ISN_Chap1_Variables_Tests_Corrige

September 16, 2019

1 ISN Chapitre 1 Variables et tests

2 Préambule

Ce fichier est un notebook Python.
Il comporte deux types de cellules :

- les cellules d'édition dans lesquelles vous pouvez saisir du texte éventuellement enrichi de mises en formes ou de liens hypertextes avec la syntaxe du langage HTML simplifié qui s'appelle Markdown. Voir http://daringfireball.net/projects/markdown/ pour la syntaxe de Markdown.
- les cellules de code où l'on peut saisir du code Python3 puis le faire exécuter avec la combinaison de touches `CTRL + RETURN`

Une cellule peut être éditée de deux façons différentes :

- en mode *commande* lorsqu'on clique sur sa marge gauche qui est surlignée alors en bleu, on peut alors :
  - changer le type de la cellule en appuyant sur `m` pour passer en cellule Markdown ou sur `y` pour passer en cellule de code
  - insérer une cellule juste au-dessus en appuyant sur `a`
  - insérer une cellule juste en-dessous en appuyant sur `b`
  - couper la cellule en appuyant sur `x` etc ...
- en mode *édition* lorsqu'on clique sur l'intérieur de la cellule.

L'aide complète sur les raccourcis claviers est accessible depuis le bouton `Help` dans la barre d'outils ci-dessus.

3 Installation de l'Ide Pyzo et de la distribution Python Anaconda

- Le plus simple est d'installer l'environnement de développement (ou Integrated Development Environment) Pyzo puis la distribution Python Anaconda en suivant le guide rapide d'installation sur le site de Pyzo. Installez bien Anaconda et non pas Miniconda (sauf sur votre clef USB éventuellement) car Anaconda inclut tous les modules nécessaires et le Jupyter Notebook si pratique que vous êtes en train d'utiliser. Par la suite nous travaillerons plutôt avec l'IDE Pyzo.

- On peut installer des modules supplémentaires depuis Pyzo avec l'un des outils conda ou pip, voir http://www.pyzo.org/install_packages.html#install-packages.
- `pygame` pose problème avec conda et pip. Dans ce cas on se rend sur le site http://www.lfd.uci.edu/~gohlke/pythonlibs/ et on télécharge le paquet wheel de pygame qu'on installe avec pip install paquet.whl
- En cas de problème contactez le professeur par la messagerie de l'ENT.

4 Devinette

```
In [1]: from random import randint

        n = randint(1, 100)
        print('Python vient de choisir un nombre entre 1 et 100.')
        r = int(input('Devinez-le : '))
        while r != n:
            if r < n:
                print('Trop petit !')
            else:
                print('Trop grand !')
            r = int(input('Essayez encore : '))
        print('Bravo, vous avez trouvé !')
```

```
Python vient de choisir un nombre entre 1 et 100.
Devinez-le : 50
Trop grand !
Essayez encore : 25
Trop grand !
Essayez encore : 12
Trop grand !
Essayez encore : 6
Trop grand !
Essayez encore : 3
Trop grand !
Essayez encore : 2
Trop grand !
Essayez encore : 1
Bravo, vous avez trouvé !
```

5 Variables et affectation

Une *variable* est l'association d'un espace de la mémoire de l'ordinateur, accessible par son nom, et d'une valeur que l'on y stocke. En Python on définit une variable par une instruction d'affectation, par exemple pour affecter la valeur 12 à la variable de nom `a` on écrit :

```
a = 12
```

Un langage de programmation fixe des contraintes pour les noms de variables. En Python les seuls caractères autorisés sont les caractères non accentués, les chiffres (sauf en première position) et le tiret bas (mais pas le tiret haut).

```
1var = 13      #nom incorrect
var1 = 13      #nom correct
var-1 = 14      #nom incorrect
var_1 = 14      #nom correct
```

Un langage de programmation n'accepte par défaut qu'un nombre fini de types de valeurs, ainsi une variable est caractérisée par son **type** accessible par la fonction `type` et qu'on peut afficher avec la fonction `print`:

```
type(a)
```

```
In [2]: a | 1 = 5


          File "<ipython-input-2-83dfe44b2ff3>", line 1
        a  1 = 5
          ^
    SyntaxError: invalid character in identifier



In [3]: u = v


        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-3-5188a734531e> in <module>()
    ----> 1 u = v


        NameError: name 'v' is not defined


In [4]: 5 = a


          File "<ipython-input-4-be8696ed204d>", line 1
        5 = a
            ^
    SyntaxError: can't assign to literal



In [5]: else = 5
```

```
  File "<ipython-input-5-49b8cd04d31e>", line 1
    else = 5
         ^
SyntaxError: invalid syntax
```

```
In [6]: print("Hello World")

Hello World
```

```
In [7]: print = 1789

In [8]: print("Hello World")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-8-fb1c7aebe012> in <module>()
----> 1 print("Hello World")


TypeError: 'int' object is not callable
```

