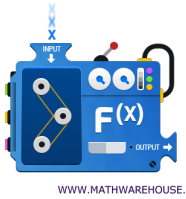
TP n° 4 : Fonctions



1 Environnement de développement

Jusqu'ici nous avons travaillé directement dans la console IPYTHON, qui est un moyen simple de dialoguer de manière interactive. Cela permet de comprendre rapidement le fonctionnement de petits programmes, d'obtenir la valeur de l'évaluation d'expressions ou de vérifier le comportement de différentes instructions.

Pour des projets plus conséquents, on veut pouvoir lire, éditer, sauvegarder et exécuter des programmes plus complexes. On utilise alors un *environnement de développement intégré* (IDE) qui permet :

- de lire, d'ouvrir et d'écrire des programmes dans un éditeur adapté ;
- d'exécuter ces programmes ;
- de trouver, comprendre et corriger les bogues [1] (*déboguer*) ;
- de consulter la documentation.

Au lycée, nous utiliserons l'IDE PYZO [2], que vous pouvez également installer chez vous.

Pour ouvrir PYZO, il suffit de cliquer sur l'icône algorithmique en bas à gauche puis de choisir l'application PYZO.

1. Défaut de conception ou de réalisation se manifestant par des anomalies de fonctionnement (anglais : *bug*). Journal officiel de la République française du 19 février 1984, p. 51741.
2. http://www.pyzo.org

Commencez par identifier les différentes fenêtres qui vous sont accessibles :

- La fenêtre principale, dans laquelle vous pouvez écrire et enregistrer votre code. Elle possède des lignes numérotées qui vous permettent de vous repérer facilement dans le fichier, notamment quand PYTHON vous signalera l'existence d'un bogue dans votre programme. Pour l'instant, le fichier s'appelle `<tmp 1>`.
- Une console IPYTHON (`Shells`), comme celle que nous avons déjà utilisée. On peut y entrer des commandes pour tester leur comportement sans que cela soit sauvegardé dans un fichier (elle doit donc être cantonnée au rôle de test rapide). C'est aussi à l'intérieur de celle-ci que sera exécuté le code de votre fichier principal.
- La fenêtre d'aide (`Interactive help`) qui offre un accès direct à l'aide de PYTHON. N'hésitez pas à l'utiliser !
- Enfin, la fenêtre de structure du code qui vous permet d'accéder directement à certaines parties de votre code une fois que votre fichier aura acquis une certaine structure (en particulier via l'écriture de fonctions).

Dans la console IPYTHON, on effectue successivement :

PYTHON
```
In [1]: foo = 42

In [2]: foo
Out[2]: 42
```

L'interprétation est *interactive*, lorsque l'on évalue l'expression `foo`, sa valeur est renvoyée et affichée dans la console.

✔ Dans l'interpréteur interactif, il est possible de rappeler une ligne entrée précédemment à l'aide des flèches ↑ et ↓, et de modifier cette ligne avant de relancer son calcul avec la touche entrée.

Dans la fenêtre principale, écrire les lignes suivantes :

PYTHON
```
bar = "42"
bar
```

Sauvegarder le fichier (par exemple dans `~/info/tp04/tp04.py`). Pour exécuter le script, on peut utiliser le menu `Exécuter` puis `Démarrer le script` ou plutôt directement le raccourci[3] clavier F5.

1. Que voyez-vous dans la console `IPython` ?
2. Est-ce que la variable `bar` y est définie ? .....

À la deuxième ligne du script, l'évaluation de l'expression `bar` est bien exécutée mais sa valeur, la chaîne de caractères `"42"`, n'est pas directement accessible. Modifier le script en :

PYTHON
```
bar = "42"
print(bar)
```

3. Exécuter. Que voyez-vous ? Expliquer.
4. Dans la console IPYTHON, quelle est la différence entre l'expression `bar`, l'expression `print` et l'expression `print(bar)` ? Quels sont leur types (utiliser la fonction `type`) ? Expliquer. .....
   .....

La fonction `print` affiche ses arguments à l'écran, séparés par une espace, puis va à la ligne. Elle ne renvoie rien (donc en fait, elle renvoie `None`).

5. Parcourir rapidement l'aide de la fonction `print` (soit dans la console avec `print?` ou `help(print)`, soit dans la fenêtre d'aide dédiée à cet effet). Comprendre le rôle des paramètres optionnels `sep` et `end`, puis deviner le résultat des expressions suivantes :
   - `print()`
   - `print("\n ")`
   - `print(0, 1, 2)`
   - `print(0, 1, 2, end="")`
   - `print(0, 1, 2, sep=' + ', end=" = ?")`

En PYTHON, le caractère « # » permet de commencer un commentaire. Tout ce qui suit ce caractère est ignoré par l'interpréteur. On commente donc... pour soi ! Et pour les autres ! Sous `Pyzo`, une ligne qui commence par `##` (deux caractères « # ») permet de délimiter des *cellules* et d'organiser son code. Ce peut être une bonne idée par exemple de délimiter les sections du TP ou les questions (par exemple `## Question 1`).

6. Écrire un commentaire simple et un délimiteur de cellule.

⚠ Il est primordial de sauvegarder régulièrement son travail (Ctrl + S). Nous ne répondrons à aucune question d'un élève ayant un fichier de travail non sauvegardé (`<tmp *>` ou icône 💾).

L'exécution d'un script le sauvegarde automatiquement, mais il est également possible de n'exécuter qu'une sélection (F9) ou une cellule (F8), ce qui ne sauvegarde alors pas votre travail.

