces en Python, vous pouvez utiliser la méthode `replace()`. Cette méthode prend deux arguments : la première est la sous-chaîne à rechercher et la seconde est la sous-chaîne par laquelle la remplacer. Si vous voulez remplacer tous les underscores par des espaces, vous pouvez faire comme suit :

```python

nombre\_universel = "Le\_nombre\_universel\_est\_42"

nombre\_universel = nombre\_universel.replace("\_", " ")

print(nombre\_universel)

```

Dans cet exemple, `replace("\_", " ")` remplace tous les underscores dans la chaîne `nombre\_universel` par des espaces. Le résultat sera : `"Le nombre universel est 42"`

**5. N'affichez que les valeurs de ce dictionnaire.**

**heroes = {"Superman" : "Clark Kent", "Batman" : "Bruce Wayne", "Spiderman" : "Tony Parker"}**

**6. Affichez uniquement les clés de ce dictionnaire.**

**heroes = {"Superman" : "Clark Kent", "Batman" : "Bruce Wayne", "Spiderman" : "Tony Parker"}**

D'accord, imagine que tu aies un grand sac rempli de différentes sortes de jouets. Chaque type de jouet est unique et a un nom particulier. Dans ce cas, le sac serait comme un dictionnaire en Python, où chaque type de jouet serait une clé et le nombre de jouets de ce type serait la valeur.

Maintenant, si tu veux savoir quels types de jouets tu as dans ton sac, tu peux demander à Python de te donner une liste de tous les noms des jouets. C'est ce que fait la commande `heroeskeyValues = list(heroes.keys())`.

`heroes.keys()` demande à Python de te montrer toutes les clés (les noms des jouets) dans le dictionnaire `heroes`. Et `list()` transforme ce que Python te donne en une liste. Donc, après avoir exécuté cette commande, `heroeskeyValues` sera une liste de tous les noms des jouets que tu as dans ton sac.

**7. Remplacez la valeur de "Spiderman" par "Peter Parker".**

**heroes = {"Superman" : "Clark Kent", "Batman" : "Bruce Wayne", "Spiderman" : "Tony Parker"}**

Pour remplacer la valeur de "Spiderman" par "Peter Parker" dans le dictionnaire `heroes`, vous pouvez simplement assigner la nouvelle valeur à la clé correspondante. Voici comment vous pouvez le faire :

```python

heroes = {"Superman" : "Clark Kent", "Batman" : "Bruce Wayne", "Spiderman" : "Tony Parker"}

heroes["Spiderman"] = "Peter Parker"

```

Dans cet exemple, `heroes["Spiderman"] = "Peter Parker"` remplace la valeur actuelle de la clé "Spiderman" par "Peter Parker" [Source 1](https://www.geeksforgeeks.org/python-replace-dictionary-value-from-other-dictionary/).

**8. Créez un dictionnaire pour construire la base de prix des produits correspondant au tableau suivant.**

**Épée laser // 229,0**

**Mitendo DX // 127,30**

**Coussin Linux // 74,50**

**Slip Goldorak // 29,90**

**Station Nextpresso // 184,60**

products = { "Laser sword": 229.0, "Mitendo DX": 127.30, "Linux cushion": 74.50, "Goldorak briefs": 29.90, "Nextpresso station": 184.60 }

**9. Calculez le prix total des articles du dictionnaire.**

**sum**

**10. Retirez l'un des articles du dictionnaire.**

Pour supprimer un élément d'un dictionnaire en Python, vous pouvez utiliser la méthode `del` ou la méthode `pop()`.

La méthode `del` permet de supprimer une clé spécifique et sa valeur associée du dictionnaire. Voici comment vous pouvez le faire :

```python

heroes = {"Superman" : "Clark Kent", "Batman" : "Bruce Wayne", "Spiderman" : "Peter Parker"}

del heroes["Spiderman"]

```

Dans cet exemple, `del heroes["Spiderman"]` supprime la clé "Spiderman" et sa valeur associée du dictionnaire `heroes`

La méthode `pop()` fonctionne de manière similaire, mais elle retourne également la valeur qui a été supprimée. Si la clé n'existe pas dans le dictionnaire, `pop()` renvoie une valeur par défaut si elle est fournie, sinon elle renvoie une erreur `KeyError`. Voici comment vous pouvez utiliser `pop()` :

```python

heroes = {"Superman" : "Clark Kent", "Batman" : "Bruce Wayne", "Spiderman" : "Peter Parker"}

removed\_value = heroes.pop("Spiderman")

