

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 moulage 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 façons. Dans ce cours tu verras :
- le mode interactif dans des notebooks
- le mode programme dans l'interface de développement (IDE) VSCode

I – Le mode interactif

Le mode interactif se compose d'un peu comme une calculatrice. Nous allons explorer ce mode à travers les notebooks.

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

Quel programme précédent si on supprime la deuxième ligne ?

Ecrits un programme qui demande l'année de naissance à l'utilisateurs puis affiche l'âge qu'il aura en 2048.

2.6 – Un programme complet

```
[ ] : # Calcul de l'âge en 2048
saliste = input("Tapez votre année de naissance : ")
âmele = int(saliste)
âge = 2048 - âmele
print("Vous aurez", âge, "ans en 2048.")
```

CORRECTION

ligne	état	interfâce	affichage : Entrée votre année de naissance :	affichage : Vous aurez 45 ans en 2048 .
2	2003	saisie	2003	affichage : 2003
3	2003	saisie année	2003	affichage : Entrée votre année de naissance :
4	2003	saisie année age	2003 45	affichage : Vous aurez 45 ans en 2048 .
5	2003	saisie année age	2003 45	affichage : Vous aurez 45 ans en 2048 .

16

- 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.

Exemple

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

Exemple

```
[ ]: # Ceci est un bloc de CODE.
# Les lignes qui commencent pas un # sont des commentaires
# Les commentaires ne seront pas interprétés par Python
#
# Les blocs de code se reconnaissent facilement car ils
# possèdent une indication dans la marge "In []" ou "Entrée []"
# Si le bloc a été exécuté, alors un compteur apparaît dans "[...]"
print("Dans le notebook, ceci est un bloc de code")
```

Pour **modifier un bloc de texte ou de code** 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** ou **Exécuter** pour interpréter ce bloc et sortir du mode *édition*

2. méthode au clavier :

une seule instruction `print` et de mettre en argument les expressions séparées par une virgule :

- `print(a,b)` affichera les deux éléments sur la même ligne : 34 55
- `print("la somme de", a, "et de", b, "vaut", a+b)` affichera sur une seule ligne les 6 expressions et donnera alors : la somme de 34 et de 55 vaut 89

2.5 – Interagir avec l'utilisateur

Pour permettre l'interaction du programme avec l'utilisateur, il faut mettre en place une **interface**.

L'interface la plus simple consiste à utiliser l'instruction `input` qui permet de récupérer des caractères tapés au clavier par l'utilisateur.

Cette instruction interrompt l'exécution du programme et attend que la saisie se termine lorsque la touche <Entrée> est appuyée.

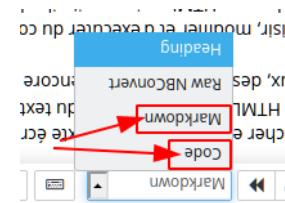
```
[ ]: saisie = input()
print("la chaîne de caractère saisie est:", saisie)
```

La chaîne de caractère ainsi récupérée pourra être convertie, si besoin, en nombre entier en utilisant l'instruction `int`.

Par exemple le programme ci-dessous demande l'âge de l'utilisateur. Le nombre saisi est converti en nombre entier puis un calcul est effectué :

```
[ ]: texte = input("ton âge ?")
age = int(texte)
print("Dans 10 ans, tu auras", age+10)
```





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.

Pour ajouter des blocs (de code ou de texte) :

Souvent quand on ouvre un notebook, aucun bloc de code n'a été exécuté. Il faut donc passer dans l'ordre par tous les blocs de code et les exécuter. C'est pour quoi, quand on lit notebook, il est pratique d'utiliser la touche **Shift** + **Entrée** qui permet de passer d'un bloc au suivant.

REMARQUE

- utiliser les flèches **↑** et **↓** pour sélectionner le bloc de texte à modifier/corriger le texte
- appuyer sur **Entrée** pour éditer le bloc différent
- utiliser les entrées deux chaines de caractères.
- pour interpréter le résultat tu peux utiliser le choix :
- **Ctrl** + **Entrée** pour interpréter le bloc ou
- **Shift** + **Entrée** pour interpréter le bloc au suivant.

