Partie 1 : Les bases du langage

Résumés Module 2 : Type de données, Variables, Opérations d'entrée-sorties de base, opérateurs de base

- Ecriture et exécution des programmes Python simples ;
- Les littéraux, les opérateurs et les expressions Python;
- Les variables et les règles correspondantes;
- Opérations d'entrée et de sortie de base.

2.1/ Ecriture et exécution de votre premier programme : print("Hello Word ! ")

1. La fonction ***print()*** est une fonction intégrée (**built-in)**. Il imprime/sort un message spécifié à l'écran/fenêtre de console.

2. Les fonctions intégrées, contrairement aux fonctions définies par l'utilisateur, sont toujours disponibles et n'ont pas besoin d'être importées. Python 3.8 est livré avec 69 fonctions intégrées. Vous pouvez trouver leur liste complète fournie par ordre alphabétique dans la bibliothèque standard Python.

3. Pour appeler une fonction (ce processus est appelé **invocation de fonction ou appel de fonction**), vous devez utiliser le nom de la fonction suivi de parenthèses. Vous pouvez passer des arguments à une fonction en les plaçant entre parenthèses. Vous devez séparer les arguments par une virgule, par exemple print("Bonjour", "monde !"). Une fonction print() "vide" affiche une ligne vide à l'écran.

4. Les chaînes Python sont délimitées par des guillemets, par exemple, "Je suis une chaîne" (guillemets doubles) ou 'Je suis aussi une chaîne' (guillemets simples).

5. Les programmes informatiques sont des collections **d'instructions**. Une instruction est une commande pour effectuer une tâche spécifique lorsqu'elle est exécutée, par exemple, pour imprimer un certain message à l'écran.

6. Dans les chaînes Python, la barre oblique inverse (**backslash**) (****) est un caractère spécial qui annonce que le caractère suivant a une signification différente, par exemple, **\n** (le caractère de nouvelle ligne) commence une nouvelle ligne de sortie.

7. Les arguments positionnels (**Positional arguments**) sont ceux dont la signification est dictée par leur position, par exemple, le deuxième argument est sorti après le premier, le troisième est sorti après le deuxième, etc.

8. Les arguments de mots-clés **(Keyword arguments)** sont ceux dont la signification n'est pas dictée par leur emplacement, mais par un mot spécial (mot-clé) utilisé pour les identifier.

9. Les paramètres ***end*** et ***sep*** peuvent être utilisés pour formater la sortie de la fonction print(). Le paramètre sep spécifie le séparateur entre les arguments affichés (par exemple, ***print("H", "E", "L", "L", "O", sep="-")***, tandis que le paramètre end spécifie ce qu'il faut imprimer à la fin de la déclaration d'impression.

2.2/ Les littéraux, les opérateurs et les expressions Python.

Les littéraux

1. Les littéraux (**Literals**) sont des notations pour représenter certaines valeurs fixes dans le code. Python a différents types de littéraux - par exemple, un littéral peut être un nombre (littéraux numériques, par exemple, 123) ou une chaîne (littéraux de chaîne, par exemple, "Je suis un littéral.").

2. Le système binaire est un système de nombres qui utilise 2 comme base. Par conséquent, un nombre binaire est composé de 0 et de 1 uniquement, par exemple, 1010 est 10 en décimal.

De même, les systèmes de numération octale et hexadécimale utilisent respectivement 8 et 16 comme bases. Le système hexadécimal utilise les nombres décimaux et six lettres supplémentaires.

3. Les entiers (**Integers ou simplement int**) sont l'un des types numériques pris en charge par Python. Ce sont des nombres écrits sans composante fractionnaire, par exemple 256 ou -1 (entiers négatifs).

4. Les nombres à virgule flottante (**Floating-point ou simplement float**) sont un autre type numérique pris en charge par Python. Ce sont des nombres qui contiennent (ou peuvent contenir) un composant fractionnaire, par exemple, 1,27.

5. Pour coder une apostrophe ou une citation à l'intérieur d'une chaîne, vous pouvez soit utiliser le caractère d'échappement, par exemple, ***'I\'m happy.'***, ou ouvrir et fermer la chaîne en utilisant un ensemble de symboles opposés à ceux que vous souhaitez encoder, par exemple, ***"I'm happy."*** pour encoder une apostrophe, et ***'He said "Python", not "typhoon"'*** pour encoder une (double) guillemet.

