

Formation Python

École Secondaire Pierre-Dupuy

Olivier Chabot, 17 Novembre 2021

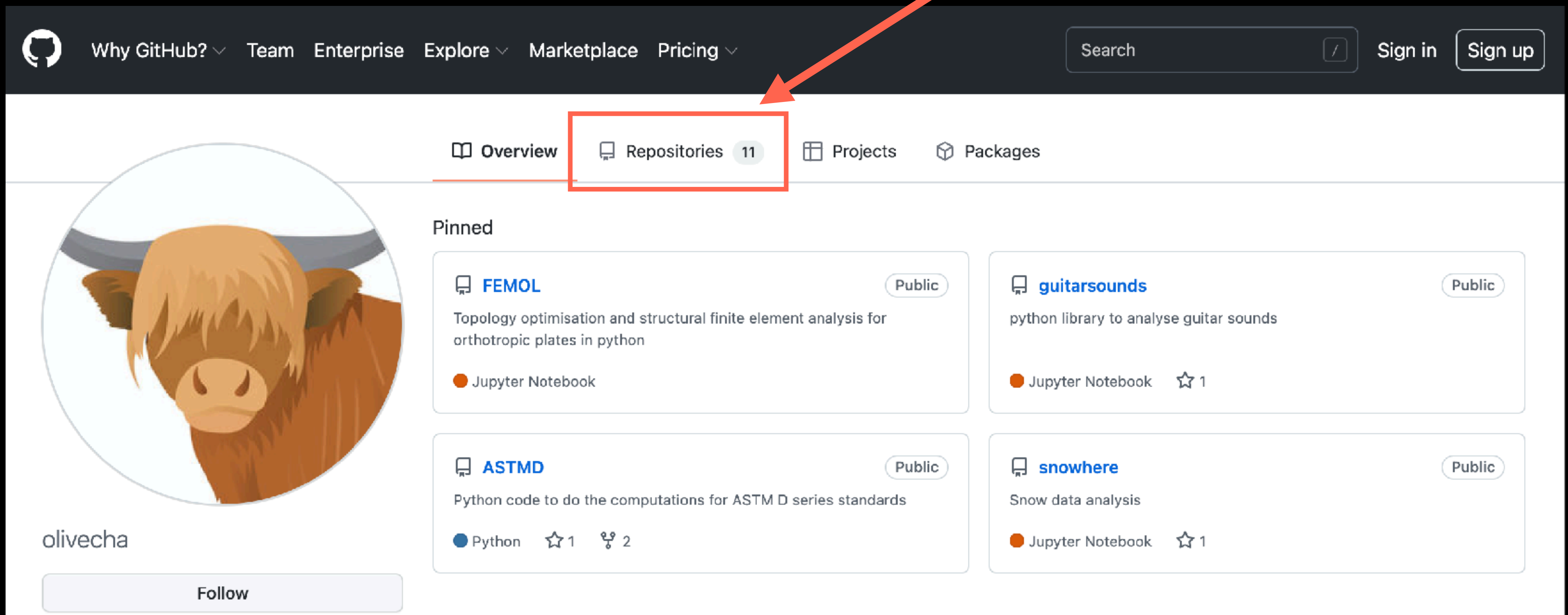
Plan de la formation

- Aller chercher le contenu du cours d'aujourd'hui
- Récapitulatif séance 2
- Les boucles
- Comprendre le jeu devine le nombre
- Les fonctions
- Votre deuxième jeu



