VARIABLES ET AFFECTATION EN PYTHON

1 POUR COMMENCER

Pyzo

Python existe sous de nombreuses présentations. Dans le cadre du cours, nous utiliserons la distribution Pyzo que l'on peut télécharger gratuitement sur le site http://www.pyzo.org

C'est cette distribution qui est actuellement utilisée aux concours.

Deux modes de saisie

Les éléments qui composent la fenêtre Pyzo peuvent être réarrangés en faisant glisser les bandeaux correspondants avec la souris. Le choix des éléments à afficher peut être déterminé à partir du menu.

Pour commencer, nous conseillons d'adopter la présentation suivante avec une séparation verticale :

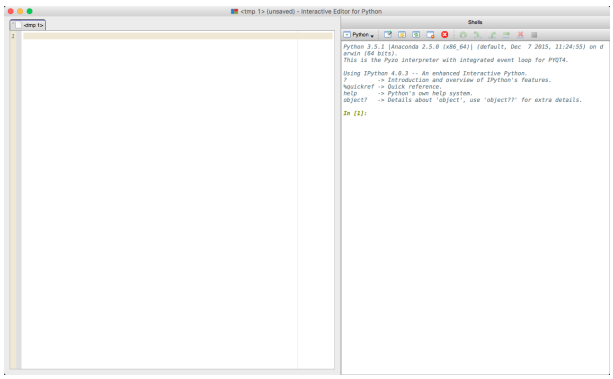


Figure 1: Fenêtre Pyzo

- **Partie de gauche :** le fichier ".py" dans lequel on écrit les instructions. Ce ficher peut être enregistré pour une utilisation ultérieure.
  Pour exécuter les instructions qui y sont écrites on peut taper sur F5 (tout est exécuté) ou F9 (seulement la sélection est exécutée).
- **Partie de droite :** le shell dans lequel apparaissent les réponses de Python lors de l'exécution du fichier. C'est aussi le mode interactif. Python réagit "en direct" à ce que l'on écrit. Il interprète chaque ligne dès que l'on appuie sur la touche "Entrée".

Instructions Python

Les instructions Python s'écrivent les unes à la suite des autres en passant à la ligne entre deux.

Si on veut placer plusieurs instructions sur une même ligne, on peut les séparer d'un point-virgule ";".

Nota : Contrairement à beaucoup d'autres langages, il n'y a pas de point virgule à mettre en fin de ligne.

Commentaires

On peut placer des commentaires dans notre programme Python pour le rendre plus lisible et expliquer ce que l'on code.

Les commentaires sont précédés du signe "#". Dès qu'il voit ce signe, Python passe directement à la ligne suivante sans s'occuper de ce qui est écrit après.

Attention : Dans la suite du cours, il est conseillé de tester tous les exemples sur une machine, pour assimiler la syntaxe.

2 AFFECTATION ET MODIFICATION

A Affection d'une variable

Affectation

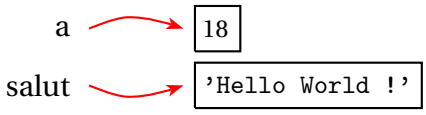
En Python, une variable est une façon de nommer un objet, ou plus précisément, de faire référence à un objet présent dans la mémoire.

Une variable ne *"contient"* pas l'objet, mais pointe vers l'objet. On parle de **référence objet**[1].

Lorsque l'on fait pointer la variable vers un objet, on dit que l'on **affecte** cette variable. En Python, l'affectation se fait avec le signe égal =.

```
# Ceci est un commentaire (précédé par dièse)
a = 18                    # la variable a pointe vers la valeur 18
salut = 'Hello World !' # la variable salut pointe vers le texte 'Hello World !'
```

Python 1: Affectation



Affichage

La commande `print` affiche la valeur vers laquelle pointe une variable[2].

Pour écrire plusieurs éléments à la suite (séparés d'un espace), on les sépare par des virgules en argument de `print`.

```
salut = 'Hello World !'
print(salut)
print('Python vous dit :', salut)
```

Python 2: Affichage

Attention : Python n'est pas un logiciel de calcul formel : il faut affecter une variable (la définir) avant de l'utiliser. C'est comme en maths.

Remarques pour les pros :

- Les variables sont des mots composés des caractères alphanumériques habituels et du signe souligné : "_". Un nom de variable ne peut pas commencer par un chiffre. N'hésitez pas à donner des noms de variable explicite avec suffisamment de lettres.
  Python est *sensible à la casse.* Cela veut dire qu'il distingue les majuscules des minuscules. Ainsi, les variables `var` et `Var` sont différentes.
- Lorsque l'on affecte une variable pour la première fois, elle est créée automatiquement par Python, il n'est pas nécessaire de la déclarer avant comme dans certains langages.
- Une fois que l'on a affecté une valeur à une variable, on peut la modifier en *écrasant* sa "valeur" par une autre. On pointe simplement la flèche vers une autre case mémoire.