On peut retrouver la fonction `print` en important de nouveau le nom `print` depuis le module `builtins`. Celui-ci est toujours importé par défaut, c'est pourquoi les fonctions `built-in` comme `print` sont toujours disponibles.

```
In [9]: from builtins import print

In [10]: print("Hello World")

Hello World
```

5.1 Exemples d'affectation

```
In [11]: a = 5

In [12]: a

Out[12]: 5

In [14]: b = a * ( a + 1 )
```

```
In [15]: b

Out[15]: 30

In [16]: a = max( 1 , a , 50 )

In [17]: a

Out[17]: 50
```

5.2 Incrémentation / Décrémentation d'une variable

```
In [18]: a = 1789

In [19]: a = a + 1

In [20]: a

Out[20]: 1790
```

Une notation raccourcie pour l'incrémentation

```
In [21]: a += 1

In [22]: a

Out[22]: 1791
```

De même pour la décrémentation

```
In [25]: a = 1789

In [26]: a = a - 1

In [27]: a

Out[27]: 1788

In [28]: a -= 1

In [29]: a

Out[29]: 1787
```

5.3 Exercice1 : permutation de variables

```
In [30]: x = 1
         y = 100
         y = x
         x = y

In [31]: x

Out[31]: 1

In [32]: y

Out[32]: 1
```

La permutation des valeurs des variables x et y n'a pas fonctionné, comme on peut le constater en affichant l'état courant du programme, qui trace l'évolution des variables x et y

```
In [4]: #Pour l'utilisation de Python Tutor dans le notebook
        from metakernel import register_ipython_magics #nécessite d'installer metakernel avec
        register_ipython_magics()

In [5]: %%tutor

        x = 1
        y = 100
        y = x
        x = y
```

```
<IPython.lib.display.IFrame at 0x7f891c124c18>
```

5.4 Entraînement 1

Comment obtenir la permutation circulaire des valeurs des trois variables x, y et z ?

```
In [1]: #Initialisation
        x = 843
        y = 844
        z = 845

        #Variable de stockage
        a = x
        x = y
        y = z
        z = a
        print(x, y, z)
```

```
844 845 843
```

Méthode pythonique

```
In [3]: x, y, z = 843, 844, 845
        x, y, z = y, z, x
        print(x, y, z)
```

```
844 845 843
```

5.5 Exercice 4 Etat courant d'un programme

```
In [6]: %%tutor

        a = 10
        b = 7
        c = a + b
        a = 3
        c = a - b
```

```
<IPython.lib.display.IFrame at 0x7f891c124b70>
```

5.6 Entrainement 2

Le programme ci-dessous échange les valeurs des variables x et y.

```
In [8]: %%tutor

        x = 731
        y = 734
        x = x + y
        y = x - y
        x = x - y
```

```
<IPython.lib.display.IFrame at 0x7f891c124c88>
```

6 Type

Quelques exemples

```
In [10]: x = 101
         type(x)
```

```
Out[10]: int
```

```
In [11]: x = 101.    #attention au point
         type(x)
```

```
Out[11]: float

In [12]: y = 101

In [13]: type(x == y)

Out[13]: bool

In [14]: x == y

Out[14]: True

In [15]: type(False)

Out[15]: bool

In [16]: a = (1, 3.14159)
         type(a)

Out[16]: tuple

In [17]: ch = "Hello world"

In [18]: type(ch)

Out[18]: str

In [21]: L = [ 1 , 'abcdefg' , 2.71 ]
         type(L)

Out[21]: list

In [22]: def f ( x ) :
             return x + 1

In [23]: type(f)

Out[23]: function
```

6.1 Opérateurs usuels

- Python dispose *d'opérateurs* arithmétiques pour réaliser des opérations sur des *entiers* ou sur des approximations décimales (en notation pointée) de nombres réels qu'on appelle des *flottants*.

```
In [25]: # L'opérateur + réalise l'addition de deux entiers
         1 + 2

Out[25]: 3

In [26]: # L'opérateur + réalise aussi l'addition de deux flottants
         2.0 + 3.5
```

```
Out[26]: 5.5

In [27]: # L'opérateur * réalise la  multiplication  de deux entiers
         3 * 2

Out[27]: 6

In [28]: # L'opérateur * réalise aussi la  multiplication  de deux flottants
         2.0 * 3.14

Out[28]: 6.28

In [29]: # L'opérateur - réalise la soustraction de deux entiers ou de deux flottants
         (10 - 4, 3.14 - 0.14)  #on utilise des parenthèses pour créer un couple de valeurs

Out[29]: (6, 3.0)

In [30]: # L'opérateur ** réalise l'exponentiation d'un entier par un entier, idem pour les fl
         (2 ** 3, 2.0 ** 0.5)

Out[30]: (8, 1.4142135623730951)

In [31]: # L'opérateur // retourne le quotient de la division euclidienne de deux entiers
         15 // 4

Out[31]: 3

In [32]: # L'opérateur / retourne l'approximation du quotient par le flottant le plus proche
         (15 / 4, 2/3)

Out[32]: (3.75, 0.6666666666666666)

In [33]: # L'opérateur % retourne le reste de la division euclidienne de deux entiers
         15 % 4

Out[33]: 3
```

