LSIN200A - Fondements de l'informatique 1

Coline Gianfrotta

coline.gianfrotta@ens.uvsq.fr

5 février 2026

Déroulement du cours

- Cours: jeudi 11h20 - 12h50 salle Jungle (bât. Descartes)
- TD: jeudi 13h50 - 17h00 salle Jungle (bât. Descartes)

Environnement de programmation

Vous devez venir avec un **ordinateur** pour chaque séance (ordinateur personnel ou cartable numérique)

Pour programmer il faut installer des logiciels:

- *Un interpréteur* : **python3** (pour exécuter vos programmes)
- *Un éditeur de texte* : **Visual Studio Code** (pour écrire vos programmes)
- *Des bibliothèques* : numpy, matplotlib, jupyter (pour enrichir vos programmes)

La procédure d'installation vous sera fournie d'ici la semaine prochaine.

Evaluations

Ce qui est attendu pour cette UE :

- Savoir lire du code
- Comprendre l'exécution de programmes simples et imbriqués
- Retranscrire une description d'actions du langage courant en langage Python
- Etre capable, face à un problème simple, de le résoudre en utilisant la programmation

Premières définitions

Un algorithme

Suite finie d'instructions et d'opérations permettant de résoudre un problème.

Pour donner un algorithme:

- Input: les données en entrée du problème
- Output: la solution du problème
- les étapes qu'on doit effectuer pour résoudre le problème

Exemple: Vérifier si un mot est un palindrome

Input: un mot avec N caractères
Output: OUI ou NON

Commencer à la position i=1
Tant que i <= N/2

> Vérifier que le caractère à la position i du mot coïncide avec le caractère à la position N-i+1.
>
> Si ce n'est pas le cas, renvoyer NON
>
> Passer à la position suivante

Renvoyer OUI

Premières définitions

Un programme

Ensemble d'instructions et d'opérations destinées à être exécutées par un ordinateur

Pour écrire un programme, il faut utiliser un *langage de programmation*. Par exemple, Python.

Programme en Python pour détecter un palindrome:

```
mot = "Bonjour"
longueur = len(mot)
is_palindrome = True
for i in range(longueur // 2) ## on parcourt jusqu'à la moitié
    if mot[i] != mot[longueur - 1 - i]:
        ◦ is_palindrome = False # dès qu'une différence est
          trouvée
print(is_palindrome)
```

A-t-on besoin de tout cela ?

Dans la vie de tous les jours :

- les applications web : banque, courses en ligne etc.
- les jeux video
- les applications mobiles : Calculatrice, WhatsApp etc
- l'intelligence artificielle : ChatGPT

Une multitude de langages de programmation selon le besoin informatique auquel on doit répondre :

- C, C++, Java, C#, JavaScript, PHP, Python, Kotlin etc.

Python

- Créé par Guido van Russom (Pays-Bas)
- Première version de Python en 1991
- Parmi les caractéristiques du langage :
  - multiplateforme : Windows, Linux, MacOs, Android...
  - gratuit et open-source
  - langage interprété : les instructions d'un programme sont exécutées ligne par ligne par un interpréteur

Les variables

Variable

élément de mémoire qui stocke une valeur susceptible d'être modifiée.
Une variable est identifiée par un nom

```
x = 2 # affectation d'une valeur à une variable
y = x # affectation de la valeur d'une variable à une autre
# variable
print(x) # affichage de la valeur d'une variable
print(y)
y = 3
print(x,y)
```

- 2 et 3 sont des objets de type entier
- les variables x et y sont des identifiants qui font référence à l'un de ces objets
- règle de style: on met un espace après une virgule mais pas avant

Echanger deux variables

Méthode fausse

```
x = 2
y = 3
x = y
y = x
print(x, y)
```

Première méthode

```
x = 2
y = 3
z = x
x = y
y = z
print(x, y)
```

Deuxième méthode

```
# plus pythonesque
# utilisation de tuples
x = 2
y = 3
x, y = y, x
print(x, y)
```

Règles de nommage des variables

- Caractères autorisés:
  - caractère alphabétique (A à Z) majuscule ou minuscule
  - chiffres (0 à 9)
  - caractère souligné _ (touche 8 du clavier)
  - c'est tout ! pas d'espace, pas de caractères avec accents etc...
- Le nom ne doit pas commencer par un chiffre
- Python est sensible à la casse: les variables toto et toTo sont différentes
- Ne pas utiliser un mot réservé du langage comme print et toutes les fonctions natives que nous verrons

Règles de nommage des variables

Bonnes pratiques:

- sauf exceptions, donner un nom explicite aux variables tel que `nb_de_vie`
- suivre cet exemple, c'est-à-dire séparer les mots par des _ sans majuscule
- éviter d'autres formes telles que `NbDeVie`, ou au moins être cohérent dans tout le programme

Types de données

Le type d'une variable détermine:

- les opérations qu'on peut lui appliquer
- les valeurs qu'elle peut prendre

Exemples

```
x = 2
y = 3
z = x + y
x = "Bonjour tout le monde"
```

En Python, le typage est **dynamique** et **implicite**:

