01-cours_python_variables_print_input

January 11, 2020

1 Prise en main de l'environnement, variables, interactions avec l'utilisateur

1.1 Objectifs

Le but de cette séance est d'exécuter quelques commandes simples et de se familiariser avec quelques notions de base d'algorithmique.

1.2 Les différentes versions de Python

- Python 2.7 (ne sera plus supportée fin 2019)
- Python 3.x (actuellement Python 3.7)

1.3 Pour bien s'organiser...

1.3.1 Quelques notions

En programmation informatique pour les noms de fichiers (mais aussi pour les noms de variables, de fonctions...) : - Pas d'accent - Pas d'espaces

1.3.2 Mettre en place son organisation avec des répertoires

- Choisir un répertoire de son ordinateur
- Dans ce dernier, créer un répertoire cours
- Dans ce dernier, créer un répertoire python
- Dans ce dernier, créer un répoertoire "seance1"

1.3.3 Les fichiers Python

- Chaque programme écrit sera enregistré dans un fichier qui portera un nom avec l'extension ".py".
- Pour chaque exercice proposé, vous l'enregistrerez dans le répertoire "seance1" avec le nom qui vous convient, par exemple *exercice1.py* (ou *exercice1_affectation_variable.py* si vous préférez un nom plus parlant).

1.3.4 Vue d'ensemble

Note : ce ne sont pas des conseils, ce sont des obligations !

1.4 Utilisation d'un environnement de développement (IDE) : Spyder

Pour écrire un programme en Python, on peut (mais ce n'est pas obligatoire !) utiliser un environnement de développement (IDE en anglais) qui permet notamment d'afficher les mots-clefs en couleur, de consulter la valeur d'une variable, de bénéficier de l'auto-complétion… Celui que nous utiliserons répond au nom de *Spyder*.

Dans une fenêtre de l'éditeur Spyder, taper le magnifique programme suivant :

```
# Cool, this is my first python script!
print("Hello world!")
```

- L'enregistrer dans le répertoire : "cours - python - seance1" et le nommer *hello_world.py*
- L'exécuter…
- Noter également qu'en plus de la fenêtre principale, vous avez également une fenêtre nommée "console IPython". Elle peut être très pratique pour effectuer des tests rapides de petits programmes (Python).

1.5 Les variables

1.5.1 Présentation

- Une variable est une référence vers une zone de contenu (adresse mémoire de l'ordinateur). C'est donc une façon de nommer un contenu pour pouvoir le retrouver ultérieurement.
- Lorsqu'on exécute des calculs, qu'onappelle des fonctions ou qu'on effectue divers traitements, il est souvent intéressant de pouvoir en conserver le résultat. C'est tout le principe des variables : garder la trace d'une valeur.
- Les noms de variables sont des identificateurs arbitraires, de préférence assez courts tout en étant aussi explicites que possible, de manière à exprimer clairement ce que la variable est censée référencer (la sémantique de la donnée référencée par la variable).

1.5.2 Les noms de variables (détail)

- Une variable ne peut être composée que de lettres minuscules ou majuscules, de chiffres et du symbole souligné "_".
- Une variable ne peut commencer par un chiffre.
- De plus le langage Python est sensible à la casse (les majuscules et les minuscules). Ainsi var1, Var1 et VAR1 sont des variables différentes.
- Ne jamais utiliser de mot clé réservé comme nom de variable (même si Python l'autorise). Exemple: *list*, *str*,... Si dans votre éditeur, le nom de votre variable apparaît en couleur, posez-vous des questions...
- Ne jamais mettre d'accents dans les noms de variables.
- On utilisera toujours des noms de variables explicites (*toto*, *var* etc. sont à proscrire).
- Conventions de nommage:
  - ma_variable
  - MA_CONSTANTE
  - ma_fonction
  - MaClasse

Ces conventions de nommages ne sont pas obligatoires mais recommandées. Elles sont régies par la PEP8 (Python Extension Proposals).

