Formation Python

École Secondaire Pierre-Dupuy

Olivier Chabot, 11 Novembre 2021

Plan de la formation

- Pourquoi Python ?
- Faire fonctionner python sur votre machine
- Les blocs de construction d’un programme
- Votre premier jeu

Pourquoi apprendre à programmer avec Python ?

Python

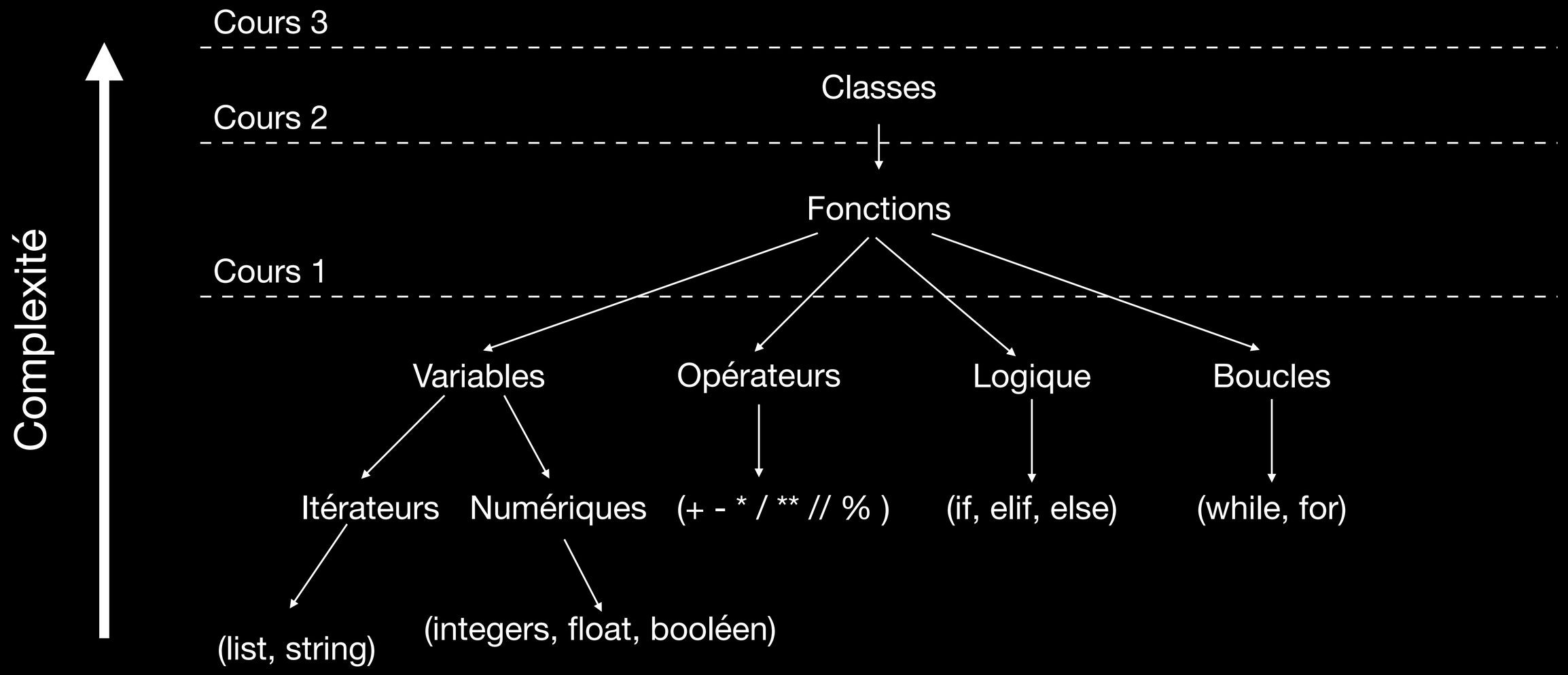
```python
"""
Example de la syntaxe accessible de Python
"""

# Objectif : Créer une liste de nombres pairs

longueur_liste = 5  # Nombre de chiffres dans la liste
premier_nombre = 2   # Premier nombre
dernier_nombre = premier_nombre + longueur_liste * 2
liste_de_nombres_pairs = []

for nombre in range(premier_nombre, dernier_nombre, 2):
    liste_de_nombres_pairs.append(nombre)

print(liste_de_nombres_pairs)
# [2, 4, 6, 8, 10]
```

