Chap. II Les fonctions

BRUNO DARID

14 août 2019

PLAN

1 Repères historiques **1**

2 Notion de fonction : première approche **2**

3 Passer des arguments et récupérer une valeur **2**

4 Concevoir une fonction **3**

5 Fonctions de la bibliothèque standard **4**
5.1 À retenir 5

6 CORRECTION **7**
6.1 E1C2 : maximum 7
6.2 E2C2 : volume d'un ballon 7
6.3 E3C2 : longueur d'un tweet 7
6.4 P1C2 : distance euclidienne 8

1 Repères historiques

Fig. 1 – John McCarthy

John McCarthy *(1927-2011)* auteur du langage Lisp en 1958, dont la principale construction est la définition de **fonctions**. Il joua un rôle majeur dans la programmation en intelligence artificielle, écrivant un des premiers programmes jouant aux échecs.

2 Notion de fonction : première approche

Supposons que l'on doive afficher les valeurs des puissances de 2 les plus utilisées en architecture machine (8, 16, 32 et 64). On peut utiliser le programme du chapitre précédent :

```
N = 8
p = 1
for c in range(N):
    p = p * 2
print(p)
N = 16
p = 1
for c in range(N):
    p = p * 2
print(p)
N = 32
p = 1
for c in range(N):
    p = p * 2
print(p)
N = 64
p = 1
for c in range(N):
    p = p * 2
print(p)
```

Procéder de cette façon met en évidence plusieurs problèmes. On peut citer :

— les répétitions ;

— la difficulté de maintenance du code *(que devrait-on faire si on souhaite ensuite les puissances de 16 ?)*

La solution consiste à isoler la partie de code qui se répète, lui donner un nom et l'appeler lorsque c'est nécessaire.

3 Passer des arguments et récupérer une valeur

Le langage python (*comme les autres langages*) possède une construction permettant de résoudre leproblème précédent : la **définition de fonction**.

```
def nom_fonction(paramètre(s)):
    bloc_instructions
```

Le nom de la fonction doit **commencer par une lettre**, ne doit **pas être un mot reservé** et doit être autant que possible explicite. Le bloc d'instructions, qui constitue le corps de la fonction, **DOIT** être indenté. La fonction peut avoir 0, 1 ou plus de paramètres.

Créons une première version d'une fonction destinée à afficher les puissances de 2.

```
def puissance(n):
    p = 1
    for c in range(n):
        p = p * 2
    print(p)
```

Lorsqu'on exécute ce code, il ne se passe **rien**. À ce stade on a définit la fonction, il faut maintenant l'appeler pour que tout son code soit exécuté. L'appel d'une fonction consiste à écrire **son nom suivi de parenthèses ouvrantes-fermantes** à l'intérieur desquelles on place d'éventuels **arguments**.

```
puissance(8)
puissance(16)
puissance(32)
puissance(64)
```

La définition `puissance(n)` proposée ne correspond pas exactement à une fonction (on devrait plutôt parler ici de *procédure*). En effet, une fonction au sens strict devrait fournir (on dit **retourner**) une valeur. Python dispose de l'instruction `return` qui permet de retourner une valeur. Une version améliorée de la définition de `puissance(n)` ainsi que son appel pourrait être :

```
def puissance(n):
    p = 1
    for c in range(n):
        p = p * 2
    return p

print(puissance(8))
print(puissance(16))
print(puissance(32))
print(puissance(64))
```

4 Concevoir une fonction

En plus d'un nom explicite une fonction devrait être convenablement documentée. Cette documentation devra comporter des **spécifications**, c'est-à-dire les hypothèses faites sur les arguments, leur relation avec le résultat retourné. On reviendra plus en détails sur cette notion de spécifications dans le cours d'algorithmique.

En python, la documentation suit immédiatement la définition de la fonction et est encadrée de trois double quotes """. Cette partie constitue la **docstring**.

