PYTHON

I. INSTALLATION

TÉLÉCHARGEMENT

Si Python n'est pas encore installé sur votre ordinateur, allez sur le site https://edupython.tuxfamily.org/ (taper « edupython » sur Google.) Dans /Téléchargement, téléchargez le fichier d'installation de la dernière version, qui apparaît un premier. Exécuter le Setup. (Soyez patient.)

ALTERNATIVE

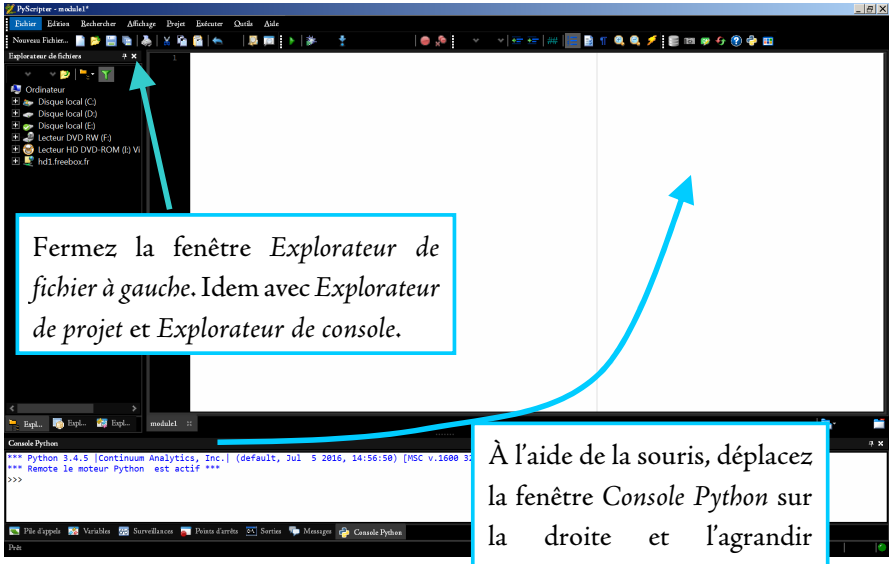
Vous pouvez parfaitement utiliser Python sur un smartphone. Par exemple, sur Android, *Pydroid*. Il suffit de télécharger l'appli. Tous les programmes d'initiation donnés en exercice tournent sur *Pydroid 3*

EXÉCUTION

Exécutez le logiciel *Edupython* par le truchement d'un double clic gauche de la souris sur son icône :  .

On découvre à l'écran un assortiment assez disparate de fenêtres et de menus, pour la plupart inutiles. Je vous recommande de nettoyer tout cela en vous laissant guider par les instructions du paragraphe suivant.

NETTOYAGE



/Affichage/Barre d'état : décocher.

/Outils/Options/Options de l'éditeur

/Afficher :

/Bordure : décocher *Visible*

/Bord droit/Largeur : mettre 0.

/Color Theme : choisir *Black Paste*. Cliquer sur *Apply Theme* puis *Ok*.

/Outils/Options/Options de l'IDE :

/*Autocomplétion du code* (cliquer sur le +) : décocher *Complétion à la saisie*, *Complétion des mots clefs Python* et *Complétion automatique de l'editeur de code*.

