

Méthodologie de programmation – Introduction

Initiation à l'algorithmique et à la programmation

Revekka Kyriakoglou

Objectifs du cours

L'objectif de ce cours est d'apprendre les bases de la programmation, principalement avec Python, afin de savoir :

- comprendre et écrire un programme simple ;
- décomposer un problème en étapes ;
- manipuler des variables et des données ;
- utiliser des conditions, des boucles et des fonctions ;
- détecter et corriger des erreurs ;
- produire un code lisible et structuré ;
- progressivement concevoir des algorithmes.

Principe du cours :

On apprend à programmer en programmant.

Évaluation

Ce cours sera évalué à l'aide de :

- 1 contrôles continus ;
- 2 petites interrogations / exercices ;
- 3 projet ou examen final.

Pourquoi Python ?

- Syntaxe relativement simple et lisible.
- Très utilisé dans de nombreux domaines.
- Nombreuses bibliothèques disponibles.
- Grande communauté et beaucoup de documentation.
- Disponible sous Windows, macOS et Linux.
- Gratuit et open source.

De quoi avons-nous besoin ?

Pour programmer en Python, il nous faut :

- l'interpréteur Python ;
- un éditeur de texte ou un environnement de développement ;
- un répertoire dans lequel enregistrer nos programmes ;
- un terminal pour exécuter nos programmes.



Pour vérifier que Python est installé :

```
python3 -version
```

Créer un répertoire de travail

- 1 Ouvrir un terminal.
- 2 Retourner dans le répertoire personnel si nécessaire :

```
cd
```

- 3 Créer le répertoire :

```
mkdir -p prog/python/cours
```

- 4 Se déplacer dans ce répertoire :

```
cd prog/python/cours
```

Éditeur

Vous pouvez utiliser l'éditeur de votre choix.

- Visual Studio Code
- Gedit
- Sublime Text
- PyCharm
- tout autre éditeur de texte

L'important est de pouvoir créer et modifier des fichiers texte avec l'extension `.py`.

Un fichier source Python

Un programme Python est généralement enregistré dans un fichier dont le nom se termine par :

`.py`

Par exemple :

`bonjour.py`

Un fichier Python peut contenir notamment :

- des commentaires ;
- des instructions ;
- des expressions ;
- des appels de fonctions ;
- plus tard : des fonctions définies par le programmeur.

Hello World

```
1 # Mon premier programme Python
2 print("Hello, World!")
```

Que fait ce programme ?

- La première ligne est un **commentaire**.
- La deuxième ligne demande à Python d'afficher un texte.
- `print()` est une fonction déjà fournie par Python.

Commentaires

Un commentaire commence par le caractère #.

```
# Ceci est un commentaire
```

Les commentaires sont ignorés lors de l'exécution.

Ils servent à :

- expliquer une partie du programme ;
- rappeler une décision importante ;
- rendre le code plus facile à relire.



Les chaînes entre triples guillemets ne sont pas des “commentaires multilignes” au sens strict : ce sont des chaînes de caractères, souvent utilisées comme *docstrings*.

Exécuter un programme

Python est principalement un langage **interprété**.

Pour exécuter le fichier `bonjour.py` :

```
python3 bonjour.py
```

L'interpréteur Python :

- 1 lit les instructions ;
- 2 vérifie qu'elles peuvent être comprises ;
- 3 les exécute.

Une partie normale de la programmation

Faire des erreurs est normal.

Lorsqu'un programme ne fonctionne pas, on cherche à répondre à trois questions :

- 1 Où se trouve le problème ?
- 2 Quel type de problème est-ce ?
- 3 Comment le corriger ?

Ce processus s'appelle le **débogage** (*debugging*).

Trois grandes familles d'erreurs

- **Erreur de syntaxe** : Python ne comprend pas la manière dont le programme est écrit.
- **Erreur à l'exécution** : le programme démarre, mais une instruction provoque une erreur.
- **Erreur logique** : le programme s'exécute, mais le résultat obtenu n'est pas celui attendu.

Erreur de syntaxe

1

```
print("Bonjour"
```

Il manque une parenthèse.

Python affiche un message d'erreur qui permet généralement de localiser le problème.

Erreur à l'exécution

```
1 age = "20"  
2 print(age + 1)
```

Le programme tente ici d'additionner une chaîne de caractères et un entier.

Cela produit une erreur de type :

TypeError

Erreur logique

```
1 prix = 100
2 reduction = 20
3
4 prix_final = prix - reduction / 100
5 print(prix_final)
```

Le programme est syntaxiquement valide et s'exécute.

Mais le calcul n'est pas celui que nous voulions effectuer.