C++

```cpp
// Exemple de la synthaxe moins accessible de C++
#include <iostream>
const int longueur_liste = 5;
int premier_nombre = 2;
int liste_de_nombres_pairs [longueur_liste];
int dernier_nombre = premier_nombre + longueur_liste * 2;
int nombre = premier_nombre;

int main() {
    // Créer la liste
    for (int i=1 ; i < longueur_liste; i++) {
        liste_de_nombres_pairs[i] = nombre;
        nombre = nombre + 2;
    }
    // Montrer la liste
    for (int i = longueur_liste - 1; i >= 0; i--) {
        std::cout << liste_de_nombres_pairs[i];
    }

    return 0;
}
```

- Syntaxe compréhensible
- Gratuit
- Répandu partout

Les blocs de construction d'un programme



Les *Types* numériques dans Python

Les opérations possibles sur une variable sont définies par son *type*

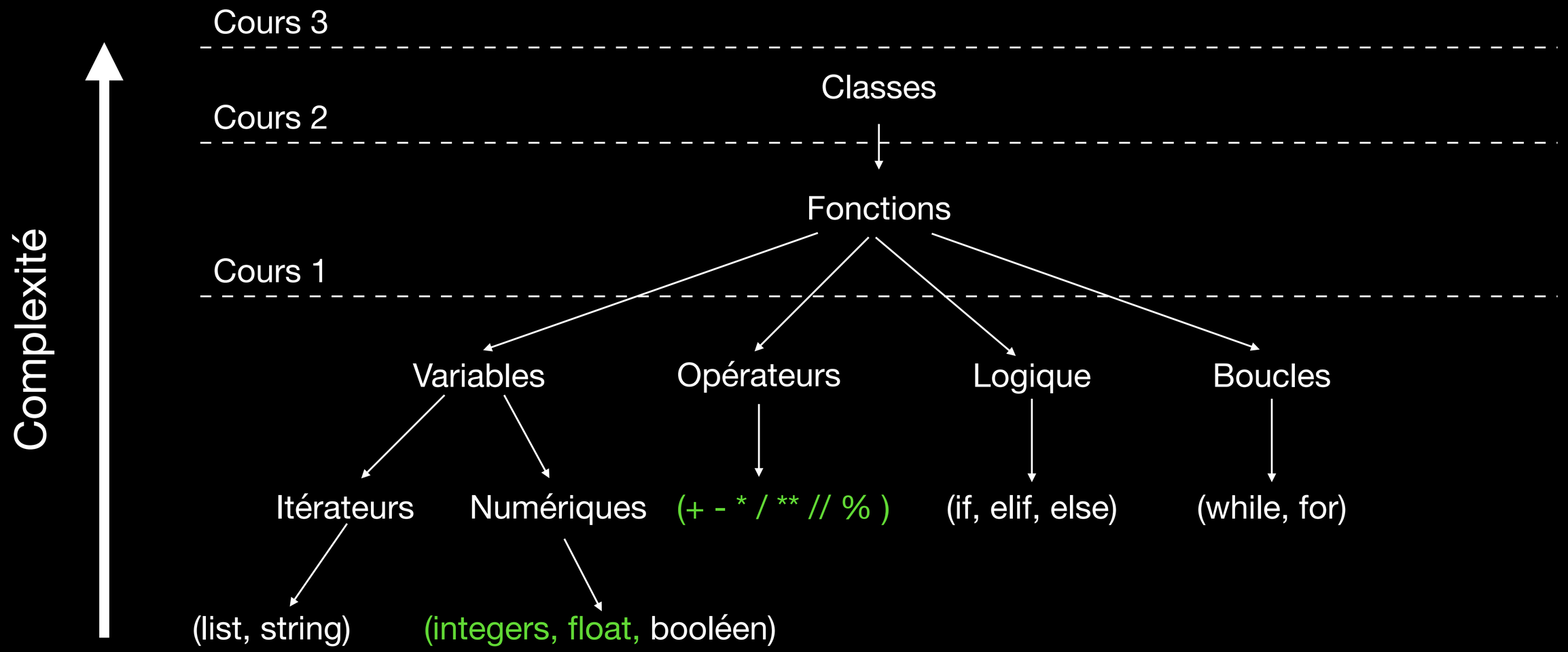
- **int** : représente un nombre entier « integer »
  - ex : `a = 5`, donc a est un **int**
- **float** : représente un nombre avec décimale « floating point number »
  - ex : `b = 3.1415`, donc b est un **float**
- **Exemple :**
  - si `a = 10`, `b = 5` et `c = a/b`, quel est le type de `c` ?