1.5.3 Affectation

L'**instruction** d'affectation est notée = en Python : `nom = valeur`. Le nom de la variable à modifier est placé dans le membre de gauche du signe =, la valeur qu'on veut lui attribuer dans le membre de droite.

Ainsi, les deux membres du signe = ne jouent pas le même rôle. Le membre de droite de l'affectation est d'abord **évalué** sans être modifié. Puis la valeur obtenue est affectée à la variable dont le nom est donné dans le membre de gauche de l'affectation. Cette opération ne modifie que le membre de gauche de l'affectation.

Le membre de droite d'une affectation peut être une **constante** ou une **expression** évaluable.

[3]:
```
# Exemple d'affectation d'une variable
x = 1
```

```
y = 1 + 2/3
z = y - x
z = z + 4  # On peut aussi écrire en Python z += 4
print(z)  # L'instruction print permet d'effectuer un affichage à l'écran
```

```
4.666666666666666
```

Notons que, pour utiliser une variable dans une expression, elle doit avoir été préalablement définie:

[8]:
```
# Programme correct
x = 1
y = x
```

[9]:
```
# Programme incorrect
z = t + 3
```

```
␣
↪---------------------------------------------------------------------------

NameError                                 Traceback (most recent call␣
↪last)

<ipython-input-9-2539b134b630> in <module>
  1 # Programme incorrect
----> 2 z = t + 3


NameError: name 't' is not defined
```

1.5.4 Exercices

Instruction versus expression

Parmi les lignes ci-dessous indiquer quelles sont des instructions et quelles sont des pures expressions :

- $1 + 3$
- $x = 2$
- $print(x)$
- $cos(x)$
- x
- $y = sin(x)$
- $z = x + 2$
- $4 = p$

Correction :

- $1 + 3$: expression

- x = 2 : instruction
- print(x) : instruction
- cos(x) : expression
- x : expression
- y = sin(x) : instruction
- z = x + 2 : instruction
- 4 = p : incorrect !

Exécuter un algorithme dans sa tête

Sans utiliser Python, indiquer le contenu de la variable c à la fin du programme ci-dessous :

```
a = 1
b = 3
c = a - b
c = c + 3
```

Calcul de la moyenne

Soient quatre nombre : 8, 20, 30 et 2. En utilisant six variables, calculer la moyenne et l'afficher.

[4]:
```
n1 = 8
n2 = 20
n3 = 30
n4 = 2
somme = n1 + n2 + n3 + n4
moyenne = somme / 4
print(moyenne)
```

```
15.0
```

Un grand classique de la programmation : l'échange de variables

Soient deux variables x = 1 et y = 2. On souhaite avoir à la fin du programme x = 2 et y = 1. Le professeur de Python a commencé à écrire un programme mais il ne fonctionne pas, il faut donc l'aider :

[5]:
```
x = 1
y = 2
y = x
x = y
print(x)
print(y)
```

```
1
1
```

[6]:
```
# Première version : utiliser une variable temporaire
x = 1
y = 2
tmp = y
y = x
```

```
x = tmp
print(x)
print(y)
```

```
2
1
```

[7]:
```
# Deuxième version : utiliser les "facilités" du langage Python
x = 1
y = 2
x, y = y, x
print(x)
print(y)
```

```
2
1
```

A la librairie

Un libraire propose une réduction de 3.5% sur le prix hors taxes (HT) d'un livre à 12.35 € HT. Sachant que la taxe sur la valeur ajoutée (TVA) sur les livres est de 5.5%, écrire un programme qui permettra d'afficher le prix au client.