Télécharger des fichiers de **github**

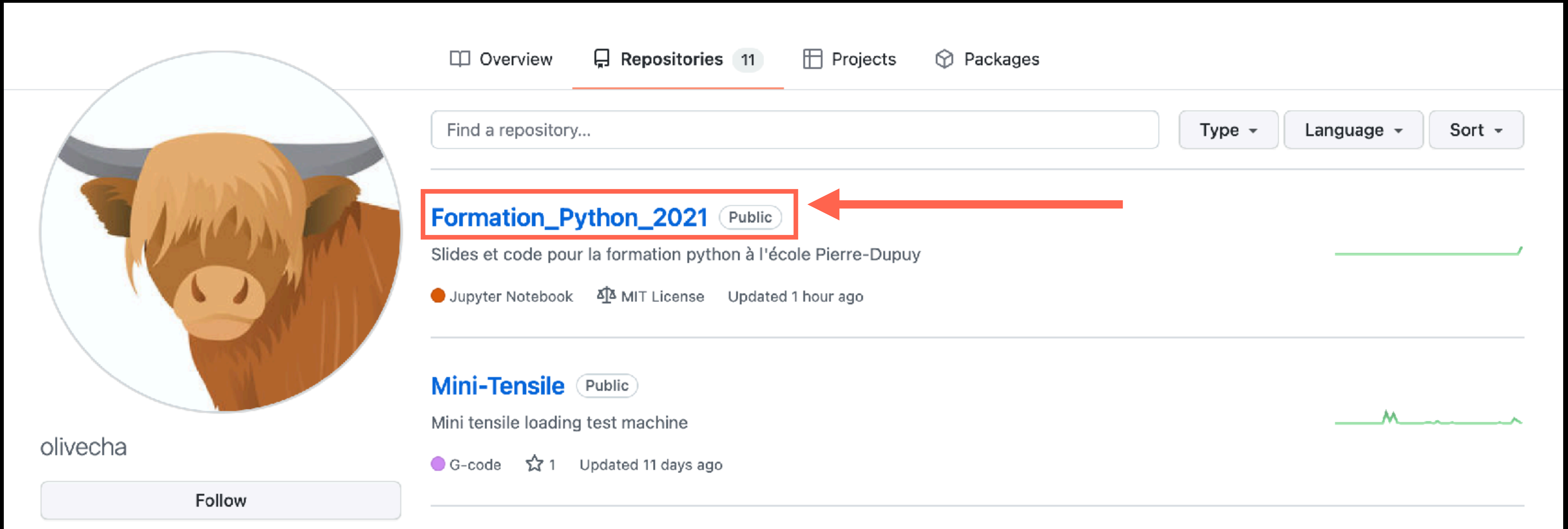
- Se rendre sur <https://github.com/olivecha>



The screenshot shows the GitHub profile page for user **olivecha**. The navigation bar at the top includes links for Why GitHub?, Team, Enterprise, Explore, Marketplace, and Pricing, along with a search bar and Sign in / Sign up buttons. The user's profile picture is a cartoon illustration of a Highland cow. The 'Repositories' tab is selected and highlighted with a red box, with a red arrow pointing to it from the text above. Below the tabs, the 'Pinned' section displays four repositories:

Repository Name	Description	Language	Stars	Forks	Public
FEMOL	Topology optimisation and structural finite element analysis for orthotropic plates in python	Jupyter Notebook	0	0	Yes
guitarsounds	python library to analyse guitar sounds	Jupyter Notebook	1	0	Yes
ASTMD	Python code to do the computations for ASTM D series standards	Python	1	2	Yes
snowhere	Snow data analysis	Jupyter Notebook	1	0	Yes

Répertoire **github** de la formation



The screenshot shows the GitHub profile of a user named 'olivecha'. The profile includes a circular avatar of a Highland cow, the username 'olivecha', and a 'Follow' button. The 'Repositories' tab is selected, showing 11 repositories. A search bar is present with the placeholder text 'Find a repository...'. Two repositories are listed: 'Formation_Python_2021' and 'Mini-Tensile'. The repository 'Formation_Python_2021' is highlighted with a red box and a red arrow pointing to it from the right. It is a public repository containing Jupyter Notebooks, licensed under MIT, and was updated 1 hour ago. The repository description is 'Slides et code pour la formation python à l'école Pierre-Dupuy'. The repository 'Mini-Tensile' is also public, contains G-code, has 1 star, and was updated 11 days ago. Its description is 'Mini tensile loading test machine'.

Overview Repositories 11 Projects Packages

Find a repository...

Type Language Sort

Formation_Python_2021 Public

Slides et code pour la formation python à l'école Pierre-Dupuy

Jupyter Notebook MIT License Updated 1 hour ago

Mini-Tensile Public

Mini tensile loading test machine

G-code ☆ 1 Updated 11 days ago

olivecha

Follow

Téléchargement du dossier .zip

olivecha / Formation_Python_2021 Public

Notifications

Star 0

Fork 0

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

main

1 branch

0 tags

Go to file

Code

About

Slides et code pour la formation python à l'école Pierre-Dupuy

Readme

MIT License

Releases

No releases published

Packages

No packages published

Clone

HTTPS GitHub CLI

https://github.com/olivecha/Formation_

Use Git or checkout with SVN using the web URL.

