Lycée Saint-Louis
1Bio3

Septembre 2023
Informatique

TP 1 : Variables, fonctions, tests

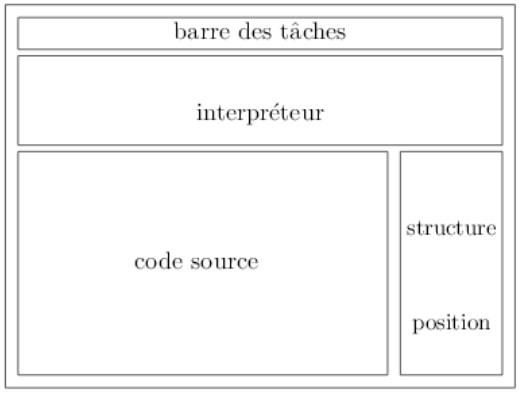
1 Démarrage de Python — Interface de Pyzo

— Allumer l'ordinateur. Le bureau Windows s'affiche à l'écran.

— Créer un nouveau dossier nommé *Informatique* dans lequel seront rangés les différents TP de cette année. Pour cela, aller dans le menu *Démarrer*, sélectionner *Poste de travail*, puis *Mes documents*, *Créer un nouveau dossier*.

— Dans le dossier *Informatique*, créer un nouveau dossier nommé *TP1*. Au début de chaque nouveau TP, on commencera par créer un nouveau dossier rangé dans *Informatique*, où seront sauvegardés les différents fichiers créés lors de la séance.

— Depuis le bureau Windows, lancer *Pyzo*. Une fois l'application ouverte, votre écran est divisé en plusieurs parties (pas forcément dans la même configuration que sur ce schéma) :



- Une partie « interpréteur », appelée **console** ou **shell**, qui lit du code écrit en Python et le traduit en langage machine pour que celle-ci puisse l'exécuter et vous donner un résultat.
- Une partie « code source », qui est un fichier comportant des programmes écrits en langage Python. Il s'agit en fait d'un fichier texte qui peut être utilisé par l'interpréteur.
- Éventuellement d'autres parties que nous ignorerons pour le moment.

N'hésitez pas à fermer les petites fenêtres qui ne sont pas l'interpréteur et le code source, et à redimensionner ces dernières pour une meilleure lisibilité.

2 Le mode direct de la console et les commandes élémentaires

Le mode direct d'utilisation a lieu dans la fenêtre *console* (*shell* en anglais). Elle permet d'utiliser Python comme une calculatrice ou d'exécuter quelques instructions.

2.1 Affectations et variables

Une variable est une sorte de « boîte » destinée à recevoir une valeur. Pour donner une valeur à une variable (ce que l'on appelle **affectation**, **assignation** ou encore **instanciation**), on écrit :

```
nomvariable = valeur
```

Attention : Le symbole d'affectation `=` n'est pas symétrique comme en mathématiques.

Le nom d'une variable commence par une lettre suivie ou non de caractères. Par exemple, `a` et `var1` peuvent être des noms de variables mais `5var` ne peut pas en être un. Python est sensible à la casse (c'est-à-dire qu'il distingue les majuscules et les minuscules).

Exemples :

- `x = 10` : la variable `x` reçoit la valeur entière 10.
- `mois = 'septembre'` : la variable `mois` reçoit la chaîne de caractères « septembre ».
- `7 = x` : c'est une erreur, car 7 n'est pas une variable mais une valeur. On ne peut pas y affecter `x`.
- `x = x+1` : affecte à la variable `x` son ancienne valeur plus 1. Si par exemple `x` valait 10 avant cette instruction, alors `x` vaut 11 après. Autre syntaxe : `x += 1`
- `x = 3.5` : affecte à x la valeur $3,5$. Remarquer que Python utilise un point pour la virgule des décimaux.

Lors qu'une variable a été déclarée, on peut ensuite récupérer la valeur qu'elle contient en tapant simplement le nom de la variable dans la console et en appuyant sur *Entrée*.

