Programmer un ordinateur, c'est quoi ?

Programmer, c'est créer des programmes (suite d'instructions données à l'ordinateur) ! Un ordinateur sans programme ne sait rien faire. Il existe différents langages qui permettent de programmer un ordinateur, mais le seul directement utilisable par le processeur est le langage machine (suite de 1 et de 0). Aujourd'hui (presque) plus personne ne programme en langage machine (trop compliqué).

Les informaticiens utilisent des instructions (mots souvent en anglais) en lieu et place de la suite de 0 et de 1. Ces instructions, une fois écrites par le programmeur, sont "traduites" en langage machine. Un programme spécialisé assure cette traduction. Ce système de traduction s'appellera interpréteur ou bien compilateur, suivant la méthode utilisée pour effectuer la traduction.

Il existe 2 grandes familles de langages de programmation :

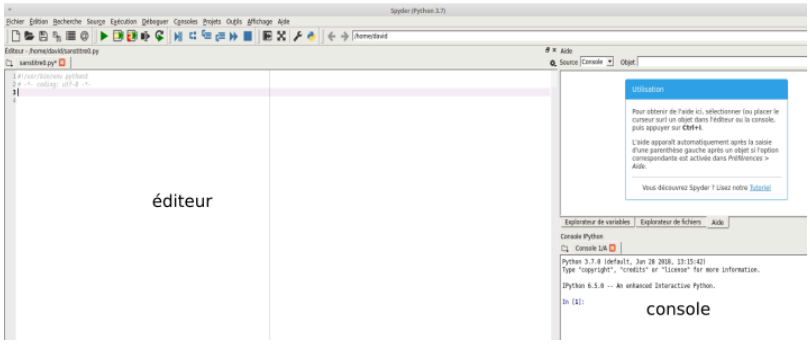
- Les langages de bas niveau sont très complexes à utiliser, car très éloignés du langage naturel, on dit que ce sont des langages « proches de la machine », en contrepartie ils permettent de faire des programmes très rapides à l'exécution. L'assembleur est le langage de bas niveau. Certains "morceaux" de programmes sont écrits en assembleur encore aujourd'hui.
- Les langages de haut niveau sont eux plus "faciles" à utiliser, car plus proches du langage naturel (exemple : si a=3 alors b=c). Exemples de langages de haut niveau : C, C++ , Java, Python...

En NSI, notre langage de prédilection sera Python.

Pour écrire nos programmes, nous utiliserons le logiciel Spyder.

Prise en main de Spyder

Une fois Spyder lancé, vous devriez obtenir quelque chose qui ressemble à cela :



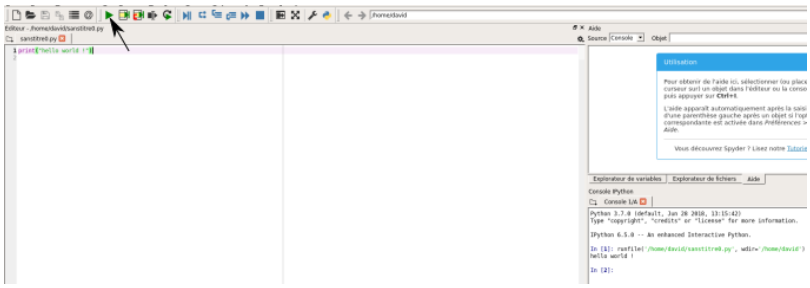
Spyder se divise en plusieurs fenêtres, deux fenêtres vont principalement nous intéresser : la fenêtre "éditeur" et la fenêtre "console".

À faire vous-même 1

Dans la fenêtre "éditeur", saisissez le programme suivante :

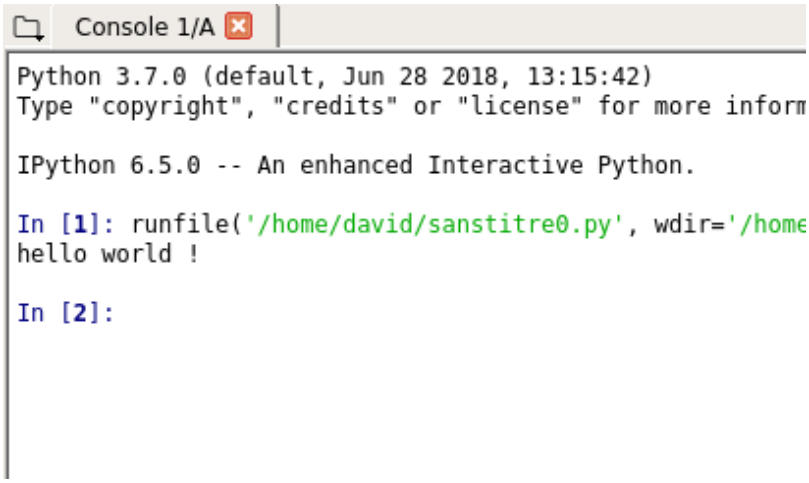
```
print("hello world !")
```

Cliquez sur le "triangle vert" afin d'exécuter le programme qui vient d'être saisi.



