La modularité

Yao Aristide

Janvier 2018
1.0

Table des matières

Objectifs

A la fin de cette leçon, l'apprenant sera capable de :

- Décrire le fonctionnement d'une fonction
- Écrire une fonction
- Écrire des programmes simplifiés et factorisés avec les fonctions
- Utiliser un module
- Présenter l'utilité d'un module dans la conception d'un programme

Introduction

Le chapitre sur les structures répétitives a montré l'intérêt de factoriser le code. La modularité permet aussi au programmeur de définir une seule fois un bout de code qui fonctionne correctement et de le réutiliser autant de fois que nécessaire. Dans cette leçon, une autre technique de factorisation de code sera présentée. Il s'agit des fonctions. Avec les fonctions, on pourra écrire en une seule fois un *bout de code* et le réutiliser à plusieurs endroits d'un programme. Par la suite, on verra comment utiliser les fonctions qui existent déjà pour construire des programmes avancés

Les fonctions

Objectifs

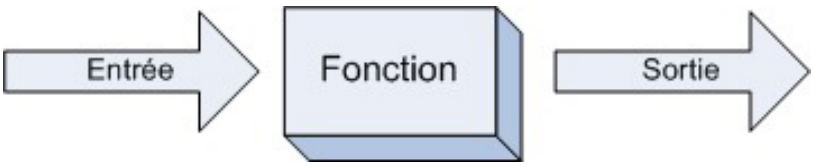
- Décrire l'architecture d'une fonction en général
- Écrire une fonction en python
- Utiliser une fonction déjà déclaré et la réutiliser dans un programme

1. Architecture d'un fonction

Définition : Fonction

Une fonction est un *morceau de code* qui exécute des actions définies dans un bloc d'instructions et renvoie un résultat. Une fonction possède en général une entrée et une sortie.

Architecture

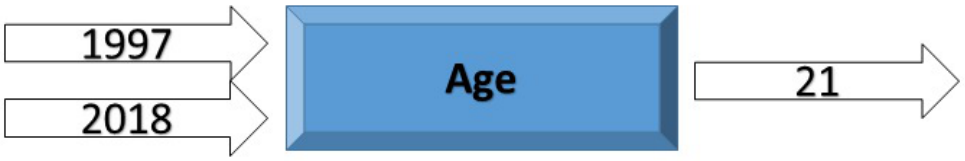


architecture d'une fonction

Lorsqu'on veut faire appel à une fonction dans un programme, il y a trois étapes :

- *L'entrée* : La fonction reçoit des informations sur lesquelles elle doit effectuer des opérations. Ces informations sont appelés *paramètres*.
- *Le corps* : Le corps de la fonction est l'ensemble des calculs, opérations, traitements appliqués sur l'entrée.
- *La sortie* : Une fois que la fonction fini ses calculs, elle renvoie un résultat. C'est ce résultat retournée par la fonction qui est appelé sortie.

Exemple : fonction qui calcul l'age



Architecture d'une fonction qui calcule l'age d'un individu

Pour calculer l'age, la fonction a besoin de *deux valeurs en entrée* qui sont: la date de naissance (1997) et l'année courante (2018). Elle fournit en *sortie* l'age qui est 21

2. Déclaration de fonction

Syntaxe : Python 2 ou 3

def nom_de_la_fonction(`parametre`1, `parametre`2, ..., `parametre`n) :

[indentation] bloc d'instructions

Explication de la syntaxe

Dans la déclaration d'une fonction on trouve dans l'ordre les éléments clés suivant :

- *def*: ce mot-clé est l'abréviation de "*define*" qui signifie "définir" en Français. Il est indispensable pour indiquer à Python que l'on voudrait créer ou définir une fonction.
- *nom_de_la_fonction*: désigne le nom de la fonction qu'on veut définir, elle suit les même règles de nommage que les variables.
- *(parametre1, parametre2, ..., parametren)* : désigne *l'entrée* qui est fournie à la fonction afin qu'elle puisse réaliser ses calculs. Les parenthèses sont obligatoires mais les paramètres sont facultatifs.
- *:* : ils clôturent la ligne et indique que la ligne suivante est le début de l'écriture du bloc d'instructions. Il est le délimiteur de la fonction.

Exemple

```
>>> def calculAge(annee_naissance, annee_courante):
        age = annee_courante - annee_naissance
        print("Votre avez",age,"ans")
```

Fonction python pour afficher l'age

Attention

Déclarer une fonction et une variable qui porte le même nom entraîne que l'un écrase l'autre en fonction de l'ordre de déclaration.

Une variable et une fonction ne doivent pas porter le même nom.