REMARQUE

Pour afficher plusieurs expressions sur une seule ligne, il suffit d'utiliser

Ce programme affiche deux entiers à l'écran, sur deux lignes :

CORRECTION

```
a = 34
b = 21 + a
print(a)
print(b)
```

Détaillez ce que produit l'exécution du programme suivant :

ACTIVITE

Un programme est généralement constitué de plusieurs instructions. Chaque instruction est écrite sur une ligne.

2.4 – Séquence d'instructions

La chaîne et un entier est définie et est équivalente à concatener n fois "Hello" + "Hello" et après interprétation produit comme résultat "HelloHelloHello". La chaîne de caractère. Ici l'expression est équivalente à "Hello" + "Hello" + "Hello" * 3 est une expression valide car l'opération * entre une chaîne et un entier est définie et est équivalente à concatener n fois "Hello" + "Hello" + "Hello".

 ACTIVITÉ

Accède à la version *notebook* de ce chapitre (<https://pa.dilla.fr/15>).

1. **Corrige** la faute d'orthographe de ce bloc de texte.
2. **Ajoute** juste en dessous un bloc de code et **écris** `y = 1+2`.
3. Puis fait **calculer** le résultats par l'interprète Python.

1.2 – Arithmétique

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

Exemple

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

REMARQUE

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. Par exemple :

$$1+2 \quad * \quad 3$$
 renvoie 7 et non pas 9!

L'instruction `print("Bonjour tout le monde")` affiche le message Bonjour tout le monde à l'écran.

Le texte écrit entre guillemets est appelé une **chaîne de caractères**.

REMARQUE

Les guillemets englobants ne sont pas affichés.

Les guillemets peuvent être doubles "..." ou simples '...'.

Les guillemets ouverts doivent être impérativement fermés sinon il y aura une exception (erreur) de type `syntaxError`.

Une chaîne de caractère est arbitraire et n'est pas interprétée par Python.

 ACTIVITÉ

Indiquer ce qu'affichent les deux instructions suivantes :

- `print(1+3)`
- `print("1+3")`

Attention aux opérations avec les chaînes de caractères.

- "Hello" + 2 est une expression **invalidé** car l'opération + entre une chaîne et un entier n'a aucun sens.
- "Hello" + "World" est une expression **valide** car l'opération + entre deux chaînes de caractères les concatènent (c'est-à-dire les assemblent et produisent une chaîne de caractère contenant le texte "HelloWorld" (sans espace))
- "Hello" * "World" est une expression **invalidé** car l'opération * n'a pas

metz.

On peut donner à l'instruction print un message à afficher, écrit entre guillemets.

2.3 – Affichage des textes

- soit des **entiers relatifs** (appelés entiers)
 - soit des **entiers décimaux** (appelés flottants).
- En Python, les nombres sont de deux types :

1.3 – Les nombres

Et en Python ?

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

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



CORRECTION

Dans le bloc de code ci-dessous, écris 1 + * 2.
Indique le type d'erreur affiché.



2.1 – VSCode

Cette suite d'instruction s'appelle un programme ou encore un **code source**.

Pour écrire des programmes, le plus simple est d'utiliser un **environnement de développement** (IDE). Cette année, nous allons essentiellement utiliser VSCode.

VSCode.

- en mode sans débogage : <Ctrl> + <F5>
- en mode débogage : <F5>

Puis une fois votre programme écrit, pour l'exécuter :

- Sauvegarder le <Ctrl> + <S> en donnant un nom qui a pour extension .py (par exemple test.py).
- Après avoir ouvert le logiciel, créer un nouveau fichier <Ctrl> + <N>.

2.2 – Affichage

En mode programme, les résultats calculés ne sont plus affichés. Il faut utiliser pour cela une instruction d'affichage.

En Python, elle s'appelle print. Ainsi le programme test.py contenant l'unique ligne print(3) affiche 3.