Spyder va vous demander d'enregistrer le programme, enregistrez-le dans un dossier qui vous servira de dossier de travail

Vous devez voir le message "hello world !" apparaître dans la console



Notion de variable

Définition du mot ordinateur d'après "Le Petit Larousse" :

"Machine automatique de traitement de l'information, obéissant à des programmes formés par des suites d'opérations arithmétiques et logiques."

Qui dit "traitement de l'information", dit donc données à manipuler. Un programme "passe" donc son temps à traiter des données. Pour pouvoir traiter ces données, l'ordinateur doit les ranger dans sa mémoire (RAM - Random Access Memory). La RAM se compose de cases dans lesquelles nous allons ranger ces données (une donnée dans une case). Chaque case a une adresse (ce qui permet au processeur de savoir où sont rangées les données).

Alors, qu'est-ce qu'une variable ?

Eh bien, c'est une petite information (une donnée) temporaire que l'on stocke dans une case de la RAM. On dit qu'elle est "variable", car

c'est une valeur qui peut changer pendant le déroulement du programme.

Une variable est constituée de 2 choses :

- une valeur présente en mémoire (par exemple le nombre entier 5)
- un nom

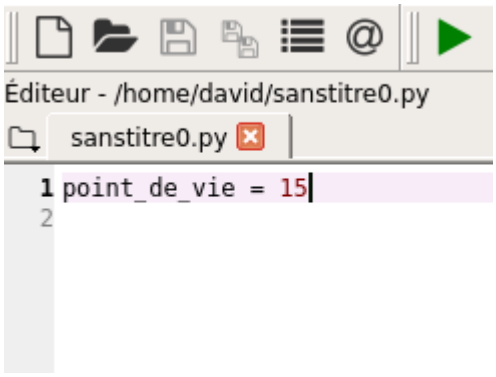
On dira donc qu'une variable est l'association d'un nom et d'une valeur

```
i = 12
```

Grâce à cette ligne, nous avons défini une variable qui porte le nom i. Ce nom i est associé à la valeur 12.

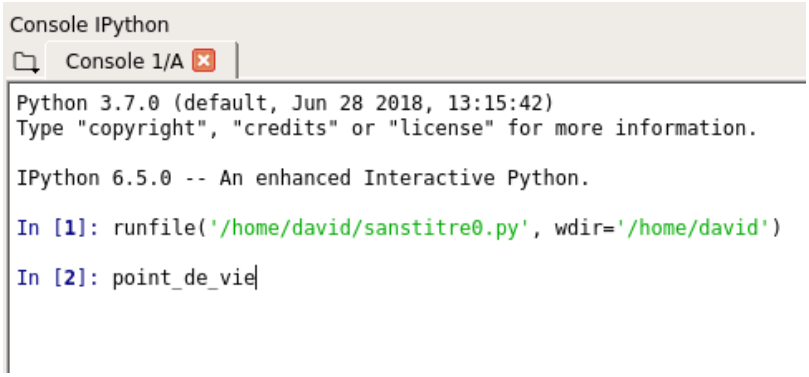
Dans la partie "éditeur" de Spyder, saisissez le code suivant :

```
point_de_vie = 15
```

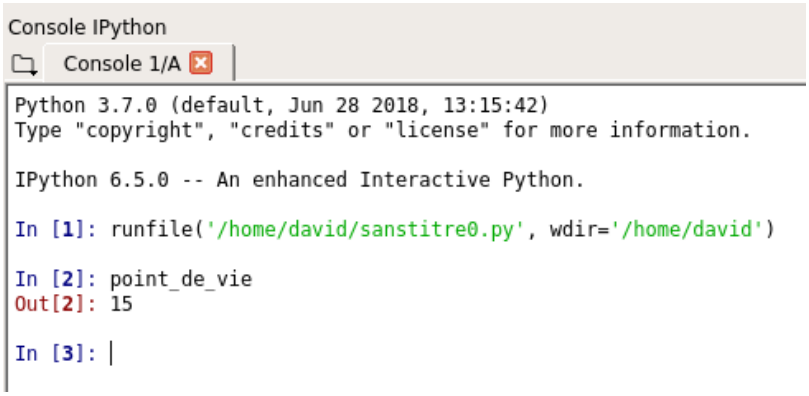


Après avoir exécuté le programme en cliquant sur le triangle vert, il est possible de connaitre la valeur associée à un nom en utilisant la partie "console" de Spyder.

