

Chap. 2 – Programmer en Python (pa.dilla.fr/15)

D'après toi, pourquoi HTML et CSS ne sont **pas** des langages de programmation ?

HTML et CSS sont incapables d'effectuer le moindre calcul.

On peut aussi remarquer qu'il manque des instructions essentielles comme les répétitions, les branchements conditionnels ou encore les affectations.

Python est un langage de programmation qui peut être utilisé de différentes façon. Dans ce cours nous verrons :

- le mode *interactif* dans des notebooks
- le mode *programme* dans l'interface de développement (IDE) VSCodium.

1 – Le mode interactif

Le mode interactif se comporte un peu comme une calculatrice. Nous allons explorer ce mode à travers les **notebooks**.

1.1 – Les notebooks

Un *notebook* est une page HTML qui contient des blocs de texte et des blocs de code :

- les blocs de **texte** permettent d'afficher et de modifier du texte écrit en langage `Markdown`. Cette simplification du HTML permet d'afficher du texte (normal, gras, italique, etc.), des listes, des tableaux, des liens, du code ou encore des média comme les images et les vidéos.
- les blocs de **code** permettent de saisir, modifier et d'exécuter du code Python. L'interprète Python est chargé avec la page HTML puis ensuite il calcule et affiche les résultats.

La page de ce cours consultable à l'adresse <https://pa.dilla.fr/15> **est un notebook.**

Dans le notebook, ceci est un bloc de texte avec des mots en *italiques* et d'autres en **gras**.

```
[ ]: # Ceci est un bloc de CODE.  
# Les lignes qui commencent pas un # sont des commentaires  
# Les commentaires ne seront pas interprétés par Python  
  
print("Dans le notebook, ceci est un bloc de code")
```

Dans la suite, tu vas apprendre quelques manipulations sur les notebooks et revoir des éléments **essentiels** de Python.

Comment modifier un bloc de texte ou de code ?

Pour cela tu peux utiliser la souris ou le clavier :

1. méthode avec la souris :

- double-cliquer sur ce bloc de texte pour l'*éditer*,
- modifier/corriger le texte
- cliquer sur le bouton Run pour interpréter ce bloc et sortir du mode *édition*

2. méthode au clavier :


- utiliser les flèches <↑> et <↓> pour sélectionner le bloc de texte à modifier
- appuyer sur <Entrée> pour *éditer* le bloc
- modifier/corriger le texte
- pour interpréter le résultats tu peux utiliser au choix :
 - <Ctrl> + <Entrée> pour interpréter le bloc ou
 - <Shift> + <Entrée> pour interpréter le bloc *et* passer au bloc suivant.

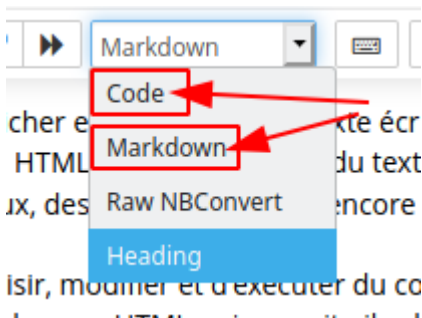
Accède à la version *notebook* de ce chapitre.

1. **Corrige** la faute d'orthographe de ce bloc de texte.
2. **Écris** dans le bloc de code ci-dessous les trois caractères `1+2` puis fait calculer le résultats par l'interprète Python.

[]:

Dans un notebook, tu peux ajouter autant de bloc que tu le souhaites. Pour cela,

1. tu peux utiliser dans les outils le bouton + .
2. Si tu souhaites que le bloc soit du code, sélectionne Code dans la liste déroulante.
3. Si tu souhaites un bloc de texte, sélectionne alors Markdown.



1.2 – Arithmétique

En Python, on peut saisir des combinaisons arbitraires d'opérations arithmétiques, en utilisant notamment les quatre opérations les plus communes.

[]:

```
2 + 5 * (10 - 1 / 2)
```

Quelques remarques

1. Les symboles des quatre opérations sont + (addition), - (soustraction), * (multiplication) et / (division).
2. La priorité des opérations est usuelle et on peut utiliser les parenthèses.
3. Les espaces sont purement décoratifs : ils rendent plus lisibles mais n'agissent pas sur la priorité des opérations

```
[ ]: 1+2      * 3
```

Remarque bien que l'interprète Python n'accepte pas les expressions qui sont mal formées ou comportent des erreurs.

Dans le bloc de code ci-dessous, **écris** `1 + * 2`.

Indique le type d'erreur affiché.