Exercice 1. Prévoir sur papier l'effet des instructions suivantes. On remplira pour cela un tableau avec les valeurs des différentes variables, étape par étape.
Vérifier ensuite avec l'ordinateur : taper chaque ligne dans la console en validant à chaque fois avec *Entrée* ; puis demander à Pyzo d'afficher les valeurs des variables en tapant simplement leurs noms, comme expliqué ci-dessus.

```
1 x = 1
2 variable = x
3 x = 2*x
4 variable = variable*variable-3*x
5 x = variable-x
6 y = 'coucou'
```

Affectation simultanée : Il est possible d'affecter simultanément plusieurs variables. Par exemple, la commande

```
x, abc = 3, 'truc'
```

affecte à la variable `x` la valeur 3 et à la variable `abc` la chaîne de caractères `'truc'`.

Exercice 2. Que contiennent les variables `x`, `y` et `z` après les deux lignes suivantes ?

```
1 x,y,z = 2,4,6
2 z,x,y = x,y,z
```

Vérifier en tapant la commande `x, y, z`.

2.2 Manipulation de la console

- À l'aide des flèches ↑ et ↓, on peut sélectionner d'anciennes lignes de commande de la console.
- En sélectionnant dans la barre de menu : *Shell*, puis *Redémarrer*, on redémarre la console. Cela a en particulier pour effet d'effacer le contenu des variables.

Tester ces commandes.

2.3 Notion de type

Chaque variable possède un **type** (qui correspond à la nature de la valeur qu'elle contient). Voici quelques types usuels :

Nature	*Type en Python*	*Exemple*
Entier	`int` (abréviation de *integer*)	2
Flottant (nombre à virgule)	`float`	3.5
Chaîne de caractères	`str` (abréviation de *string*)	`'bonjour'`
Booléen	`bool`	`True, False`
Liste	`list`	`[1, 4, 9, x]`

Pour connaître le type d'une variable `A`, on peut utiliser l'instruction `type(A)`.

Exercice 3. Exécuter **dans la console** les commandes suivantes et observer en particulier l'effet des commandes $+, *$ et/ selon le type des variables auxquelles elles s'appliquent. Expliciter également la différence entre $=$ et $==$.

```
u, v, w = 2,3,3
u == v
v == w
p = (u == v or v <= w)
type(p)
p == (v == w)
```

```
x, y, z, t = 3,'oui','non','et'
x*y+z+2*t
```

```
x, y = 25, 25.0
type(x), type(y)
y/3
x//3, x%3
```

```
x = 3.1
x**2-x*x == 0      #x**y calcule x puissance y
x**10-(x*x*x*x*x)**2 == 0
```

On remarquera que la dernière égalité est en réalité vraie. Cela montre la difficulté de tester informatiquement l'égalité entre deux flottants.

Conversion d'un type

— Les commandes `int`, `float` et `eval` permettent de convertir une chaîne de caractères :
  - en entier pour `int` (par exemple `int('123')` renvoie 123)
  - en flottant pour `float` (par exemple `float('123')` renvoie 123.0)
  - en entier ou en flottant pour `eval` suivant ce qu'il semble le plus adapté (par exemple `eval('123')` renvoie 123).

— Ces commandes permettent aussi de convertir un flottant en entier et réciproquement.

— La commande `str` transforme un entier ou un flottant en chaîne de caractères (par exemple `str(123)` renvoie `'123'`).

2.4 Opérations élémentaires

— **Pour les nombres**
  — Opérations usuelles : $+$, $-$, $*$, $/$.
  — `x**y` : élévation de `x` à la puissance `y`.
  — `x//y` : quotient dans la division euclidienne de `x` par `y` ($14 = \mathbf{3} \times 4 + \mathbf{2}$: 14//4 renvoie 3).
  — `x%y` reste dans la division euclidienne de `x` par `y` (14%4 renvoie 2).

— **En logique**
  — `x==y` renvoie `True` si `x` et `y` sont égaux et `False` sinon.
  — $<$, $>$: strictement supérieur, strictement inférieur.
  — $>=$, $<=$: supérieur ou égal, inférieur ou égal.
  — `!=` (ou $<>$) : différent.
  — `or, and, not` : ou, et, non.

— **Pour les chaînes de caractères**
  — $+$ réalise la concaténation de deux chaînes de caractères. Par exemple, `'mot1'+'mot2'` renvoie `'mot1mot2'`.
  — `3*'mot1'` renvoie `'mot1mot1mot1'`.