```

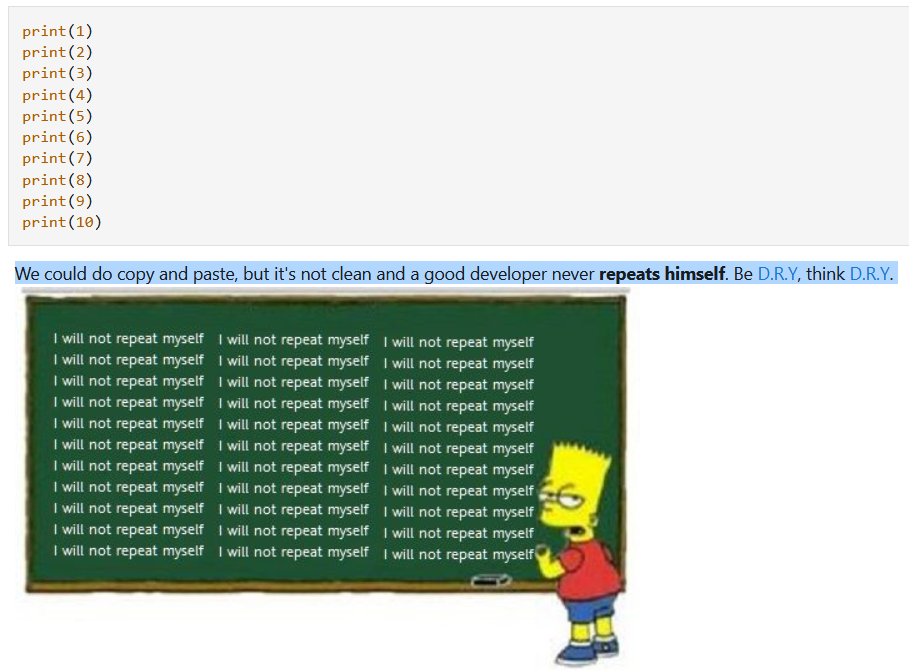
Dans cet exemple, `heroes.pop("Spiderman")` supprime la clé "Spiderman" et sa valeur associée du dictionnaire `heroes` et stocke la valeur supprimée dans la variable `removed\_value

7.Boucles et itération

**Nous allons maintenant aborder un élément plus complexe et plus obscur que les conditions, j'ai nommé : les boucles !**

**Les boucles sont un élément essentiel de la programmation. Elles permettent de répéter des instructions un certain nombre de fois. Elles permettent également d'éviter les répétitions. Imaginons que l'on veuille afficher les chiffres de 1 à 10. Sans les boucles, nous devrions écrire quelque chose comme ceci :**

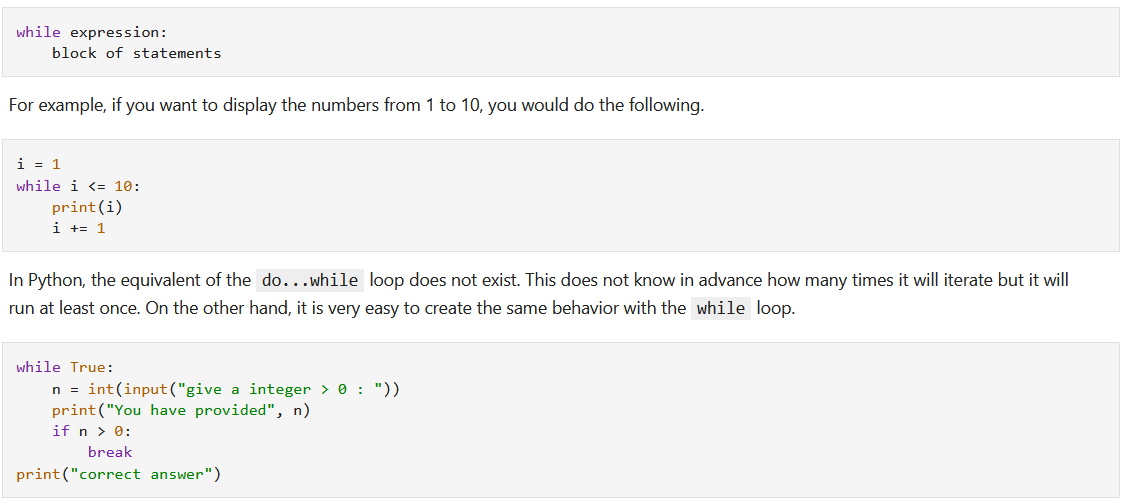
**Nous pourrions faire du copier-coller, mais ce n'est pas propre et un bon développeur ne se répète jamais. Soyez D.R.Y, pensez D.R.Y.**