Dans le cas qui nous intéresse ici, tapez point_de_vie dans la console



Après avoir appuyé sur la touche "Entrée", vous devriez voir la valeur associée au nom point_de_vie s'afficher dans la console.



N.B. : Dans la suite la procédure sera toujours la même :

- Vous utiliserez la partie "éditeur" pour saisir votre programme
- vous utiliserez la partie "console" pour afficher la valeur associée à un nom

---

À faire vous-même 3

Écrire un programme dans lequel on associe la valeur 12 au nom point_de_force. La valeur associée à point_de_force devra ensuite être affichée dans la console.

Nous venons de voir qu'un nom pouvait être associé à nombre entier, mais il peut aussi être associé à un nombre à virgule :

```
i = 5.2
```

Prenez bien garde, nous utilisons un point à la place d'une virgule (convention anglo-saxonne).

Un nom peut donc être associé à plusieurs types d'entités (pour l'instant nous n'en avons vu que deux, mais nous en verrons d'autres plus loin) : les nombres entiers ("integer" en anglais, abrégé en "int") et les nombres à virgule ("float" en anglais). Il est possible de connaitre le type de l'entité à l'aide de l'instruction "type".

À faire vous-même 4

Testez le programme suivant :

```
a = 5.2
b = 12
```

tapez type(a) puis type(b) dans la console

```
In [3]: runfile('/home/david/sanstitre0.py', wdir='/home/david')

In [4]: type(a)
Out[4]: float

In [5]: type(b)
Out[5]: int

In [6]:
```

Comme vous pouvez le constater, le type de la grandeur associée à a et le type de la grandeur associée à b s'affichent dans la console

---

Un peu de calculs

Un ordinateur est bien évidemment capable d'effectuer des opérations mathématiques (arithmétiques).

Les signes utilisés sont classiques : +, - , * (multiplication), / (division), // (division euclidienne) ou encore % (modulo : reste d'une division euclidienne).

Il est tout à fait possible d'effectuer des opérations directement avec des nombres, mais il est aussi possible d'utiliser des variables.

À faire vous-même 5

Essayez d'écrire un programme qui additionnera le contenu de 2 variables (nom des variables : a et b). Le résultat de cette opération devra être associé à une variable nommée resultat (atention pas d'accent dans les noms de variable). Testez votre programme en utilisant la console pour vérifier la valeur associée à resultat.

À faire vous-même 6

D'après vous, que fait ce programme ?

```
a = 11
a = a + 1
```

Vérifiez votre réponse en exécutant le Programme (utilisation dans console pour déterminer la valeur associée au nom a à la fin du programme)

Détaillons ce qui se passe dans le "À faire vous-même 6":

- nous créons une variable : le nom a est associé à l'entier 11
- nous affichons à l'écran la valeur associée à a (c'est-à-dire 11)

La suite est un peu plus complexe, mais très importante à comprendre. Il va falloir lire la ligne a = a + 1 de droite à gauche, décortiquons cette ligne :

- a + 1 : nous prenons la valeur actuelle associé au nom a (c'est-à-dire 11) et nous ajoutons 1 à 11, à droite de l'égalité nous avons donc maintenant la valeur 12

- nous associons la valeur qui vient d'être calculée au nom a
- nous affichons à l'écran la nouvelle valeur associée à a

Ce raisonnement peut être généralisé pour éviter des erreurs parfois difficiles à corriger : dans une égalité, commencer toujours par évaluer l'expression se trouvant à droite du signe égal.

exposant, racine carrée, fonctions trigonométriques

Il est aussi possible d'effectuer des calculs plus complexes en utilisant par exemple des exposants, des racines carrées, des fonctions trigonométriques...

Pour utiliser ces fonctions mathématiques plus avancées, il est nécessaire d'ajouter une ligne au début de votre programme :

```
import math
```

Cette ligne permet d'importer (et donc d'utiliser) le module math (ce module contient toutes les fonctions mathématiques "classiques").