2.5 L'aide

— Lorsqu'on souhaite se documenter sur une commande connue, nommée `nomcommande`, on tape directement, derrière le *prompt*, l'instruction `help(nomcommande)`.

— On peut également utiliser l'aide interactive. On y accède en sélectionnant dans le menu *Outils* puis *Aide interactive*. Une nouvelle fenêtre s'ouvre alors à l'écran qui fournit des informations sur les commandes en cours d'écriture dans la console ou le fichier code source.

En utilisant l'aide, déterminer la fonctionnalité de la commande `abs`.

3 Code source

Python est fourni avec de nombreux programmes parfois très élaborés. Pour en écrire de nouveaux, les exécuter et les corriger, on écrit un code source. Il s'agit de la partie « code source » de l'écran. On sélectionne dans la barre de menu *Fichier* puis *Nouveau* pour créer un nouveau fichier ou *Ouvrir* pour ouvrir un fichier déjà existant. Il est possible d'ouvrir simultanément plusieurs fichiers de code source.

— Pour enregistrer votre fichier, cliquez dans la barre de menu sur *Fichier*, puis sélectionner *Enregistrer sous*. Nommez votre fichier et enregistrez-le dans le dossier TP1.

— Écrire dans le fichier (code source) les lignes suivantes :

```
1 """Premier programme Python"""
2 print('Bonjour et bienvenue')
3 b = input('Entrer une chaine de caracteres')
4 print('Vous venez de taper :', b)
5 d = input('Entrer un entier')
6 print('Ce que vous venez de taper a pour type:',type(d))
7 d = eval(d)
8 print('Son carre vaut:',d**2)
```

La première ligne du fichier, appelée *docstring* dans le vocabulaire Python, doit décrire succinctement le contenu du fichier et le rôle du programme. Il s'agit d'une habitude de programmation. Cette ligne ne sera pas interprétée par Python lors de l'exécution du programme.

— Pour exécuter ce programme, cliquez droit sur l'onglet du fichier ouvert dans l'éditeur et sélectionnez *Exécuter cet onglet*.

— A quoi sert l'instruction `d = eval(d)` ? Que se passe-t-il si on la supprime ?

4 Entrées, sorties

4.1 Entrée : la fonction input

Une entrée consiste à interrompre un programme pour saisir une valeur au clavier. En Python, on utilise la fonction `input`. La valeur saisie au clavier est alors considérée comme une chaîne de caractères.
Attention : La valeur saisie au clavier est **toujours** considérée comme une chaîne de caractères, même si on ne tape pas les guillemets (ou les apostrophes) qui la délimitent.

Exemple :

— Un programme exécute l'instruction `a = input()`. Il s'arrête et attend une saisie dans l'interpréteur.

— L'utilisateur tape 14 au clavier et appuie sur la touche *Entrée*.

— Le programme reprend et affecte à la variable `a` la valeur `'14'`, qui est une chaîne de caractères.
Si l'on veut convertir `a` en l'entier 14, on utilise l'instruction `a = int(a)`.
Si l'on veut convertir `a` en le flottant 14.0, on utilise l'instruction `a = float(a)`.

Pour une saisie plus lisible, on peut faire afficher un message comme « Valeur de a ? » à l'aide de l'instruction `a = input('Valeur de a ?')`.

4.2 Sortie : la fonction `print`

Une sortie consiste en l'affichage d'un résultat pendant l'exécution d'un programme ou en fin de programme. Les résultats sont affichés dans l'interpréteur. En Python, on utilise la fonction `print`.

Exemples :

- `print(x)` affiche la valeur de la variable `x`.
- `print('x est égal à', x)` affiche le message « x est égal à », suivi de la valeur de la variable `x`.
- `print('Bonjour', 252, 2==2)` affiche à l'écran la chaîne de caractères « Bonjour », l'entier 252 puis le booléen `True`.

Exercice 4. Écrire (dans la partie « code source ») un programme qui demande à l'utilisateur d'entrer deux réels et affiche le double de l'inverse de la somme de leurs cubes.

