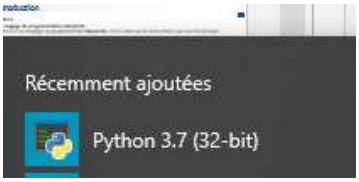
PYTHON - Variables

Installation python sous windows (pour les PC personnels)

Pour l'installation de Python, quel que soit votre système d'exploitation, vous devez vous rendre sur le **site officiel de Python** : https://www.python.org/

Sous Windows

1. Cliquez sur le lien Download dans le menu principal de la page.
2. Sélectionnez la version de Python que vous souhaitez utiliser (je vous conseille la dernière en date).
3. Enregistrez puis exécutez le fichier d'installation et suivez les étapes.
4. Une fois l'installation terminée, vous pouvez vous rendre dans le menu Démarrer>Tous les programmes.
   Python devrait apparaître dans cette liste



Premiers pas sur l'interpréteur de commandes

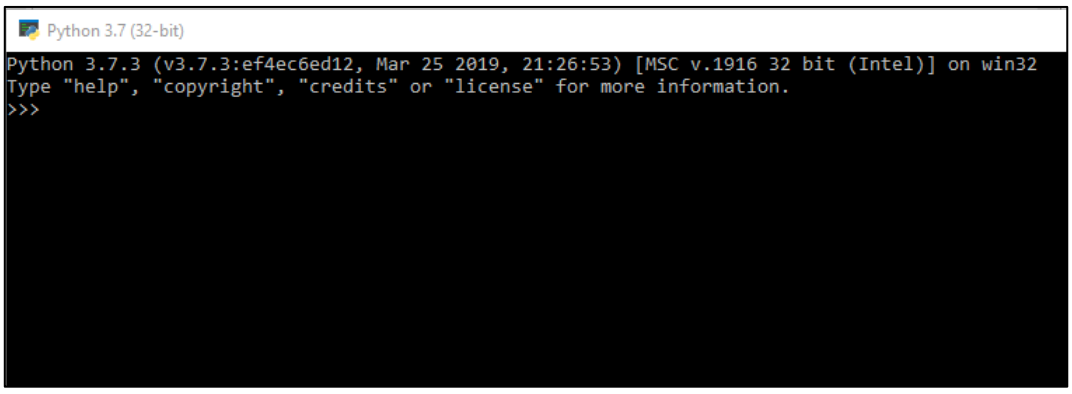
Lancer Python sous Windows

Nous allons désormais utiliser Python sous Windows.

Vous avez plusieurs façons d'accéder à la ligne de commande Python

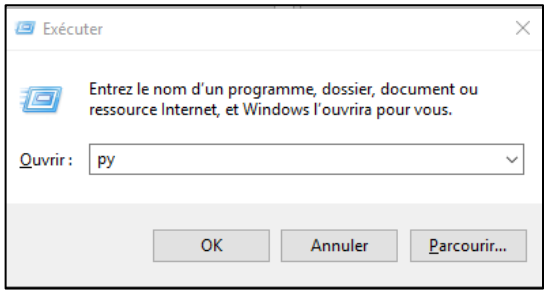
1- la plus évidente consiste à passer par les menus Démarrer>Tous les programmes>Python 3.7>Python.

vous obtenez une console d'interprétation de Python.



2- Vous pouvez également passer par la ligne de commande Windows en tapant le raccourci Windows + R

Dans la fenêtre qui s'affiche, **tapez « py »** et la ligne de commande Python devrait s'afficher de nouveau.



Pour **fermer l'interpréteur** de commandes Python, vous pouvez taper « **exit()** » puis appuyer sur la touche **Entrée**.

Interpréteur de commandes de Python

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

L'interpréteur de commandes va nous permettre de tester directement du code.

- Je saisis une ligne d'instructions,
- j'appuie sur la touche Entrée de mon clavier,
- je regarde ce que me répond Python (s'il me dit quelque chose),
- puis j'en saisis une deuxième,
- une troisième…

Cet interpréteur est particulièrement utile pour comprendre les bases de Python et réaliser nos premiers programmes.

la série de trois chevrons (>>>) signifient : « je suis prêt à recevoir tes instructions ».