6. Les valeurs booléennes sont les deux objets constants ***True*** et ***False*** utilisés pour représenter les valeurs de vérité (dans les contextes numériques, ***1*** est ***True***, tandis que ***0*** est ***False***.

REMARQUE

Il y a un autre littéral spécial qui est utilisé en Python : le littéral ***None***. Ce littéral est un objet appelé ***NoneType***, et il est utilisé pour représenter l'absence d'une valeur.

Les opérateurs et les expressions

1. Une **expression** est une combinaison de valeurs (ou de variables, d'opérateurs, d'appels à des fonctions - vous les découvrirez bientôt) qui s'évalue à une valeur, par exemple 1 + 2.

2. Les **opérateurs** sont des symboles spéciaux ou des mots-clés capables d'opérer sur les valeurs et d'effectuer des opérations (mathématiques), par exemple, l'opérateur * multiplie deux valeurs : x * y.

3. Opérateurs arithmétiques en Python : ***+*** (addition), ***-*** (soustraction), ******* (multiplication), ***/*** (division classique - renvoie toujours un flottant), ***%*** (module - divise l'opérande gauche par l'opérande droit et renvoie le reste de l'opération , par exemple, ***5 % 2 = 1***), ******** (exponentiation - opérande gauche élevé à la puissance de l'opérande droit, par exemple, ***2 ** 3 = 2 * 2 * 2 = 8***), ***//*** (division entièrer - renvoie un nombre résultant de la division, mais arrondi au nombre entier inférieur le plus proche, par exemple, ***3 // 2,0 = 1,0***)

4. Un opérateur **unaire** est un opérateur avec un seul opérande, par exemple, ***-1*** ou ***+3***.

5. Un opérateur **binaire** est un opérateur avec deux opérandes, par exemple, ***4 + 5***, ou ***12 % 5***.

6. Certains opérateurs agissent avant d'autres - **la hiérarchie des priorités** :

- unaire + et - ont la priorité la plus élevée
- puis : **, puis : *, / et %, puis la priorité la plus basse : binaire + et -.

7. Les sous-expressions entre **parenthèses** sont toujours calculées en premier, par exemple, ***15 - 1 * (5 * (1 + 2)) = 0***.

8. L'opérateur d'exponentiation utilise la reliure à droite, par exemple, ***2 ** 2 ** 3 = 256***.

2.3/ Les variables et les règles correspondantes

1. Une **variable** est un emplacement nommé réservé pour stocker des valeurs dans la mémoire. Une variable est créée ou initialisée automatiquement lorsque vous lui attribuez une valeur pour la première fois.

2. Chaque variable doit avoir un nom unique - un **identifiant**. Un nom d'identifiant légal doit être une séquence de caractères non vide, doit commencer par le trait de soulignement (_) ou une lettre, et il ne peut pas être un mot-clé Python. Le premier caractère peut être suivi de traits de soulignement, de lettres et de chiffres. Les identifiants en Python sont sensibles à la casse.

3. Python est un langage à **typage dynamique**, ce qui signifie que vous n'avez pas besoin d'y déclarer de variables. Pour affecter des valeurs aux variables, vous pouvez utiliser un opérateur d'affectation simple sous la forme du signe égal (**=**), c'est-à-dire ***var = 1***.

4. Vous pouvez également utiliser des opérateurs d'affectation composés (opérateurs de raccourci) pour modifier les valeurs affectées aux variables, par exemple, ***var += 1***, ou ***var /= 5 * 2***.

5. Vous pouvez affecter de nouvelles valeurs à des variables déjà existantes à l'aide de l'opérateur d'affectation ou de l'un des opérateurs composés, par exemple :

```
... var = 2
... imprimer(var)
... var = 3
... imprimer(var)
... var += 1
... imprimer(var)
```

6. Vous pouvez combiner du texte et des variables à l'aide de l'opérateur ***+*** et utiliser la fonction **print()** pour afficher des chaînes et des variables, par exemple :

```
... var = "007"
... print("Agent " + var)
```

2.4/ Commentaires.

