I_{01} Programmation modulaire

PCSI 2020 – 2021

I Exécution d'un programme python

De façon générale, sauf structures particulières, un programme s'exécute de haut en bas ligne par ligne.
Lors d'une affectation (`a = 3 * 5` par exemple), python calcule d'abord ce qui se trouve à droite puis enregistre le résultat dans la variable à gauche du signe égal.
Pour observer l'exécution d'un programme, vous pouvez utiliser l'outil en ligne python tutor
http://www.pythontutor.com/
Démonstration 1
Toutefois, on a souvent à refaire plusieurs fois une opération similaire ou identique (afficher un message d'erreur, calculer le minimum d'un ensemble de nombres, trier un ensemble d'objet ...). Plutôt que de ré-écrire plusieurs fois la même chose, on préfèrera écrire une fonction.
Intérêt des fonctions et des modules :

- Ne pas ré-écrire/copier-coller du code (avec les risques d'erreurs que ça comporte).
- Faciliter la compréhension de l'algorithme général et la maintenance du code en séparant les différents morceaux de code dans différentes fonctions. Exemple : pour calculer le pgcd de deux nombres $n1$ et $n2$:

```
1 liste_facteur_premier1 = decompose(n1)
2 liste_facteur_premier2 = decompose(n2)
3 liste_facteur_commun = compare(liste_facteur_premier1,
     liste_facteur_premier2)
4 pgcd = produit_liste(liste_facteur_commun)
```

  Ainsi, il suffit de coder une seule fois comment décomposer un nombre en facteur premier et on peut l'appliquer à $n1$ et à $n2$. De plus, si on trouve un moyen plus performant pour faire cette décomposition, il suffit de changer le code à un seul endroit.
- Permettre le travail d'équipe . Une personne s'occupe d'une fonction $f1$, une autre s'occupe d'une autre fonction $f2$ et n'a pas à se préoccuper de comment est programmé en détail $f1$ (il a juste besoin de savoir « ce que fait $f1$ » d'où l'importance de commenter ses fonctions au début).

II Fonction

1. Définition

Définition : 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 peut renvoyer une ou plusieures valeurs.

2. Pseudo-code

En pseudo-code et dans les langages typés, on précise généralement le type que renvoie la fonction et le type des arguments. Par exemple `float divise(int a, int b)` signifierait que la fonction divise prend comme argument deux entiers et renvoie un flottant.

```
Function nom_de_la_fonction(argument1,argument2)
    corps de la fonction;
    Renvoie resultat1,resultat2;
end
```

3. Python

Le mot clé pour définir une fonction est `def`. Le mot clé pour indiquer que la fonction est finie et que l'on renvoie le résultat est `return`.
L'instruction return arrête le déroulement de la fonction : le code situé après le return ne s'effectuera pas.
⚠ La syntaxe générale est la suivante :

```
def nom_de_la_fonction(arguments):
    '''commentaire indiquant le but de la fonction'''
    ligne_de_calcul1
    ligne_de_calcul2
    ...
    return resultat_de_la_fonction
#l'intérieur de la fonction se reconnait car il est décalé de 4 espaces vers la droite
#on indique donc que l'on n'est plus dans la fonction en ne décalant plus de 4 espaces
```

Attention, si vous voulez que la fonction vous renvoie une valeur, à ne pas oublier `return`.

3.a. Exemple avec un argument

```
def polynome(x):
    ''' calcule et renvoie (x**2-x+1)**2 '''
    resultat = x**2 - x + 1
    return resultat**2
a = polynome(4)
b = polynome(3)
c = polynome(a-b)
```

Lien python tutor 1
Lien python tutor avec ligne inutile

3.b. Avec plusieurs arguments

Pour indiquer que la fonction prend plusieurs arguments, il suffit de les séparer par une virgule.

```
def poly2(x, y):
    z = x**2 - y**3 + 2
    return z
p = poly2(3,-1)
```

Python tutor

3.c. Avec Arguments optionnels

Certains arguments peuvent avoir une valeur par défaut (la valeur la plus fréquemment utilisée). L'utilisateur peut ensuite choisir de rentrer la valeur normalement, ou pas (auquel cas le paramètre prend sa valeur par défaut). Les arguments avec valeur par défaut sont forcément les derniers.

```
def poly3(x, y = 0):
    z = x**2 - y**3 + 2
    return z
p = poly3(3,-1)
q = poly3(3) #on ne rentre pas y, donc il prend la valeur par défaut
```

Python tutor

3.d. Fonction qui renvoie plusieurs valeurs

```
def division_entiere(nombre,diviseur):
    quotient=nombre//diviseur
    reste=nombre-quotient*diviseur
    return quotient,reste
q,r = division_entiere(10,3)
print(q,r)
r = division_entiere(10,3)
print(r)
```

Python tutor
Le résultat est en fait renvoyé sous forme d'un tuple.
Nous verrons lors du prochain chapitre comment utiliser un tel objet.