****

**While**

**La boucle while répète plusieurs fois la séquence d'actions jusqu'à ce que la condition ait la valeur False. La condition est donnée avant le corps de la boucle et est vérifiée avant chaque exécution du corps de la boucle. En règle générale, la boucle while est utilisée lorsqu'il est impossible de déterminer à l'avance le nombre exact d'itérations de la boucle.**

**La syntaxe de la boucle while dans le cas le plus simple se présente comme suit :**

****

**Par exemple, si vous voulez afficher les chiffres de 1 à 10, vous ferez ceci.**

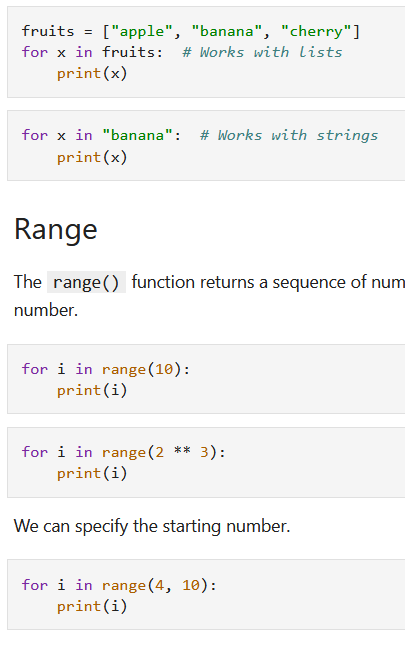
**En Python, l'équivalent de la boucle do...while n'existe pas. Celle-ci ne sait pas à l'avance combien de fois elle va itérer mais elle s'exécutera au moins une fois. Par contre, il est très facile de créer le même comportement avec la boucle while.**

**For**

**Une boucle for est utilisée pour parcourir une séquence (c'est-à-dire une liste, un tuple, un dictionnaire, un ensemble ou une chaîne de caractères).**

**Contrairement au mot-clé for dans d'autres langages de programmation, elle fonctionne davantage comme une méthode d'itérateur telle qu'on la trouve dans d'autres langages de programmation orientés objet.**

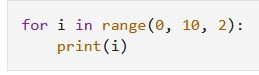
**La boucle for permet d'exécuter un ensemble d'instructions, une fois pour chaque élément d'une liste, d'un tuple, d'un ensemble, etc.**

****

**Range**

**La fonction range() renvoie une séquence de nombres, commençant à 0 par défaut, et incrémentée de 1 (par défaut), et se terminant à un nombre spécifié.**

**Nous pouvons également spécifier l'intervalle entre chaque.**

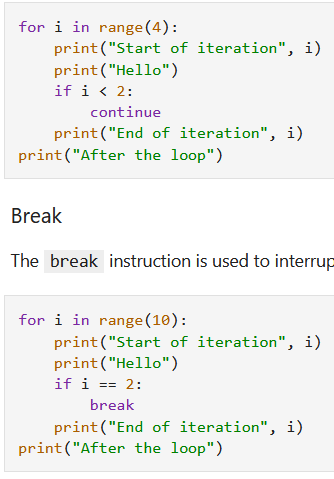
****

**Continuer et rompre**

**Continuer**

**L'instruction continue permet de passer plus tôt au tour suivant de la boucle. Elle passe à l'itération suivante de la boucle.**

**Interrompre**

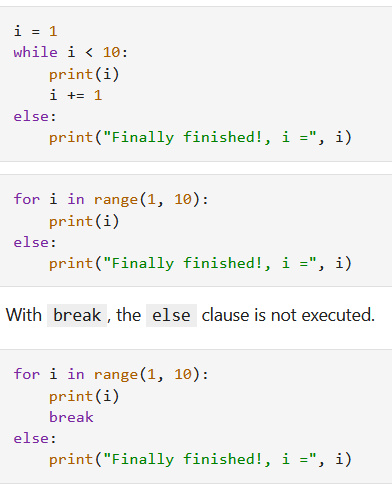
****

**L'instruction break est utilisée pour interrompre l'exécution d'une boucle (while ou for). Elle sort de la boucle et passe à l'instruction suivante.**

**Else**

**Les instructions de boucle comportent une clause else. Elle est exécutée lorsque la boucle se termine par l'épuisement de la liste (dans les boucles for) ou lorsque la condition devient False (dans les boucles while), mais pas lorsque la boucle est interrompue par une instruction break. Ceci est illustré dans la boucle suivante.**