Modification d'une variable

```
a = 'glop'
b = 'pas glop'
a = b                     # on écrase la variable a pour lui donner la valeur 'pas glop'
print(a); print(b)
maVariable = 1
print(maVariable)
print(mavariable)         # Renvoie une erreur car la casse est différente
```

Python 3: Modification d'une variable

Attention : Le signe égal n'est pas symétrique pour Python. Dites vous que = se traduit par "pointe vers" (voir la figure 2).

Explication de la figure 2 :

Au début, on affecte la valeur 1 à la variable `a` et la valeur 8 à la variable `b`. Python crée donc ces deux variables et les fait pointer vers des cases mémoires dont l'une contient la valeur 1 et l'autre, la valeur 8.

[1] Ceci sera important lors des copies de listes que nous verrons plus tard.
[2] On peut aussi simplement écrire le nom de cette variable dans le shell et exécuter.

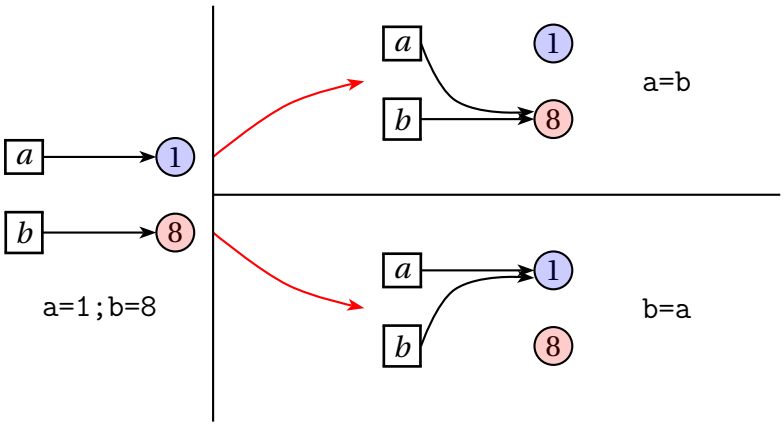


Figure 2: Non symétrie de l'affectation

Si on leur donne l'instruction `a=b` (illustration d'en haut), alors on modifie la variable `a` pour lui donner la valeur de `b`. Le pointeur de `b` ne change pas, par contre, celui de `a` va pointer vers la même case mémoire que celui de `b`.

À l'inverse, si on place l'instruction `b=a` (illustrée en bas), cette fois ci, c'est la variable `b` qui est modifiée et va faire référence à la même case mémoire que `a`.

Remarques pour les pros :

- Python optimise la mémoire et dans l'instruction `a=b`, on fait pointer `a` vers la même case mémoire que `b`. Cela évite d'avoir deux fois la même information en deux endroits du disque différents. Nous verrons que ceci n'est pas sans influence sur le comportement de Python lorsque nous manipulerons des objets plus complexes.
- Dans l'exemple, plus aucune variable ne pointe vers la case mémoire qui contient le 8. On peut donc libérer cet emplacement mémoire pour y placer d'autres informations plus tard. Python s'en occupera pour nous à travers le GC :*garbage collector* ou ramasse-miettes. Le rôle de cet outil intégré à Python est de détecter les cases mémoires vers lesquelles plus aucune variable ne pointe pour les remettre à disposition.
  Tous les langages ne possèdent pas de GC (qui n'a pas que des qualités), ainsi en est-il du langage C pour lequel c'est au programmeur de faire ce travail.

B Affections simultanées

On peut affecter plusieurs variables d'un coup, en les séparant par des virgules (*tuple*). On peut mettre ou non des parenthèses autour du uplet. De manière général, les parenthèses faciliteront la lecture.

```
a,b,c = 1,'deux',3
print(a); print(b); print(c)
(a,b,c) = ('un',2,'trois')
print(a); print(b); print(c)
print('je compte',a,b,c)
```

Python 4: Affectations simultanées

C Échange de deux variables

L'affectation simultanée permet d'échanger facilement les valeurs de deux variables. C'est une spécificité de Python qui est très pratique[3].

```
a = 1
b = 5
(a,b) = (b,a)
print(a); print(b)
```

Python 5: Échange de variables

[3] En général, dans les autres langages il faut faire appel à une variable auxiliaire

3 FONCTIONS

À partir de maintenant, nous aurons besoin d'exécuter plusieurs lignes à la fois. Nous les écrirons donc d'abord dans la partie fichier avant de les compiler (par F5 ou F9).
Pour tester les fonctions ainsi écrites, on peut taper directement dans le shell.

Définition 3.1 :

Une **fonction** est une suite finie d'instructions qui reçoit un argument et renvoie un résultat.

Remarque : On utilise le terme de **procédure** pour désigner une fonction en informatique lorsqu'elle ne renvoie pas de résultats. Dans ce cours, nous assimilerons les deux termes[4].

