Expressions numériques (Programmation 01)

I – Comment programmer avec Python ?

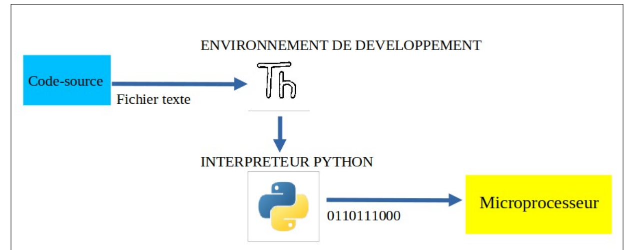
Plusieurs solutions existent :

Utiliser simplement l'**interpréteur Python**

Utiliser un **Environnement de développement** (**IDE** en Anglais) (comme **Thonny** par exemple)

Utiliser une **solution en ligne**, comme **Baston**

https://basthon.fr/) (https://python.infoforall.fr



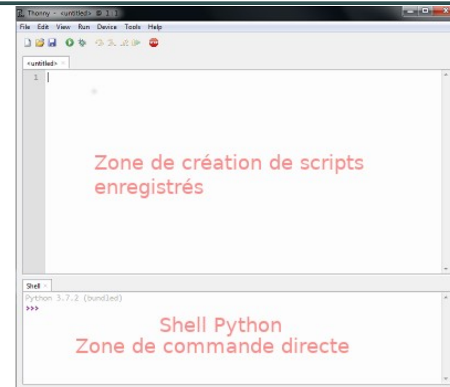
II – Thonny

2.1-Distinction entre la console interactive et la zone de programmation

Thonny est l'**environnement de développement** avec lequel nous allons travailler (en anglais, IDE pour Integrated Development Environment).

La **zone programmation en haut** : pour créer des programmes.

La zone **Console interactive en bas** : les trois chevrons >>> indiquent que l'interpréteur est prêt à exécuter immédiatement votre demande.



2.2 - Les types d'erreurs

L'interpréteur Python va échouer à comprendre votre demande à la moindre faute de "grammaire", la syntaxe. Plusieurs cas se présentent :

1. **Pas d'erreur** : la demande est bien formulée et la réponse de l'ordinateur correspond à ce que nous voulions qu'il fasse.
```
>>> 3.2 + 0.2
3.4
```
2. **Erreur de syntaxe** : notre demande ne correspond pas aux règles de la syntaxe du langage de programmation. L'interpréteur Python ne parvient pas à comprendre votre demande :
```
>>> Ca va bien ?
SyntaxError: invalid syntax
```
3. **Erreur d'exécution** : la demande est bien formulée mais provoque une erreur lors de l'exécution.
```
>>> 12 / (3 - 3)
ZeroDivisionError: division by zero
```
Ici, la division est bien formulée mais on se rend compte pendant l'évaluation qu'il s'agit d'une division par zéro.

4. **Erreur sémantique** : tout se "passe bien" en apparence, mais le sens que vous pensez avoir donné à votre demande ne correspond pas au sens que l'interpréteur lui donne.
```
>>> 5,2 + 1,2
(5, 3, 2)
```
L'**énorme** problème de ce type d'erreur ? Il est impossible à l'interpréteur Python de signaler que sa réponse ne correspond pas à ce qui se passe dans votre tête !

2.3 - Fonctionnement de la console

La console a été programmée pour afficher le résultat final obtenu lorsqu'on lui demande d'évaluer quelque chose. C'est un choix de conception, rien d'autre.

L'algorithme de fonctionnement du programme gérant la console est une boucle infinie :

1. ATTENTE : il attend que l'utilisateur valide sa ligne de commande avec l'appui sur ENTREE
2. INTERPRETATION : il tente d'interpréter et réaliser la ligne qu'il vient de recevoir
3. AFFICHAGE : il affiche
   - soit le résultat du travail demandé.
   - soit un **message d'erreur**
4. Il revient à l'étape 1

III - Expression comportant un opérateur

3.1 DEF. : Expression - Valeur - Opérateur

Une **expression** caractérise une séquence de **valeurs** associées à des **opérateurs**. Que fait l’intrepréteur d'une expression ? Il l'évalue.

```
>>> 4 + 5
9
```

- 4 et 5 sont des valeurs.
- + est un opérateur.
- 4 + 5 est une expression.
- 9 est la valeur calculée à partir de l'expression.

Les symboles Python des opérateurs classiques sur les nombres :

- **+** symbolise l'addition
- **-** symbolise la soustraction
- * symbolise la multiplication
- **/** symbolise la division
- ****** symbolise la puissance

3.2 DEF. : Evaluation et priorité d'évaluation

Lorsque l'interpréteur rencontre une **expression**, il l'évalue et obtient donc une nouvelle valeur. Cette action sur l'expression se nomme une **évaluation** .

L'évaluation est **séquentielle** et **progressive** sur une ligne : l'interpréteur Python n'évalue pas la ligne d'un seul bloc, il effectue certaines opérations en priorité.