2 Tuples

Dans cette section, on pourra travailler directement dans la console interactive ou dans le fichier principal en l'exécutant lorsque nécessaire.

7. Définir un tuple `eleve` comportant comme composantes votre prénom, votre nom et votre âge.
8. Définir les variables `prenom`, `nom`, `age` en déstructurant ce tuple.
9. Reprendre l'exercice n° 3 du TP n° 3 et le résoudre avec une seule instruction PYTHON, en utilisant une affectation simultanée des trois variables.

3. Sur votre ordinateur personnel, c'est peut-être Ctrl+F5, mais c'est facile à changer.

3 Fonctions

3.1 Syntaxe

La syntaxe pour définir une fonction est :

PYTHON
```
def nom_fonction(arg1, arg2, ...):
    # instructions
    # à
    # réaliser
    return valeur
```

Par exemple :

PYTHON
```
def multiplier(a, b):
    return a * b
```

On peut ensuite faire un *appel* à la fonction :

PYTHON
```
quinze = multiplier(3, 5)
print(quinze)
```

10. Définir la fonction `successeur` qui à un entier n associe l'entier $n+1$. Exécuter le script et vérifier son bon fonctionnement.
11. Définir une fonction `produit_des_differences` qui à quatre entiers a, b, c, d associe l'entier $(a-b)*(c-d)$. Exécuter le script et vérifier son bon fonctionnement.

3.2 Arguments optionnels

Il est possible d'avoir des arguments facultatifs (optionnels) à condition de leur donner une valeur par défaut :

PYTHON
```
def racine(x, n=2):
    return x ** (1 / n)
```

12. Que vaut `racine(2)` ? Et `racine(3, n=3) * racine(3, 3)**2` ?

Les paramètres optionnels doivent suivre les paramètres obligatoires et il vaut mieux toujours les nommer.

☞ **Remarque 1**

Cas particulier : pas d'espaces autour du « = ».

3.3 Chaînes de documentation (*docstring*)

En PYTHON, les docstrings décrivent les modules, les classes et les fonctions. Ce sont des chaînes de caractères qui sont placées en tout premier dans une fonction.

PYTHON
```
def reponse():
    """Réponse à la grande question sur la vie,
    l'univers et le reste."""
    return 42
```

L'avantage par rapport à un commentaire classique, c'est que les docstrings sont récupérables dynamiquement via la fonction primitive `help`.

13. Le vérifier en entrant : `help(reponse)`
14. Quel est le type de `reponse` ? .....
15. Ajouter une docstring à la fonction `multiplier`.

✔ On prendra l'habitude de documenter ses propres fonctions de cette manière. Il faut toujours ajouter une docstring à vos fonctions !

4 Bibliothèques

En PYTHON, on peut utiliser plusieurs milliers de fonctions déjà implémentées qui répondent à différents besoins (calcul scientifique, création de jeux, traitement du signal ou d'images, conception d'interfaces graphiques, envoi de courriels, etc.). Pour pouvoir s'y retrouver, elles sont organisées en *bibliothèques* ou *modules* thématiques. Par exemple, nous avons déjà rencontré la bibliothèque `math` qui contient des fonctions et des constantes utilisées en analyse. Il existe un très grand nombre [4] de bibliothèques, dont certaines vous seront certainement très utiles pour vos TIPE.

Les fonctions de la *bibliothèque standard* (`print`, `divmod`, `abs`, etc.) sont accessibles directement. Pour utiliser les fonctions d'un module, il faut préalablement importer celui-ci. Par exemple :

PYTHON
```
import math
```

Comme nous l'avons déjà vu, on peut ensuite utiliser les fonctions d'un module en les préfixant du nom du module :

PYTHON
```
racine_de_deux = math.sqrt(2)
```

Cette notation permet d'éviter tout risque d'homonymie :

PYTHON
```
pa, pe, pi, po, pu = ("a", "e", "i", "o", "u")
2 * math.pi
```

Cependant, lorsque l'on utilise un très grand nombre de fois une fonction ou une constante d'une bibliothèque, écrire le nom de la bibliothèque à chaque fois peut être un peu pénible. On peut importer directement les fonctions dont on a besoin avec la syntaxe suivante :

4. On pourra regarder `https://pypi.org`.

PYTHON
```
from math import cos, sin, tan, acos, asin, atan, pi
pi == 4 * atan(1)  # Vrai mais ça pourrait être faux !
```

Pire, on peut importer directement toutes les fonctions de la bibliothèque avec la syntaxe `from math import *`, mais cela est vraiment dangereux ! À éviter absolument, en mode non interactif.

✔ Privilégiez la forme `module.nom_fonction` sauf éventuellement en mode interactif lorsque vous utilisez PYTHON comme calculatrice.

5 Exercices

EXERCICE 1

Définir en PYTHON les fonctions suivantes :

1. $f : x \mapsto \dfrac{x}{1+x^2}$
2. $g : x \mapsto \ln x + x^{x+1}$
3. $h : x \mapsto \sin(\pi \lfloor x \rfloor)$
4. $u : (x,y) \mapsto x^{-y}$
5. $v : (x,y,z) \mapsto (x+y) \times z$
6. $w : (x,y,z,t) \mapsto (y-x, t-z)$
7. $k : ((x,y),(z,t)) \mapsto (y-x, t-z)$

EXERCICE 2

Écrire une fonction `norme(vecteur)` qui prend en entrée un vecteur (x,y) représenté par un couple (un tuple à deux composantes) de flottants et qui renvoie sa norme euclidienne $\sqrt{x^2+y^2}$.