Voici quelques exemples :

- math.pow(x,a) permet de calculer x à la puissance a (il est aussi possible de directement écrire x**a)
- math.cos(x) permet de calculer le cosinus de l'angle x (l'angle x doit être en radian) (nous avons la même chose pour le sinus ou la tangente)
- math.sqrt(x) permet de calculer la racine carrée de x

Si vous avez besoin d'autres fonctions mathématiques, je vous invite à consulter la documentation de Python : https://docs.python.org/3/library/math.html (https://docs.python.org/3/library/math.html)

À faire vous-même 7

Quelles sont les valeurs associées aux noms suivants : d, e, f, g, h et i après l'exécution du programme suivant :

```
import math
a = 5
b = 16
c = 3.14 / 2
d = b / a
e = b // a
f = b % a
g = math.pow(a,2)
h = math.sqrt(b)
i = math.sin(c)
```

Vérifiez vos réponses à l'aide de la console

À noter qu'il est tout à fait possible de "mélanger" des nombres entiers et des nombres à virgules ("3.14 / 2") dans une opération.

À faire vous-même 8

Quel est le type du résultat d'une addition d'un integer et d'un float ? Utilisez la console Python pour vérifier votre réponse.

---

chaînes de caractères

On peut aussi associer des noms à des suites de caractères que l'on appelle "chaîne de caractères".

À faire vous-même 9

Tester le code suivant :

```
ma_chaine = "Bonjour le monde !"
```

Vérifiez que le nom ma_chaine est associé à la chaîne de caractères "Bonjour le monde !"

Le signe + et les chaînes de caractères

L'utilisation du signe + ne se limite pas à l'addition. Il est aussi utilisé pour la **concaténation**.

D'après Wikipédia :

« Le terme concaténation (substantif féminin), du latin cum («avec») et catena(«chaîne, liaison»), désigne l'action de mettre bout à bout au moins deux chaînes. »

Comme vous avez pu le deviner en lisant la définition ci-dessus, la concaténation va concerner les chaînes de caractères.

À faire vous-même 10

Quelle est la chaîne de caractère associée au nom mon_expression après l'exécution du programme ci-dessous ? Validez votre réponse en testant ce programme.

```
a = "Hello"
b = "World"
mon_expression = a + b
```

---

chaînes de caractères et variables

Il est aussi possible de concaténer une chaîne de caractères et une ou plusieurs variables :

À faire vous-même 11

Testez le code suivant :

```
ma_chaine_1 = "Bonjour "
ma_chaine_2 = "le "
res = ma_chaine_1 + ma_chaine_2 + "monde!"
```

Les 2 noms ma_chaine_1 et ma_chaine_2 sont associés à 2 chaînes de caractères, nous avons donc bien ici une concaténation.

Mais que se passe-t-il si une des deux variable est de type nombre (entier ou flottant) ?

À faire vous-même 12

Testez le code suivant :

```
mon_nombre = 5
res = "Nombre de personnes : " + mon_nombre
```

Comme vous pouvez le constater, nous avons droit à une erreur. En effet, il n'est pas possible de concaténer une chaîne de caractères et un nombre.

Python nous offre 2 solutions :

- l'utilisation de la méthode "str"
- l'utilisation des "fstring"

La méthode (nous verrons plus loin la notion de méthode) "str" permet de transformer un nombre en chaîne de caractères (si la

transformation n'est pas possible, nous aurons une erreur)

À faire vous-même 13

Testez le code suivant :

```
mon_nombre = 5
mon_nombre = str(mon_nombre)
```

Quel est le type de la valeur associée à mon_nombre après l'exécution du programme ci-dessus ?

À faire vous-même 14

Testez le code suivant :

```
mon_nombre = 5
res = "Nombre de personnes : "  + str(mon_nombre)
```

Tout fonctionne, car maintenant nous avons bien une concaténation entre 2 chaînes de caractères.

Les "fstring" (nouveauté de Python 3.5), permettent de résoudre ce problème de combinaison variable-chaîne de caractères.

À faire vous-même 15

Testez le code suivant :

```
mon_nombre = 5
res = f"Nombre de personnes : {mon_nombre}"
```

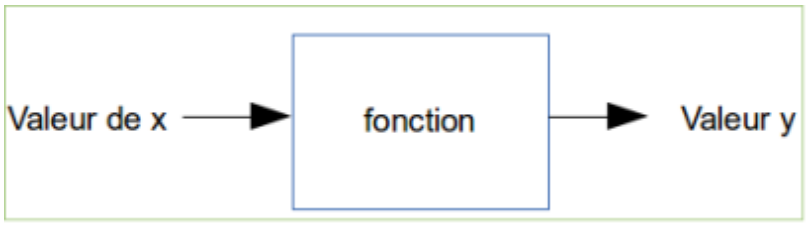
Notez la présence du "f" juste avant le guillemet et des accolades qui encadrent le nom de la variable. Il est possible, dans une même chaîne

de caractères d'avoir plusieurs noms de variable.