Lorsque l'on fait des erreurs de syntaxe car l'expression est mal formée, l'interprète affiche `SyntaxError`. Pour aider l'utilisateur, l'interprète montre l'endroit exact de l'erreur avec le symbole `^` qui pointe sous le `*`.

Pour répondre aux questions suivantes, tu peux ajouter des blocs de texte ou des blocs de code...

En mathématique, est-il possible de **diviser un nombre par zéro** ?

Et en **Python** ?

1.3 – Les nombres

En Python, les nombres sont de deux *types* :

- soit des **entiers relatifs** (appelés *entiers*)
- soit des **nombres décimaux** (appelés *flottants*).

Attention, les nombres flottants s'écrivent à l'*anglo-saxonne* avec un point `.`. Par exemple le nombre 1,2 s'écrit en Python `1.2`

Indique ce qu'affiche les commandes suivantes :

1. Commande 1 : `1,2 + 3,4`.
2. Commande 2 : `1,2 * 3`

D'après toi, à quoi sert le symbole virgule `,` en Python ?

En Python :

- les *entiers* sont de **taille arbitraire** ;
- les *flottants* en revanche ont une **capacité limitée** et sont souvent des approximations qui ne représentent alors qu'une partie des nombres décimaux.

Quand tu en as le choix, travaille de préférence avec les entiers.

1.4 – Autres opérations

Détermine le type de nombre que l'on obtient lorsque l'on divise entre eux deux nombres entiers.

La division avec l'opération `/` donne un nombre flottant. Ainsi `10/2` donne `5.0` et `7/2` donne `3.5`. Ces deux résultats sont de types *flottant* car ils sont écrits avec une virgule.

Effectue à la main les divisions euclidiennes :

- $\frac{7}{2}$
- $\frac{98}{10}$

`7/2` donne 3 et il reste 1. L'égalité $7 = 3 \times 2 + 1$ est vérifiée.

`98/10` donne 9 et il reste 8. L'égalité $98 = 9 \times 10 + 8$ est vérifiée.

Pour effectuer la **division entière**, il faut utiliser l'opération `//`.

`a // b` peut être vu comme le plus grand nombre entier inférieur ou égal à a/b .

Pour obtenir le **reste de la division euclidienne**, on utilise l'opération `%` (qui se dit "*modulo*").

Vérifie à l'aide des opérations *division entière* et *modulo* les résultats de l'activité précédente.

```
[ ]: 7//2
```

```
[ ]: 7%2
```

```
[ ]: 98 // 10
```

```
[ ]: 98 % 10
```

L'opération `**` calcule la *puissance* d'un nombre.

Ainsi, 10^3 s'écrit `10 ** 3`.

Calculer dans un bloc de code 2^{1024} .

Dans un autre bloc de code, **calculer** $2,0^{1024}$.

Propose une explication à tes observations.

```
[ ]: 2**1024
```

```
[ ]: 2.0**1024
```

1.3 – Variables

Les résultats calculés peuvent être **mémorisés** afin d'être utilisés plus tard.

```
[ ]: a = 1 + 1
```

La notation `a =` permet de donner un nom à l'information mémorisée. Ici, l'interprète Python calcule le résultat de `1 + 1` et le mémorise dans la **variable** `a`.

Aucun résultat n'est affiché. On accède à la valeur mémorisée en utilisant le nom `a`.

```
[ ]: a
```

Affiche maintenant le résultat du calcul `a * (a + 1)`.

```
[ ]: a * (a+1)
```

Le symbole `=` utilisé pour définir la variable `a` désigne une opération d'**affectation**.

Ce symbole attend à *gauche* un nom de variable et à *droite* une expression.

On peut donner une nouvelle valeur à la variable `a` avec une nouvelle affectation. Cette valeur *remplace* la précédente.

```
[ ]: a = 3  
a * (a+1)
```

Le calcul de la nouvelle valeur de `a` peut utiliser la valeur courante de `a`.

```
[ ]: a = a + 1  
a
```

Un nom de variable peut être formé de plusieurs caractères (lettre, chiffres et tiret bas). Il est recommandé de :

- ne pas utilisé de caractères accentués
- se limiter aux caractères minuscules.

```
[ ]: cube = a * a * a  
ma_variable = 42  
ma_variable2 = 2021
```

1.4 – État

2 – Le mode programme

2.1 – VSCodium

2.2 – Affichage

2.3 – Affichage des textes

2.4 – Séquence d'instructions

2.5 – Interagir avec l'utilisateur

2.6 – Un programme complet

2.7 – Représentation de l'exécution

3 – Exemple de bibliothèque `Turtle`