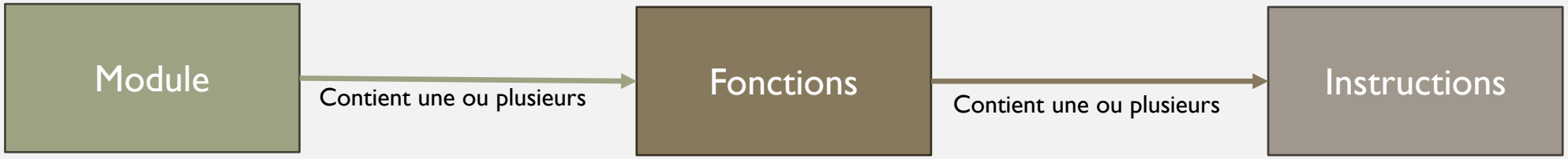
PROGRAMMATION PYTHON

Chapitre 6: Modules

Sommaire

Introduction

On appelle module un groupe de fonctions stockées dans un ou plusieurs fichiers par exemple les fonctions mathématiques peuvent être placées dans un module dédié aux mathématiques.



Un module en Python identifie ses fonctions par leurs signatures qui est le nom de la fonction. Lors de la création d'un module il faut veiller à ne pas avoir deux fonctions différentes avec le même nom.

Les modules vont nous permettre de scinder l'ensemble de nos fonctions dans plusieurs fichiers, sans les modules l'on serait obligé d'écrire tout le code dans un seul fichier ce qui devient contraignant pour un gros projet.

Python propose des modules et fonctions de base comme le module ***math, collections…*** Dans un projet on aura besoin d'écrire des modules et fonctions spécifiques selon nos besoins et notre organisation d'où l'objet de ce chapitre.

La méthode *import*

Lorsqu'on veut utiliser un module, il faut l'importer grâce à l'instruction ***import nom_du_module.*** Tous les modules internes de Python comme le module ***math*** ne sont pas chargés par défaut quand on démarre l'interpréteur de Python. Par exemple, pour utiliser les fonctions du module ***math*,** il faut l'importer.

```
>>> # Essaie d'avoir la documentation de math sans l'importer
>>> help(math)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    help(math)
NameError: name 'math' is not defined
>>>
>>> # Essaie après import
>>> import math
>>> help(math)
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

        The result is between 0 and pi.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

        The result is between -pi/2 and pi/2.

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.

        The result is between -pi/2 and pi/2.
```

La méthode *import*

Quand on importe un module pour utiliser une de ses fonctions, il faut précéder la fonction du nom du module. Par exemple pour avoir le carré d'un nombre avec la fonction ***pow*** du module ***math*** on aura:

```
>>> help(math.pow)
Help on built-in function pow in module math:

pow(x, y, /)
    Return x**y (x to the power of y).

>>> # Affichage de la doc de la fonction pow
>>> import math
>>> help(math.pow)
Help on built-in function pow in module math:

pow(x, y, /)
    Return x**y (x to the power of y).

>>> # Utilisation de la fonction pow pour calculer le carré
>>> math.pow(3, 2)
9.0
>>> math.pow(2, 2)
4.0
```

Dans l'exemple précédent on affiche la documentation de la fonction ***pow*** avant de l'utiliser quand on connait déjà le rôle de la fonction on peut s'en passer. Quand on veut afficher la documentation il faut éviter de mettre les parenthèses dans lors de l'utilisation de ***help*** du type ***help(math.pow())*** qui n'affiche pas la documentation de la fonction ***pow***.

Espace de nom spécifique

Quand on import un module par exemple ***import math***, cela crée un espace de nom dénommé « math » qui contient les variables et les fonctions du module math.

Quand on exécute ***math.pow(3, 2),*** on précise à Python que nous souhaitons exécuter la fonction ***pow*** contenue dans le module ***math*** d'espace de nom ***math*.**

Grâce aux espaces de nom, on peut avoir dans un autre module où dans l'espace de nom principal une autre fonction ***pow*** sans conflit car chaque fonction lors de l'appel spécifie son espace de nom.