Variables

Une variable est un nom associé à une valeur.

```
1 first_name = "Revekka"  
2 age = 25
```

On peut ensuite réutiliser ces valeurs :

```
1 print(first_name)  
2 print(age)
```

Pourquoi utiliser des variables ?

Les variables permettent de :

- mémoriser des informations ;
- réutiliser des valeurs ;
- effectuer des calculs ;
- rendre le programme plus général ;
- donner du sens aux données.

Typage des variables en Python

En Python, il n'est pas nécessaire de déclarer explicitement le type d'une variable.

```
1 age = 25
2 name = "Alice"
3 weight = 59.5
```

Python détermine automatiquement le type de chaque valeur :

- age est de type `int`,
- name est de type `str`,
- weight est de type `float`.

On parle de **typage dynamique**.

Quelques types fondamentaux

Pour commencer, nous utiliserons surtout :

- `int` : nombres entiers ;
- `float` : nombres à virgule ;
- `str` : chaînes de caractères ;
- `bool` : valeurs `True` ou `False`.

Les listes, tuples, dictionnaires et ensembles seront introduits plus tard.

Exemples de types

```
1 age = 25
2 weight = 59.5
3 name = "Alice"
4 is_student = True
```

Pour connaître le type d'une valeur :

```
1 print(type(age))
2 print(type(name))
```

Une variable peut référencer des valeurs de types différents

En Python, une variable n'est pas liée définitivement à un type.

```
1 x = 10
2 print(type(x))      # int
3
4 x = "bonjour"
5 print(type(x))      # str
```

Ici, le nom `x` fait d'abord référence à un entier, puis à une chaîne de caractères.



À retenir

En Python, le type est associé à la **valeur**. Une variable peut donc faire référence successivement à des valeurs de types différents.

Annotations de type

Python permet d'ajouter des **annotations de type** :

```
1 age: int = 25
2 weight: float = 59.5
3 name: str = "Alice"
```



Ces annotations indiquent le type attendu, mais Python reste un langage à typage dynamique. Ils servent à indiquer explicitement le type attendu des variables, des paramètres et des valeurs de retour d'une fonction, améliorant ainsi la fiabilité et la clarté du code.



Le moteur d'exécution Python n'applique pas les annotations de type pour les fonctions et les variables. Elles peuvent être utilisées par des outils tiers tels que les contrôleurs de type, les IDE, les analyseurs de code, etc.

Opérations arithmétiques

```
1 a = 10
2 b = 3
3
4 print(a + b)      # addition
5 print(a - b)      # soustraction
6 print(a * b)      # multiplication
7 print(a / b)      # division
8 print(a // b)     # division entiere
9 print(a % b)      # reste
10 print(a ** b)    # puissance
```

Expressions

Une expression combine des valeurs, des variables et des opérateurs.

```
1 length = 5
2 width = 3
3
4 area = length * width
5 print(area)
```

?

Quelle valeur contient la variable area ?

Opérations sur les chaînes

On peut également manipuler du texte.

```
1 first_name = "Alice"  
2 last_name = "Martin"  
3  
4 full_name = first_name + " " + last_name  
5 print(full_name)
```

L'opérateur + concatène ici des chaînes de caractères.

Afficher avec print()

La fonction `print()` permet d'afficher une valeur.

```
1 x = 10
2 print("La valeur de x est :", x)
```

Elle peut afficher :

- du texte ;
- des nombres ;
- des variables ;
- le résultat d'une expression.

Lire une valeur avec input()

La fonction `input()` permet de demander une valeur à l'utilisateur.

```
1 name = input("Entrez votre nom : ")  
2 print("Bonjour", name)
```



`input()` renvoie toujours une chaîne de caractères (`str`).

Conversion de type

Si nous voulons effectuer un calcul, il peut être nécessaire de convertir la valeur obtenue avec `input()`.

```
1 age = int(input("Quel age avez-vous ? "))
2
3 next_age = age + 1
4
5 print("L'annee prochaine, vous aurez",
6       next_age, "ans.")
```

Conversions courantes

```
1 x = int("42")
2 y = float("3.14")
3 z = str(25)
4 b = bool(1)
```

- `int()` : convertir vers un entier ;
- `float()` : convertir vers un nombre à virgule ;
- `str()` : convertir vers du texte ;
- `bool()` : convertir vers un booléen.

Exemple : aire d'un rectangle

```
1 length = float(input("Longueur : "))
2 width = float(input("Largeur : "))
3
4 area = length * width
5
6 print("Aire =", area)
```

Ce programme utilise maintenant :

- des variables ;
- des types ;
- `input()` ;
- une conversion ;
- une opération ;
- `print()`.

Exercice

Écrire un programme qui demande :

- le prénom ;
- l'année de naissance ;

puis affiche un message contenant ces informations.

Extension : calculer un âge approximatif à partir de l'année de naissance.