Les fonctions

Les fonctions permettent de décomposer un programme complexe en une série de sous-programmes plus simples. De plus, les fonctions sont réutilisables : si nous disposons d'une fonction capable de calculer une racine carrée, par exemple, nous pouvons l'utiliser un peu partout dans notre programme sans avoir à la réécrire à chaque fois (on parle de factorisation du code)

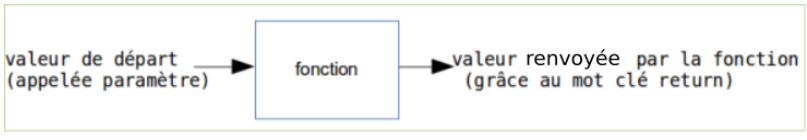
La notion de fonction en informatique est comparable à la notion de fonction en mathématiques.



Si nous avons y = 3x+2, pour une valeur donnée de x, nous aurons une valeur de y.

Exemple : x=4 donc y= 14 (y = 3.4+2=14, attention ici le point correspond au signe "multiplié").

La fonction en informatique est basée sur la même idée :



Voici la syntaxe employée en Python pour définir une fonction :

```
def nom_de_la_fonction(parametre):
        instruction_1
        instruction_2
        return y
suite programme
```

La fonction renvoie la valeur associée à y.

ATTENTION : Notez bien la présence du décalage entre la première ligne et les lignes suivantes. Ce décalage est appelé indentation, l'indentation permet de définir un bloc de code. Dans l'exemple ci-dessus, l'indentation nous permet de savoir que "instruction_1", "instruction_2" et "return y" constituent un bloc de code, ce bloc correspond au contenu de la fonction. "suite programme" ne fait pas partie de la fonction, car il n'est pas indenté. Pour indenter du code, il y a 2 solutions : mettre 4 espaces ou utiliser une tabulation. En Python il est conseillé d'utiliser les 4 espaces, mais ce n'est pas une obligation. Une chose est sûre, une fois que vous avez choisi une méthode, n'en changé surtout pas au cours d'un même programme !

Codons notre exemple (y=3x+2) en créant une fonction ma_fonction :

```
def ma_fonction(x):
        y = 3 * x + 2
        return y
```

Pour "utiliser" la fonction ma_fonction, il suffit d'écrire : ma_fonction (4) (dans ce cas précis, notre fonction renverra le nombre 14).

À faire vous-même 16

Testez le programme suivant. Quelle est la valeur associée à solution après l'exécution du programme :

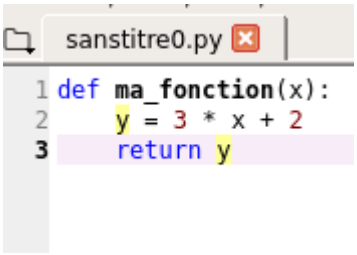
```
def ma_fonction(x):
        y = 3 * x + 2
        return y
solution = ma_fonction(4)
```

Il faut savoir qu'au moment de l'exécution de votre programme le code ma_fonction(4) sera systématiquement remplacé par la valeur renvoyée par la fonction (toujours dans notre exemple le ma_fonction(4) sera remplacé par le nombre 14).

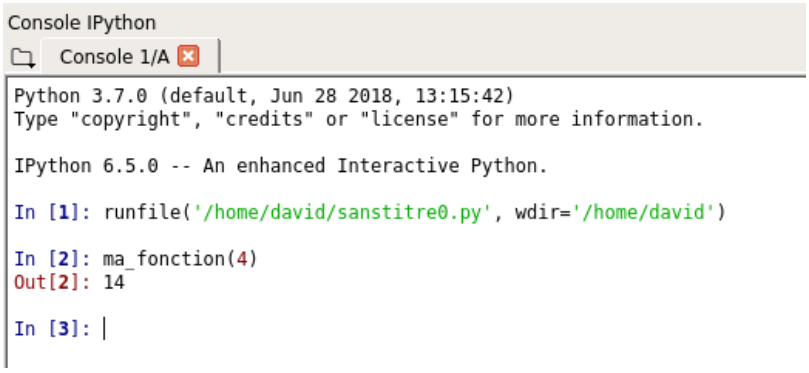
À faire vous-même 17

Codez en Python la fonction $y = x^2+2x+10$

Il est possible d'écrire une fonction dans l'éditeur :



et d'utiliser la console (après exécution du programme) pour obtenir la valeur renvoyée par une fonction



Il est possible de faire passer plusieurs paramètres à une fonction.