Exemple

```
def puissance(x,n):
    """
    Calcule x à la puissance n; on suppose x > 0 et n >= 0
    """
    p = 1
    for c in range(n):
        p = p * x
    return p

print(puissance(16,2))
```

```
256
```

Cette documentation est en outre accessible via la fonction `help()`

```
help(puissance)
```

```
Help on function puissance in module __main__:
puissance(x, n)
    Calcule x à la puissance n; on suppose x > 0 et n >= 0
```

5 Fonctions de la bibliothèque standard

Tous les grands langages de programmation proposent des fonctions toutes faites. Ces fonctions écrites dans le langage et fournies avec lui sont regroupées dans la **bibliothèque standard**. Dans la bibliothèque standard de python on trouve par exemple, la bibliothèque *math* ou *random*. Pour pouvoir utiliser les fonctions d'une bibliothèque, il faut d'abord l'importer avec l'instruction `import`.

```
import math
```

Pour obtenir la documentation d'une fonction de la bibliothèque (on dit aussi *module*), on utilise la fonction `help()`.

```
help(math.radians)
```

Pour connaître les fonctions disponibles dans une bibliothèque, on utilise la fonction `dir()`.

```
dir(math)
```

Pour appeler une fonction d'une bibliothèque, on doit préfixer son nom par le **nom de la bibliothèque suivi d'un point**.

```
import math
#Affichage du sinus de pi/4
print("Sinus de pi/4: ", math.sin(math.pi/4))
```

```
Sinus de pi/4:  0.7071067811865476
```

On peut raccourcir l'écriture précédente, même si ce n'est pas très recommandable, avec la construction suivante :

```
from math import *

print("Sinus de pi/4:", sin(pi/4))
```

5.1 À retenir

Une fonction permet une écriture plus concise du code, une maintenance plus aisée. On la déclare de la manière suivante :

```
def nom_fonction(paramètre(s)):
    bloc_instructions
```

L'appel se fait en écrivant le nom de la fonction suivi d'éventuels arguments entre parenthèses. Si la fonction ne retourne pas de valeur on lui préfèrera le nom de *procédure*. La bibliothèque standard contient des fonctions prêtes à l'emploi. L'appel se fait de la même manière, après l'importation de la bibliothèque avec l'instruction `import`.

E1C2 : maximum
Définissez une fonction maximum(n_1, n_2, n_3) qui renvoie le plus grand de 3 nombres n_1, n_2, n_3 fournis en arguments. Par exemple, l'exécution de l'instruction :

```
print(maximum(2,5,4))
```

doit donner le résultat 5.
E2C2 : volume d'un ballon

1. Écrire une fonction `volume_ballon` qui prend en paramètre un rayon r et qui retourne le volume V d'un ballon de rayon r. Rappel : $V = \frac{4}{3} \times \pi r^3$
2. Documenter la fonction. Afficher le volume d'un ballon de football ($r = 11$ cm) puis de basket ($r = 12.4$ cm) en arrondissant les résultats (voir la fonction `round()`de la bibliothèque standard).

E3C2 : longueur d'un tweet

La longueur maximale d'un message sur Tweeter est de 280 caractères. On souhaite écrire un programme qui indique le nombre de caractères d'un message qu'on souhaiterait publier.

1. Que réalise la fonction `len()` de la bibliothèque standard ?
2. Écrire une fonction longueur_message(msg) qui prend comme paramètre un message (*une chaine de caractère*) et qui renvoit le nombre de caractères de ce message.
3. Écrire le programme qui demande à l'utilisateur de rentrer son message et qui affiche la longueur de celui-ci. Utiliser la fonction précédente.

P1C2 : distance euclidienne (France IOI)
Sur les feuilles à motifs que vous leur avez imprimées tout à l'heure, les jeunes gens souhaiteraient calculer la distance entre deux motifs, ce qui ce fait facilement si l'on perçoit la feuille comme un repère orthonormé. Ce que doit faire votre programme :

Écrivez une fonction qui prend en paramètre les coordonnées (x_A, y_A) et (x_B, y_B) de deux points et retourne la distance euclidienne entre ces deux points. On rappelle que la distance euclidienne entre deux points est égale à :

$$\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

Utilisez ensuite cette fonction dans un programme qui lit quatre nombres décimaux x_A, y_A, x_B et y_B tapés au clavier, puis affiche la distance entre les deux points correspondants.

On pourra utiliser la fonction $sqrt(x)$ de la bibliothèque math, qui retourne la racine carrée du paramètre x.