PYTHON – Input Print

Les fonctions intégrées

Utiliser une fonction

Une fonction exécute un certain nombre d'instructions déjà enregistrées.

C'est donc un groupe d'instructions écrites pour faire une action précise et auquel vous donnez un nom.

Vous n'avez plus ensuite qu'à appeler cette fonction par son nom, autant de fois que nécessaire (cela évite les répétitions).

La plupart des fonctions ont besoin d'au moins un paramètre pour travailler sur une donnée ; ces paramètres sont des informations que vous passez à la fonction afin qu'elle travaille dessus.

Les fonctions s'utilisent en respectant la syntaxe suivante :
nom_de_la_fonction(parametre_1,parametre_2,…,parametre_n).

- Vous commencez par écrire le **nom** de la fonction.
- Vous placez entre **parenthèses** les **paramètres** de la fonction. Si la fonction n'attend aucun paramètre, vous devrez quand même mettre les parenthèses, sans rien entre elles.

La fonction « type »

La fonction « type » permet de savoir de quel type est une variable

Syntaxe : `type(nom_de_la_variable)`

La fonction renvoie le type de la variable passée en paramètre

Si vous saisissez dans l'interpréteur les lignes suivantes

```
>>> a = 6
>>> type(a)
```

Python vous indique donc que la variable est de type entier :

```
<class 'int'>
```

Vous pouvez faire le test sans passer par des variables

```
>>> type(6.4)
<class 'float'>
>>> type("un essai de type")
<class 'str'>
>>>
```

La fonction « print »

Le fonction « print » permet d'afficher la valeur d'une ou plusieurs variables

NB : L'interpréteur affiche déjà bien la valeur de la variable car il affiche automatiquement tout ce qu'il peut, pour pouvoir suivre les étapes d'un programme.

Cependant, quand vous ne travaillerez plus avec l'interpréteur, taper simplement le nom de la variable n'aura aucun effet. De plus, l'interpréteur entoure les chaînes de caractères de délimiteurs et affiche les caractères d'échappement, tout ceci encore pour des raisons de clarté

La fonction print est dédiée à **l'affichage** uniquement.

Le nombre de ses paramètres est variable, c'est-à-dire que vous pouvez lui demander d'afficher une ou plusieurs variables

```
>>> a = 3
>>> print(a)
3
>>> a = a + 3
>>> b = a - 2
>>> print("a =", a, "et b =", b)
a = 6 et b = 4
```

Le premier appel à print se contente d'afficher la valeur de la variable a, c'est-à-dire « 3 »

Dans le deuxième appel à print, on passe quatre paramètres : deux chaînes de caractères et les variables a et b. Quand Python interprète cet appel de fonction, il va afficher les paramètres dans l'ordre de passage, en les séparant par un espace

La fonction « print » associée aux mot-clés sep et end :

- sep : permet de séparer chaque paramètre (espace ' ' par défaut)
- end : permet de terminer l'affichage (saut de ligne \n par défaut)

```
>>> print("Voici", "un", "exemple", "d'appel", sep=" % ", end=" -\n")
Voici % un % exemple % d'appel -
>>>
```

Les variables sont toutes séparées par la valeur de ***sep*** (pourcent) et l'affichage se termine par la valeur de ***end*** (tiret puis saut de ligne)

Fonction pour convertir une variable vers un autre type

Pour convertir une variable vers un autre type, il faut utiliser le nom du type comme une fonction

int() : Permet de convertir une valeur en un nombre entier.

str() : Permet de convertir une valeur en chaîne de caractère

float() : Permet de convertir une valeur en nombre flottant

Exemple :

```
>>> str = "1234"
>>> nb = int(str)
>>> str
'1234'
>>> nb
1234
```

La fonction « input »

input() est une fonction qui va permettre d'interagir avec l'utilisateur.

input() accepte un paramètre **facultatif** : le message à afficher à l'utilisateur.

```
>>> nombre = input("Saisissez un nombre : ")
Saisissez un nombre : 9
```

Dès que le programme rencontre une instruction **input**, l'exécution s'interrompt, affiche le message (Saisissez un nombre ici) et attend que l'utilisateur frappe une valeur sur son clavier.