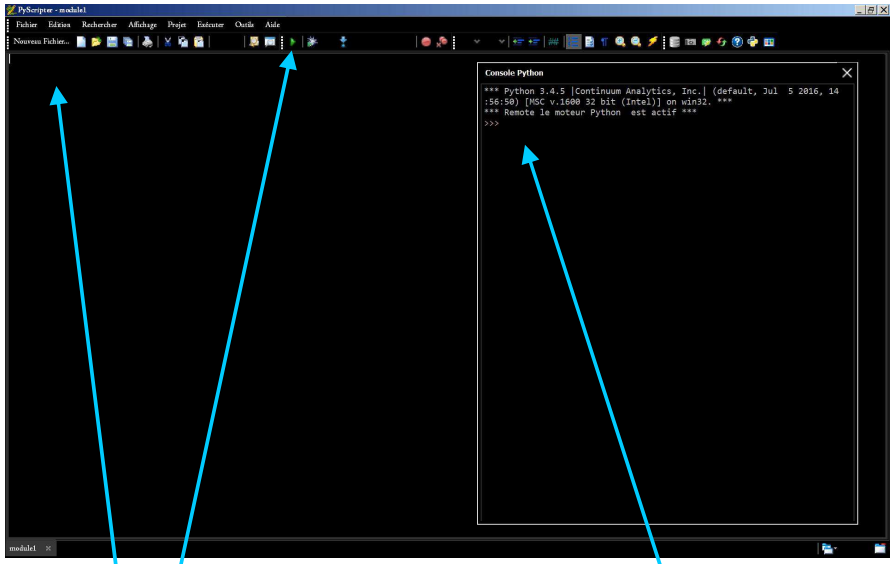
/Console Python : cocher *Effacer la console avant exécution*.

/Editeur : décocher *Afficher les conseils de code*

II. PREMIERS PAS

PREMIER PROGRAMME

Après avoir procédé au nettoyage précédent, vous devriez obtenir à peu près ce qui suit :



Dans l'*éditeur*, taper **`print("Hello World!")`**, puis *Entrée*. Ensuite, aller dans /Exécuter/Exécuter ou bien cliquer sur le triangle vert de la barre menu outil  .

Lors de son *exécution*, le programme affiche la *chaîne de caractères* « Hello World ! » dans la *console*.

Ça y est, vous avez écrit votre premier programme en Python !

UTILISATION DE LA CONSOLE

Si vous n'avez qu'une ligne d'**instruction** à exécuter, vous pouvez la taper directement dans la *console*. Elle sera immédiatement exécutée. (Si la console n'est pas visible, /Affichage /Explorer /Console.)

Vous allez à présent taper dans la console les instructions suivantes (avec *Entrée* en fin de ligne) et observer :

```
>>> a=2
>>> b=3
>>> a+b
>>> print(a+b)
>>> print("a+b")
>>> print("a+b est égal à",a+b)
>>> c=a*b
>>> print(c)
```

Dans la console, au lieu d'écrire `print(c)`, on peut se contenter d'écrire `c` (puis Entrée). Cela ne sera pas possible à l'intérieur d'un programme. Essayez encore les lignes suivantes :

```
>>> 10/3
>>> 5**2
>>> 10<3
>>> a==b
>>> 2==a
```

Essayons à présent la fonction `input()` :

```
>>> input()
>>> a=input()
>>> print(a+a)
>>> a=input("Entrez un nombre entier")
>>> print(a+a)
```

La fonction `input()` renvoie une **chaîne de caractère** saisie par l'**utilisateur**. Même si l'*utilisateur* saisit un nombre, il est vu comme une simple succession de caractères. Pour transformer la *chaîne* en nombre entier, on peut utiliser la **fonction** `int()` :

```
>>> b=int(a)
>>> print(b+b)
```

Faites des essais vous-même, expérimentez en utilisant ce que nous avons vu. Essayez par exemple de voir si la **casse** a de l'importance en Python, c'est-à-dire de voir si Python distingue les majuscules des minuscules. Essayez aussi de voir s'il est possible d'utiliser des noms de variables de plusieurs lettres.

III. Exercices

Go

Sur la page *Python* du site *MathEnSeconde.fr*, la page où vous avez trouvé ce cours, téléchargez le fichier *Programmes d'initiation* que vous décompresserez dans un dossier vous appartenant. Vous trouverez une succession de petits programmes en commentaires desquels se trouvent vos exercices. Prenez-les dans l'ordre.

Les paragraphes qui suivent peuvent être lus avant, pendant ou après les exercices.

