

Cours 2

INTRODUCTION À LA PROGRAMMATION ET LES BASES DE PYTHON

Contenu du cours

Algorithme et programme

- Qu'est-ce qu'un programme informatique ?
- Les composantes de base d'un programme
- Les expressions et leur évaluation
- La structure générale d'un programme

Les bases du langage Python

- Syntaxe de base de Python
- Quelques fonctions utiles

Algorithme et programme

"un ensemble d'étapes qui permettent d'accomplir une tâche" (Cormen, 2013, p. 1)

Les algorithmes existent depuis très longtemps, avant même que l'idée des ordinateurs n'existe. Un des premiers algorithmes élaborés est l'anthyphérèse d'Euclide – aussi appelé l'algorithme d'Euclide – qui permet de déterminer le plus grand diviseur commun entre deux nombres naturels non nuls (PGCD).

Les algorithmes sont un peu comme une recette de cuisine, c'est l'ensemble des instructions qui forme la procédure à suivre pour arriver au résultat final.

Exemples d'algorithmes:

- « Comment faire un gâteau au chocolat ? »
- « Comment assembler les pièces séparées en un meuble ? »
- « *Est-ce que N est un nombre pair ?* »

Les algorithmes et l'informatique

Dans le cadre de l'informatique, l'algorithmique est appliquée à la programmation des ordinateurs. On veut pouvoir définir les étapes nécessaires à la résolution d'un problème pour que l'ordinateur puisse exécuter ces étapes et nous fournir le résultat.

Tous les programmes informatiques sont des constructions d'algorithmes. La maîtrise de la pensée algorithmique est un prérequis pour l'apprentissage de la programmation.

Pour la construction d'un programme informatique, les opérations d'un algorithme sont décomposées jusqu'à ce qu'on puisse les arrimer aux **instructions élémentaires** d'un langage permettant de codifier des algorithmes pour ordinateurs. C'est ce qu'on appelle **la programmation**.

Exemple d'algorithme et de programme

L'aire d'un quadrilatère (Pseudocode)

hauteur $\leftarrow$ valeur entrée par l'utilisateur

largeur $\leftarrow$ valeur entrée par l'utilisateur

aire $\leftarrow$ hauteur * largeur

afficher aire

Exemple d'algorithme et de programme

L'aire d'un quadrilatère (en code Python)

```
hauteur = float(input("Entrer la largeur: "))
```

```
largeur = float(input("Entrer la hauteur: "))
```

```
aire = hauteur * largeur
```

```
print(f"L'aire du quadrilatère est: {aire}")
```

Qu'est-ce qu'un programme informatique ?

Un programme informatique est une suite d'instructions écrites dans un langage de programmation, destinée à être exécutée par un ordinateur pour accomplir **une tâche spécifique**. Ces instructions sont souvent appelées « code ».

Exemples de programmes informatiques communs

- Un programme qui affiche "Bonjour" à l'écran.
- Un programme qui calcule la moyenne de trois nombres.
- Un programme qui trie une liste de nombres du plus petit au plus grand.
- Un programme qui simule des systèmes physiques.
- Un programme qui fait le chiffrement des données (cryptage)

Les composants de base d'un programme

- Les instructions
- Les variables
- Les données textuelles
- Les opérateurs

Les instructions

Les instructions sont les commandes élémentaires que l'ordinateur exécute. Elles peuvent inclure des opérations arithmétiques, la manipulation de données, ou encore l'affichage de résultats à l'utilisateur.

Exemple:

```
print("Bonjour, monde!")
```

Cette instruction demande à l'ordinateur d'**afficher** le texte "**Bonjour, monde!**" sur l'écran.

Les variables

Les variables sont des conteneurs qui stockent des données que vous pouvez manipuler tout au long de votre programme. Une variable a un nom, qui permet de l'identifier, et une valeur, qui est l'information qu'elle contient. Pour modifier le contenu d'une variable, on utilise l'opérateur d'affectation « = »

Exemples en Python

```
x = 10
```

```
y = 20
```

```
somme = x + y
```

Les données textuelles

L'exemple précédent nous montre des variables qui contiennent des données de type numérique. Il arrivera souvent que l'on doive manipuler des données qui n'ont aucune valeur numérique, mais qui sont en fait du texte.

