Chapitre 4 - Modularité

Objectifs du cours

- Comprendre le concept de modularité dans le développement logiciel.
- Explorer l'utilisation de fonctions et de modules en Python.
- Apprendre à créer des modules réutilisables.
- Comprendre les avantages de la modularité en termes de maintenance et de réutilisation du code.

I - Introduction

La programmation modulaire est une approche de conception logicielle qui consiste à **diviser un programme en composants autonomes et interconnectés appelés "modules"**.

Chaque module est une entité indépendante qui accomplit une tâche spécifique et communique avec d'autres modules de manière organisée.

En Python, un module est simplement un fichier contenant des définitions et des déclarations Python, et la modularité est la pratique qui consiste à organiser et structurer ces modules de manière à rendre le code plus lisible, réutilisable et facile à maintenir.

II - Rappel : Fonctions en Python

Les fonctions en Python sont un moyen fondamental d'implémenter la modularité. Une fonction est un bloc de code qui réalise une tâche spécifique et peut être appelée depuis d'autres parties du programme.

Syntaxe de Définition de Fonction

```
def nom_de_la_fonction(parametres):
  # code de la fonction
  return resultat
```

Exemple de Fonction

```
def addition(a,b):
  return a+b
```

III - Modules en Python

1) Définition d'un module

Qu'est-ce qu'un module en Python ?

Un module en Python est essentiellement un fichier contenant des définitions et des déclarations Python. Il peut contenir des variables, des fonctions, et même des classes. L'idée principale derrière les modules est de diviser votre code en morceaux logiques et autonomes pour faciliter la gestion et la réutilisation du code.

Pourquoi utiliser des modules ?

- **Réutilisation du code** : Les modules permettent de réutiliser du code sans avoir à le réécrire. Cela favorise la modularité et évite la duplication inutile du code.
- **Gestion facilitée** : En séparant le code en modules, la gestion de chaque composant devient plus facile. Vous pouvez vous concentrer sur un aspect spécifique de votre programme à la fois.

Interface et implémentation

- **Interface** : L'interface d'un module définit les fonctions, variables ou classes qui sont destinées à être utilisées par d'autres parties du programme. C'est comme un contrat décrivant comment interagir avec le module.
- **Implémentation** : L'implémentation représente le code interne du module qui réalise les fonctionnalités définies dans l'interface. L'implémentation est généralement cachée, et seules les parties nécessaires pour l'utilisation du module sont exposées dans l'interface.

2) Création d'un module simple

Structure de base

Pour créer un module en Python, suivez ces étapes :

- **Créez un fichier Python** : Commencez par créer un fichier Python avec l'extension .py. Par exemple, `mon_module.py`.
- **Définissez des fonctions ou des classes** : Dans ce fichier, définissez des fonctions, des variables ou des classes qui auront un rôle spécifique.

Exemple pratique : Fichier `operations.py`

```
#contenu du fichier operations.py
def addition(a,b):
  return a+b

def soustraction(a,b):
  return a-b
```

Importation de modules

Pour utiliser les fonctions d'un module dans un autre fichier, nous utilisons l'instruction **import**.

Exemple d'importation

```
#dans un autre fichier python
import operations

resultat_addition = operations.addition(3,5)
resultat_soustraction = operations.soustraction(2,7)
```

IV - Avantages de la modularité

1) Réutilisation du code

Principe de réutilisation

L'un des avantages clés de la modularité est la possibilité de réutiliser du code existant.
Les modules offrent une encapsulation du code, permettant à d'autres parties du programme (ou à d'autres programmes) d'utiliser des fonctionnalités spécifiques sans connaître les détails de l'implémentation.

Exemple pratique
Imaginons que vous ayez développé un module `calculs` qui contient des fonctions mathématiques utiles. En l'important dans un nouveau programme, vous pouvez utiliser ces fonctions sans avoir à réécrire le code mathématique.

```
#programme principal
import calculs

resultat = calculs.carre(6)
```

2) Maintenance facilitée

Isolation des changements

En divisant le programme en modules, les modifications apportées à un module n'affectent pas nécessairement les autres parties du programme. Cela permet une maintenance plus facile, car vous pouvez apporter des améliorations ou corriger des erreurs sans risquer de perturber l'ensemble du système.