À faire vous-même 18

Quel est le résultat renvoyé par la fonction ci-dessous si l'on saisit dans la console une_autre_fonction(5, 3)

```
def une_autre_fonction(x, b):
        y = 3 * x + b
        return y
```

Les paramètres peuvent être des chaînes de caractères (ainsi que la valeur retournée)

À faire vous-même 19

Quel est le résultat attendu après l'exécution du programme ci-dessous et la saisie dans la console de "dit_bonjour("toto", 14)" ?

```
def dit_bonjour(nom, age):
        phrase = f"Bonjour {nom}, vous avez {age} a
ns."
        return phrase
```

Attention : remarquez bien les guillemets autour du paramètre "toto" (c'est une chaîne de caractères)

Les paramètres ne sont pas obligatoires.

À faire vous-même 20

Testez la fonction suivante :

```
def ma_fon():
        return "voici une fonction qui ne sert à ri
en"
```

Il faut aussi savoir qu'une fonction ne renvoie pas forcément de valeur (le mot clé return n'est pas obligatoire). Mais si elle ne renvoie pas de valeur, que fait-elle ? Elle peut faire plein de choses, par exemple elle peut tout simplement afficher une chaîne de caractères à l'aide d'un "print". Sachez que dans certains langages, on utilise les termes méthode ou procédure pour qualifier une fonction "qui ne renvoie rien".

À faire vous-même 21

Soit le programme suivant :

```
def dit_bonjour(nom, age):
        phrase = f"Bonjour {nom}, vous avez {age} a
ns."
        print(phrase)
```

Testez la fonction dit_bonjour à l'aide de la console (avec par exemple un dit_bonjour("toto", 14))

Variables locales et variables globales

À faire vous-même 22

Analysez le programme suivant :

```
def fct():
  i=5
fct()
```

Après avoir exécuté le programme ci-dessus, déterminez la valeur associée à i en utilisant la console.

Comme vous avez pu le constater, nous avons eu droit à une erreur : "NameError: name 'i' is not defined". Pourquoi cette erreur, la variable i est bien définie dans la fonction fct() et la fonction fct() est bien exécutée, où est donc le problème ?

En fait, la variable i est une variable dite locale : elle a été définie dans une fonction et elle "restera" dans cette fonction. Une fois que l'exécution de la fonction sera terminée, la variable i sera "détruite" (supprimée de la mémoire). Elle n'est donc pas accessible depuis "l'extérieur" de la fonction (ce qui explique le message d'erreur que nous obtenons).

Autre exemple de cette notion de variable locale :

À faire vous-même 23

Analysez le programme suivant :

```
i = 3
def fct():
  i=5
fct()
```

Après avoir exécuté le programme ci-dessus, déterminez la valeur associé à i en utilisant la console.

Cette fois pas d'erreur, mais à la fin de l'exécution de ce programme, i est associé à la valeur 3. En faite dans cet exemple nous avons 2 variables i différentes : la variable i "globale" (celle qui a été définie en dehors de toute fonction) et la variable i "locale" (celle qui a été définie dans la fonction). Ces 2 variables portent le même nom, mais sont différentes.

Une variable globale peut être "utilisée" à l'intérieur d'une fonction :

À faire vous-même 24

Analysez et testez le programme suivant :

```
i = 3
def fct():
  print(i)
fct()
```

Quand on cherche à utiliser une variable dans une fonction, le système va d'abord chercher si cette variable se "trouve" dans l'espace local de la fonction, puis, s'il ne la trouve pas dans cet espace local, le système va aller rechercher la variable dans l'espace global. Pour le "print(i)" situé dans la fonction le système ne trouve pas de variable i dans l'espace local de la fonction "fct", il passe donc à l'espace global et trouve la variable i (nous avons donc 3 qui s'affiche). Il est important de bien comprendre que si le système avait trouvé une variable i dans l'espace local de la fonction, la "recherche" de la variable i se serait arrêtée là :

À faire vous-même 25

Analysez et testez le programme suivant :

```
i = 3
def fct():
  i = 5
  print(i)
fct()
```

À faire vous-même 26

Analysez et testez le programme suivant :

```
i = 3
def fct():
  i = i + 1
fct()
```

Nous avons une erreur "UnboundLocalError: local variable 'i' referenced before assignment"

Pour pouvoir modifier une variable globale dans une fonction, il faut utiliser l'instruction "global" :

À faire vous-même 27

Analysez le programme suivant :

```
i = 3
def fct():
  global i
  i = i + 1
fct()
```

Après avoir exécuté le programme ci-dessus, déterminez la valeur associée à i en utilisant la console.