3.e. Fonction sans argument

Dans quelques cas, on n'a pas besoin d'arguments. Il suffit de ne pas mettre de nom de variable entre les parenthèses.

```
def ma_fonction_preferee():
    print("j'aime la physique\n"*3)
ma_fonction_prefere()
```

4. Documenter une fonction

Quand on conçoit une fonction, il est préférable de lui donner un nom explicite , car elle est susceptible d'apparaitre à de nombreux endroits dans le programme. (On n'imagine pas que les fonctions de la bibliothèque Python s'appellent f1, f2, etc.) En revanche, les noms des arguments formels peuvent être courts, car leur portée, et donc leur signification, est limitée au corps de la fonction.
Il convient par ailleurs de documenter convenablement les fonctions, en spécifiant :

- les arguments de la fonction
- les hypothèses sur ces arguments (liste non vide, fonction croissante…)
- ce que fait la fonction, qu'est-ce qu'elle renvoie comme résultat

Python propose un mécanisme pour associer une documentation à toute fonction, sous la forme d'une chaine de caractères placée au début du corps de la fonction. Ainsi on peut écrire

```
def puissance(x, n):
    """
    Fonction puissance
    arguments :
        x, nombre (flottant ou entier ou complexe) que l'on veut élever à
            la puissance n
        n, entier : puissance à laquelle on veut élever le nombre. Le
            programme fait l'hypothèse n>=0
```

```
    renvoie :
        un nombre de même type que x et qui vaut x à la puissance n
    """
    r = 1
    for i in range(n):
        r = r * x
    return r
```

La documentation d'une fonction f peut être affichée avec help(f).

5. les fonctions lambda

```
>>> f = lambda x: x*x
>>> f(5)
25
```

Ceci est une autre façon d'écrire les fonctions. C'est utilisé pour des fonctions courtes , principalement lorsque l'on veut qu'une fonction renvoie une autre fonction ou pour passer une fonction en argument d'une autre.

Exemple : pour créer la fonction produit(a,f) qui à la fonction f associe la fonction $x \mapsto f(a*x)$

```
def produit(a,f):
    return(lambda x: f(x*a))
carre = lambda x:x**2
f1 = produit(0.5, carre)
print(f1(2)) # calcule carre(2*0.5) => 1
```

Exercice n°1 écrire un programme qui prend comme argument deux fonctions f et g et renvoie la fonction $f \times g$, c'est-à-dire la fonction $x \mapsto f(x) \times g(x)$.

```
def multiplie_f(f, g): return lambda x : f(x) * g(x)
```

III Portées des variables

Lorsque le programme devient un petit peu gros, il y a beaucoup de fonctions. Dans chaque fonction, on utilise des noms de variables (i,j, mavariable). Il ne faut pas qu'il y ait de conflit entre ces différentes variables, ainsi les variables n'existent en général qu'à l'intérieur de la fonction et sont « effacées » à la sortie.

1. Principe et exemples

En programmation (et en python), on distingue deux sortes de variables : les variables globales et les variables locales.

Par exemple, dans le programme

```
>>> def f():
...     y = 8
...     return 3*y
>>> f()
24
>>> y
NameError: name 'y' is not defined
```

la variable y est locale, c'est-à-dire définie uniquement à l'intérieur de la fonction.
On dit que la portée de la variable y est limitée au corps de la fonction f.

⚠ Si on veut « forcer » une variable à être définie dans tout le programme, il faut la définir dans la fonction comme variable globale.
C'est en général une mauvaise pratique.

```
def reinitialise_x():
    x = 0
```

ne fait qu'affecter 0 à une variable locale x, cette variable n'a de sens qu'à l'intérieur de la fonction réinitialise. Au contraire

```
def reinitialise_x_qui_marche():
    global x
    x = 0
```

Considère qu'une variable x existe à l'extérieur de la fonction et va modifier sa valeur.

```
x = 5
reinitialise_x()
print(x) #affiche 5, le x de la fonction est local
reinitialise_x_qui_marche()
print(x) #affiche 0 : la fonction vient modifier la variable globale
```

Python tutor

2. Spécificité de python : définition implicite

Lorsque l'on utilise une variable dans une fonction sans lui avoir préalablement affecté une valeur, python la considère implicitement comme globale.
Exemple :

```
def f():
    a = b + 1 #b utilisé sans être défini dans la fonction : variable
       implicitement déclarée comme globale
      #a est une variable locale car définie dans la fonction
    return a
b = 3
print(f() ) #affiche 4
```

⚠ **Évitez autant que possible les variables globales, et faites leur porter un nom explicite assez long afin de bien les différencier.**

IV Les modules ou bibliothèques

Python ne propose pas beaucoup de fonctionnalité « de base » : par exemple il n'est pas possible de calculer le cosinus d'un nombre à moins de le programmer soi-même.
Toutefois, il est possible d'étendre les possibilités de base en important des modules, c'est-à-dire « en apprenant à python de nouvelles fonctions ». On parle aussi de bibliothèques.
Il existe plusieurs manières d'importer un module. La première méthode est la suivante :

```
from math import cos #importe uniquement la fonction cosinus contenue
   dans le module math
```

```
from math import cos,pi #importe la fonction cosinus et le nombre pi
cos(pi) #renverra le résultat -1.0
```

On préfère généralement les manières suivantes

```
import math#importe toute la bibliothèque math
math.cos(math.pi)# mais pour accéder aux objets il faut les préfixer
   par math.
import math as m #importe la bibliothèque math en lui donnant un nom
m.cos(m.pi)
```

L'avantage de cette méthode est de savoir exactement d'où viennent les différentes fonctions. C'est cette méthode qui doit être privilégiée systématiquement.
Les modules les plus utiles pour nous seront :

- Le module math (contient les fonctions usuelles)
- Le module random (permet de générer des nombres pseudos-aléatoires)
- Les modules numpy et scipy
  (permettent de faire des calculs scientifiques, calculs matriciels, résolutions d'équations différentielles etc...). Le module numpy redéfinit les fonctions du module math et permet de les appliquer directement sur une liste.
- Le module matplotlib et plus précisément le sous-module matplotlib.pyplot (permet de tracer des jolis graphiques)

L'utilisation de modules permet d'éviter de ré-écrire du code complexe et d'utiliser des fonctions développées par des professionnels qui ont été testées par des milliers d'utilisateurs (et que l'on peut donc espérer fiables…).

Table des matières