- **Complex** : Les nombres complexes sont aussi représentés dans python avec la syntaxe `n = a + bj`

Survol des différents *opérateurs numériques*

Nom	Opérateur	Exemple	Python
Addition	+	1 + 1 = 2	`1 + 1 = 2`
Soustraction	-	2 - 1 = 1	`2 - 1 = 1`
Multiplication	*	2 x 2 = 4	`2 * 2 = 4`
Division	/	9 / 3 = 3	`9 / 3 = 3.0`
Exposant	**	$2^2 = 4$	`2 ** 2 = 4`
Division entière	//	Combien de fois entre 3 dans 10 : 3	`10 // 3 = 3`
Reste	%	Quel est le reste de la division 7/3 : 1	`7 % 3 = 1`

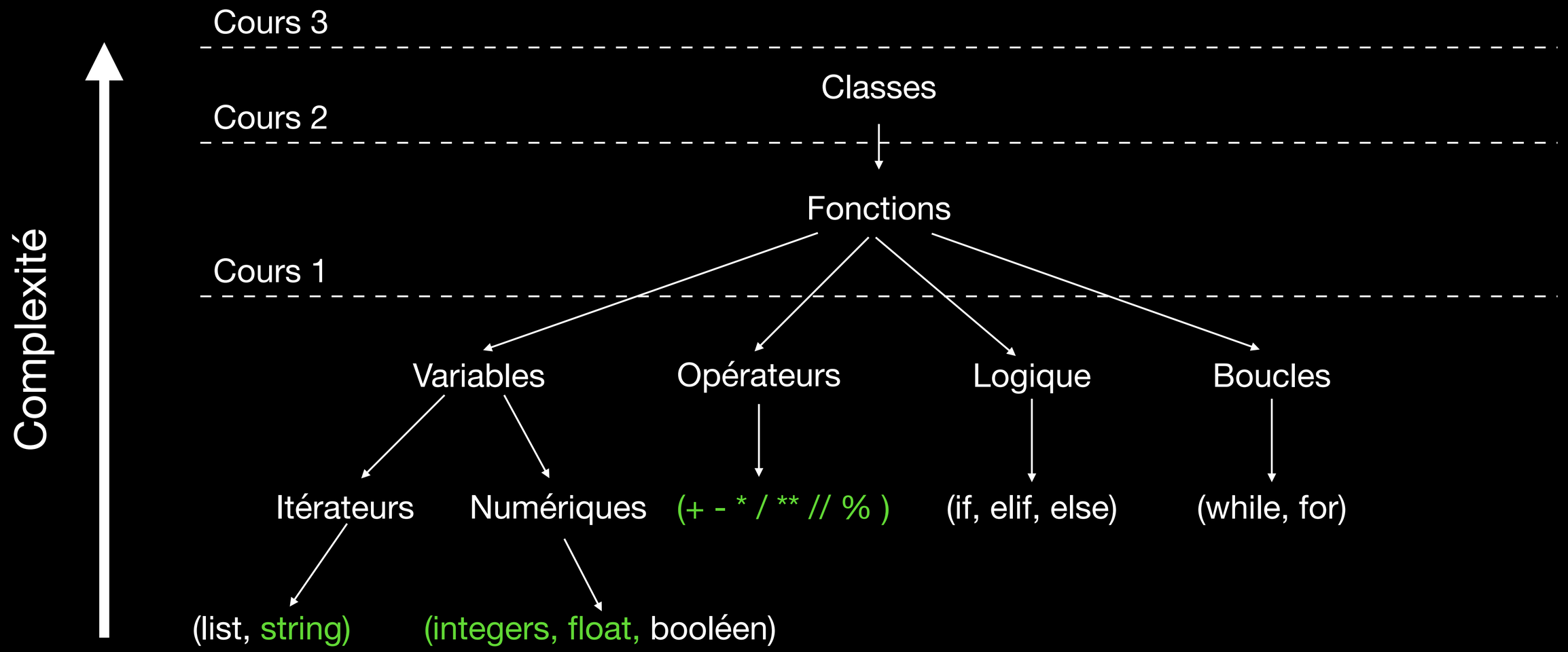
Les blocs de construction d'un programme



