

- 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 :

REMARQUE

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

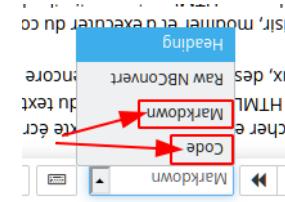
REMARQUE

- utiliser les flèches \leftarrow et \rightarrow pour sélectionner le bloc de texte à modifier/corriger le texte
- appuyer sur $\langle \text{Entrée} \rangle$ pour éditer le bloc différent
- utiliser les touches $\langle \uparrow \rangle$ et $\langle \downarrow \rangle$ pour sélectionner le bloc de texte à modifier/corriger le texte
- pour interpréter le résultat tu peux utiliser au choix :

 - $\langle \text{Ctrl} \rangle + \langle \text{Entrée} \rangle$ pour interpréter le bloc ou suivant.
 - $\langle \text{Shift} \rangle + \langle \text{Entrée} \rangle$ pour interpréter le bloc et passer au bloc suivant.



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



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.

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

CORRECTION

Détaillez ce que produit l'exécution du programme suivant :

ACTIVITE

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

2.4 – Séquence d'instructions

Une chaîne est généralement constituée plusieurs instructions. Chaque instruction est écrite sur une ligne.

La chaîne d'instruction est généralement interprétée comme produit de caractères. Ici l'expression est équivalente à "Hello" + "Hello" + "Hello".

"Hello" * 3 est une expression valide car l'opération * entre une chaîne et un entier est définie et est équivalente à concaténer n fois la chaîne de caractères. La chaîne "HelloWorld" (sans espace) est équivalente à "Hello" + "World".

Les chaînes de caractères sont assemblées et produisent une chaîne de caractères contenant le texte "HelloWorld" (sans espace).

Les chaînes de caractères sont valides car l'opération * n'a pas de sens entre chaînes de caractères.

"Hello" * "World" est une expression invalide car l'opération * n'a pas de sens entre chaînes de caractères.

"Hello" + "World" est une expression valide car l'opération + entre deux chaînes de caractères les concatène (c'est-à-dire les assemble et produisent une chaîne de caractères contenant le texte "HelloWorld" (sans espace)).


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

2.3 – Affichage des textes

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

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 **invalidée** car l'opération + entre une chaîne et un entier n'a aucun sens.

1.3 – Les nombres

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

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

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 `1:1,2 + 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.

CORRECTION

L'instruction `a = c - a` modifie la valeur de `a` en fonction des valeurs de `c` et de `a`. L'état sera :

a	b	c	d
2	2	3	-12

Puis 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>

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

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

$98/10$ donne 9 et il reste 8. L'égalité $98 = 9 \times 10 + 8$ est vérifiée.
 $7/2$ donne 3 et il reste 1. L'égalité $7 = 3 \times 2 + 1$ est vérifiée.

CORRECTION

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

ACTIVITÉ

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 float car ils sont écrits avec une virgule.

CORRECTION

Détermine le type de nombre que l'on obtient lorsqu'on divise deux nombres entiers.

ACTIVITÉ

1.4 – Autres opérations

Après l'exécution de chaque instruction, écrire le nouvel état de l'interpréteur.

$a = 7$

$d = a$

$e = b + c$

$a = c - a$

$b = b$

$trees :$

À partir de l'état $\boxed{1} \boxed{2} \boxed{3} \boxed{4}$, les instructions suivantes sont exécutées :



L'état évolue en fonction des instructions exécutées. Les instructions qui modifient l'état sont dites à effet de bord.

constitue l'état de l'interpréteur Python.

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

1.6 – Etat

$x = 1$ se représente par un emplacement x contenant la valeur 1 : $\boxed{1}$

Exemple

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

```
[ ] : cube = a * a * a
ma_variabLe2 = 42
ma_variabLe = 2021
```




ACTIVITÉ

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

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
```

La notation `a =` permet de donner un nom à l'information mémorisée. Ici, l'interpréteur 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 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
```

REMARQUE

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

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