**Avec l'instruction break, la clause else n'est pas exécutée.**

****

1- **\*\*Sort and display the list.\*\***

- Output expected: `[10, 17, 25, 38, 72]`

-2 **\*\*Add item 12 to the list and display the list.\*\***

- Output expected: `[17, 38, 10, 25, 72, 12]`

-3 **\*\*Reverse and display the list.\*\***

- Output expected: `[12, 72, 25, 10, 38, 17]`

-4 **\*\*Display the index of element 17.\*\***

- Output expected: `5`

-5 **\*\*Remove item 38 and display the list.\*\***

- Output expected: `[12, 72, 25, 10, 17]`

-6 **\*\*Display the sub-list of the 2nd to 3rd element.\*\***

- Output expected: `[72, 25]`

-7 **\*\*Display the sub-list from the beginning to the 2nd element.\*\***

- Output expected: `[12, 72]`

-8 **\*\*Display the sub-list of the 3rd element at the end of the list.\*\***

- Output expected: `[25, 10, 17]`

-9 **\*\*Display the complete sub-list of the list.\*\***

- Output expected: `[12, 72, 25, 10, 17]`

-10 **\*\*Display the last element using a negative indication.\*\***

- Output expected: `[17]`

**Pour réaliser ces opérations sur une liste en Python, vous pouvez suivre les étapes suivantes :**

1. \*\*Trier et afficher la liste.\*\*

Vous pouvez utiliser la méthode `sort()` pour trier la liste. Ensuite, vous pouvez l'afficher en utilisant la fonction `print()`.

```python

ma\_liste = [10, 17, 25, 38, 72]

ma\_liste.sort()

print(ma\_liste)

```

2. \*\*Ajouter l'élément 12 à la liste et afficher la liste.\*\*

Vous pouvez utiliser la méthode `append()` pour ajouter un élément à la fin de la liste.

```python

ma\_liste.append(12)

print(ma\_liste)

```

3. \*\*Inverser et afficher la liste.\*\*

Vous pouvez utiliser la méthode `reverse()` pour inverser l'ordre des éléments de la liste.

```python

ma\_liste.reverse()

print(ma\_liste)

```

4. \*\*Afficher l'index de l'élément 17.\*\*

Vous pouvez utiliser la méthode `index()` pour trouver l'index d'un élément dans la liste.

```python

print(ma\_liste.index(17))

```

5. \*\*Supprimer l'élément 38 et afficher la liste.\*\*

Vous pouvez utiliser la méthode `remove()` pour supprimer la première occurrence d'un élément dans la liste.

```python

ma\_liste.remove(38)

print(ma\_liste)

```

6. \*\*Afficher la sous-liste du deuxième au troisième élément.\*\*

Vous pouvez utiliser l'indexation de liste pour accéder à une sous-liste. L'indexation de liste en Python commence à 0, donc le deuxième élément est à l'index 1 et le troisième élément est à l'index 2.

```python

print(ma\_liste[1:3])

```

7. \*\*Afficher la sous-liste du début au deuxième élément.\*\*

Vous pouvez également utiliser l'indexation de liste pour cela.

```python

print(ma\_liste[:2])

```

8. \*\*Afficher la sous-liste du troisième élément à la fin de la liste.\*\*

Vous pouvez également utiliser l'indexation de liste pour cela.

```python

print(ma\_liste[2:])

```

9. \*\*Afficher la sous-liste complète de la liste.\*\*

Vous pouvez simplement imprimer la liste.

```python

print(ma\_liste)