Exemple pratique
Si une mise à jour est nécessaire dans le module `calculs`, vous pouvez effectuer cette modification sans toucher aux autres parties du programme. Les dépendances sont gérées de manière à isoler les changements.

```
#module calculs.py

def carre(x):
  return x**2

#mise a jour
def cube(x):
  return x**3
```

3) Collaboration efficace

Travailler en parallèle

La modularité facilite la collaboration entre les membres d'une équipe de développement. Chaque développeur peut travailler sur un module spécifique sans entrer en conflit avec les autres, à condition que l'interface du module reste inchangée.

Exemple pratique
Si votre projet est divisé en modules distincts, un développeur peut travailler sur l'amélioration des fonctionnalités d'un module sans interférer avec le travail d'un autre développeur sur un module différent.

V - Organiser des modules

1) Paquets (Packages)

Introduction

Les paquets (packages) en Python sont des structures permettant d'organiser les modules en hiérarchie. Un paquet est simplement un répertoire contenant un fichier spécial `__init__.py` (qui peut être vide) et des modules ou sous-packages.

Structure de base

Un exemple de structure de base avec des paquets :

```
mikael@mikael-HP-EliteBook-840-G3:~/enseignement/2024_2025/Terminale_NSI/projet/
   microbit/MNP/package$ ls -R
.:
LICENSE
README
--> src
     __init__.py
     router_MNP.py
```

VI - Création de modules réutilisables

1) Documentation

Importance de la documentation

Documenter votre code est essentiel pour permettre aux utilisateurs (y compris vous- même et d'autres développeurs) de comprendre comment utiliser votre module sans devoir examiner le code source.

Docstrings

Utilisez des docstrings (chaînes de documentation) pour documenter vos fonctions, classes et modules. Les docstrings sont des commentaires spéciaux qui peuvent être récupérés par des outils de documentation tels que Sphinx.

```
#contenu du fichier operations.py
def addition(a,b):
  '''
    Cette fonction retourne la somme de deux nombres
    Arguments : a un nombre
                b un nombre
    return : la somme des nombres a et b
  '''
  return a+b
```

2) Variable `__name__`

La variable spéciale `__name__` permet de déterminer si le module est exécuté en tant que programme principal ou importé dans un autre module.

VII - Les principaux modules Python

Module math

Fonctions Mathématiques : Le module math propose des fonctions mathématiques standard.
`math.sqrt(x)` : Renvoie la racine carrée de x.

Module datetime

Manipulation de dates et heures : Le module datetime permet la manipulation de dates et heures.
`datetime.now()` : Renvoie un objet datetime représentant la date et l'heure actuelles.

Module random

Génération de nombres aléatoires : Le module random permet de générer des nombres aléatoires.
`random.randint(a, b)` : Renvoie un entier aléatoire entre a et b inclus.

Module requests

Effectuer des requêtes HTTP : Le module requests est utilisé pour effectuer des requêtes HTTP.
`requests.get(url)` : Effectue une requête GET à l'URL spécifiée.

Module os

Interaction avec le Système d'Exploitation : Le module os fournit des fonctions pour interagir avec le système d'exploitation.
`os.listdir(path)` : Renvoie une liste des noms des fichiers et dossiers dans le chemin spécifié.

Module json

Traitement JSON : Le module json facilite le codage et le décodage de données JSON.
`json.loads(s)` : Analyse une chaîne JSON et renvoie l'objet Python correspondant.

Module sys

Fonctionnalités du Système : Le module sys fournit des fonctions et variables liées au système.
`sys.version` : Renvoie la version actuelle de Python.

Module collections

Types de Conteneurs : Le module collections propose des alternatives aux types de conteneurs intégrés.
`Counter(iterable)` : Compte le nombre d'éléments distincts dans un itérable.

Module re

Expressions Régulières : Le module re permet d'utiliser des expressions régulières.
`re.findall(pattern, string)` : Trouve toutes les occurrences du motif dans la chaîne.

Module urllib

Utilitaires pour les URL : Le module urllib propose des utilitaires pour travailler avec les URL.
`request.urlopen(url)` : Ouvre une URL et renvoie un objet http.client.HTTPResponse.

Module sqlite3

Interagir avec SQLite : Le module sqlite3 permet d'interagir avec des bases de données SQLite.
`sqlite3.connect(database)` : Connecte à la base de données spécifiée.

Module csv

Manipulation de Fichiers CSV : Le module csv facilite la manipulation de fichiers CSV.
`csv.reader(file)` : Renvoie un objet lecteur qui itère sur les lignes du fichier CSV.

VIII - Documentation en Ligne

Consultez la documentation officielle de Python en ligne pour des informations détaillées sur les modules.
Documentation Python : https ://docs.python.org/fr/3/