3. valeur par défaut d'un paramètre

Il est possible d'indiquer une valeur par défaut à un paramètre. Dans ce cas, si l'utilisateur du programme n'affecte pas de valeur à ce paramètre, il prend sa valeur par défaut.

Pour ce faire, il suffit d'affecter au paramètre une valeur dans la déclaration de la fonction.

Exemple

```
>>> def calculAge(annee_naissance, annee_courante=2018):
        age = annee_courante - annee_naissance
        print("Votre avez",age,"ans")
```

fonction avec paramètre par défaut

Ici on indique à la fonction l'instruction suivante :

Si l'utilisateur du programme ne fournit pas de valeur au paramètre annee_courante, alors utilise la valeur par défaut qui est 2018

4. Appel d'une fonction dans un programme

Après avoir déclarer une fonction, on peut faire appel à elle en utilisant la syntaxe suivante

Syntaxe

>>>nom_de_la_fonction(`parametre`1, `parametre`2, ..., `parametre`n)

Exemple

```
>>> calculAge(1997, 2018)
Votre avez 21 ans
>>> calculAge(1997, 2020)
Votre avez 23 ans
>>> calculAge(1997)
Votre avez 21 ans
```

Appel d'une fonction déjà déclarée en python

Lors du troisième appel à la fonction calculAge, on remarque que le paramètre année_courante n'a pas été fourni ; mais la valeur par défaut 2018 a été utilisé pour faire le calcul

Remarque

Techniquement, lorsqu'on fait appel à une fonction dans un programme, les paramètres fournis à la fonction sont appelés *arguments*

5. Valeur de retour d'une fonction: Sortie

La fonction calculAge écrite ci-dessus ne retourne aucune valeur. Ainsi il serait impossible de tester en dehors de la fonction si l'utilisateur est mineur ou majeur. Grâce à l'instruction *return*, il est possible de fournir une valeur de sortie à la fonction.

Exemple

```
>>> def calculAge(annee_naissance, annee_courante=2018):
        return annee_courante - annee_naissance

>>> calculAge(1997)
21
>>> print("Vous avez", calculAge(1997), "ans")
Vous avez 21 ans
```

1

2

fonction avec l'instruction return

Le premier bloc (1) représente la définition de la fonction; l'instruction *return* permet à la fonction de retourner l'age. Ainsi l'age retourné par la fonction peut être utilisé plus tard dans le programme pour réaliser des tests et faire des opérations comme l'indique le bloc (2)

Attention

L'instruction *return* ne peut être utilisée hors de la déclaration d'une fonction. Toute instruction qui se trouve après la ligne d'instruction *return* n'est pas exécutée.

6. Exercice

[Solution n°1 p 14]

Complétez ce code pour définir une fonction nommée welcome

welcome

[indentation] print("Bonjour et bienvenue")

7. Exercice

[Solution n°2 p 14]

Ordonnez ce code pour qu'il fonctionne correctement

1. [indentation] print("nous composons en algebre")
2. composition()
3. def composition() :

Réponse : _____ _____ _____

8. Exercice

[Solution n°3 p 14]

Quel est le résultat de ce code ?

```
>>> def double(x):
        print(2*x)


>>> double(5)
```

capture

9. Exercice

[Solution n°4 p 14]

Completer le code suivant pour que la fonction nommée pair affiche oui si son paramètre est un nombre pair et non sinon

pair(x)

[indentation] if x%2 == 0

[indentation][indentation] ("oui")[indentation]

[indentation]

[indentation][indentation] ("non")

10. Exercice

[Solution n°5 p 14]

Donnez le résultat d'exécution de ce code

```
>>> def addition(x,y):
        somme = x+y
        return somme
        print("fin de la fonction")


>>> print(addition(9,10))
```

capture

Les modules

II

Objectifs

- Décrire un module et son utilisation
- Utiliser les modules pour écrire des programmes

1. Introduction aux modules

Un module est un ensemble de fonctions et de variables destiné à résoudre des problèmes d'un domaine bien défini. par exemple pour utiliser les fonctions mathématiques telles que cos, sin ou tan, il faudrait utiliser le module *math*.

Pour utiliser un module dans un programme, il faut l'importer grâce au mot-clé *import*.

Remarque

Pour connaître toutes les fonctions d'un module, on peut saisir dans l'interpréteur interactif la fonction suivante :

>>>help("nom_du_module")

2. Import standard

L'import standard consiste à importer un module et à faire usage de toutes les fonctions de ce module. Dans ce cas la syntaxe suivante est utilisée

Syntaxe

>>>*import* nom_du_module

>>>*nom_du_module*.nom_de_la_fonction

import standard et espace de nom

Pour faire appel à une fonction, il faut utiliser le nom du module en préfixe. En effet l'instruction import crée un *espace de nom* dénommé *"nom_du_module"* où il regroupe toutes les fonctions et toutes les variables du module. Ainsi pour avoir accès à une fonction de ce module, il faudrait préciser d'abord son espace de nom ; c'est-à-dire qu'il faut accéder d'abord à l'espace et ensuite à la fonction de l'espace.