```

10. \*\*Afficher le dernier élément en utilisant une indication négative.\*\*

Vous pouvez utiliser l'indexation de liste avec un indice négatif pour accéder à l'élément à partir de la fin de la liste. Un indice de `-1` se réfère à l'élément à la fin de la liste.

```python

print([ma\_liste[-1]])

**```**

**### 7. Écrire un algorithme qui :**

**1. Demande à l'utilisateur d'entrer un nombre.**

**2. Assurez-vous que votre programme affiche tous les nombres jusqu'à 0. Par exemple, si l'utilisateur entre le nombre 3, votre programme affichera quelque chose comme ceci : `3,2,1,0`**

**numbers = int(input(“input a number :”))**

**while numbers >= 0:**

**print(numbers )**

**numbers -= 1**

**9. L'erreur dans votre code provient du fait que vous essayez d'imprimer une liste entière comme une chaîne de caractères. En Python, lorsque vous essayez de convertir une liste en une chaîne de caractères, elle sera convertie en une représentation de chaîne de caractères de la liste, ce qui inclut les crochets.**

**De plus, votre liste `all\_students` est une liste de listes, donc vous devez utiliser une boucle `for` imbriquée pour accéder à chaque élève individuellement. Voici comment vous pouvez corriger votre code :**

**```python**

**all\_students = [**

**["David", "Justine", "Valentin", "Axel", "Redouane"],**

**["Julie", "Stéphane", "Mostapha", "Claudiu", "Son"],**

**]**

**for student\_list in all\_students:**

**for student in student\_list:**

**print(f"{student} est un ancien élève.")**

**```**

**Dans ce code, nous utilisons une boucle `for` externe pour parcourir chaque liste d'étudiants dans `all\_students`. Ensuite, nous utilisons une boucle `for` interne pour parcourir chaque étudiant dans la liste d'étudiants actuelle. Pour chaque étudiant, nous utilisons la fonction `print()` pour afficher le nom de l'étudiant suivi de la phrase "est un ancien élève."**

**Quizz 1 - Les bases de Python**

**a) type**

**Créez une variable string contenant "toto" et une variable number contenant le nombre 10.**

**Convertissez string en unicode et number en float.**

**Inversez le contenu des variables string et number (à partir de la ligne 1).**

**b) Liste**

**Créez une liste first\_list contenant les entiers suivants : 1, 3, 2, 7, 4, 10, 46.**

**Affichez les 3 premiers éléments de first\_list - en une ligne de code.**

**Créez une liste second\_list qui ne contient que les 3e, 4e et 5e éléments de first\_list. L'indexation de votre liste doit utiliser 3 caractères au maximum (sans compter les espaces) - Vérifiez soigneusement votre résultat !**

**Concaténer first\_list et second\_list pour former third\_list.**

**En utilisant la fonction zip, associez les valeurs de first\_list et second\_list dans la variable tuple\_of\_lists.**

**Ajoutez le nombre 5 à first\_list.**

**Ajoutez un élément nul à third\_list.**

**Ecrivez une fonction qui prend comme arguments une liste my\_list et un entier n et qui renvoie le résultat de n concaténations de my\_list avec elle-même.**

**# Exemple : my\_list=[1,2,3], n=2 --> [1,2,3,1,2,3]**

**Vous souhaitez rendre le paramètre n optionnel et lui donner la valeur 2 par défaut. Comment procéder ?**

**Avec une boucle while, imprimez chaque élément de third\_list jusqu'à l'élément nul.**

**Calculez, à l'aide d'une boucle for, le nombre d'entiers pairs présents dans la liste first\_list.**

**Réécrivez l'expression suivante à l'aide d'une boucle for à plusieurs lignes.**

**even\_numbers = [element for element in first\_list if element % 2 == 0]**

**Créez une fonction qui prend une chaîne en entrée et renvoie la première lettre de la chaîne.**

**c) Dictionnaires**

**Créez un dictionnaire de voitures avec les clés et les valeurs suivantes :**

**clé valeur**

**marque Ford**

**modèle Mustang**

**année 1964**

**Accédez à la valeur année.**

**Utilisez une boucle pour afficher toutes les clés du dictionnaire.**

**Utiliser une boucle pour afficher toutes les valeurs du dictionnaire.**

**Utiliser une boucle pour afficher tous les éléments du dictionnaire. Affichez également l'index de chaque itération de la boucle (fonction enumerate).**

**À l'aide d'une compréhension du dictionnaire, créez un dictionnaire avec les clés et les valeurs suivantes :**

**clé valeur**

**cle\_1 dictionnaire 'voiture'**

**cle\_2 dictionnaire 'voiture'**

**cle\_3 dictionnaire 'voiture'**

**Exemple de compréhension d'un dictionnaire (similaire à la compréhension d'une liste) :**

**dict = {clé:valeur pour clé, valeur dans zip(liste\_1, liste\_2)}**

**d) Fonctions**

**Créez une fonction de Fibonacci appelée fibonacci (séquence de Fibonacci : 0 1 1 2 3 5 8 13 ...).**

**Elle doit retourner la liste de tous les nombres de Fibonacci jusqu'au nombre de Fibonacci qui est plus grand (ou égal) que le nombre passé en argument. Ex : fibonacci(10) return [0,1,1,2,3,5,8,13] Ex : fibonacci(5) return [0,1,1,2,3,5]**

**Exécutez votre fonction. fibonacci(100) doit être exécuté en moins d'une seconde.**

**Créez la même fonction à l'aide d'un générateur**

**Ex : generate\_fibonacci(10)**