L'instruction print admet une expression arbitraire. Elle commence d'abord par calculer le résultat de cette expression puis l'affiche à l'écran.

Par exemple l'instruction print(1+3) calcule d'abord l'expression 1+3 puis affiche 4 à l'écran.

2.3 – Affichage des textes

On peut donner à l'instruction print un message à afficher, écrit entre guillemets.

- soit des **entiers relatifs** (appelés entiers)
 - soit des **entiers décimaux** (appelés flottants).
- En Python, les nombres sont de deux types :



REMARQUE

Attention, les nombres flottants s'écrivent à l'*anglo-saxonne* avec un point 3.14 à la place de la virgule 3,14.

ACTIVITÉ

Indique ce qu'affiche les commandes suivantes :

1. Commande 1 : `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**ACTIVITÉ**

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

Indique les états suivants après l'exécution des trois dernières instructions.

CORRECTION

L'instruction `e = b + c` affecte la variable `e` pour la première fois. Celle-ci fait donc son apparition dans l'état :

a	b	c	d	e
2	2	3	-12	5

L'instruction `d = a` affecte à `d` la valeur pointée par `a`. Il est important de comprendre que `a` et `d` sont deux emplacements différents et donc indépendants. Ainsi, l'instruction suivante n'aura pas d'effet sur la variable `d`

a	b	c	d	e
2	2	3	2	5

La dernière instruction `a = 7` affecte à la variable `a` la valeur 7.

a	b	c	d	e
7	2	3	2	5

On peut voir une modélisation de l'état sur le site suivant : <https://pythontutor.com>

2 – Le mode programme

Le mode programme de Python consiste à écrire une suite d'instructions dans un fichier et à les faire exécuter par l'interprète Python.

ACTIVITÉ

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

Pour effectuer la division entière, il faut utiliser l'opération // . Ainsi, $a // b$ peut être vu comme le plus grand nombre entier inférieur ou égal à a/b .

// Z donne 3 et il reste 1. Legalite / = 3 x Z + 1 est verifiee.

CORRECTION

Effectue à la main les divisions euclidiennes 7/2 et 98/10.

ACTIVITÉ

La division avec l'opération / donne un nombre mortel. 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.

CORRECTION

Après la première instruction `a = c - a`, l'état initial devient alors :

$$\begin{aligned}a &= 7 \\d &= a \\e &= b + c \\a &= c - a\end{aligned}$$

ACTIVITÉ

A partir de l'état initial [1|2|3|12], les instructions suivantes sont exécutées :

Lettat évolue en fonction des instructions exécutées. Les instructions qui modifient les variables d'état sont affectées par la variable `last`.

L'ensemblé des associations entre des noms de variables et des valeurs

Copyright © 2014 by Pearson Education, Inc.

```
cube = a * a * a  
ma_var1ab1e = 42  
ma_var1ab1e2 = 2021
```

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

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

REMARQUE

CORRECTION

Utiliser quatre blocs de code pour afficher les résultats des calculs suivants :

```
7//2
7 % 2
98 // 10
98 % 10
```

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

Ainsi, 10^3 s'écrit $10**3$.

ACTIVITÉ

Calcule dans un bloc de code 2^{1024} puis dans un autre bloc de code : $(2.0)^{1024}$.

Propose une explication à tes observations.

CORRECTION

2^{1024} est un nombre entier et se calcule correctement.

En revanche, 2.0^{1024} est un nombre flottant trop grand pour être calculé. `OverflowError` indique que le résultat est hors intervalle.

1.5 – Variables

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

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

Une variable peut être imaginée comme un **emplacement en mémoire** portant une **étiquette** et contenant une **valeur**.

Exemple

`a = 2` se représente par un emplacement `a` contenant la valeur `2` : 2

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
```

Le bloc ci-dessous affiche le résultat du calcul `a * (a + 1)` :

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

Le symbole `=` utilisé en Python pour définir la variable `a` désigne une opération d'**affectation**. En algorithme, on note l'affectation par le symbole '`←`', ce qui donne par exemple : `a ← 2`.

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
```