Le *Type* chaînes de caractères « string »

- Les guillemets ('' ou "") permettent de créer une chaine de caractère
    - message = ‘Bonjour chers enseignants’, le type de la variables message est *string*
- Certains opérateurs sont compatibles avec les *strings :*
    - Addition : `‘a’ + ‘b’ = ‘ab’`
    - Multiplication : `‘ab’ * 3 = ‘abab’`
- **Exemple 2** : Manipulation de chaines de caractère

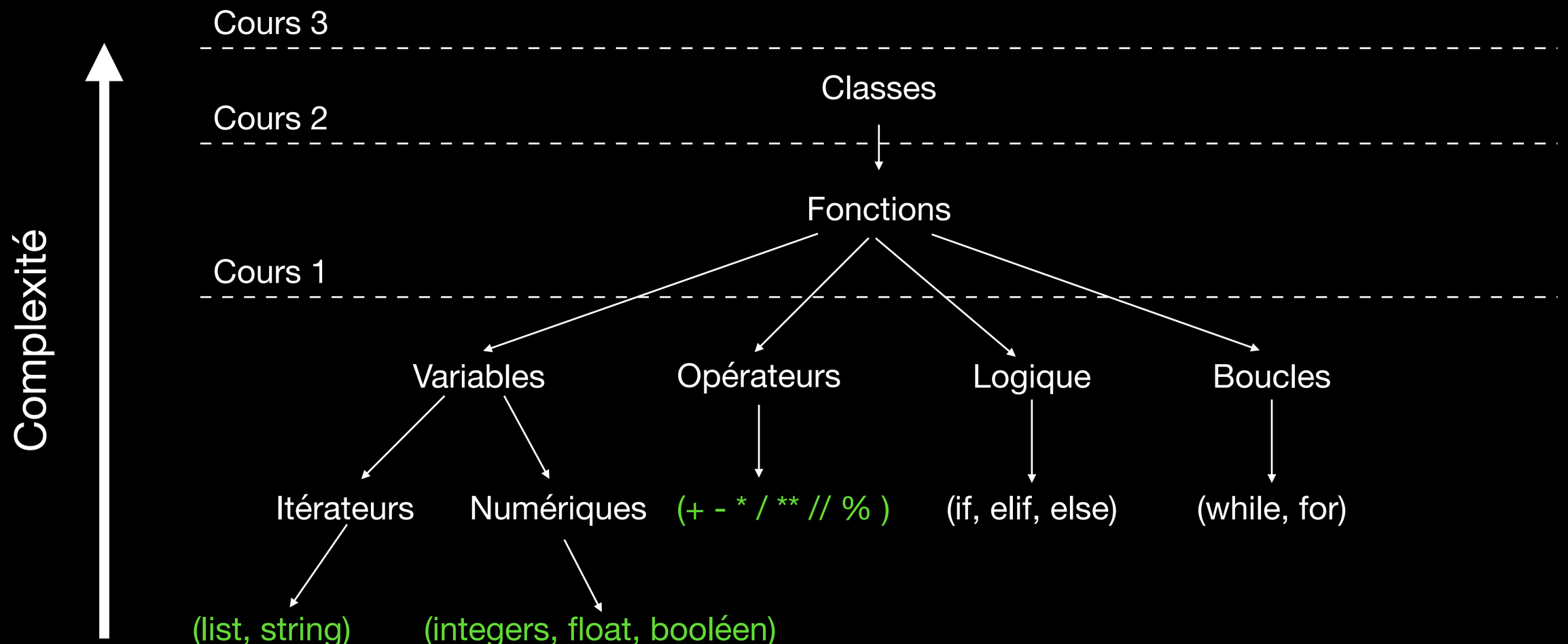
Les blocs de construction d'un programme