[23]:
```
# Initialisation des variables
prix_ht = 12.35
tva = 5.5
taux_reduction = 3.5

# Calcul de la taxe et de la réduction
reduction = prix_ht * taux_reduction / 100
prix_ht_reduit = prix_ht - reduction
taxe = prix_ht_reduit * tva / 100

# Calcul du résultat final
prix_vente = prix_ht_reduit + taxe
print(prix_vente)
```

```
12.57322625
```

1.5.5 Le typage dynamique

Jusqu'à présent nous n'avons manipulé que des variables de type entier ou décimal. Le fait d'exécuter une instruction d'affectation définit en même temps le type de la variable. Exemples :

```
nombre_entier = 22
nombre_decimal = 12.5
chaine_caracteres = "Bonjour"  # Ou 'Bonjour' cela est équivalent
```

C'est ce qu'on appelle le **typage dynamique**. Notez que cela n'est pas valable pour tous les langages informatiques. C'est une particularités de Python.

Les types de base Les types de base (*non mutables*) de variables sont :

- ***int*** pour les entiers,
- ***float*** pour les décimaux (nombres flottants)
- ***str*** pour les chaînes de caractères.
- ***bool*** pour les booléens

Pourquoi le type est il important ? C'est lui qui va déterminer le type d'opération qu'on va pouvoir effectuer ainsi que le résultat :

[10]:
```
# Opérations sur les nombres
a = 1
b = a + 2  # Entier + Entier
print(b)
c = 1.4
d = b + c  # Entier + Décimal
print(d)
```

```
3
4.4
```

[11]:
```
# Opérations sur les chaînes de caractères
prenom = "Wolfgang Amadeus"
nom = "Mozart"
separateur = " "  # C'est un espace
nom_complet = prenom + separateur + nom  # Concaténation de chaînes
print(nom_complet)
```

```
Wolfgang Amadeus Mozart
```

Il n'est pas possible de mixer certaines opérations si les types sont différents :

[14]:
```
a = 1
b = 2
c = a + b  # OK -> 3
d = "1"
e = "2"
f = d + e  # OK -> "12"
print(c)
print(f)
g = a + e  # Erreur
```

```
3
12
```

```
␣
↪---------------------------------------------------------------------------
```

```
TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-14-c3ff953c1c7b> in <module>
  7 print(c)
  8 print(f)
----> 9 g = a + e  # Erreur


TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Pour effectuer des opérations entre variables de types différents, il faut effectuer une **conversion explicite** :

[17]:
```
# Conversion str -> int
a = 1
b = "2"
c = a + int(b)
print(c)
# Conversion int -> str
chaine = "Nombre = "
valeur = 12
resultat = chaine + str(valeur)
print(resultat)
```

```
3
Nombre = 12
```

On peut utiliser la *fonction* Python *type()* pour connaître le type d'une variable en Python

[12]:
```
nom = "Mozart"
print(type(nom))
age = 78
print(type(age))
condition = True
print(type(condition))
```

```
<class 'str'>
<class 'int'>
<class 'bool'>
```

Remarque: dans un programme, même si **ce n'est pas recommandé** pour des questions de lisiblité, grâce au typage dynamique, il est possible qu'un même nom de variable prenne plusieurs types :

[16]:
```
variable_fourre_tout = "toto"
print(type(variable_fourre_tout))
variable_fourre_tout = 123
```

```
print(type(variable_fourre_tout))
```

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

1.5.6 Exercices

Construction d'une chaîne de caractères

Afficher le message suivant "La voiture roule à 125.5 km/h.".

Utiliser :

- Deux variables de type chaine (str)
- Une variable de type décimal (float).

On utilisera également une variable de type chaîne pour stocker le résultat avant de l'afficher (avec la fonction *print*).

[19]:
```
debut_phrase = "La voiture roule à "
vitesse = 125.5
fin_phrase = " km/h."
resultat = debut_phrase + str(vitesse) + fin_phrase
print(resultat)
```