On peut reconnaître les données textuelles dans un programme par le fait qu'elles sont entourées de guillemets.

```
nom = "Lambert"
```

```
prenom = "Hippolyte"
```

```
nom_complet = prenom + nom
```

```
#resultat: "HippolyteLambert"
```

Les opérateurs de base

Les opérateurs sont utilisés pour réaliser des opérations sur les variables et les valeurs. Les plus simples et intuitifs à comprendre sont certainement les opérateurs arithmétiques: +, -, *, /, //.

```
a = 11
```

```
b = 5
```

```
somme = a + b
```

```
# addition (résultat: 16)
```

```
produit = a * b
```

```
# multiplication (résultat: 55)
```

```
quotient_entier = a // b
```

```
# division entière (résultat: 2)
```

```
quotient_reel = a / b
```

```
# division réelle (résultat 2.2)
```

Expressions

Les instructions d'un programme sont constituées de plusieurs expressions. En termes informatiques on peut définir une expression comme suit:

« une expression est un élément de syntaxe qui combine un ensemble de lexèmes retournant une valeur. C'est une combinaison de littéraux, de variables, d'opérateurs, et de fonctions qui est évaluée (ou calculée) en suivant les règles de priorité et d'associativité du langage de programmation pour produire (ou retourner) une nouvelle valeur. »¹

¹[https://fr.wikipedia.org/wiki/Expression_\(informatique\)](https://fr.wikipedia.org/wiki/Expression_(informatique))

Évaluation d'une expression

L'évaluation des expressions est essentielle dans l'exécution des programmes. Plus particulièrement, l'exécution de chaque instruction amène l'évaluation des expressions qui la composent. Par exemple:

```
x = 5
```

Pour savoir quelle valeur doit être affectée à la variable `x`, on doit déterminer la valeur de l'expression qui est à droite du symbole d'affectation (« = »). Dans l'exemple précédent, l'expression est simple à évaluer: il s'agit de la valeur entière 5. C'est donc cette valeur qui sera affectée à la variable `x`.

Les instructions des programmes sont souvent beaucoup plus complexes. Regardons par exemple l'instruction suivante, qui calcule l'aire d'un rectangle:

```
aire = largeur * hauteur
```

Cette instruction diffère de la précédente puisqu'elle utilise deux variables du programme: `largeur` et `hauteur`. Ainsi, avant de pouvoir effectuer la multiplication, l'ordinateur doit d'abord consulter l'espace-mémoire associé à chaque variable pour connaître la valeur numérique à utiliser dans le calcul. On a donc la forme intermédiaire suivante pour l'évaluation de l'expression:

```
    5      10  
aire = largeur * hauteur
```

```
aire = 50
```

Évaluation d'une expression

Lorsque plusieurs opérateurs mathématiques sont impliqués dans une expression, leur évaluation suit les règles de priorité et d'associativité habituelles: les parenthèses ont priorité sur tout, les multiplications et divisions ont priorité sur les additions et les soustractions. Lorsque 2 opérateurs ont la même priorité, ils sont évalués de gauche à droite

Exercices

Énumérez toutes les expressions dans ce qui suit:

1. « Coucou »
2. $y_2 - y_1 / x_2 - x_1$
3. $b * b - 4 * a * c$
4. $i = i + 1$

Exercices

Évaluez toutes les expressions dans ce qui suit:

1. « Coucou »
2. $y_2 - y_1 / x_2 - x_1$ (sachant que $y_1 = 5$, $y_2 = 10$, $x_1 = 20$, $x_2 = 32$)
3. $b * b - 4 * a * c$ (sachant que $a = 2$, $b = 6$, $c = 4$)
4. $i = i + 1$ (sachant que $i = 11$)

Évaluation d'une expression

Le résultat d'une opération a toujours un type, on doit donc s'assurer que les expressions produisent un résultat qui est compatible avec la variable à qui récupère ledit résultat.