L'interruption peut durer quelques secondes, quelques minutes ou plusieurs heures : la seule chose qui fera exécuter la suite des instructions, c'est que la touche **Entrée** ait été enfoncée.

Aussitôt que c'est le cas, il se passe deux choses.

- tout ce qui a été frappé avant la touche Entrée (une suite de lettres, de chiffres, ou un mélange des deux) est rentré dans la variable qui suit l'instruction input (ici, la variable *nombre* récupère la valeur *9*).
- Et ensuite, immédiatement, la machine exécute la suite des instructions (s'il y en a).

La fonction **input() renvoie toujours une chaîne de caractères**.

Si vous souhaitez que l'utilisateur entre une valeur numérique, vous devrez donc convertir la valeur entrée (qui sera donc de toute façon de type string) en une valeur numérique du type qui vous convient, par l'intermédiaire des fonctions intégrées **int()** (si vous attendez un entier) ou **float()** (si vous attendez un réel).

```
>>> annee = input("Entrez une année : ")
Entrez une année : 2019
>>> type(annee)
<class 'str'>
>>> annee
'2019'
>>> # On veut convertir la variable en un entier,
>>> annee = int(annee)
>>> type(annee)
<type 'int'>
```

Générer des nombres aléatoires

Python est capable de générer des nombres aléatoires.

Pour utiliser cette fonctionnalité il faut **importer le module random** de Python. (Nous verrons plus tard ce qu'est un module, pour l'instant, contentez-vous de l'importer)

Ce module contient, entre autres, la **fonction randint(a, b)** qui permet de générer un entier compris entre a (inclus) et b (inclus).

Exemple :

```
>>> import random
>>> random.randint(0, 9)
8
>>> random.randint(0, 9)
5
>>> random.randint(0, 9)
0
>>> random.randint(0, 9)
2
>>>
```

Autres fonctions intégrées

Une liste complète est accessible ici : https://docs.python.org/3/library/functions.html

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Écrire les programmes Python dans des fichiers

l'interpréteur est très pratique :

il propose une manière interactive d'écrire un programme, qui permet de tester le résultat de chaque instruction.

Mais l'interpréteur a aussi un défaut :

le code que vous saisissez est effacé à la fermeture de la fenêtre (pas de sauvegarde).

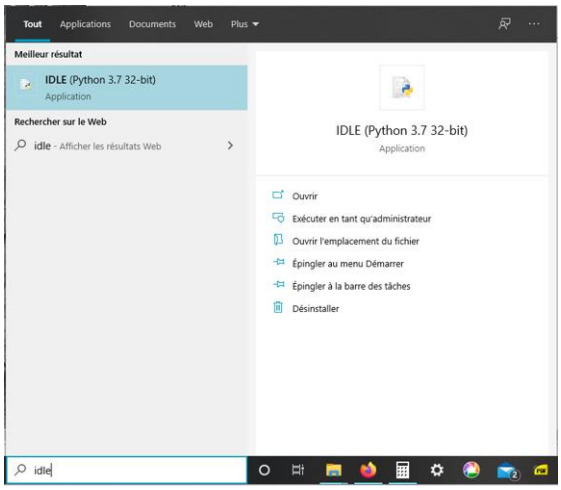
Or, si on rédiger des programmes relativement complexes, devoir réécrire le code entier de son programme à chaque fois qu'on ouvre l'interpréteur de commandes est assez lourd…

La solution ?