```
La voiture roule à 125.5 km/h.
```

1.5.7 Synthèse

L'affectation d'une variable a donc pour effet de réaliser plusieurs opérations en mémoire :

- Créer et mémoriser une valeur particulière
- Créer et mémoriser un nom de variable et établir un lien (par un système interne de pointeurs) entre le nom de la variable et l'emplacement mémoire de la valeur correspondante
- Attribuer un type déterminé à la variable

1.6 Fonctions d'entrées/sorties

1.6.1 La fonction *print*

La fonction ***print(...)*** permet d'afficher à l'écran un message passé en **argument** (ou en paramètres) - c'est-à-dire entre les parenthèses. Voici quelques exemples :

[21]:
```
# Avec un seul argument (une seule chaîne de caractères)
print("Bonjour")

# Avec un seul argument et des variables
age = 20
## Cela fonctionne
print("Vous avez " + str(age) + " ans")
## Cela ne fonctionne pas
print("Vous avez " + age + " ans")
```

```
# Cela fonctionne et c'est mieux (plus lisible)
print("Vous avez {0} ans".format(age))
# Cela est encore mieux
print(f"Vous avez {age} ans")  # A partir de Python 3.6

# Avec plusieurs arguments (3)...
print("Vous avez", age, "ans")
```

```
Bonjour
Vous avez 20 ans
Vous avez 20 ans
Vous avez 20 ans
Vous avez 20 ans
```

1.6.2 La fonction *input*

La fonction ***input(...)*** provoque une interruption de l'exécution d'un programme pour que l'utilisateur puisse entrer une valeur. Le programme reprend son cours quand l'utilisateur appuie sur la touche *Entrée*.

- L'**argument** passé à la fonction *input* est le message qui sera affiché sur l'écran de l'utilisateur.
- La fonction *input* **renvoie** une valeur (qui est celle saisie par l'utilisateur). La plupart du temps on la stockera dans une variable. (Notez au passage que la fonction *print* ne renvoie aucune valeur).

Exemple :

[ ]:
```
nom = input("Quel est votre nom ?")
```

Attention: la fonction ***input(...)*** **renvoie toujours une chaîne de caractères**. Si l'on veut effectuer des opérations mathématiques sur une variable entrée par l'utilisateur (par exemple), il faut préalablement la convertir en effectuant un transtypage (cast, en anglais). Pour les trois types principaux, les transtypages s'effectuent en utilisant les fonctions suivantes : * *int(...)* * *float(...)* * *str(...)*

[3]:
```
annee_naissance = input("Entrez votre année de naissance : ")
annee_actuelle = input("Entrez l'année actuelle : ")

# Calcul de l'âge avec conversion
age = int(annee_actuelle) - int(annee_naissance)


print("Vous avez " + str(age) + " ans")
# Instruction équivalente avec conversion implicite
print(f"Vous avez {age} ans")
```

```
Entrez votre année de naissance : 1999
Entrez l'année actuelle : 2019
Vous avez 20 ans
```

```
Vous avez 20 ans
```

1.6.3 Exercice : une calculatrice vraiment basique

Créer un programme qui demande deux entiers a et b à l'utilisateur. Le programme doit calculer puis afficher les résultats suivants :

- la somme de a et de b
- la soustraction de b par a
- le produit de a et de b
- la quotient de b par a
- le reste quand b est divisé par a
- le résultat de a puissance b

[1]:
```
valeur_a = input("Valeur de a :")  # valeur_a est une chaîne
valeur_b = input("Valeur de b :")  # valeur_b aussi
a = int(valeur_a)  # a est désormais un entier
b = int(valeur_b)  # b aussi
somme = a + b
print(f"Somme de {valeur_a} et {valeur_b} : {somme}")
soustraction = a - b
print(f"Soustraction de {valeur_a} et {valeur_b} : {soustraction}")
multiplication = a * b
print(f"Multiplication de {valeur_a} et {valeur_b} : {multiplication}")
quotient = b / a
print(f"Quotient de {valeur_b} par {valeur_a} : {quotient}")
reste = b % a
print(f"Reste de {valeur_b} par {valeur_a} : {reste}")
puissance = a**b
print(f"{valeur_a} puissance {valeur_b} : {puissance}")
```