Open with GitHub Desktop

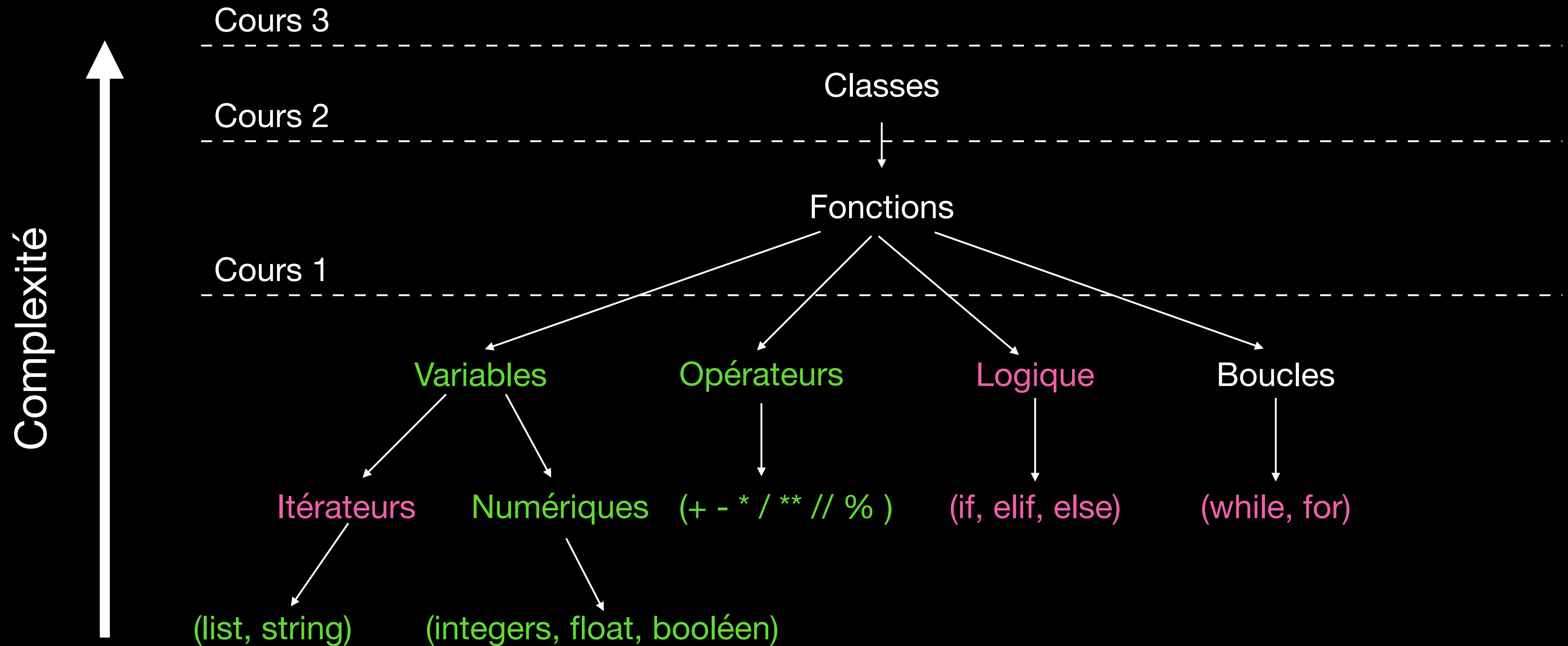
Download ZIP

Rappel sur les différents types

Les types sont la façon qu'utilise l'ordinateur pour représenter des variables

Nom	Signature	Exemples
Nombre entier	int	type(1) = int, liste[a], type(a) = int
Nombre à virgule	float	type(4/2) = float, type(3.1415) = float
Booléen	bool	type(1 < 2) = bool, if a: type(a) = bool
String	str	type('abc') = str, my_string='abc', my_string[0] = 'a'
Liste	list	type([1, 2, 3]) = list, a = [1, 2, 3], a[0] = 1

Récapitulatif de la séance 2



La structure (if, elif, else)

```
if condition1:
    'Exécute si la condition 1 est vraie'

elif condition2:
    """
    Exécute si la condition 1 est fausse
    et la condition 2 est vraie
    """

else:
    """
    Exécute si toutes les conditions
    sont fausses
    """
```


La structure for/in

```
# Pour chaque variable dans un itérateur: <-- Ne pas oublier le ':'  
for each_variable in iterator:  
    #-> Le code intenté s'exécute pour chaque variable  
    do_something(each_variable)  
# Ici on sort de la boucle  
do_one_last_thing()
```

Exemple 0 : Boucle for (impression)

```
# Une liste de noms (string)
mes_noms = ['Jay', 'Sam', 'Nina', 'Pollo']

# Pour chaque nom dans mes noms
for chaque_nom in mes_noms:
    # Imprimer le nom
    print(chaque_nom)
```