1. Les commentaires peuvent être utilisés pour laisser des informations supplémentaires dans le code. Ils sont omis lors de l'exécution. Les informations laissées dans le code source s'adressent à des lecteurs humains. En Python, un commentaire est un morceau de texte qui commence par **#**. Le commentaire s'étend jusqu'à la fin de la ligne.

2. Si vous souhaitez placer un commentaire qui s'étend sur plusieurs lignes, vous devez placer **#** devant chaque ligne. De plus, vous pouvez utiliser un commentaire pour marquer un morceau de code qui n'est pas nécessaire pour le moment (voir la dernière ligne de l'extrait ci-dessous), par exemple :

```
... # Ce programme imprime
... # une introduction à l'écran.
... print("Bonjour!") # Appel de la fonction print()
... # print("Je suis Python.")
```

3. Chaque fois que cela est possible et justifié, vous devez donner des noms auto-commentaires (**self-commenting names**) aux variables, par exemple, si vous utilisez deux variables pour stocker une longueur et une largeur de quelque chose, les noms de variables longueur et largeur peuvent être un meilleur choix que myvar1 et myvar2.

4. Il est important d'utiliser des commentaires pour rendre les programmes plus faciles à comprendre et d'utiliser des noms de variables lisibles et significatifs dans le code. Cependant, il est tout aussi important de ne pas utiliser de noms de variables qui prêtent à confusion ou de laisser des commentaires contenant des informations erronées ou incorrectes !

5. Les commentaires peuvent être importants lorsque vous lisez votre propre code après un certain temps (faites-nous confiance, les développeurs oublient ce que fait leur propre code) et lorsque d'autres lisent votre code (peuvent les aider à comprendre ce que font vos programmes et comment ils le font plus vite).

2.5/ Opérations d'entrée et de sortie de base.

1. La fonction **print()** envoie des données à la console, tandis que la fonction **input()** récupère les données de la console.

2. La fonction **input()** est livrée avec un paramètre facultatif : la chaîne d'invite. Il vous permet d'écrire un message avant la saisie de l'utilisateur, par exemple :

```
... name = input("Entrez votre nom : ")
... print("Bonjour, " + nom + ". Ravi de vous rencontrer !")
```

3. Lorsque la fonction **input()** est appelée, le déroulement du programme est arrêté, le symbole d'invite continue de clignoter (il invite l'utilisateur à agir lorsque la console passe en mode d'entrée) jusqu'à ce que l'utilisateur ait saisi une entrée et/ou appuyé la touche Entrée.

REMARQUE

Vous pouvez tester la fonctionnalité de la fonction **input()** dans toute sa portée localement sur votre machine. Pour des raisons d'optimisation des ressources, nous avons limité le temps maximum d'exécution du programme dans Edube à quelques secondes. Allez dans Sandbox, copiez-collez l'extrait ci-dessus, exécutez le programme et ne faites rien - attendez quelques secondes pour voir ce qui se passe. Votre programme devrait s'arrêter automatiquement après un court instant. Maintenant, ouvrez IDLE et exécutez le même programme - voyez-vous la différence ?

Astuce : la fonctionnalité mentionnée ci-dessus de la fonction input() peut être utilisée pour inviter l'utilisateur à terminer un programme. Regardez le code ci-dessous :

```
... name = input("Entrez votre nom : ")
... print("Bonjour, " + nom + ". Ravi de vous rencontrer !")
... print("\nAppuyez sur Entrée pour terminer le programme.")
... input()
... print("LA FIN.")
```

4. Le résultat de la fonction input() est une chaîne. Vous pouvez ajouter des chaînes les unes aux autres à l'aide de l'opérateur de concaténation (**+**). Découvrez ce code :

```
... num_1 = input("Entrez le premier chiffre : ") # Entrez 12
... num_2 = input("Entrez le deuxième nombre : ") # Entrez 21
... print(num_1 + num_2) # le programme renvoie 1221
```

5. Vous pouvez également multiplier (* - **réplication**) des chaînes, par exemple :

```
... my_input = input("Entrez quelque chose : ") # Exemple d'entrée : hello
... print(my_input * 3) # Résultat attendu : hellohellohello
```