Parfois on a besoin d'importer un module avec un nom différent pour l'espace de nom avec le mot clé ***as*.**

import nomModule* => importe avec comme espace de nom *nomModule
import nomModule as nomEspaceDeNom* => import avec comme espace de nom *nomEspaceDeNom

```
>>> # Import avec espace de nom par défaut
>>> import math
>>> help(math.pow)
Help on built-in function pow in module math:

pow(x, y, /)
    Return x**y (x to the power of y).

>>> math.pow(2,2)
4.0
```

```
>>> # Import avec espace de nom spécifique
>>> import math as mathematique
>>> help(mathematique.pow)
Help on built-in function pow in module math:

pow(x, y, /)
    Return x**y (x to the power of y).

>>> mathematique.pow(2,2)
4.0
```

La méthode from … import

Parfois, on a juste besoin d'importer un nombre restreint de fonctions d'un module dans ce cas on utilise la syntaxe ***from nomModule import nomFonction,…***

Avec cette méthode, on a plus besoin de spécifier le nom du module quand on appelle la fonction.

```
>>> # Importer uniquement la fonction de puissance
>>> from math import pow
>>> help(pow)
Help on built-in function pow in module math:

pow(x, y, /)
    Return x**y (x to the power of y).

>>> pow(2, 2)
4.0
```

```
>>> # Importer les fonctions valeur absolue fabs et puissance pow du module mathématique
>>> from math import pow, fabs
>>> help(fabs)
Help on built-in function fabs in module math:

fabs(x, /)
    Return the absolute value of the float x.

>>> help(pow)
Help on built-in function pow in module math:

pow(x, y, /)
    Return x**y (x to the power of y).
```

Quand on veut importer toutes les fonctions d'un module, on utilise la syntaxe: ***from nomModule import ****

```
>>> # Importer tous les fonctions du module mathématiques
>>> from math import *
>>> pow(2,2)
4.0
>>> help(fabs)
Help on built-in function fabs in module math:

fabs(x, /)
    Return the absolute value of the float x.
```

Exercices

Mini Projet 1

Créer un module ***multiplication*** qui contiendra les fonctions suivantes:

- *addition(nbre1, nbre2):* qui retourne le résultat de nbre1 + nbre2
- *soustraction(nbre1, nbre2):* qui retourne le résultat de nbre1 - nbre2
- *multiplication(nbre1, nbre2):* qui retourne le résultat de nbre1 * nbre2
- *division(nbre1, nbre2):* qui retourne le résultat de nbre1 / nbre2 attention nbre2 doit être différent de 0

Créer un deuxième module ***helper*** qui contiendra les fonctions suivantes:

- *afficheMenu():* qui affiche le menu des actions (addition, soustraction, multiplication, division)
- *saisirAction(minAction, maxAction):* qui demande le numéro de l'action à réaliser numéro compris entre *minAction* et *maxAction* par exemple dans notre cas çà sera entre 1 et 4
- *saisirNombre(typeNbre):* qui va demander à l'utilisateur de saisir un nombre *typeNbre* permet de dire si c'est le nombre 1 ou le nombre 2 qui doit être renseigné.

Créer un fichier main.py qui va utiliser l'ensemble des fonctions des modules précédents pour faire réaliser une calculatrice simple.

Exercices

Mini Projet 2

En Python, pour générer un nombre aléatoire on utilise le module ***random*** comme ci-dessous:

```
>>> # Génération de nombre aléatoire
>>> import random
>>> random.randint(0, 100)
20
>>> random.randint(0, 100)
55
```

Le code précédent génère un nombre aléatoire entre 0 et 100 inclus.
Utilisez le code précédent, pour réaliser le mini-projet du nombre magique:
Coder un jeu de nombre magique où l'utilisateur doit deviner un nombre aléatoire généré en 10 tentatives maximum. Le jeu doit être constitué de trois modes, le mode facile (nombre magique entre 0 et 100), le mode moyen (nombre magique entre 0 et 1000) et le mode difficile (nombre magique entre 0 et 10000). Après choix du mode faire deviner le nombre magique à l'utilisateur quand l'utilisateur entre un nombre inférieur au nombre magique afficher « Le nombre magique est plus grand que nombreSaisi ». Quand l'utilisateur entre un nombre supérieur au nombre magique afficher « Le nombre magique est plus petit que nombreSaisi ». Sinon afficher « Bingo vous avez trouvé le nombre magique en x essais ». En cas d'échec afficher un message d'échec et demander si l'utilisateur veut recommencer si oui réafficher le menu.
L'organisation des fonctions et modules est libre.

FIN CHAPITRE 6