En faite, l'utilisation de "global" est une (très) mauvaise pratique, car cette utilisation peut entrainer des "effets de bord".

Effet de bord

On parle d'effet de bord quand une fonction modifie l'état d'une variable globale. Dans notre exemple ci-dessus la fonction fct modifie bien la valeur associée à i : avant l'exécution de fct, i est associé à la valeur 3, après l'exécution de la fonction fct, i est associé à la valeur 4. Nous avons donc bien un effet de bord.

Les effets de bord c'est "mal" ! Mais pourquoi est-ce "mal" ?

Les effets de bords provoquent parfois des comportements non désirés par le programmeur (évidemment dans des programmes très complexes, pas dans des cas simplistes comme celui que nous venons de voir dans le "À faire vous-même 27"). Ils rendent aussi parfois les programmes difficilement lisibles (difficilement compréhensibles). À cause des effets de bord, on risque de se retrouver avec des variables qui auront des valeurs qui n'étaient pas prévues par le programmeur. On dit aussi qu'à un instant donné, l'état futur des variables est difficilement prévisible à cause des effets de bord. En résumé, on évitera autant que possible l'utilisation du "global".

Un paradigme de programmation se propose d'éviter au maximum les effets de bords : la programmation fonctionnelle. Nous étudierons ce paradigme de programmation en terminale.

les expressions et les booléens

Si quelqu'un vous dit que "4 est égal à 5", vous lui répondez quoi ? "c'est faux". Si maintenant la même personne vous dit que "7 est égal à 7", vous lui répondrez bien évidemment que "c'est vrai".

En Python, ces deux «affirmations» ("4 est égal à 5" et "7 est égal à 7") s'écriront "4 == 5" et "7 == 7" (notez bien le double signe égal).

À faire vous-même 28

Dans la console, tapez :

```
4 == 5
```

En Python, "4 == 5" est appelé une expression, une expression est soit vraie ("True"), soit fausse ("False").

Pour l'instant nous avons vu deux grands types de données : les nombres (entier ou flottant) et les chaînes de caractères, il existe un troisième type tout aussi important que les deux premiers : les booléens. Un booléen est un type de données qui ne peut prendre que deux valeurs : vrai ("True") ou faux ("False"). Une expression (comme par exemple "4 == 5") est soit True, soit False.

ATTENTION : notez le double égal "==" qui permet de distinguer une expression et une affectation (association d'une valeur à un nom (variable)). Le fait de confondre le "simple égal" et le "double égal" est une erreur classique qu'il faut éviter.

À faire vous-même 29

Soit le programme suivant :

```
a = 4
b = 7
```

Quel est le résultat attendu après l'exécution de ce programme si vous saisissez dans la console "a==b" ?

Il est possible d'utiliser aussi l'opérateur "différent de" !=

À faire vous-même 30

Soit le programme suivant :

```
a = 4
b = 7
```

Quel est le résultat attendu après l'exécution de ce programme si vous saisissez dans la console "a!=b" ?

Notez aussi l'existence des opérateurs :

- "strictement inférieur à" <
- "strictement supérieur à" >
- "inférieur ou égal à" <=
- "supérieur ou égal à" >=

À chaque fois ces opérateurs sont True (vrai) ou False (faux).

À faire vous-même 31

Soit le programme suivant :

```
a = 4
b = 7
```

Quel est le résultat attendu après l'exécution de ce programme si vous saisissez dans la console "a<b" ?

Les conditions

Nous allons maintenant étudier une structure fondamentale en programmation le « si ..... alors.....sinon.....».

L'idée de base est la suivante :

```
si expression:
        suite_instruction1
sinon:
        suite_instruction2
```

Si "expression" est True alors "suite_instruction1" est exécuté et "suite_instruction2" est ignoré.

Sinon (sous-entendu que "expression" est False) "suite_instruction2" est exécuté et "suite_instruction1" est ignoré.

Notez l'indentation "suite_instruction1" et de "suite_instruction2"

À faire vous-même 32

Soit le programme suivant :

```
a = 4
b = 7
if a < b:
        print("Je suis toto.");
        print("Je n'aime pas titi.")
else:
        print("Je suis titi.")
        print("Je n'aime pas toto.")
print("En revanche, j'aime le Python.")
```

Quel est le résultat attendu après l'exécution de ce programme ?

Vérifiez votre hypothèse en testant le programme.