Le *Type liste*

- Les parenthèses carrées [ ] permettent de créer une liste :
    - noms = [‘Samuel’, ‘Damien’, ‘Maxime’]
- On peut accéder aux élément de la liste avec des indexes :
    - Les indexes commencent à zéro et vont jusqu’à n-1
    - `nom = noms[0]  -> nom = ‘Samuel’`
    - `nom2 = noms[2] -> nom2 = ‘Maxime’`
- Les opérateurs * et + fonctionnent de la même façon que pour les *strings*
    - `[1] + [2] = [1, 2]` et `[1, 2] * 2 = [1, 2, 1, 2]`
- **Exemple 3 :** Des formes avancés de listes et d’indexes

Les blocs de construction d'un programme



Opérateurs booléens

- Les opérateurs booléens (`<, >, ==, >=, <=, !=`) permettent de comparer des variables
  - Ex : `a = (1 == 1) a = True` Dans ce cas le *type* de `a` est *booléen*

Opérateur	Signification	Exemple Vrai	Exemple Faux
`<`	Plus petit que	`3 < 5 = True`	`5 < 3 = False`
`>`	Plus grand que	`7 > 4 = True`	`10 > 20 = False`
`==`	Égal	`2 == 2 = True`	`2 == 1 = False`
`<=`	Plus petit ou égal	`2 <= 2 = True`	`3 <= 2.9 = False`
`>=`	Plus grand ou égal	`3 >= 3 = True`	`3 >= 4 = False`
`!=`	Pas égal	`2 != 1 = True`	`1 != 1 = False`