IV. Syntaxe

Casse

Comme vous avez pu l'observer, Python distingue les majuscules des minuscules, aussi bien pour les variables que pour le reste. Ainsi, la variable `Nombre` sera distinguée de la variable `nombre`. Et si vous écrivez `Print()` au lieu de `print()`, vous aurez droit à un message d'erreur lors de l'exécution.

Variables

Les **variables** peuvent être formées de plusieurs lettres. On recommande d'ailleurs de leur donner des noms explicites afin de faciliter la lecture du programme.

Commentaires

Dans le texte d'un programme, tout ce qui suit le caractère # (« croisillon » ou « hash ») est considéré comme du commentaire et donc n'est pas pris en compte lors de l'*exécution* du programme.

Égal

Affectation

Le signe = sert à **affecter** une valeur à une variable.

`a=3` a en gros le sens qu'a en mathématiques un : « on pose $a=3$ ».

À gauche du signe *égal*, il y a donc toujours une variable. À droite, en revanche, on peut trouver une constante ou une expression.

`a=a+1` n'est pas une égalité fausse. C'est une instruction qui demande à la machine de calculer la valeur de l'expression `a+1` en utilisant la valeur contenue sur le moment dans la variable `a`. Une

fois cette valeur calculée, elle est affectée à la variable `a`. Ainsi, par cette instruction, la variable a voit sa valeur augmentée d'une unité.

Comparaison

Pour utiliser le signe égal dans le sens qu'il a en mathématiques, on écrit ==.

Par exemple, `2==1+1` renvoie la valeur `True` et 2==1 revoie la valeur `False`.

Puissance

En Python, l'opérateur puissance se note `**`. Ainsi, si vous tapez `2**3` dans la console, vous obtiendrez `8`.

V. Variables

Principe

En Python, on peut créer une variable sans la déclarer et changer son **type** en cours de programme. Ainsi, `a=2` crée la variable entière `a` (si elle n'était pas déjà créée) et lui attribue la valeur 2. Il est possible, juste après, de donner l'instruction `a="Bonjour !"`. Et `a` deviendra alors une chaîne de caractère juste après avoir été un nombre.

Type

Voici les *types* de variables les plus simples. Il y en a beaucoup d'autres et en plus on peut en créer de nouveaux, mais nous n'en sommes pas là.

`int`

De l'anglais *integer*, entier. Il s'agit des entiers relatifs. Les entiers, en Python, n'ont pas vraiment de limite de taille (à ceci près que la quantité de mémoire disponible est limitée). Vous pouvez ainsi

faire des calculs avec de très grands nombres entiers sans que Python ne passe à une valeur approchée, comme elle le ferait sur votre calculatrice.

Par exemple print(2**(2**15)) affichera dans la console un entier qui s'écrit avec 9865 chiffres.

`float`

Nombres à virgules flottantes. Disons que ça correspond très grossièrement aux nombres réels, mais en valeurs approchées. Là, en revanche, la taille du nombre est limitée et la précision (nombre de chiffres après la virgule) aussi.

`str`

De l'anglais *string*, chaîne. Il s'agit des chaînes de caractères. Ce sont des « phrases ». Les suites de lettres ou de signes. Elles sont écrites entre guillemets.

`bool`

De *booléen*, du nom du mathématicien Georges Bool, créateur de la logique moderne.

Ce type (pas le mathématicien) n'a que deux valeurs : vrai et faux, notées True et False.

Conversion

On a parfois besoin de faire passer une variable d'un type à l'autre. Par exemple, lorsqu'une chaîne de caractère représentant un entier (comme `"151"`) a été récupérée par le fonction `input()` on a souvent besoin de la convertir en entier. On peut, comme nous

l'avons vu, utiliser la fonction `int()`.

Pour faire le contraire, on peut utiliser la fonction `str()`.

VI. Instructions d'entrée et sortie

`print()` Cette *fonction* affiche à l'écran la valeur donnée entre parenthèses. Ou les valeurs, à la suite, lorsqu'elles sont indiquées, séparées par une virgule.

Pour éviter le retour à la ligne, on peut entrer comme dernière valeur : `end=""`.