```
Valeur de a :12
Valeur de b :3
Somme de 12 et 3 : 15
Soustraction de 12 et 3 : 9
Multiplication de 12 et 3 : 36
Quotient de 3 par 12 : 0.25
Reste de 3 par 12 : 3
12 puissance 3 : 1728
```

1.6.4 Exercice : calculer la surface d'une pièce

Ecrire un programme qui demande à l'utilisateur d'entrer successivement la largeur puis la longueur d'une pièce d'une maison. Une fois que les valeurs ont été lues, le programme doit afficher la surface correspondante.

[2]:
```
# On utilise la fonction float pour effectuer la conversion
# explicite en float, car la valeur saisie n'est pas forcément entière
largeur = input("Largeur de la pièce (m) : ")
```

```
longueur = input("Longueur de la pièce (m) : ")
surface = float(largeur) * float(longueur)
print(f"La surface de la pièce est : {surface} m²")
```

```
Largeur de la pièce (m) : 12
Longueur de la pièce (m) : 4.5
La surface de la pièce est : 54.0 m²
```

1.7 Les commentaires dans un programme

Le langage Python permet au programmeur de placer des commentaires dans son code : des commentaires sont des lignes de texte qui ne seront pas interprétées lors de l'exécution du programme. En Python, une ligne est en commentaire si elle commence par le caractère dièse. Les lignes suivantes sont en commentaires :

```
# Voici une ligne en commentaire
# print("si on décommente cette ligne, je m'affiche à l'exécution")
# On peut aussi mettre un commentaire sur la même ligne :
print("Bonjour") # Ici on est poli
```

Pour écrire un commentaire sur plusieurs lignes, on va utiliser cette notation:

```
"""
Ceci est
Un commentaire
Sur plusieurs lignes
"""
```

- Un des moyen d'analyser du code est d'en commenter l'ensemble des lignes, puis de décommenter une à une les lignes pour en observer le fonctionnement. A coupler avec l'utilisation de la fonction *print* pour afficher dans la console les résultats intermédiaires.
- Il est indispensable de commenter son code:
  - Description générale de ce que fait le programme.
  - Commentaires internes au code.
- Cependant les commentaires doivent rester pertinents.

```
# Commentaire non pertinent
# Initialisation de la variable price
prix = 100

# Commentaire pertinent
# Un nombre premier est un entier naturel qui admet exactement deux
# diviseurs distincts entiers et positifs (qui sont alors 1 et lui-même)
# ...
```

- Un code trop commenté est un code douteux (on fait trop compliqué).
- Il faut donc trouver le juste équilibre.

1.8 Pour aller plus loin

1.8.1 Python en ligne de commande

La console Python Pour des raisons de confort, nous avons utilisé Spyder pour écrire et exécuter un programme. Cependant cela n'est pas obligatoire, on peut très bien utiliser la console Python : - Sous Windows, taper *cmd* après avoir ouvert le menu de démarrage - Sous Linux / Mac, ouvrir un terminal Taper ensuite la commande :

```
python
```

Si l'installation de Python s'est effectuée correctement, de nouvelles lignes apparaissent dans l'invite de commande, par exemple :

```
Python 3.6.5 |Anaconda 4.1.1 (64-bit)
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

La première commande que nous allons exécuter à l'aide de Python est très simple et consiste à afficher une chaîne de caractères. Rentrer la commande :

[3]:
```
print("Hello World!")
```

```
Hello World!
```

Cette façon de faire peut être utile si l'on doit (ou si l'on veut) effectuer des opérations simples ou des tests. A noter que la console Python sert aussi de calculatrice, on peut par exemple effectuer le calcul suivant :

[2]:
```
(2 + (3 * (10.5 / 4)) ** 2) / 11
```

[2]:
```
5.8196022727272725
```

Exécuter un programme Python

A l'aide de l'invite de commande, se déplacer dans le répertoire contenant un programme écrit précédemment avec Spyder (indication : utiliser la commande "cd"). Ensuite exécuter la commande : `python mon_programme.py`.