Fonctions, modules et fichiers en Python

Pierre Jaumier

Fonctions, modules et fichiers

- Apprendre à organiser son code en fonctions
- Découvrir les modules standards de Python
- Lire et écrire des fichiers texte

Pourquoi organiser le code ?

> "Une bonne organisation du code rend ton programme plus lisible, réutilisable et maintenable."

Quelques bonnes raisons : - Réutiliser un bloc de code plusieurs fois → **fonctions** - Séparer la logique métier de l'interface → **modules** - Sauvegarder ou lire des données → **fichiers**

En Python, tout cela est simple et intuitif !

Définir une fonction avec `def`

```
def saluer(nom):
    print(f'Bonjour {nom} !')
```

- `def` = mot-clé pour définir une fonction
- `saluer` = nom de la fonction
- `(nom)` = paramètre
- `print(...)` = corps de la fonction

Appel :

```
saluer('Alice')  # Bonjour Alice !
```

Retourner une valeur avec `return`

```
def carre(x):
    return x * x

resultat = carre(5)
print(resultat)  # 25
```

- `return` permet de renvoyer une valeur utilisable ailleurs
- Une fonction peut aussi ne rien retourner (ex: affichage)

```
def nada():
    pass

print(nada())  # None
```

Portée des variables (local vs global)

```
def exemple():
    local_var = 'locale'
    print(local_var)

global_var = 'globale'

exemple()       # Affiche 'locale'
print(global_var)  # Affiche 'globale'
# print(local_var)  #   Erreur : variable inconnue
```

Les variables définies dans une fonction sont **locales**
Les variables définies en dehors sont **globales**

Arguments par défaut

```
def dire_bonjour(nom='inconnu'):
    print(f'Bonjour {nom} !')

dire_bonjour()         # Bonjour inconnu !
dire_bonjour('Bob')    # Bonjour Bob !
```

- Très utile pour rendre les paramètres optionnels
- À comparer avec Java/C++ où il faut surcharger les fonctions

Modules – Importer des bibliothèques

```
import math

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

- `import` permet d'utiliser du code externe
- Des milliers de modules disponibles dans la **bibliothèque standard**

Exemples utiles : - `math`, `random`, `datetime`, `os`, `sys`

Module `math` – Opérations mathématiques

```
import math

print(math.sqrt(16))    # Racine carrée → 4.0
print(math.pi)          # Valeur de   → 3.14159...
print(math.ceil(2.3))   # Arrondi supérieur → 3
print(math.floor(2.7))  # Arrondi inférieur → 2
```

Module `random` – Tirage aléatoire

```
import random

print(random.randint(1, 6))     # Un entier entre 1 et 6
print(random.random())          # Un float entre 0 et 1
print(random.choice(['a', 'b', 'c']))  # Choix au hasard
```

Astuce : Utile pour jeux, tests, simulations

Module `datetime` – Manipuler les dates et heures

```python
from datetime import datetime

# Date actuelle
now = datetime.now()
print(now.strftime('%Y-%m-%d %H:%M'))  # Format personnalisé

# Créer une date spécifique
date = datetime(2025, 4, 5, 10, 30)
print(date)
```

Principales méthodes : - `.now()` → date actuelle - `.strftime()` → formater la date - `.timedelta()` → calculer des durées

Module `os` – Interagir avec le système

```python
import os

print(os.getcwd())              # Chemin du dossier courant
os.makedirs('nouveau_dossier', exist_ok=True)  # Créer un dossier
print(os.listdir('.'))          # Lister les fichiers
```

Utile pour : - Gérer des dossiers/fichiers - Travailler avec des chemins dynamiques

Module `sys` – Contrôler l'environnement d'exécution

```python
import sys

print(sys.platform)         # OS utilisé (win32, linux, darwin...)
print(sys.version)          # Version de Python
sys.exit()                  # Quitter le programme
```

Autres usages : - Lire depuis l'entrée standard (`sys.stdin`) - Écrire sur la sortie standard (`sys.stdout`) - Gérer les arguments passés au script (`sys.argv`)

Mini-projet : Générateur de mot de passe

Écrivez un programme qui génère un mot de passe aléatoire de 10 caractères.

```
import random
import string

caracteres = string.ascii_letters + string.digits + '!@#$%&'
mdp = ''.join(random.choices(caracteres, k=10))
print(f'Votre mot de passe : {mdp}')
```

Utilise : - `random.choices()` pour choisir plusieurs caractères - `string` pour avoir facilement lettres et chiffres

Bonnes pratiques

Importez uniquement ce que vous utilisez

```
from math import sqrt  # plutôt que import math si tu n'utilises qu'une seule fonction
```

Documentez vos imports

```
# Pour gérer les dates de création
from datetime import datetime
```

N'utilisez pas trop de modules inutiles
→ Gardez votre code propre et compréhensible

Importer une partie d'un module

```
from random import randint

print(randint(1, 6))  # Un nombre entre 1 et 6
```

Avantages : - Évite de taper `random.randint()` à chaque fois - Gagne en lisibilité si peu de fonctions nécessaires

À utiliser avec parcimonie si conflit de noms possible

Créer son propre module

Créez un fichier `mymodule.py` :

```
def info():
    print('Ceci est un module personnalisé')
```

Utilisez-le dans un autre script :

```
import mymodule

mymodule.info()  # Ceci est un module personnalisé
```

C'est ainsi qu'on structure un projet en plusieurs fichiers

Lire un fichier texte

```
with open('exemple.txt', 'r') as fichier:
    contenu = fichier.read()
    print(contenu)
```

- `'r'` = mode lecture
- `with` = gère automatiquement la fermeture du fichier
- `.read()` = lit tout le contenu

Comparaison avec Java : pas besoin de try-catch, Python gère ça avec `with`

Écrire dans un fichier

```
with open('sortie.txt', 'w') as fichier:
    fichier.write('Première ligne\\n')
    fichier.write('Deuxième ligne\\n')
```

- `'w'` = mode écriture (remplace le contenu existant)
- Utilisez `'a'` pour ajouter sans effacer (`append`)
- `\n` = caractère de saut de ligne

Lire ligne par ligne

```
with open('donnees.csv', 'r') as fichier:
    for ligne in fichier:
        print(ligne.strip())
```

- Utile pour traiter des fichiers volumineux
- `strip()` supprime les espaces inutiles et les sauts de ligne

Mini-projet : Compteur de mots

Écrivez un programme qui : 1. Lit un fichier texte 2. Compte combien de mots contient ce fichier

```
with open('texte.txt', 'r') as f:
    contenu = f.read()
    mots = contenu.split()
    print(f"Nombre de mots : {len(mots)}")
```

Idéal pour pratiquer : - Lecture de fichiers - Manipulation de listes - Fonctions simples

4 – Bonnes pratiques

Utilisez des fonctions pour : - Regrouper du code réutilisable - Améliorer la lisibilité

Organisez votre code en modules quand : - Le projet devient gros - Vous voulez partager des outils

Travaillez avec `with` pour les fichiers : - Plus sûr - Plus propre

5 – Conclusion

Ce que vous avez appris aujourd'hui : - Définir et utiliser des fonctions - Importer et créer des modules - Lire et écrire des fichiers texte

Prochaines étapes : - Programmation orientée objet