Exercices

Déterminez le type du résultat de l'évaluation de chacune des expressions globales qui suivent:

1. « Coucou »
2. $y2 - y1 / x2 - x1$ (sachant que $y1 = 5$, $y2 = 10$, $x1 = 20$, $x2 = 32$)
3. $b * b - 4 * a * c$ (sachant que $a = 2$, $b = 3$, $c = 4$)

Structure générale d'un programme

Un programme bien structuré suit généralement une séquence logique et cohérente :

1. **Initialisation** : Initialisation des variables, obtention des entrées auprès de l'utilisateur, etc...
2. **Traitement** : Exécution des instructions principales du programme
3. **Sortie** : Affichage des résultats ou enregistrement de données en fonction du but du programme.

```
# 1. Initialisation
nom = input("Entrez votre nom: ")

# 2. Traitement
message = "Bonjour " + nom

# 3. Sortie
print(message)
```

Ce programme demande à l'utilisateur de saisir son nom, puis lui affiche un message de salutation.

Conclusion

En résumé, l'algorithmique constitue le socle sur lequel repose l'ensemble de la programmation informatique.

À travers des siècles d'évolution, les algorithmes ont façonné la manière dont nous abordons les problèmes complexes et structurons les solutions. De l'anthyphérèse d'Euclide à nos applications modernes, les algorithmes sont au cœur de la capacité des ordinateurs à réaliser des tâches variées et sophistiquées.

Comprendre et maîtriser l'art de l'algorithmique est donc essentiel pour tout programmeur, car c'est cette compréhension qui permettra de concevoir des programmes efficaces et robustes, capables de répondre aux besoins variés du monde numérique actuel.

Les bases de Python

LA SYNTAXE ET EXEMPLES

Introduction

Python est un langage de programmation très populaire en raison de sa simplicité et de sa lisibilité. Il est utilisé dans de nombreux domaines, y compris les sciences, l'ingénierie, et l'analyse de données.

Dans cette section, nous allons explorer les éléments de base qui vous permettront de créer vos premiers programmes en Python.



Python vs les autres langages

Simplicité et lisibilité : La syntaxe de Python est claire et proche du langage humain.

Communauté et ressources : Il existe une vaste communauté d'utilisateurs, ce qui signifie que vous pouvez trouver beaucoup de documentation et de support.

Polyvalence : Python est utilisé dans une variété de domaines, des simples scripts aux projets complexes d'intelligence artificielle.

Syntaxe de base de Python

Python utilise une syntaxe qui est simple et directe. Voici quelques éléments fondamentaux.

- Les commentaires
- Les variables
- Les types de données
- Les opérations de base
- Quelques fonctions utiles

Les commentaires

Un commentaire est une ligne de code qui ne sera pas exécutée : elle n'est là que pour **documenter** votre programme à l'intérieur du fichier de code.

Une ligne de commentaire débute par le caractère **#**

```
# La ligne suivante assigne une valeur par défaut à la variable 'a'  
a = 1  
  
# La ligne suivante ne sera pas exécutée, car elle débute par un #  
# a = 53
```

Les variables

Les variables en Python servent à stocker des informations que vous pouvez réutiliser plus tard dans votre programme.

```
# Déclaration de variables
# Ici, x est une variable qui stocke le nombre entier 5.
x = 5

# texteBonjour est une variable qui stocke une chaîne de caractères,
# c'est-à-dire du texte.
texte_bonjour = "Bonjour"
```

Les types de données

Les variables en Python servent à stocker des informations que vous pouvez réutiliser plus tard dans votre programme.