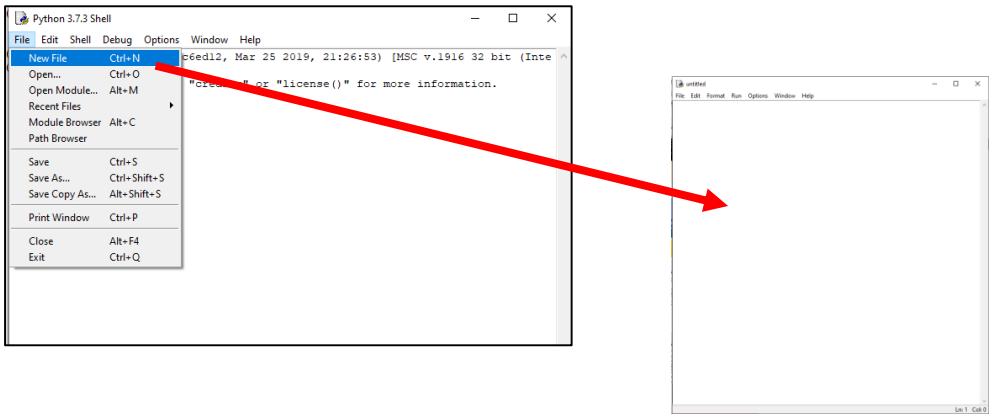
Mettre notre code dans un fichier que nous pourrons lancer à volonté, comme un véritable programme

Voici la démarche :

1. Ouvrez la console IDLE



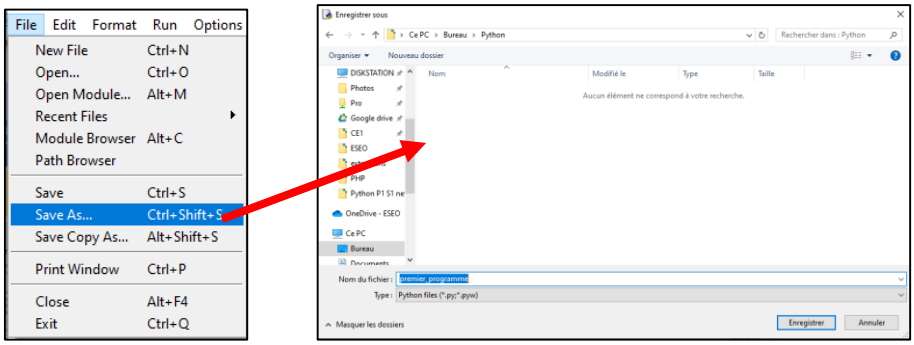
2. Dans le menu File, on choisit l'entrée *New File* (raccourci: Ctrl+N). Une nouvelle fenêtre s'ouvre.



3. On commence par sauvegarder ce fichier script : Dans le menu File, on choisit l'entrée *Save AS* (raccourci: Ctrl+Shift+S). L'enregistrer dans **votre** dossier Python sous le nom que vous avez choisi : *premier_programme.py* par exemple .

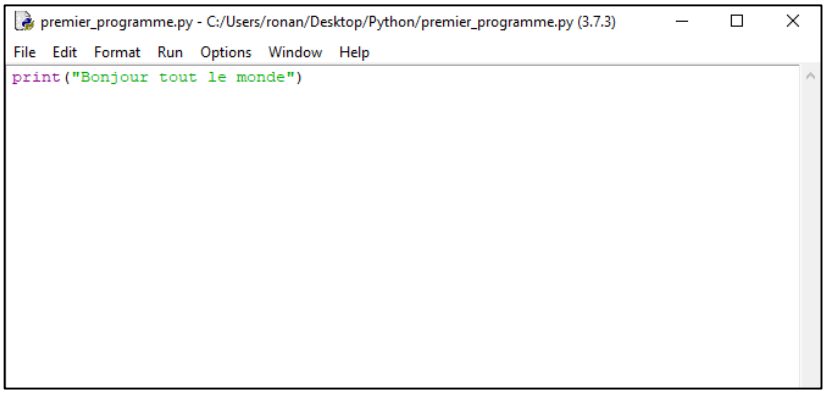
Veillez à organiser votre travail ! (C:/Python/3-input-print/ex1.py)