- **Example 4 :** Opérateurs booléens

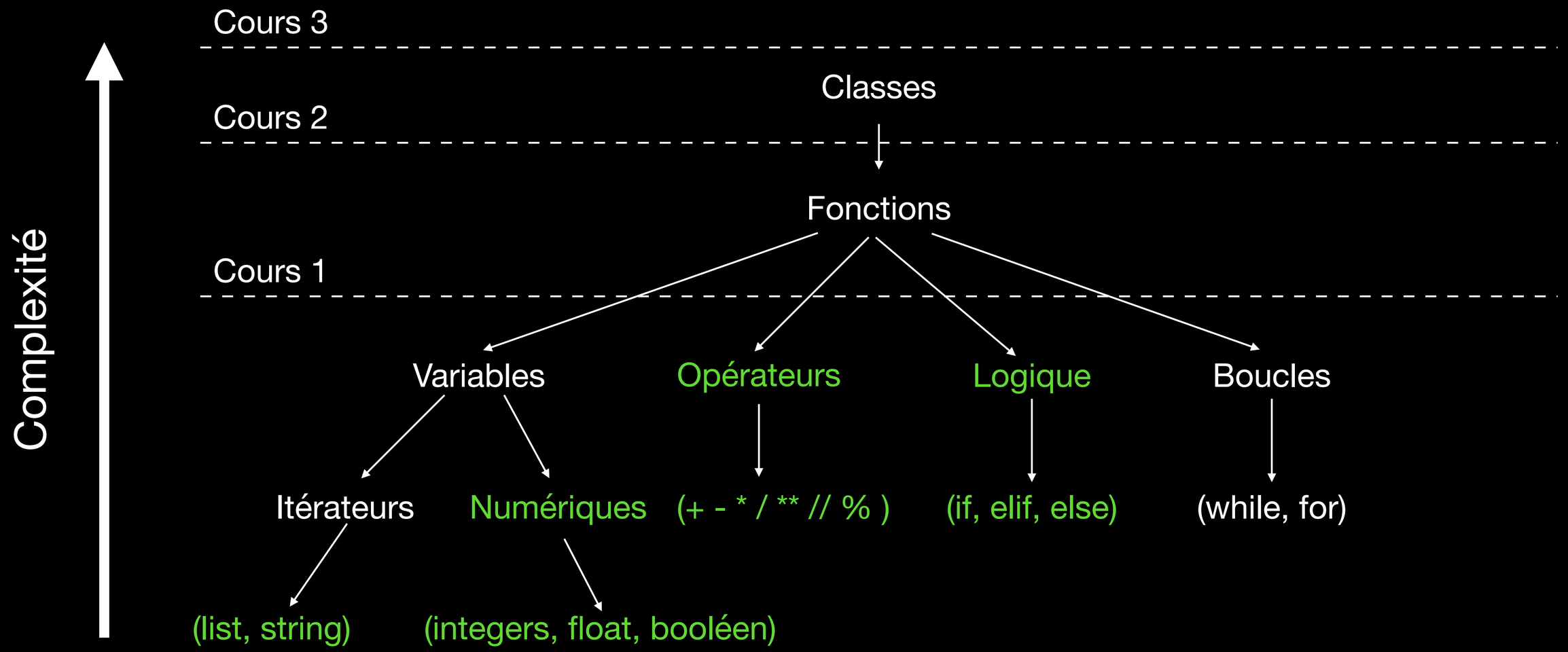
Expressions conditionnelles

- Des décisions peuvent êtres prises lors de l'exécution d'un code avec l'expression (`if, elif, else`)

```python
number = 2

# Je suis évalué en premier
if number == 0:
    # Si la condition est vraie le segment suivant est exécuté et le code termine
    print('Le nombre est zéro')

# Si la première condition n'est pas vraie je suis évalué
elif number % 2 == 0:
    # Si la condition est vraie le segment suivant est exécuté et le code termine
    print('Le nombre est pair')

# Un nombre arbitraire de "elif" peut être utilisé
elif number == 1001:
    print('Le nombre est 1001')

# Si TOUTES Les conditions précédents ne sont pas vraies le code qui suit est exécuté
else:
    print('Le nombre est impair')
```

- **Quizz 1 :** Expressions conditionnelles

Les blocs de construction d'un programme



La boucle *for*

- L’itération : Faire quelque chose un certain nombre de fois un changeant possiblement des variables à chaque fois.
- La boucle *for* permet d’**itérer** sur un **itérateur** (C’est un peu abstrait)
- Ex : On veut prendre une liste de nombres et savoir combien d’entre eux sont pairs
- Dire dans nos mots comment on s’y prendrait :