- le type de la variable est déterminé lors de son affectation
- le type de la variable peut changer au cours du temps

Types de données

Les types prédéfinis que l'on va manipuler au début sont:

- les nombres entiers, flottants...
- le type booléen
- les chaînes de caractères

```
a = 5
print(a, type(a))
a = 3.14
print(type(a))
print(type("hello world"))
```

Les nombres

- entiers: de précision arbitraire
- flottants: utilise le point comme séparateur décimal

Exemples

```
x = 1
y = 2.0
z = 2e+20
```

- Sur les nombres, on utilise les opérateurs arithmétiques usuels: $+$, $-$, $*$, etc..
- L'utilisation de flottants et d'entiers dans une expression numérique donne une valeur flottante

Exemples

```
print(type(5 + 2.0))
print(type(7 * 1.0))
```

Opérateurs sur les nombres

- L'opérateur / effectue une division flottante. Il renvoie un flottant même si le numérateur et le dénominateur sont des entiers
- L'opérateur // effectue une division entière. On arrondit par l'entier le plus proche.
- L'opérateur ** effectue l'exponentiation à la puissance
- L'opérateur % donne le reste d'une division

Qu'affiche-t-on ?

```
a = 11/2
print(a)
b = 11//2
print(b)
c = 10**10
print(c)
d = 11%2
print(d)
```

```
print(type(a+b))
print(type(a+c)
```

Opérateurs d'affectation augmentés

Ajouter la valeur de b à la variable a:

```
a, b = 2, 3
a += b # identique à a = a+b
print(a)
```

Incrémenter (ajouter 1 à) une variable:

```
a += 1
print(a)
```

Autres opérateurs: $-=$, $*=$, etc...

Conversion de types

Fonctions de conversion (utilisable quand cela a un sens):

- int: conversion en entier
- float: converson en flottant

Exemples

```
x = 1 # int
y = 3.14 # float
#conversion de int ‘a
float:
a = float(x)
#conversion de float ‘a
int:
b = int(y)
```

```
print(type(a))
print(type(b))
print(a)
print(b)
```

Qu’affiche-t-on ?

Représentation des flottants

- En Python, représentés suivant le standard IEEE 754
- Sur machine, les flottants sont représentés par des fractions en base 2.
- La précision est limitée, alors la plupart du temps on manipule une approximation

```
print(float(10**309))
(0.1).as_integer_ratio()
print(0.1+0.2)
3 * 0.1 == 0.3
```

Type booléen

- Utile pour les instructions de contrôle d'un programme.
- Deux valeurs : `False` et `True`.
- Opérateurs logiques : `and, or, not`
- Opérateurs de comparaison :
  - $>$, $<$
  - $>=$ supérieur ou égal, $<=$ inférieur ou égal
  - $==$ test d' égalité
  - $!=$ test de non égalité

Exemples

```
a = (5 < 2)
print(a, type(a))
```

Les opérateurs logiques

a	not a
True	False
False	True

a	b	a and b
True	True	True
True	False	False
False	True	False
False	False	False

a	b	a or b
True	True	True
True	False	True
False	True	True
False	False	False

- L'opérateur `not` fait passer une valeur de `False` à `True`, ou inversement.
- L'opérateur `and` retourne `True` si et seulement si les deux opérandes sont vrais.
- L'opérateur `or` retourne `True` si au moins l'un des opérandes est vrai.

Type booléen

Quels opérateurs dans l'exemple suivant ?

Qu'affiche-t-on ?

Exemple

```
a = -0.5
res = (a <= 0) or (a > 1 and a < 3 and a != 2)
print(res)
```

Chaînes de caractères (String)

- Suite de caractères entre guillemets (simple ou double)
- Des opérations sur les chaînes de caractères sont fournies nativement

```
s1 = "hello"
s2 = 'world'
print(s1, s2)
print(s1 + s2)
print(len(s1))
print(s1 > s2)
print(s1 in s2)
print(s1.upper())
str(3)+str(5)
```

Quelle est la signification de ces opérations ?

Concaténation de chaînes de caractères

- L'opérateur + fait la concaténation de deux chaînes de caractères.
- Attention à ne pas mélanger les types !
- Vous noterez le caractère retour à la ligne `\n`.

```
"hello" + "3" # et non pas "hello"+3
print("hello" * 3)
print("hello\n"*3)
```

Récupérer une valeur tapée au clavier par un utilisateur

```
s = input("Entrer un texte\n")
print(s)
s = input("entrer un nombre entier\n")
print("le nombre qui le précède est", s - 1)
s = input("entrer un nombre entier\n")
print("le nombre qui le précède est", int(s) - 1)
```

- Avec `input`, on récupère une chaîne de caractères
- Fonctions de conversion (utilisable quand cela a un sens):
  - `int`: conversion en entier
  - `float`: conversion en flottant
  - `str`: conversion en chaînes de caractères