À faire vous-même 33

Écrire une fonction qui prend en paramètre un age. Si age est supérieur ou égal à 18 ans, la fonction devra renvoyer la chaîne de caractères "Bonjour, vous êtes majeur.". Si age est inférieur à 18 ans, la fonction devra renvoyer "Bonjour, tu es mineur."

À faire vous-même 34

Soit le programme suivant :

```
def annonce(num, prov, dest):
        if dest != "0":
                msg = f"le train n° {num} en proven
ance de {prov} et à destination de {dest}, entre en
gare."
        else:
                msg = f"le train n° {num} en proven
ance de {prov} entre en gare. Ce train est terminus
Triffouillis-les-Oies."
        return msg
```

Quel est le résultat attendu après l'exécution de ce programme si vous saisissez dans la console annonce("4557", "Paris", "Marseille") ? Et si vous saisissez dans la console annonce("5768", "Bonneville", "0") ? Vérifiez votre réponse en testant ce programme.

À faire vous-même 35

Vous êtes gérant d'un magasin et vous désirez écrire un programme Python qui calculera automatiquement le montant de la facture des clients. Tout client qui achète au moins 5 fois le même article se voit octroyer une remise de 5 % (uniquement sur le montant de l'achat de cet article). Afin de simplifier le problème, on considère qu'un client n'achète qu'un seul type d'article. Écrire une fonction qui prend en paramètre le prix unitaire de l'article et le nombre d'articles achetés. Cette fonction doit renvoyer le montant de la facture.

Le "or", le "and" et le "not"

Un if peut contenir plusieurs conditions, nous aurons alors une structure de la forme :

```
si expression1 op_logique expression2:
        suite_instruction1
sinon:
        suite_instruction2
```

« op_logique » étant un opérateur logique.

Nous allons étudier 2 opérateurs logiques : le "ou" (noté en Python "or") et le "et" (noté en Python "and").

Par exemple (expression1 or expression2) est vrai si expression1 est vraie ou expression2 est vraie.

Autre exemple (expression1 and expression2) est faux si expression1 est vraie et expression2 est faux.

Les résultats peuvent être regroupés dans ce que l'on appelle une table de vérité :

table de vérité pour le "ou"

expression1	expression2	expression1 or expression2
vrai	vrai	vrai
vrai	faux	vrai
faux	vrai	vrai
faux	faux	faux

table de vérité pour le "et"

expression1	expression2	expression1 and expression2
vrai	vrai	vrai
vrai	faux	faux
faux	vrai	faux
faux	faux	faux

À faire vous-même 36

Soit le programme suivant :

```
a = 5
b = 10
if a > 5 and b == 10:
        print ("Toto")
else:
        print("Titi")
if a > 5 or b == 10:
        print ("Tata")
else:
        print("Tutu")
```

Quel est le résultat attendu après l'exécution de ce programme ? Vérifiez votre réponse en testant ce programme.

Il existe aussi l'opérateur logique "not" ("non" en français) :

table de vérité pour le "non"

expression	not(expression)
vrai	faux
faux	vrai

La boucle while

La notion de boucle est fondamentale en informatique. Une boucle permet d'exécuter plusieurs fois des instructions qui ne sont présentes qu'une seule fois dans le code.

La structure de la boucle while est la suivante :

```
while expression:
        instruction1
        instruction2
suite programme
```

Tant que l'expression s'évalue à "True", les instructions à l'intérieur du bloc (partie indentée) seront exécutées.

À faire vous-même 37

Soit le programme suivant :

```
i = 0
while i < 10:
        print(f"i vaut : {i}")
        i = i + 1
print("C'est terminé.")
```

Quel est le résultat attendu après l'exécution de ce programme ? Vérifiez votre réponse en testant le programme

---

À faire vous-même 38

Vous allez créer "un générateur automatique de punition" :

Écrire une fonction qui prendra 2 paramètres : une chaîne de caractère et un nombre entier

Par exemple :

Si on passe comme paramètres à notre fonction : "Je ne dois pas discuter en classe" et 3

La fonction devra permettre d'afficher :

Je ne dois pas discuter en classe

Je ne dois pas discuter en classe

Je ne dois pas discuter en classe

À faire vous-même 39

Écrire une fonction permettant d'afficher une table de multiplication. Cette fonction devra prendre en paramètre la table désirée.

Par exemple si l'on passe le paramètre 3 à la fonction, la fonction devra permettre d'afficher :

1 x 3 = 3

2 x 3 = 6

...

...

10 x 3 = 30

FICHE REVISION (fiches/1/01_python_les_bases.pdf)

Auteur : David Roche