Premières instructions

1) Saisir un nombre

Vous avez pu voir sur notre premier test que Python n'aimait pas les suites de lettres qu'il ne comprend pas. Par contre, l'interpréteur aime les nombres :

```
>>> 7
7
>>>
```

ce retour indique que l'interpréteur a bien compris et que votre saisie est en accord avec sa syntaxe

vous pouvez saisir des nombres à virgule : on utilise ici la notation anglo-saxonne, c'est-à-dire que le point remplace la virgule :

```
>>> 9.5
9.5
>>>
```

2) Opérations courantes

Addition, soustraction, multiplication, division

Pour effectuer ces opérations, on utilise respectivement les symboles +, -, * et /.

```
>>> 3 + 4
7
>>> -2 + 93
91
>>> 9.5 + 2
11.5
>>> 3.11 + 2.08
5.1899999999999995
>>>
```

Division entière et modulo

Pour la division, le résultat est donné avec une virgule flottante

```
>>> 10 / 5
2.0
>>> 10 / 3
3.3333333333333335
>>>
```

Veuillez remarquer ce qui est la règle dans tous les langages de programmation, à savoir que les conventions mathématiques de base sont celles qui sont en vigueur dans les pays anglophones :

le séparateur décimal est donc toujours un point, et non une virgule comme chez nous.

```
>>> 20.5 / 3
>>> 8,7 / 5 # Erreur !
>>>
```

Il existe deux autres opérateurs qui permettent de connaître le résultat d'une division entière et le reste de cette division.

Le premier **opérateur** utilise le symbole **« // ».** Il permet d'obtenir la partie entière d'une division

```
>>> 10 // 3
3
>>>
```

La partie entière de la division de 10 par 3 est le résultat de cette division, sans tenir compte des chiffres au-delà de la virgule (en l'occurrence, 3)

L'opérateur « % », que l'on appelle le « modulo », permet de connaître le reste de la division

```
>>> 10%3
1
>>>
```

Pour obtenir le modulo d'une division, on « récupère » son reste. Dans notre exemple, 10/3 = 3 et il reste 1

Puissance

En Python, il est possible de calculer des puissances avec l'opérateur **

```
>>> 5**2
25
>>> 2 ** 7
128
>>>
```

seule limite, la capacité de la machine (à 2**1000000 on la plante!)

Hiérarchie des opérations

Lorsqu'il y a plus d'un opérateur dans une expression, l'ordre dans lequel les opérations doivent être effectuées dépend de règles de priorité.

Sous Python, les règles de priorité sont les mêmes que celles qui vous ont été enseignées au cours de mathématique.

```
>>> 7 + 3 * 4
>>> (7+3)*4
>>>
```

Les Variables

Une variable

une variable est une petite **information temporaire** qu'on stocke dans la mémoire vive (RAM). On dit qu'elle est **« variable » car c'est une valeur qui peut changer** pendant le déroulement du programme.
Par exemple, un nombre de vies restant au joueur (5) risque de diminuer au fil du temps. Si ce nombre atteint 0, on saura que le joueur a perdu.

En langage Python, une **variable** est constituée de deux choses :

- une **valeur** : c'est le nombre qu'elle stocke (par exemple 5) ;
- un **nom** : c'est ce qui permet de la reconnaître.

En programmant en Python, on n'aura pas à retenir l'adresse mémoire : à la place, on va juste indiquer des noms de variables.

Python fera la **conversion entre le nom de la variable et l'adresse mémoire**.

Noms des variables

En langage Python, chaque variable doit donc avoir un **nom**

1) Contraintes

- il ne peut y avoir que des **lettres minuscules**, **majuscules** (a → z , A → Z) et des **chiffres** (0 → 9)
- Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les caractères spéciaux tels que $, #, @, etc. sont interdits
- votre nom de variable doit **commencer par une lettre**
- les **espaces** sont **interdits**. À la place, on peut utiliser le caractère « **underscore** » _ (qui ressemble à un trait de soulignement). C'est le seul caractère différent des lettres et chiffres autorisé
- **passer en majuscule le premier caractère de chaque mot**, à l'exception du premier mot constituant la variable.
  La variable contenant mon âge se nommerait alors **monAge**.
- le langage Python fait la différence entre les majuscules et les minuscules, il **respecte la « casse »,** ce qui signifie que des lettres majuscules et minuscules ne constituent pas la même variable (la variable AGE est différente de aGe, elle-même différente de age).

