Python en classe de seconde

Laboratoire de mathématiques du lycée Roland garros

Python : Utilisation de la console, les variables

Il y a deux manières d'écrire du Python :

- **Mode interactif dans un shell Python**, vous tapez une ligne dans la console puis vous appuyez sur entrer.
- **Dans un fichier texte** (fichier.py), vous écrivez une suite d'instructions (lignes) dans un fichier en vous servant d'un éditeur de texte et dites à Python d'exécuter les instructions du fichier dans le shell.

Durant cette séance nous allons seulement utiliser la console.
Évaluez les expressions encadrées ci-dessous, marquez les affichages obtenus et commentez les.

I) Opérations et types de données

Les trois caractères >>> constituent le signal d'invite, lequel vous indique que Python est prêt à exécuter une commande.

```
>>> 5+3
>>> 2 - 9
>>> 7 + 3 * 4
>>> (7+3)*4
>>> 20 / 3
>>> 20 // 3
>>> 20 % 3
>>> 17 % 4
```

```
>>> 2**3
>>> 2019**2019
>>> 2e3
>>> 8,7 / 5
>>> 5 + 3.0
>>> 0.1 + 0.7
>>> 3 < 5
>>> 5 < 3
```

```
>>> 0 < 3 < 8
>>> 1 + 1 = 2
>>> 1 + 1 == 2
>>> age
>>> "age"
>>> sqrt(16)

>>> from math import *
>>> sqrt(16)
```

Il existe en Python des **constantes**. Les valeurs fixes telles que les nombres, les chaînes etc sont appelées « constantes » parce que leur valeur ne change pas. Il y a :

- Les constantes numériques qui sont de type **entier** ou **flottant** (nombre à virgule), Python les reconnaît directement.
- Les **chaînes de caractères** à écrire entre deux symboles d'apostrophe simple ' ou double ".
- Les **booléens** qui sont soit vrai (True), soit faux (False). Ils servent à tester des conditions.

Vous avez fait des opérations sur les entiers (résultats exacts) et les flottants (résultats approchés), Python peut aussi faire des opérations sur des chaînes de caractères (il redéfinit les opérations) et des booléens.

```
>>> "age" + 1
>>> "age" + "1"
```

```
>>> "1" + "1"
>>> 3*"age"
```

```
>>> "a" in "age"
>>> "b" in "age"
```

```
>>> 3<5 and 4<2
>>> 3<5 or 4<2
```

En Python l'expression `type(objet)` permet de connaître le type d'un objet.

Entier
```
>>> type(3)
```

Flottant
```
>>> type(3.5)
```

Chaîne de caractères
```
>>> type("age")
```

Booléen
```
>>> type(3<5)
```

II) Les variables

Une **variable** est une zone de la mémoire de l'ordinateur dans laquelle une **valeur** est stockée. Aux yeux du programmeur, cette variable est définie par un **nom**, alors que pour l'ordinateur, il s'agit en fait d'une adresse, c'est-à-dire d'une zone particulière de la mémoire. En Python, la **déclaration** d'une variable et son **initialisation** (c'est-à-dire la première valeur que l'on va stocker dedans) se font en même temps.
Quand on affecte un contenu à une variable, elle prend le **type** de l'objet qu'elle contient.

1) Noms de variables

Les noms des variables sont des noms que vous choisissez vous-même assez librement.
Sous Python, les noms de variables doivent en outre obéir à quelques règles simples :

- Un nom de variable est une séquence de lettres (majuscules ou minuscules) et de chiffres, qui doit toujours commencer par une lettre ;
- Seules les lettres ordinaires sont autorisées, pas d'accent, d'espaces etc, seul le caractère spécial _ est autorisé ;
- La casse est significative, prenez l'habitude d'écrire l'essentiel des noms de variables en minuscules, n'utilisez les majuscules qu'à l'intérieur même du nom pour en augmenter la lisibilité, comme dans tableDesMatieres ;
- On ne peut pas utiliser non plus une liste de 33 mots qui sont utilisés par le langage.

2) Affectation (ou assignation)

Affecter une variable c'est lui attribuer une valeur. En Python, l'opération d'affectation est représentée par le signe `=` .

```
>>> n = 7
>>> msg = "Bonjour"
>>> pi_2 = 3.14
```

Après avoir effectué ces trois affectations, elles ont eu pour effet chacune de réaliser plusieurs opérations dans la mémoire de l'ordinateur :

- Créer et mémoriser un **nom de variable** ;
- lui attribuer un **type** particulier ;
- Créer et mémoriser une **valeur** particulière ;
- Établir un **lien** entre le nom de la variable et sa valeur correspondante.

