Introduction

L'exécution du code print("Hello Python World!") affiche le message sur l'écran. L'interpréteur python analyse l'extension du fichier pour s'assurer qu'il s'agit bien d'un code Python. L'interpréteur lit ensuite l'ensemble du code ligne par ligne et exécute les différentes instructions.

Variables

La variables permettent de stocker des valeurs. On peut déclarer une variable à travers l'utilisation de l'opérateur "=" qui permet d'assigner une valeur à une variable.

Exemple:

message = "Hello Python World!"

print(message)

```
message = "Hello Python World!"
print(message)
```

```
Hello Python World!
```

Nomenclature

Pour nommer une variable, on doit adhérer à un certain nombre de règles:

- Le nom des variables ne doit contenir que les lettres, les nombres et les underscores "_", il peut commencer avec une lettre ou un underscore.

- Les espaces ne sont pas autorisés pour les noms des variables, mais peuvent être remplacés avec un underscore
- Eviter l'utilisation des mots clés Python comme noms de variables
- Les noms de variables doivent être concis et descriptifs
- Faire attention lors de l'utilisation de la lettre l et O car ils peuvent être confondues facilement avec les chiffres 1 et 0 respectivement.

** Erreurs

Quand l'interpréteur détecte une erreur dans un programme, une erreur est affichée sur la console avec le message d'erreur ainsi que la ligne où l'erreur a été détectée.

```
print(mesage)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-1dd115d53185> in <module>()
----> 1 print(mesage)

NameError: name 'mesage' is not defined
```

SEARCH STACK OVERFLOW

Les variables sont souvent assimilés à des boites qui peuvent être utilisées pour stocker des valeurs.

Strings

La plupart des programmes collectent des données, les traitent après leur stockage. Parmi les types de données les plus utilisés, on trouve **string**.

Un **string** est un ensemble de caractères entourés de simples ou doubles quotes.

'Ceci est une chaîne de caractères' "Ceci est une chaîne de caractères"

Cette flexibilité permet d'utiliser des apostrophes dans les variables **string**

Exemple:

"L'ensemble de données à stocker"

Changement de casse

Parmi les tâches les plus simples qu'on peut appliquer à une chaîne de caractères est le changement de la casse des mots composants la chaîne. Pour changer la casse d'une variable

string, il suffit d'utiliser la méthode **title**. L'utilisation d'une méthode peut se faire en ajoutant le nom de la méthode après le nom de variable séparés par un "."

```
name = "dupont lewis"
print(name.title())
```

```
Dupont Lewis
```

D'autres méthodes peuvent être utilisées pour mettre la chaîne de caractères en majuscules ou en minuscules.

```
name = "dupont lewis"
print(name.lower())
print(name.upper())
```

```
dupont lewis
DUPONT LEWIS
```

Utilisation des variables à l'intérieur des strings Dans certaines situations, on aura besoin d'utiliser la valeur d'une variable à l'intérieur d'une chaîne de caractères.

```
first_name = "dupont"
last_name = "lewis"

full_name = f"{first_name} {last_name}"
print(full_name)
```

```
dupont lewis
```

Le symbole f avant la chaîne de caractères dénote **format**, la chaîne de caractère obtenue est appelée f-string, elle permet de remplacer les variables entre accolades par leurs valeurs. Elles peuvent être utilisées pour composer des messages.

```
print(f"Hello, {first_name.title()}!")
```

```
Hello, Dupont!
```

Ajout des espaces et des tabulations

Dans la programmation, les espaces "whitespaces" refèrent à tous les caractères qui ne sont pas visibles sur l'écran comme les espaces, tabulations et la fin de ligne.

Pour ajouter la tabulation, il suffit d'utiliser "\t".

```
print("Python")
print("\tPython")
```

```
Python
        Python
```

Pour ajouter une nouvelle ligne, il suffit d'utiliser "\n".

```
print("Languages:\nPython\nC\nJavaScript")
```

```
Languages:
Python
C
JavaScript
```

Eliminer les espaces

Afin d'effectuer le nettoyage des données, on aura besoin des fois des values sans les espaces. Python permet d'eliminer les espaces superflus en utilisant les fonctions **rstrip** pour enlever les espaces à droite et **lstrip** pour enlever les espaces à gauche.

```
message = 'Ceci est un message qui se termine avec un espace '
message.rstrip()
```

```
'Ceci est un message qui se termine avec un espace'
```

Afin d'écraser la valeur de message avec le contenu sans espace, il suffit d'assigner le résultat de la fonction à la variable.

```
message = message.rstrip()
```

```
message
```

```
'Ceci est un message qui se termine avec un espace'
```

On peut également éliminer les espaces qui sont au début des données avec **lstrip**.

```
message = " Ceci est un message qui a un espace au début."
```

```
message.lstrip()
```

```
'Ceci est un message qui se termine avec un espace'
```

Nombres

Les nombres sont utilisés pour représenter les valeurs numériques, ils peuvent être de type **integer** ou **float**.

Les variables de type **integer** peuvent stocker des valeurs entières alors que les variables de type **float** peuvent stocker des valeurs décimales.

Pour **integer**, on peut utiliser les opérations **+** pour l'addition, **-** pour la soustraction, ***** pour la

```
3 + 5
```

```
8
```

```
3 * 4
```

```
12
```

```
-4 / 3
```

```
-1.3333333333333333
```

```
4 ** 2
```

```
16
```

On peut utiliser les mêmes opération avec les **float**.

```
0.2 + 0.3
```

```
0.5
```

```
0.1 * 0.1
```

```
0.010000000000000002
```

```
Quand on divise deux entiers **integer**, le résultat final est **float**
même si les deux entiers sont divibles.
```

```
4/2
```

```
2.0
```

On peut séparer les chiffres avec des espaces soulignés "underscore" afin de les rendre plus lisibles.

```
amount = 30_000_00
```

```
amount
```

```
3000000
```

Affectation multiple

On peut affecter plusieurs valeurs à plusieurs variables en utilisant les virgules sur une seule ligne. Ceci produit un code plus concis et facile à lire lors de l'initialisation des variables. L'interpréteur Python va affecter les valeurs aux variables dans l'ordre choisi.

```
x, y, z = 0, 0, 0
```

```
x
```

```
0
```

Constantes

Une constante, comme une variable, peut stocker des valeurs qui ne peuvent pas changer durant le cycle de vie du programme. Python n'a pas de type précis pour les constantes, mais on utilise comme convention des noms de variable en majuscules.

```
TVA = 0.2
```

Commentaires

Les commentaires sont très utiles pour la programmation. Lorsque les programmes deviennent plus longs et compliqués, on ajoute des notes tout au long du programme afin de décrire l'approche globale suivie pour résoudre un problème spécifique. Afin d'ajouter des commentaires, il suffit d'utiliser #.

```
# Ceci est un commentaire
# sur plusieurs lignes
```

Double-cliquez (ou appuyez sur Entrée) pour modifier

Zen

Les programmeurs Python expérimentés évitent la complexité et préfèrent la simplicité au maximum. La philosophie de la communauté des développeurs est contenu dans "The Zen of Python" de Tim Peters. On peut consulter à l'aide de la commande #import this#.

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```