POUR chaque chiffre DANS la liste : SI le chiffre est pair on compte +1

- **Exemple 5 :** La boucle for

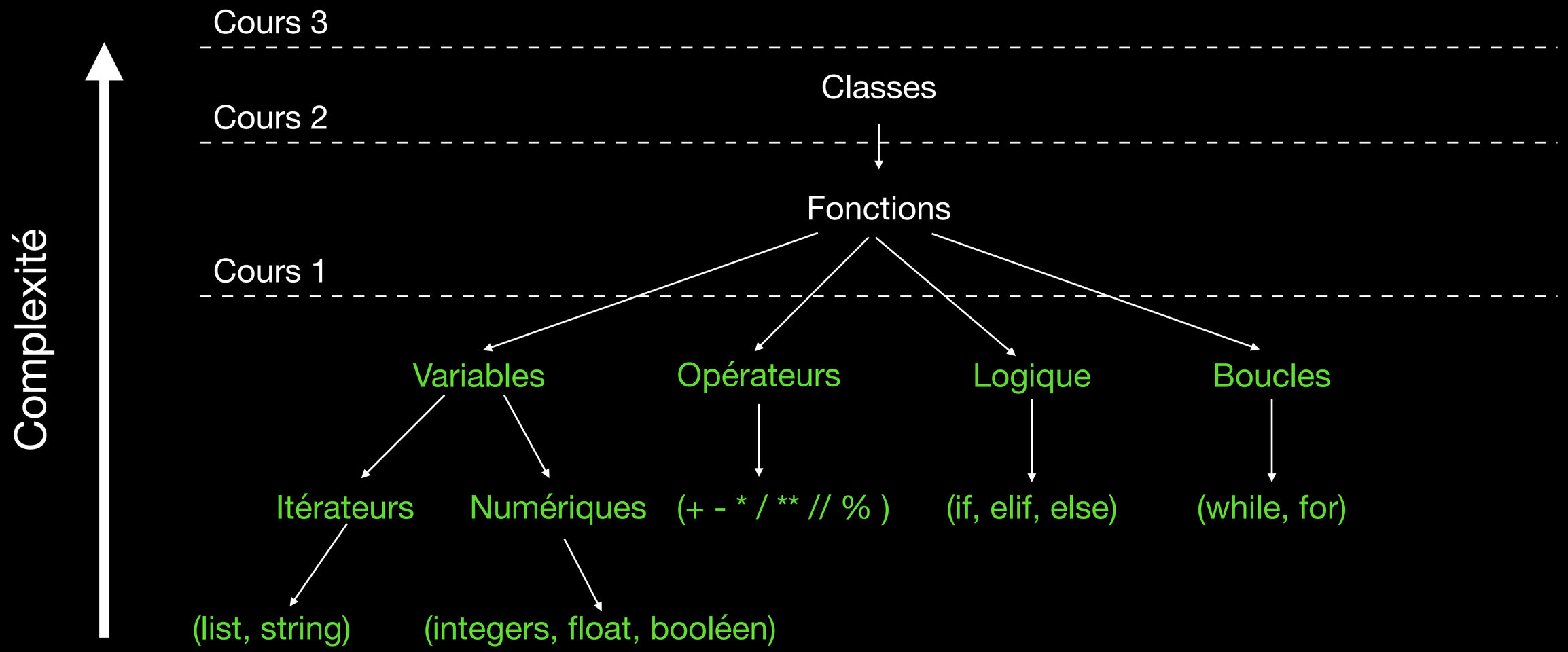
La boucle *while*

- Permet d’itérer comme la boucle *for* mais sur un nombre de fois arbitraire
- Utilisé pour des algorithmes ou pour faire rouler un programme jusqu’à ce qu’on l’arrête
- Attention aux boucles infinies
- Exemple : un algorithme vraiment peu efficace pour trouver la distance entre deux nombres

PENDANT que le nombre 1 est PLUS PETIT que le nombre 2 : ajouter +1 au nombre 1

- **Exemple 6 :** La boucle while

Les blocs de construction d'un programme



Votre premier Jeu : devine mon nombre

- L’ordinateur choisi un nombre entre 0 et 10
- L’utilisateur essaie des chiffres et l’ordinateur lui donne des indices tant qu’il ne l’a pas

4 Outils nécessaires pour construire le jeu :

- `string = input('input string')` Demande un input à l’utilisateur en imprimant l’input string
- `integer = int(variable)` Transforme un variable d’un certain type en integer (si possible)
- `random` Package permettant de générer des chiffres aléatoires
- `random int = random.randint(a, b)` Fonction du package random qui permet de générer un chiffre aléatoire entre a et b (inclus)