Python: Les modules

Les modules en Python sont des fichiers contenant des définitions et des instructions. Ils permettent de diviser un programme en plusieurs fichiers pour une meilleure organisation et réutilisation du code. Un module peut définir des fonctions, des classes et des variables, ainsi que des blocs de code exécutables.

5GMS 6TTR 5TTR Exploration

Qu'est-ce qu'un module?

Définition

Un module est un fichier contenant du code Python avec l'extension `.py`. Il peut contenir des fonctions, des classes, des variables, et même d'autres modules.

Avantages

- **Réutilisation de code**: Facilite la réutilisation de code entre différents projets.
- **Organisation**: Permet une meilleure organisation du code en séparant les différentes parties logiques.
- **Espace de noms**: Évite les conflits entre noms de variables, fonctions, etc.

Utilisation des Modules Standards

Python est fourni avec une bibliothèque standard de modules. Voici comment les utiliser :

Importation

Pour utiliser un module, il faut d'abord l'importer avec le mot-clé `import`.

Exemple:

```
1 | import math
```

Accès aux Fonctions

Après avoir importé un module, tu peux accéder à ses fonctions et variables en utilisant la notation pointée.

Exemple:

```
print(math.sqrt(16))  # Affiche 4.0
```

Création de tes Propres Modules

Créer un Fichier Module

1. **Écris du code**: Ouvre un éditeur de texte, écris ton code (par exemple, des fonctions ou des variables).
2. **Enregistre**: Sauvegarde le fichier avec un nom descriptif et l'extension `.py`.

Exemple de fichier `salutations.py`:

```
def bonjour(nom):
    return f"Bonjour, {nom}!"

def au_revoir(nom):
    return f"Au revoir, {nom}!"
```

Utilisation de ton Module

Après avoir créé ton module, tu peux l'importer et utiliser ses fonctions dans d'autres scripts Python.

Exemple:

```
import salutations

print(salutations.bonjour("Alice"))  # Affiche "Bonjour, Alice!"
```

Importation Avancée

Importation Sélective

Si tu veux seulement importer certaines fonctions d'un module, tu peux le faire directement.

Exemple:

```
from math import sqrt
print(sqrt(16))  # Affiche 4.0
```

Alias de Module

Tu peux aussi donner un alias aux modules importés pour raccourcir les noms.

Exemple:

```
1 | import numpy as np
```

Chemins de Module

Python recherche les modules dans les répertoires listés dans `sys.path`. Pour ajouter des répertoires à cette liste, tu peux modifier la variable d'environnement `PYTHONPATH` ou utiliser `sys.path.append(chemin)` dans ton script.

Exemple:

```
1 | import sys
2 | sys.path.append('/chemin/vers/ton/module')
```

Bonnes Pratiques

- **Noms descriptifs**: Donne des noms clairs et descriptifs à tes modules et fonctions.
- **Documentation**: Documente tes modules et fonctions avec des docstrings.
- **Teste ton code**: Écris des tests pour vérifier que ton module fonctionne comme prévu.
- **Respecte PEP 8**: Suis les conventions de style de code de Python pour la lisibilité.

Conclusion

Les modules sont un outil puissant en Python qui te permettent de structurer ton code de manière logique, de réutiliser du code et de partager des fonctionnalités entre différents programmes. En apprenant à les utiliser et à les créer, tu pourras rendre tes projets Python plus organisés, efficaces et faciles à maintenir.

Exercice

Créez un module appelé `utils` avec les fonctions suivantes:

`celsius(temp)`: a function to convert temperatures from fahrenheit to Celsius. `pairs_impairs(liste)`: une fonction qui compte le nombre de nombres pairs et impaires dans une liste et affiche le résultat sur la console. `suite(debut, fin)`: une fonction qui renvoie (`return`) une liste de nombres compris entre `debut` et `fin` (inclus) qui sont divisibles par 7 et par 5.

Créez un fichier Python qui importe votre module et appelle/teste vos fonctions avec 3 cas différents à chaque fois.

© 2025 Mr. Lambrechts