Il faut bien comprendre qu'il ne s'agit en aucune façon d'une égalité, Python aurait pu choisir un autre symbolisme , tel que n ← 7. Pour affecter 7 à n, on écrit `n = 7` et on peut lire « n reçoit 7 », on n'écrira pas `7 = n` !
La variable `n` peut ne pas toujours contenir la valeur 7, on peut la réaffecter en écrivant par exemple `n = 5` .

3) Afficher la valeur d'une variable

A la suite de ce que l'on vient de faire, on dispose donc de trois variables : n, msg et pi_2.
Pour afficher leur valeur à l'écran, il existe deux possibilités :

- On entre le nom de la variable, puis « enter ». Python répond en affichant la valeur correspondante :

```
>>> n

>>> msg

>>> pi_2
```

  Il s'agit cependant là d'une fonctionnalité secondaire de l'interpréteur, qui est destinée à vous faciliter la vie lorsque vous faites de simples exercices à la ligne de commande.
- On utilise la fonction `print()` (qui servira à l'intérieur d'un programme) :

```
>>> print(n)

>>> print(msg)

>>> print(pi_2)
```

4) Affectations multiples

Sous Python, on peut assigner une valeur à plusieurs variables simultanément. Exemple :

```
>>> x = y = 7
>>> x

>>> y
```

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

```
>>> a, b = 4, 8.33
>>> a

>>> b
```

5) Opérateur et expression

On manipule les valeurs et les variables qui les référencent en les combinant avec des opérateurs pour former des expressions. Exemple :

```
>>> a, b = 7.3, 12
>>> y = 3*a + b/5
>>> y
```

Dans cet exemple, nous commençons par affecter aux variables `a` et `b` les valeurs `7.3` et `12` .
La seconde ligne de l'exemple consiste à affecter à une nouvelle variable `y` le résultat d'une expression.
Dans une expression, ce que vous placez à la gauche du signe `=` doit toujours être une variable et non une expression. Ainsi par exemple, l'instruction `m + 1 = b` est tout à fait **illégale**.
Par contre, écrire `a = a + 1` est inacceptable en mathématique, alors que cette forme d'écriture est très fréquente en programmation. L'instruction `a = a + 1` signifie en l'occurrence « augmenter la valeur de la variable `a` d'une unité » (ou encore : « incrémenter `a` »). Que valent les variables à la fin des programmes suivants? Compléter les tableaux.

```
>>> a = 4
>>> a = a + 1
```

a

```
>>> a = 2
>>> m = 5
>>> a = m + a
>>> a = a + 1
>>> m = a*m
>>> m = 2*m
```

a	m

```
>>> a, b = 3, 7
>>> a = b
>>> b = a
```

a	b

```
>>> a, b = 3, 7
>>> b = a
>>> a = b
```

a	b

Proposer un programme qui puisse échanger le contenu de deux variables `a` et `b` .

6) Composition

Jusqu'ici nous avons examiné les différents éléments d'un langage de programmation, à savoir : les variables, les expressions et les instructions, mais sans traiter la manière dont nous pouvons les combiner les unes aux autres.
Par exemple si vous savez comment additionner deux nombres et comment afficher une valeur, vous pouvez combiner ces deux instructions en une seule :

```
>>> print(17 + 3)
```

Cela n'a l'air de rien, mais cette fonctionnalité qui paraît si évidente va vous permettre de programmer des algorithmes complexes de façon claire et concise. Exemple :

```
>>> h, m, s = 15, 27, 34
>>> print("Le nombre de secondes ecoulees depuis minuit est",h*3600 + m*60 + s)
```

Python : Utilisation d'un script, les fonctions informatiques

Lors de la première séance, on a travaillé directement dans la console, tout le travail effectué a été perdu (variables etc).
Pour pouvoir sauvegarder des programmes, il va falloir rédiger les séquences d'instructions dans un **éditeur de texte**.
Ainsi vous écrirez un **script** qui pourra être sauvegardé sous la forme fichier**.py**.
Il faudra ensuite appeler le fichier dans la console en faisant par exemple : `from fichier import *`

Une fonction informatique possède un **nom**, un ou plusieurs **arguments** et renvoie un ou plusieurs **résultats**.
On peut enregistrer les fonctions dans des scripts que l'on exécute dans la console.
On peut faire appel plusieurs fois à la même fonction en donnant des valeurs aux arguments (dans le bon ordre).
Il faut respecter la syntaxe comme sur l'exemple ci-dessous :

fichier.py

```
def nom(argument1, argument2, ...):
    instructions
    return (resultat1, resultat2, ...)
```

Exercice 1 :

Écrire le script ci-dessous :

exo1.py

```
def f(x):
    return 3*x-1
```

Exécuter le dans la console et compléter l'affichage :

```
>>> from exo1 import *
>>> f(1)

>>> f(2)
```

Quelle est la nature de cette fonction?