Exemple : Trouver la racine carré d'un nombre

```
>>> import math
>>> math.sqrt(25)
5.0
```

importation du module math

la fonction sqrt (square root, qui signifie racine carrée en Français) du module math renvoie en sortie un nombre réel qui représente la racine carré de l'argument 25

Complément

Le jeu du pendu consiste à retrouver un nombre choisi aléatoirement après un certains nombre de tentatives. Le cœur de ce jeu est la génération du nombre aléatoire. Plutôt que d'écrire personnellement une fonction qui permet de générer un nombre aléatoire, il est possible d'utiliser *le module random* qui fournit une fonction capable de générer un nombre aléatoire dans un intervalle.

```
>>> import random
>>> nombre_aleatoire = random.randint(1, 1000)
>>> print(nombre_aleatoire)
669
>>> nombre_aleatoire_1 = random.randint(1, 1000)
>>> print(nombre_aleatoire_1)
512
```

Générer un nombre aléatoire grâce a la fonction randint du module random

3. Import spécifique

L'import spécifique consiste à importer seulement un certain nombre de fonction d'un module. Cela se fait grâce à la syntaxe suivante :

Syntaxe

>>>*from* nom_du_module *import* nom_de_la_fonction

>>>nom_de_la_fonction

Ici plutôt que de créer un espace de nom du module, l'instruction *from* va chercher directement la fonction à l'intérieur du module et la charge dans l'interpréteur. Ainsi la fonction est accessible au même titre que la fonction *print* ou la fonction *type* sans saisir son espace de nom

Exemple

```
>>> from math import sqrt
>>> sqrt(25)
5.0
```

import spécifique de la fonction sqrt

Remarque

Il est possible d'importer plusieurs fonctions d'un module en séparant chaque fonction par une virgule.

>>>*from* math *import* sqrt, fabs

fabs est une fonction qui renvoie la valeur absolue d'un nombre fourni en paramètre.

4. Import personnalisé

Faire une importation personnalisée consiste à faire une importation spécifique en donnant un alias à la fonction ; c'est-à-dire en remplaçant le nom de la fonction par un autre nom grâce au mot-clé *as*.

Syntaxe

>>>*from* nom_du_module *import* nom_de_la_fonction *as* nouveau_nom

>>>nouveau_nom

Exemple

```
>>> from math import sqrt as racine_carre
>>> racine_carre(25)
5.0
```

import personnalisé de la fonction sqrt

5. Exercice

[Solution n°6 p 14]

Quel module est utilisé dans ce code

>>>import math

>>>numero = 10

>>>print(math.sqrt(numero))

6. Exercice

[Solution n°7 p 14]

Compléter le code pour importer seulement les fonctions cos et sin du module math

math cos

7. Exercice

[Solution n°8 p 14]

Quel est le résultat de ce code ?

>>>print(math.sqrt(25))

- ◯ une erreur
- ◯ 5.5
- ◯ 5
- ◯ 25
- ◯ 5.0

Conclusion

A l'issue de cette leçon, l'apprenant est capable de créer des programmes factorisés et simplifiés à l'aide des fonctions, et de faire usage des modules pour exploiter toute la puissance de Python. La leçon suivante s'intéressera aux structures de données.

Solutions des exercices

> Solution n° 1

Exercice p. 8

```
def welcome ():
[indentation] print("Bonjour et bienvenue")
```

> Solution n° 2

Exercice p. 8

```
1. def composition() :
2. [indentation] print("nous composons en algebre")
3. composition()
```

> Solution n° 3

Exercice p. 8

```
10
```

> Solution n° 4

Exercice p. 8

```
def pair(x) :
[indentation] if x%2 == 0
[indentation][indentation] print ("oui")[indentation]
[indentation] else:
[indentation][indentation] print ("non")
```

> Solution n° 5

Exercice p. 9

```
19
```

> Solution n° 6

Exercice p. 12

```
>>>import math
>>>numero = 10
>>>print(math.sqrt(numero))
math
```

> Solution n° 7

Exercice p. 12

```
from math import sin, cos
```

Exercice p. 12

> Solution n°8

- ○ une erreur
- ○ 5.5
- ○ 5
- ○ 25
- ◉ 5.0