Les opérateurs arithmétiques suivent des *règles de précédence* (priorité de l'exponentiation sur la multiplication et de la multiplication sur l'addition). Ces priorités peuvent être changées dans un calcul à l'aide de parenthèses.

```
In [35]: ( 1 + 2 * 3 ** 2) / 2

Out[35]: 9.5

In [36]: 1 + 2 * 3 ** 2 / 2

Out[36]: 10.0
```

Pour utiliser des fonctions mathématiques spécifiques, comme `cos` ou `sin`, il faut les importer depuis un module.

Le module `math` contient la plupart des fonctions usuelles.

```
In [37]: cos(4)

---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-37-2125acb124c4> in <module>()
----> 1 cos(4)


NameError: name 'cos' is not defined


In [38]: from math import cos
         cos(4)

Out[38]: -0.6536436208636119
```

7 Test

Un *test* est composé d'une *instruction de contrôle* puis d'un *bloc d'instructions*.

En `Python`, l'instruction de contrôle commence par le mot clef `if` suivi d'une condition à valeur booléenne (`True` ou `False`) et se termine par le symbole `:`.

Le bloc d'instructions qui suit s'exécute si et seulement si la condition de l'instruction de contrôle a pour valeur `True` . Il correspond à un *embranchement* dans le flux d'instructions.

En `Python`, un *bloc d'instructions* commence à la première ligne suivant le symbole `:` et son imbrication dans le reste du programme est caractérisée par son niveau d'*indentation*. Toutes les instructions d'un même bloc doivent avoir le même niveau d'indentation. Le flux principal d'instructions est collé contre la marge puis chaque niveau d'indentation est décalé de quatre espaces ou une tabulation vers la droite.

```
# flux parent
if conditionA:
    #bloc d'instructions exécuté si la valeur de conditionA est True
    if conditionB:
        #bloc d'instructions imbriqué, exécuté si conditionA  est à True
        # et si conditionB est à True
# retour au flux parent
```

Si la condition du *test* n'est pas vérifiée, on peut prévoir l'exécution d'un bloc d'instructions alternatif après un `else :`.

```
# flux parent
if condition:
    #bloc d'instructions exécuté si la valeur de condition est True
else:
    #bloc d'instructions exécuté si la valeur de condition est False
#retour au  flux parent
```

On peut aussi tester d'autres conditions mutuellement exclusives pour modéliser un choix entre plus de deux alternatives. Chaque condition testée dans un `elif condition :` commande un bloc d'instructions et un bloc d'instructions par défaut peut se trouver après un `else :`.

```
# flux parent
if condition1:
    #bloc d'instructions exécuté si la valeur de condition1 est à True
elif condition2:
    #bloc d'instructions  si  condition2 est True (et condition1 est à False)
elif condition3:
    #bloc d'instructions si  condition3 est True (et condition1 et condition2 sont à False)
else :
    #bloc d'instruction si toutes les conditions précédentes sont à False
#retour au flux parent
```

7.1 Entrainement 3

```
In [41]: %%tutor
         x, y, z = 2, 10, 10
         if x == 2:
             y = 5
         else:
             y = 6
             z = 7
```

```
<IPython.lib.display.IFrame at 0x7f891c158e80>
```

```
In [42]: %%tutor
         x, y, z = 2, 10, 10
         if x == 2:
             y = 5
         else:
             y = 6
         z = 7
```

```
<IPython.lib.display.IFrame at 0x7f891c158860>
```

7.2 Exercice 5 Maximum de deux nombres

```
In [44]: x = int(input('x ? '))
         y = int(input('y ? '))
         if x >= y:
             print(x)
         else:
             print(y)
```

```
x ? 4
y ? 6
6
```

7.3 Entraînement 4 Conjecture de Syracuse

```
In [5]: n = int(input('n ?'))
        if n % 2 == 0:
            print(n // 2)
        else:
            print(3 * n + 1)
```

```
n ?4
2
```