- **int** (entier) : 5, 100, -20
- **float** (nombre à virgule flottante) : 5.0, 3.14159265, -0.01
- **str** (chaîne de caractères) : "Bonjour", 'Python'
- **bool** (booléen) : True, False

Indications de types

Dans un programme Python, on peut « donner » un type à chacune des variables, lors de leur première utilisation (déclaration):

- pour une variable numérique entière, on utilise le type *int*:

```
annee : int = 2026
```

- pour une variable numérique réelle, on utilise le type *float*:

```
pi : float = 3.14159265
```

- Pour une chaîne de caractère, on utilise le type *str*:

```
message : str = « Bonjour »
```

Les opérations de base

Vous pouvez effectuer des opérations arithmétiques simples avec Python.

```
# Addition
```

```
a : int = 5 + 3
```

```
# Soustraction
```

```
b : int = 10 - 2
```

```
# Multiplication
```

```
c : float = 4.2 * 7.39
```

```
# Division entière
```

```
d : int = 20 // 4
```

Quelques fonctions utiles

Pour afficher un message à l'écran, il suffit d'utiliser la fonction **print(message)**.

```
# Afficher un message  
print("Bonjour, monde!")
```

Vous pouvez également écrire un programme qui demande une entrée à l'utilisateur à l'aide de la fonction **input(message)**.

```
# Demander le nom de l'utilisateur  
nom : str = input("Quel est votre nom? ")  
  
# Afficher un message personnalisé  
print("Bonjour, " + nom + "!")
```

Les chaînes formatées

Il peut parfois être utile d'insérer des valeurs qui sont calculées par nos programmes à l'intérieur des messages que l'on fait afficher à l'écran.

Par exemple, après avoir déterminé l'âge de l'utilisateur, on pourrait vouloir afficher un message qui ressemble à: "Vous avez 27 ans", où le nombre 27 est contenu dans une variable du programme.

Les chaînes formatées permettent ce genre de manipulations. On reconnaît une telle chaîne par le fait qu'il y a une lettre 'f' devant le premier guillemet qui délimite la chaîne:

```
# l'âge de l'utilisateur est contenu dans la variable age
age : int = 27

# On affiche la chaîne à l'écran
print(f"Vous avez {age} ans")
```

Transformer les données d'un type

Comme mentionné précédemment, l'utilisation de la fonction **input()** permet à l'utilisateur d'entrer des données afin qu'elles puissent être utilisées dans les calculs réalisés par le programme. Cependant, on doit également porter attention au fait que la fonction **input()** recueille ce que l'utilisateur entre au clavier sous forme de donnée textuelle!

```
# Le code ci-dessous produira une erreur lors de l'exécution!  
donnee : str = input("Veuillez entrer un nombre entre 0 et 15")  
total = donnee + 10
```

La variable « **donnee** » contient une valeur textuelle, et que l'on tente de **l'additionner** à la **valeur numérique 10**.

Transformer les données d'un type

Pour pallier à ce problème, on peut utiliser les fonctions suivantes pour transformer la valeur textuelle en valeur numérique!

int() - construit un nombre entier à partir d'une valeur de type **int**, **float** ou **str** (à condition que la chaîne représente un nombre entier)

float() - construit un nombre flottant à partir d'une valeur de type **int**, **float**, ou **str** (à condition que la chaîne représente un nombre flottant ou un entier)

str() - construit une chaîne de caractères à partir d'une grande variété de types de données, y compris les **str**, **int** et **float**

```
donnee : int = int(input("Veuillez entrer un nombre entre 0 et 15"))  
total = donnee + 10
```

Développement graduel

Le développement graduel est une méthode de travail qui vise à faciliter la vie du programmeur et de la programmeuse dans la mise en place de son programme.

L'idée générale consiste simplement à commencer tout d'abord à déterminer les grandes étapes du programme et à les décrire en des termes très vagues.

On s'attarde par la suite à rendre la description de chaque étape de plus en plus précise, jusqu'à ce que l'on obtienne quelque chose qui ressemble à une instruction pouvant être exécutée par un ordinateur.

Exemple: calculer pente entre 2 points

Exercice: calculer le sinus d'un angle à partir de la longueur des côtés

Laboratoire #1