✎

Pour les exercices suivants, écrivez les scripts dans des fichiers, exécutez les dans la console comme pour l'exercice précédent et répondez aux questions.

Exercice 2 :

On a programmé une fonction nommée `g` .

exo2.py

```
def g(a,b,x):
    return a*x+b
```

① Quels sont les arguments de cette fonction?

✎

② Quel va être l'affichage si on demande `g(1,2,3)` dans la console?

✎

③ Que retourne l'instruction `g(2,1,3)==7` dans la console?

✎

Exercice 3 :

En prévision des soldes, un commerçant s'apprête à modifier ses étiquettes.

① Un article coûte 40 € et subit une réduction de 30 %, quel est son nouveau prix?

② Voici le programme d'une fonction :

exo3.py

```
def solde(prix,baisse):
    return prix*(1-baisse/100)
```

a) Quelle est la valeur affichée dans la console si on saisit `solde(40,30)` ?

b) Quel est le rôle de ce programme?

c) Que doit saisir le commerçant pour solder à 60 % un article à 55 €?

Exercice 4 :

exo4.py

```
def vecteur(A,B):
    (xA,yA) = A
    (xB,yB) = B
    return (xB-xA,yB-yA)
```

Quelle est la valeur affichée dans la console si on saisit `vecteur((2,3),(5,7))` ?

Quel est le rôle de cette fonction?

Exercice 5 :

① Écrire une fonction `moyenne` (dans le fichier exo5.py) qui à deux nombres `a` et `b` associe leur moyenne arithmétique.

② Compléter la fonction `milieu` pour qu'elle renvoie les coordonnées du milieu du segment [AB].

exo5.py

```
def milieu(A,B):
    (xA,yA) = A
    (xB,yB) = B
    return (          ,          )
```

③ Chercher une autre réponse à la question précédente en utilisant la fonction `moyenne` .

exo5.py

```
def milieu(A,B):
    (xA,yA) = A
    (xB,yB) = B
    return (          ,          )
```

L'instruction if

Python exécute les instructions d'un programme les unes après les autres, dans l'ordre où elles ont été écrites à l'intérieur du script sauf lorsqu'il rencontre une instruction conditionnelle comme l'instruction `if` décrite ci-après.
Une telle instruction va permettre au programme de suivre différents chemins suivant les circonstances.
Pour la suite de la séance tapez les scripts, exécutez les dans la console et marquez les affichages dans les cadres.

I) Exécution conditionnelle if

```
a=150
if a>100 :
    print("a dépasse la centaine")
```

```
a=80
if a>100 :
    print("a dépasse la centaine")
```

Lorsque vous finissez d'entrer la seconde ligne sans oublier de mettre le caractère « : », vous constatez que Python commence la troisième en mettant une indentation, c'est le même comportement que lorsque vous écrivez une fonction (il faut absolument garder cette indentation sinon il y aura une erreur à la compilation).
L'expression que vous avez placée après `if` est ce que nous appelleront désormais une *condition*.
L'instruction `if` permet de tester la validité de cette *condition* (qui est évaluée par un booléen).
Si la *condition* est évaluée à vraie (True), alors l'instruction que nous avons indentée après le « : » est exécutée, si la condition est évaluée à fausse (False), cette ligne ne s'exécutera pas.
Cas particuliers où la *condition* est le booléen True ou le booléen False :

```
if True :
    print("Exécutée car cond vraie")
```

```
if False :
    print("Non exécutée car fausse")
```

II) Opérateurs de comparaisons et de logique

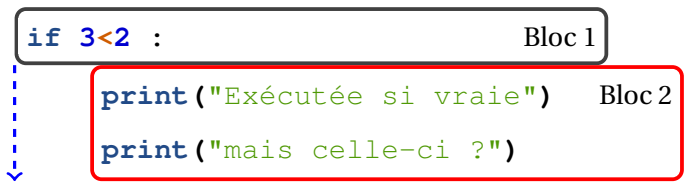
La *condition* évaluée après l'instruction `if` peut contenir les opérateurs de comparaison ou de logique suivants :