La syntaxe est la suivante

```
def maFonction(arg):
  suite d'instructions
  suite d'instructions
  ...
  return resultat
```

Python 6: Syntaxe d'une fonction

La fonction est donc composée :

- d'une ligne d'entête commencée par le mot clef **def** et terminée par deux-points,
- d'un bloc d'instructions indenté par rapport à la ligne d'entête
- d'une instruction **return** qui renvoie le résultat (aussi indentée)

Remarques :

- Pour éviter les problèmes d'indentation, il faut demander à Pyzo de transformer automatiquement les tabulations en espaces (depuis le menu).
  En général, on choisit une indentation de 2 ou 4 espaces. C'est à vous de choisir.

```
def␣maFonction(arg):
␣␣suite␣d'instructions
␣␣suite␣d'instructions
␣␣...
␣␣return␣resultat
```

Python 7: Syntaxe d'une fonction avec les espaces

- Après **return**, la fonction ne lit plus rien dans le bloc. Toutes les instructions indentées qui suivent seront ignorées.

```
def plus1(nombre):
  nombre += 1           # ajoute 1 à la variable nombre
  return nombre
  print('blablabla')

# instructions pour tester à écrire dans le shell
plus1(4)
plus1(-7)
plus1(3.75)

b = 9
plus1(b)
print(b)
```

Python 8: Fonction incrémentielle

[4] En Python, tout est fonction : si on ne demande pas de renvoyer une valeur, alors Python de lui-même renvoie la valeur `none`.

Variables globales - variables locales (*plus subtile, peut être lu dans un second temps*)

On remarque que lorsque l'on applique une fonction à une (ou des) variable, Python crée une **nouvelle variable locale** pour ne pas modifier la précédente. Cette variable *locale* n'a d'existence qu'au sein du bloc d'instructions.

Remarque pour les pros : La méthode d'affectation pour les listes fait que la variable locale et la variable globale, bien de différentes, pointent souvent vers le même objet mémoire contenant la liste. Ainsi, la modification de la variable locale affecte aussi la variable globale.
Nous verrons plus loin comment contourner cette difficulté ou au contraire l'utiliser à notre profit.

```
def empile(liste,elt):
  liste += [elt]
  return liste

empile([4],2)
empile([4],[1,2])
liste=[]
empile(liste,4)
liste                #la liste a été modifiée
```

Dans le corps de la fonction, Python, peut utiliser sans problème toutes les variables globales qui ont été définies.

Par contre, dès qu'il définit une nouvelle variable à l'intérieur de la fonction, cette variable est **locale** : cette variable n'a d'existence qu'à l'intérieur de la fonction.

Il est possible de forcer Python à définir une variable **globale** avec le mot clef `global`.

```
def createALocal():
  A = 5
def createA():
  global A
  A = 5

# dans le shell
A = 4

createALocal()
print(A)       # affiche 4 : la variable globale n'est pas modifiée.

createA()
print(A)       # affiche 5 : la variable globale est modifiée.
```

Python 9: Définition d'une variable globale au sein d'une procédure.

4 IMPORT DE BIBLIOTHÈQUES

Au gré des besoins, des programmeurs ont crée de très nombreuses fonctions que l'on peut rajouter à Python. Ils ont placé ces fonctions au sein de **bibliothèques** ou librairies.
Pour utiliser ces fonctions, il faut charger la librairie concernée.

```

pi                      # pi n'est pas défini dans Python par défaut -> erreur

# importer une bibliothèque, mais sans 'sortir' ses objets

import math             # on importe la bibliothèque math
pi                      # pi n'existe toujours pas de Python -> erreur
math.pi                 # pi existe dans la bibliothèque math-> donne valeur approché
    e.

# importer un objet d'une bibliothèque et le 'sortir'
from math import pi     # pour importer la commande pi de la bibliothèque math
pi                      # -> donne valeur approchée
cos(pi)                 # commande cos non définie -> erreur
```

```
14
15 # importer tous les objets d'une bibliothèque et les 'sortir'
16 from math import *      # pour tout importer de la bibliothèque math
17 cos(pi)
```

Python 10: Import d'une bibliothèque

Parfois, lorsque l'on importe plusieurs bibliothèques, on peut avoir des conflits entre des fonctions qui sont définies différemment selon la bibliothèque. il est alors préférable d'importer la bibliothèque suivant la première procédure : `import <bibliothèque>`.
Mais à chaque fois que l'on utilise une fonction de cette bibliothèque, il faut alors la faire précéder du nom de cette bibliothèque et d'un point. Comme c'est un peu long, on donne alors un *alias*, un surnom à cette bibliothèque.

```
1 randint(1,10)   #nombre entier aléatoire entre 1 et 10 (inclus)
2 import random
3 random.randint(1,10)
4
5 import random as rnd
6 rnd.randint(1,10)
```

Python 11: Import d'une bibliothèque avec un alias

Bibliothèque	**Description**
math	Outils de base en mathématiques
random	Nombres (pseudo-)aléatoires
numpy	Tableaux
matplotlib	Outils graphiques
pylab	*méta*-bibliothèque qui en contient beaucoup d'autres

Table 1: Bibliothèques usuelles