Et veillez à choisir un dossier de sauvegarde que vous pourrez retrouver d'une séance à l'autre ! (Cloud, clé USB, etc…)

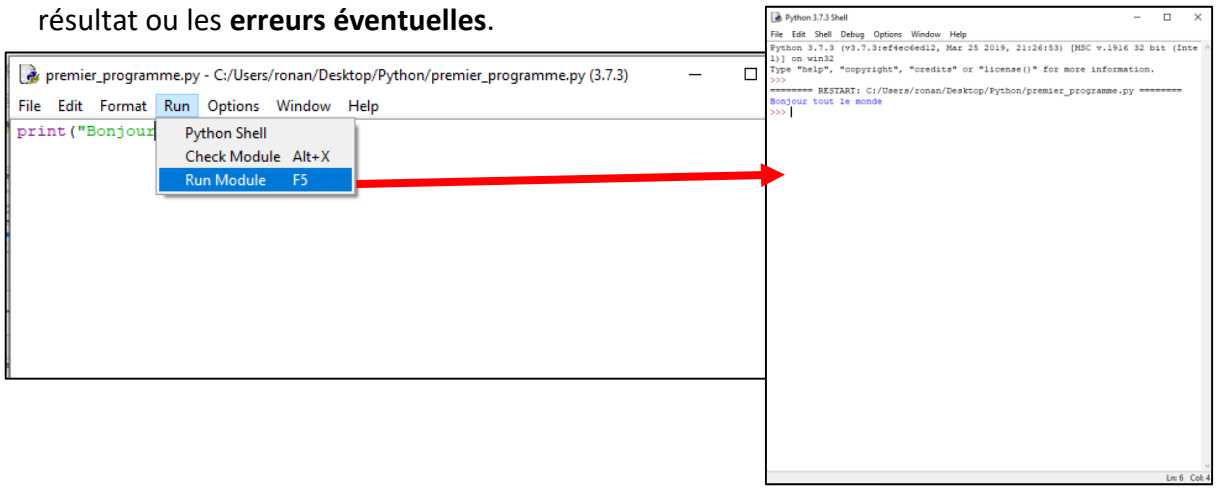


4. On peut alors écrire les instructions dans la fenêtre script.

Exemple: Taper dans la fenêtre script l'instruction ***print("bonjour tout le monde")***



5. Dans le menu Run, on choisit l'entrée *Run Module* (Raccourci F5). On bascule alors dans la console Shell. Le fichier script est alors exécuté et la console reste ouverte pour que vous puissiez voir le résultat ou les **erreurs éventuelles**.



Comment structurer correctement son programme

```
# Programme calcul TVA
# initialisation
TAUXTVA = 0.20
nb = 0
# instructions
```

1/ Un programme commence par le nom du programme et/ou une courte description
 # Programme calcul TVA

2/ on initialise les constantes

TAUXTVA = 0.20

3/ on initialise les variables, c'est-à-dire qu'elles reçoivent une valeur initiale

nb = 0

4/ on écrit les instructions du programme : l'ordre des instructions est primordial car elles sont exécutées dans l'ordre dans lequel elles apparaissent dans le programme. On dit que l'exécution est **séquentielle**.

Une fois que le programme a fini une instruction, il passe à la suivante.

Comment débugger son programme : la trace

La trace d'un programme représente la valeur des différentes informations d'un programme durant son exécution.

Il est indispensable de vérifier la trace d'un programme afin de vérifier qu'il fonctionne, afin de le tester.

1- Choisir les variables sur lesquelles on va effectuer le test

2- effectuer la trace : pour chaque variable testée, calculer la valeur obtenue après chaque instruction du programme

- Soit à la main (sur une feuille)
- Soit en utilisant différents print() sur la variable à tester, tout au long du programme

```
print("test nb : ", nb)
```

3- analyser les résultats obtenus : correspondent-ils à ce que l'on attendait ?