5 Instructions conditionnelles

La forme générale d'une instruction conditionnelle est la suivante :

Si *condition* **alors**	**if** *condition*:
instructions1	instructions1
sinon	**else**:
instructions2	instructions2
fin si	Suite du programme
Suite du programme	

Si la variable booléenne *condition* a pour valeur logique *Vrai*, le jeu d'instructions 1 est exécuté. Sinon c'est le jeu d'instructions 2 qui est exécuté. Les lignes « sinon instructions2 » sont optionnelles.

- La fin de l'instruction du branchement conditionnel se traduit en Python par le retour à l'indentation[1] initiale.
- **Attention** à ne pas oublier les « deux points ».
- Lorsqu'on souhaite utiliser plusieurs instructions conditionnelles imbriquées, on peut abréger :

  else:
  if *condition2*:

  en :

  elif *condition2*:

Exercice 5. Écrire un programme qui demande à l'utilisateur d'entrer les valeurs de deux nombres x et y puis affiche la valeur du plus grand des deux.

1. L'**indentation** (c'est-à-dire le décalage typographique entre le début de deux lignes successives) est habituellement fixée à 4 espaces.

Exercice 6. Écrire un programme qui demande à l'utilisateur d'entrer les valeurs de trois réels a, b et c, calcule les racines réelles du polynôme (de degré inférieur ou égal à 2) $x \mapsto ax^2 + bx + c$ et affiche l'un des messages suivants :

— le polynôme admet deux racines réelles, qui sont : ... ;

— le polynôme admet exactement une racine réelle, qui est : ... ;

— le polynôme n'admet pas de racines réelles ;

— tout réel est racine.

Pour calculer la racine carrée d'un nombre positif x, on pourra utiliser la commande `sqrt(x)`. Celle-ci fait partie du module `math`. Pour l'importer et pouvoir l'utiliser, il faut placer la ligne `from math import sqrt` en tête de programme.

Exercice 7. Écrire un programme qui demande à l'utilisateur d'entrer trois réels a, b et c et affiche la valeur du maximum de ces trois réels.

6 Fonctions

Redémarrer la console pour effacer les valeurs des variables affectées (cf. section 2.2)

En informatique, une **fonction** est une portion de code représentant un sous-programme, qui effectue une tâche ou un calcul relativement indépendant du reste du programme et qui renvoie une valeur. En Python, la syntaxe est la suivante.

def nomfonction(arguments):
    instructions
    return résultat

— La première ligne permet de définir le nom de la fonction et de préciser le ou les **arguments** (aussi appelés **paramètres**) à entrer par l'utilisateur pour le calcul du résultat.
Attention : La première ligne se termine par deux points.

— Lors des instructions internes à la fonction, une variable `résultat` est définie.

— L'instruction `return résultat` renvoie sa valeur, qui pourra être affichée ou bien utilisée dans la suite du programme.

— Les arguments et les variables définies dans le jeu d'instructions intérieur à une fonction sont à **portée locale**, c'est-à-dire qu'ils ne peuvent être utilisées qu'à l'intérieur de la fonction. Leur contenu est vidé une fois la fonction exécutée (leurs valeurs ne sont pas sauvegardées).

Habitude de programmation :
Un programme est souvent constitué de nombreuses fonctions. Pour se repérer dans un programme, il faut prendre l'habitude d'écrire, après la ligne de déclaration (celle débutant par `def`) un commentaire (entre deux symboles """) précisant le type des arguments d'entrée et celui de la variable en sortie et résumant en français le calcul ou l'action réalisé par la fonction. On dit que l'on documente la fonction. Dans la console Python, on peut alors faire afficher la documentation en utilisant la commande `help(nomfonction)`.

Exemple :

— Pour déterminer le maximum entre deux réels, on peut utiliser le script suivant.

```
1 x,y = eval(input('Entrer deux nombres :'))
2 if x>=y:
3     print('Le maximum de ces deux nombres est',x)
4 else:
5     print('Le maximum de ces deux nombres est',y)
```

À l'exécution de ce programme, la chaîne de caractères « Entrer deux nombres : » s'affiche à l'écran. L'utilisateur donne deux réels séparés par une virgule, puis s'affiche la chaîne « Le maximum de ces deux nombres est » suivi du maximum.