Si vous voulez communiquer avec l'interpréteur, il faut donc savoir comment il fonctionne.

Voici **les règles de priorité** que respecte l'interpréteur :

1. Si l'expression comporte des **parenthèses** (...), Python commence par cette opération.
2. Sinon, l'opérateur prioritaire est l'**opérateur de puissance** **.
3. Ensuite, on passe à l'opérateur de **multiplication** * ou de division /.
4. Enfin, l'opérateur d'**addition** + ou de soustraction - sont les opérateurs les moins prioritaires.
5. En cas d'égalité de priorité, on commence par l'opérateur **le plus à gauche** d'abord.

Exemple 1 et 2

```
10 + (5 + 3) * 2
10 + 8 * 2
10 + 16
26
```

```
5 + 10 / 2 * 3
5 + 5.0 * 3
5 + 15.0
20.0
```

3.3 Quotient // et reste % de division euclidienne (désignées parfois comme "divisions entières")

Rappels sur les divisions euclidiennes

Il s'agit de diviser **a** par **b**, en trouvant :

- Le quotient, c'est à dire le nombre de paquets égaux qu'on peut faire en divisant **a** par **b**.
- Le reste, c'est à dire le nombre d'éléments restants après création des paquets.
- Division euclidienne de 10 par 5 :

```
     10 | 5
        |---
reste 0 | 2
```

Le quotient vaut 2 et le reste vaut 0.

On peut également le justifier en écrivant 10 = 2*5 + 0

Opérateur Python //

Python, quel est le quotient de la division euclidienne de 10 par 5 ?"

```
>>> 10 // 5
2
```

Opérateur Python %

"Python, quel est le reste de la division euclidienne de 10 par 5 ?"

```
>>> 10 % 5
0
```

Division entière de 14 par 3

```
     14 | 3
        |---
reste 2 | 4
```

On peut aussi le justifier avec 14 = 4*3 + 2

Et voici pour Python :

```
>>> 14 // 3
4
```

```
>>> 14 % 3
2
```

Priorité

L'opérateur // et l'opérateur % possèdent l**a même priorité que la multiplication** ou la division. Lorsque les deux sont présents, c'est donc l'ordre d'apparition qui tient lui de priorité. **Attention, ils ne sont pas interchangeables.**

```
>>> 1407 % 5 // 2
1
```

```
>>> 1407 // 2 % 5
3
```

IV – Application à la résolution de problèmes

4.1 Chiffres de l'unité, dizaine et centaine

Unité

Il suffit de faire une division euclidienne par 10 et de récupérer le reste.

```
>>> 1234 % 10
4
```

```
>>> 489 % 10
9
```

Dizaine

Il suffit de diviser le nombre par 10. Le chiffre de la dizaine se retrouve alors à la position de l'unité. Pour récupérer ce chiffre, il suffit donc de faire une deuxième division euclidienne par 10 et de récupérer le reste.

```
# En deux étapes
>>> 1234 // 10
123
>>> 123 % 10
3
```

```
# En une étape
>>> 1234 // 10 % 10
3
```

Centaine

Il suffit de diviser le nombre par 100. Le chiffre de la centaine se retrouve alors à la position de l'unité. Pour récupérer ce chiffre, il suffit donc de faire une deuxième division euclidienne par 10 et de récupérer le reste.

```
# En deux étapes
>>> 1234 // 100
12
>>> 12 % 10
2
```

```
# En une étape
>>> 1234 // 100 % 10
2
```

4.2 Position d'une case sur un plateau

Imaginons qu'on dispose d'un plateau de jeu de 4 cases sur 4 cases. Les cases sont numérotées de 0 à 15, comme indiqué ci-dessous : la ligne du haut porte le numéro 0, la ligne du dessous le numéro 1...
La colonne de gauche porte le numéro 0 puis les autres 1, 2 et 3.

```
Ligne 0 -->  0  1  2  3
Ligne 1 -->  4  5  6  7
Ligne 2 -->  8  9  10 11
Ligne 3 --> 12  13 14 15
```

Numéro de ligne

Comment trouver le numéro de ligne d'une case dont on connaît le numéro ? Il suffit de récupérer le **quotient de la division euclidienne du numéro de la case par 4**.

```
>>> 3 // 4
0
```

```
>>> 4 // 4
1
```

```
>>> 7 // 4
1
```

```
>>> 8 // 4
2
```

Numéro de colonne

Comment trouver le numéro de colonne d'une case dont on connaît le numéro ? Il suffit de récupérer le **reste de la division euclidienne du numéro de la case par 4**.

```
>>> 3 % 4
3
```

```
>>> 4 % 4
0
```

```
>>> 7 % 4
3
```

```
>>> 8 % 4
0
```

V – Conclusion

- L'interpréteur n'a aucune marge pour comprendre une demande mal formulée (**erreur de syntaxe**)

- Il est possible que la syntaxe soit bonne mais que vous ayez mal compris la façon dont l'interpréteur va la comprendre : c'est votre faute, pas celle de l'interpréteur (**erreur sémantique**)

Les 4 piliers de l'informatique