Voici quelques exemples de noms de variables corrects

nombreDeVies, prenom, nom, numero_de_telephone, numeroDeTelephone

2) Mot-clés réservés

Enfin, certains mots-clés de Python sont **réservés**, c'est-à-dire que vous ne pouvez pas créer des variables portant ce nom

voici la liste pour Python 3 :

and	del	from	none	true
as	elif	global	nonlocal	try
assert	else	if	not	while
break	except	import	or	with
class	false	in	pass	yield
continue	finally	is	raise	
def	for	lambda	return	

Affecter une valeur à une variable

En Python, pour donner une valeur à une variable, il faut écrire **nom_de_la_variable = valeur**.

```
>>> mon_age = 21
```

On dit en effet qu'on a **affecté** la valeur 21 à la variable mon_age

NB : Les espaces séparant « = » du nom et de la valeur de la variable sont facultatifs

Afficher la valeur d'une variable

On peut afficher la valeur d'une variable en la saisissant simplement dans l'interpréteur de commandes

puis <Enter>.

Python répond en affichant la valeur correspondante

```
>>> mon_age
21
>>>
```

Si vous modifiez la variable puis que vous la réaffichez, la valeur de la variable aura changé :

```
>>> mon_age = mon_age + 2
>>> mon_age
23
>>>
```

Affecter à d'autres variables

On peut également attribuer à une variable la valeur d'une autre variable

Par exemple :

```
>>> toto = mon_age
>>> toto
23
>>>
```

Signifie que l'on attribue la valeur de la variable mon_age à la variable toto

Signifie que la *valeur* de toto est maintenant celle de mon_age.

Notez bien que cette instruction n'a en rien modifié la valeur de mon_age

Règles :

- Si la variable n'est pas créée, Python s'en charge automatiquement.
- Si la variable existe déjà, l'ancienne valeur est supprimée et remplacée par la nouvelle

Vous pouvez aussi affecter à d'autres variables des valeurs obtenues en effectuant des calculs sur la première variable

essayons d'affecter une valeur à une autre variable d'après la valeur de mon_age

```
>>> toto = mon_age * 2
>>> toto
46
>>>
```

Comme mon_age contenait 23, toto vaut maintenant 46.

Et de même que précédemment, mon_age vaut toujours 23.

Les types de données

Python a besoin de connaître quels types de données sont utilisés pour savoir quelles opérations il peut effectuer avec.

Python associe à chaque donnée un type, qui va définir les opérations autorisées sur cette donnée.

1) Les nombres entiers

Python différencie les entiers, des nombres à virgule flottante.

Pour un ordinateur, les opérations que l'on effectue sur des nombres à virgule ne sont pas les mêmes que celles sur les entiers, et cette distinction reste encore d'actualité.

Le type entier se nomme **int** (qui correspond à l'anglais « integer », c'est-à-dire entier).

La forme d'un entier est un nombre sans virgule

```
>>> 3
```

2) Les nombres flottants

Les flottants sont les nombres à virgule.

Ils se nomment **float** (ce qui signifie « flottant » en anglais).

La syntaxe d'un nombre flottant est celle d'un nombre à virgule (n'oubliez pas de remplacer la virgule par un point).

Si ce nombre n'a pas de partie flottante mais que vous voulez qu'il soit considéré par le système comme un flottant, vous pouvez lui ajouter une partie flottante de 0.

```
>>> 3.152
>>> 52.0
```

3) Les booléens

Un booléen est une information Vraie ou Fausse.

Ce type sera très utile lorsque nous commencerons à comparer des valeurs entre elles.

vrai s'écrit **T**rue et faux s'écrit **F**alse (pensez aux majuscules !).

Exemple :

Un est inférieur à deux

```
>>> 1 < 2
True
```

Un est supérieur à 2

```
>>> 1 > 2
False
```

le type booléen est très économique en termes de place mémoire occupée, puisque pour stocker une telle information binaire, un seul bit suffit (1 ou 0).

4) Les chaînes de caractères

La chaîne de caractères permet de stocker une série de lettres, une phrase.

Elle se nomme **str** (abréviation de « string » qui signifie chaîne en anglais).

On peut écrire une chaîne de caractères de différentes façons :

- entre **guillemets** ("ceci est une chaîne de caractères") ;
- entre **apostrophes** ('ceci est une chaîne de caractères') ;
- entre **triples guillemets** ("""ceci est une chaîne de caractères""") ou entre **triples apostrophes** (''')

On peut stocker une chaîne de caractères dans une **variable** :

```
>>> ma_chaine = "Bonjour !"
>>> ma_chaine
'Bonjour!'
```

Echappement

Si vous utilisez les délimiteurs simples (le guillemet ou l'apostrophe) pour encadrer une chaîne de caractères, il se pose le problème des guillemets ou apostrophes que peut contenir ladite chaîne.

```
>>> chaine = 'J'aime le Python!'
SyntaxError: invalid syntax
>>>
```

vous obtenez une message d'erreur SyntaxError

Ceci est dû au fait que l'apostrophe de « J'aime » est considérée par Python comme la fin de la chaîne et qu'il ne sait pas quoi faire de tout ce qui se trouve au-delà.