— On peut également obtenir ce maximum par la fonction suivante.

```
1 def maxi(x,y):
2     """si x et y sont deux nombres flottants ou entiers, maxi(x,y) renvoie
3                                                     le maximum de x et y."""
4     if x >= y:
5         return x
6     else:
7         return y
```

Recopier cette fonction dans l'éditeur (code source), puis l'exécuter : il ne se passe rien de visible, car l'interpréteur a simplement « pris acte » de l'existence de la fonction, sans l'exécuter. Il faut garder à l'esprit qu'une fonction n'est qu'un **mode d'emploi** pour effectuer un certain calcul. Lorsque vous exécutez ce code, vous ne faites que dire à l'ordinateur « voici le mode d'emploi à suivre pour calculer un maximum », mais vous n'êtes pas en train de lui demander d'en calculer un. Ceci pourra être fait plus tard :
Taper `maxi(45,87)` dans la console. À ce stade l'ordinateur se réfère au « mode d'emploi » que vous lui avez fourni en remplaçant `x` (le premier argument) par 45 et `y` par 87 ; il effectue toutes les instructions et vous renvoie la valeur demandée.
Taper `x` ou `y` dans la console pour observer que les variables `x` et `y` sont inconnues de Python : les variables `x` et `y` sont locales et ne sont déclarées que dans la fonction.
Taper `10 + maxi(45,87)` : le résultat renvoyé par la fonction `maxi` peut être utilisé dans d'autres calculs.

Remarque : Différence entre `return` et `print` :

— Si l'on souhaite que la fonction renvoie une valeur qui pourra être utilisée par un autre morceau de code du programme, on utilise l'instruction `return`.

— Si l'on souhaite **afficher** une valeur à l'écran, on utilise l'instruction `print`.

	Interrompt la fonction	Renvoie une valeur (pouvant être utilisée ailleurs)	Affiche toujours un résultat
`return`	Oui	Oui	Non
`print`	Non	Non	Oui

Exercice 8. On considère les six scripts suivants. Répondre sans l'ordinateur, puis vérifier.

Script 1
```
a = 5
print(a*a + 1)
```

Script 2
```
a = 5
return a*a + 1
```

Script 3
```
def f():
    return "2"
```

Script 4
```
def g(x):
    print(x*x + 1)
```

Script 5
```
def h(x):
    return x*x + 1
```

Script 6
```
def i(x, y):
    return x + 2*y
```

1. Lesquels définissent des fonctions ?
2. Lequel a une syntaxe incorrecte ?
3. On exécute le script 1. Quel résultat obtient-on ?
4. On exécute l'appel `f()`. Quel résultat obtient-on ?
5. On exécute l'instruction `f() + "1"`. Quel résultat obtient-on ?
6. On exécute l'appel `g(5)`. Qu'obtient-on ?

7. On exécute l'appel `g(5) + g(3)`. Qu'obtient-on ?
8. On exécute l'appel `h(5)`. Qu'obtient-on ?
9. On exécute l'appel `h(5) + h(3)`. Qu'obtient-on ?
10. On exécute l'appel `i(4,-2.5)`. Qu'obtient-on ?

Exercice 9. Écrire une fonction nommée f qui prend en entrée un réel x et renvoie le nombre $\frac{x^2+x+1}{4}$. Tester votre fonction pour quelques valeurs de x.

Exercice 10. On considère la fonction f définie sur $\mathbb{R}$ par $f(x) = \begin{cases} \frac{1}{2}x + 1 \text{ si } x \geqslant 4 \\ 0 \text{ si } x < 4 \end{cases}$ pour tout $x \in \mathbb{R}$.
Parmi les fonctions suivantes, laquelle ou lesquelles renvoient la bonne valeur de $f(x)$?

```
1  def f1(x):
2      if x >= 4:
3          return 0.5*x + 1
4      return 0
5
6  def f2(x):
7      if x >= 4:
8          return 0.5*x + 1
9      else:
10         return 0
11
12 def f3(x):
13     if x >= 4:
14         y = 0.5*x + 1
15     else:
16         y = 0
17     return y
```