`print("Bonjour !")` affiche la chaîne de caractère `Bonjour !`.

`print(a)` affiche la valeur de la variable `a`.

`print(a,"plus",b)` affiche la valeur de `a`, le mot « plus », puis la valeur de `b`.

`input()` Cette fonction demande à l'utilisateur d'entrer une chaîne de caractère, en affichant le message indiqué entre les parenthèses. ; puis elle attend la réponse. Elle renvoie comme résultat la réponse saisie par l'utilisateur.

```
n=input("Entrez un entier")
n=int(n)
n=n+1
print("Son successeur est",p)
```


VII. Instructions de contrôle

Boucle itérative

Instruction `for`

Comme avec Scratch, vous pouvez répéter un groupe d'instructions dix fois sans avoir pour autant à l'écrire dix fois dans votre programme.

```
for i in range(1,11):
    n=i**2
    print(n)
```

Ce qui diffère d'avec Scratch, c'est que vous devez choisir une variable pour compter les **itérations** (répétitions) de la boucle. Ici, la variable *i* va prendre successivement les valeurs de 1 inclus à 11 exclu.

Notez les deux points « : » à la fin de la ligne *for*. Ils introduisent le décalage sur la droite des deux lignes concernées par le for. Ce décalage s'appelle **indentation**. Il s'agit de décaler vers la droite tout le bloc de lignes concerné. Vous pouvez effectuer ce décalage en utilisant des espaces (le nombre que vous voulez mais le même pour tout le bloc de lignes), ou plus simplement la touche de *tabulation*.

CONDITION

Instruction `If`


```
a=input("Saisissez un entier différent de 10")
a=int(a)
if a==10:
    print("Vous le faites exprès ?")
disp("Vous avez choisi le nombre",a)
```

Notez là encore les deux points « : » et l'indentation. Notez aussi le double égal.

Le bloc d'instructions *indenté* qui suit le if n'est exécuté que si la *condition* (ici a==10) est vérifiée.

If… else

Un peu plus élaboré, le si… sinon, qui suit la syntaxe suivante :

```
if <condition>:
    <instructions>
else:
    <instructions>
```

Il existe aussi un si…sinon si…sinon si…sinon. (`If… elif… elif… else`.)

BOUCLE CONDITIONNELLE

Instruction `While`.

Pas de repeat until en Python

VIII. FONCTIONS

PRINCIPE

La notion informatique de *fonction* ressemble un peu à la notion mathématique du même nom : on peut entrer une valeur (ou plusieurs) et la fonction sort une valeur (ou plusieurs) calculée à partir de la valeur entrée. D'une certaine façon on peut voir aussi une fonction informatique comme un « sous-programme » : un 'petit' programme qui peut être utilisé par un programme plus large.

On peut définir soi-même une fonction, dans un programme (en général au début) à l'aide des instructions `def` et `return`.
`def` indique que l'on va définir une nouvelle fonction et `return` indique la valeur qui sera **renvoyée** par la fonction, le **résultat** (en mathématiques, on parlerait d'image).

EXEMPLE

```
def carré(n):
    c=n**2
    return c
```

Si vous exécutez le programme précédent, il ne se passera rien de visible. Mais ensuite, vous pourrez utiliser la fonction `carré()` soit dans la console, soit dans la suite du programme. Si par exemple vous tapez `carré(5)` dans la console, elle affichera `25`.

AUTRE EXEMPLE

```
def moyenne(a,b,c):
    s=a+b+c
```

```
    m=s/3
    return m
```

Vocabulaire

Les valeurs entrées dans la fonction sont appelés **arguments**.

Les variables qui représentent ces paramètres dans la définition de la fonction sont appelés **paramètres**.

Ainsi, si vous écrivez `moyenne(5,6,9)`, les nombres 5, 6 et 9 sont des *arguments*. Mais les lettres `a`, `b` et `c` utilisées à l'intérieur de la fonction (et dont le nom n'est pas visible à l'extérieur) sont des *paramètres*.