Pour pallier ce problème, il faut **échapper** les apostrophes se trouvant au cœur de la chaîne.

On **insère** un **caractère anti-slash « \ » avant** les apostrophes contenues dans le message

```
>>> chaine = 'J\'aime le Python!'
>>> chaine
"J'aime le Python!"
>>>
```

Pour pallier ce problème, on peut aussi alterner intelligemment entre les **guillemets** et les **apostrophes** :

```
>>> chaine2 = "J'aime le Python!"
>>> chaine2
"J'aime le Python!"
```

On doit également échapper les guillemets si on utilise les guillemets comme délimiteurs

```
>>> chaine3 = "\"Le seul individu formé, c'est celui qui a appris comment apprendre (...)\" (Karl Rogers, 1976)"
>>> chaine3
'"Le seul individu formé, c\'est celui qui a appris comment apprendre (...)" (Karl Rogers, 1976)'
```

Enfin, pour écrire un véritable anti-slash dans une chaîne, il faut l'échapper lui-même (et donc écrire « \\ »).

```
>>> chaine4 = "Python\variables"
SyntaxError: invalid syntax
>>> chaine4 = "Python\\variables"
>>> chaine4
'Python\\variables'
```

Saut de ligne « \n »

Le caractère « \n » symbolise un saut de ligne

```
>>> chaine="essai\nsur\nplusieurs\nlignes"
>>> chaine
'essai\nsur\nplusieurs\nlignes'
>>>
```

Pour l'instant, l'interpréteur affiche les sauts de lignes comme on les saisit, c'est-à-dire sous forme de « \n ».

Nous verrons plus tard comment afficher réellement ces chaînes de caractères.

5) Typage dynamique

On notera donc que sous Python, il n'est pas nécessaire d'écrire des lignes de programme spécifiques pour définir le type des variables avant de pouvoir les utiliser. Il vous suffit en effet d'assigner une valeur à un nom de variable pour que celle-ci soit **automatiquement créée avec le type qui correspond au mieux à la valeur fournie**

On dira à ce sujet que le typage des variables sous Python est un **typage dynamique**, par opposition au typage statique qui est de règle par exemple en C++ ou en Java

Incrémentation

L'incrémentation désigne l'augmentation de la valeur d'une variable d'un certain nombre

`variable = variable + 1`	Cette syntaxe est claire et intuitive mais assez longue
`variable += 1`	L'opérateur+=revient à ajouter à la variable la valeur qui suit l'opérateur.

Les opérateurs -=, *= et /= existent également, bien qu'ils soient moins utilisés

```
>>> a = 2
>>> a += 1
>>> a
3
>>> a += 3
>>> a
6
```

Permutation

Python propose un moyen simple de permuter deux variables (échanger leur valeur)

```
>>> a = 5
>>> b = 32
>>> a,b = b,a # permutation
>>> a
32
>>> b
5
>>>
```

après l'exécution de la ligne 3, les variables a et b ont échangé leurs valeurs.

Affectation à plusieurs variables

On peut aussi affecter une même valeur à plusieurs variables

```
>>> x = y = 3
>>> x
3
>>> y
3
>>>
```

On peut aussi effectuer des affectations parallèles à l'aide d'un seul opérateur

```
>>> a,b,c = 4, 8.33, "bonjour"
>>> a
4
>>> b
8.33
>>>
```

Dans cet exemple, les variables a et b prennent simultanément les nouvelles valeurs 4 et 8,33 et la variable c prend la valeur "bonjour"

Commentaires

Les commentaires sont des messages qui sont ignorés par l'interpréteur et qui permettent de donner des indications sur le code.

En Python, un commentaire

- débute par un dièse (« # »)
- se termine par un saut de ligne.

Tout ce qui est compris entre ce # et ce saut de ligne est ignoré.

Un commentaire peut donc occuper

- la totalité d'une ligne (on place le # en début de ligne)
- une partie seulement, après une instruction (on place le # après la ligne de code pour la commenter plus spécifiquement)

```
>>> # Premier exemple
>>> a = 5
>>> b = 6 # Deuxieme exemple
>>>
```

Constantes

Les variables dont la valeur ne change jamais au cours du temps, au cours de l'exécution d'un programme, s'appellent **variables constantes,** ou tout simplement **constantes.**

Par convention, les constantes sont écrites tout en **majuscule**

```
>>> A = 12
>>> B = 14
>>>
```