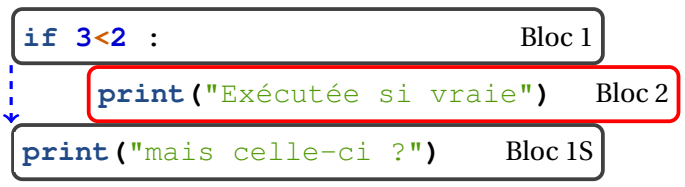
`x==2`	évaluée à vraie si x est égal à 2
`x!=4`	évaluée à vraie si x est différent de 4
`x>-1`	évaluée à vraie si x est strictement supérieur à -1
`x>=3`	évaluée à vraie si x est supérieur ou égal à 3
`x<5`	évaluée à vraie si x est strictement inférieur à 5
`x<=2`	évaluée à vraie si x est inférieur ou égal à 2
`not (x==3)`	évaluée à vraie si x est différent de 3
`1<x and x<3`	évaluée à vraie si x est dans l'intervalle]1 ;3[
`x==2 or x>3`	évaluée à vraie si x=2 ou x strictement plus grand que 3
`"x" in "age"`	évaluée à vraie si `"x"` vaut `"a"`, `"g"` ou `"e"`

III) Instructions composées, les blocs d'instructions

```
if 3<2 :
    print("Exécutée si vraie")
    print("mais celle-ci ?")
```

```
if 3<2 :
    print("Exécutée si vraie")
print("mais celle-ci ?")
```





La condition est fausse donc le bloc 2 n'est pas exécuté, on passe au bloc suivant. Un bloc est repéré par son indentation.
Dans le premier cas il n'y a pas de bloc après le bloc 2 donc rien ne s'affiche, dans le deuxième cas on retourne dans le bloc 1 pour exécuter l'instruction.

1) Instructions imbriquées

Prog1.py

```
if x>1 :
    print("Plus que 1")
    if x<100 :
        print("Moins que 100")
```

Prog2.py

```
if x>1 :
    print("Plus que 1")
```

Prog3.py

```
if x>1 :
    print("Plus que 1")
if
    print("Moins que 100")
```

① Tapez `x=110` dans la console puis exécutez Prog1.py. Affichage ?

② Tapez `x=-10` dans la console puis exécutez Prog1.py. Affichage ?

③ Est-ce-que pour une certaine valeur de `x`, Prog1.py peut afficher seulement « Moins que 100 » ?

④ Délimitez les blocs de Prog1.py

⑤ Complétez Prog2.py pour qu'il affiche « Moins que 100 » à tous les nombres plus petits que 100.

⑥ Complétez Prog3.py pour qu'il soit équivalent à Prog1.py.

2) Instructions mutuellement exclusives

```
def f(x):
    if x>2 :
        return "Plus que 2"
    else :
        return "2 ou moins"
```

① Que renvoient `f(4)`, `f(-1)` et `f(2)` ?

② Que fait l'instruction `else` ?

③ Délimitez les blocs à partir de la deuxième ligne.

Créez une fonction `inverse` de paramètre `x` qui renvoie « 0 n'a pas d'inverse » si `x` vaut `0` et renvoie l'inverse de x sinon.

Créez une fonction `absolue` de paramètre `x` qui renvoie la valeur absolue de `x` c'est-à-dire `x` si `x` est positif et l'opposé de `x` si `x` est négatif.

On peut aussi créer un programme qui possède plus d'une instruction alternative en utilisant l'instruction `elif` (contraction de `else if`), les instructions du bloc qui le suivent ne seront exécutées que si la condition qui suit elif est évaluée à vraie **et** que les conditions précédentes sont évaluées à fausses :

```
def g(x):
    if x<2 :
        return "Petit"
    elif x<10 :
        return "Moyen"
    else :
        return "Grand"
```

① Que renvoient `g(4)`, `g(-1)` et `g(22)` ?

② Pour quelles valeurs de `x` la fonction `g` renvoie « Moyen » ?

③ Délimitez les blocs à partir de la deuxième ligne.

On considère la fonction mathématique h définie pour tout nombre réel par :

$$h(x) = \begin{cases} -x-1 & \text{si} \quad x \leq -2 \\ 0,5x+2 & \text{si} \quad -2 < x \leq 4 \\ -0,5x+8 & \text{si} \quad x > 4 \end{cases}$$

Écrivez une fonction informatique en langage Python correspondant à la fonction h. Placez des points dans le graphique en vous aidant de cette fonction puis dessinez la courbe représentative de la fonction h sur $[-4;8]$.