Exemple 1: boucle for (sommmation)

```
# Define a list
my_list = [2, 3, 4, 1, 6]
# Define the sum variable
my_sum = 0
# Pour chaque nombre dans ma liste:
for each_number in my_list:
    # La valeur de chaque nombre change à chaque
    # itération
    my_sum = my_sum + each_number
# Le print ne s'exécute qu'une fois
print(my_sum)
```

Les itérateurs

Simplement : Un objet qui peut être utilisé dans un boucle FOR

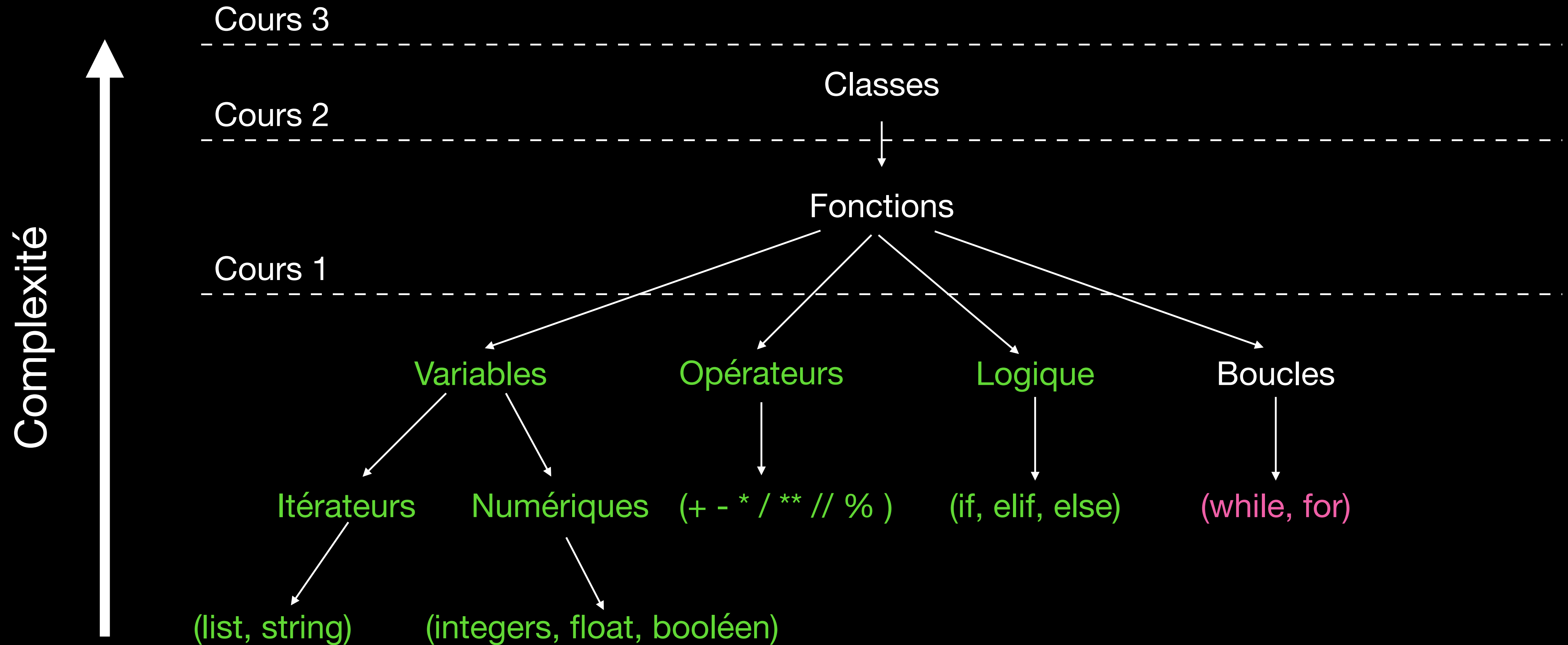
- Les objets pour lesquels on peut utiliser la notation :

```
for thing in iterator:  
    ""  
  
    Do something  
  
    ""
```

Exemple 2 : Les itérateurs

On pourra voir plus tard pourquoi c'est cool et ça mérite un nom

Encore quelques notions sur les boucles



La *méthode* `.append()` des listes

Interlude pour voir un essentiel de la boucle **for**

- Méthode :
`liste.append(variable)`
- Ajoute une variable à la **fin** de la `liste`
- Très utile dans les boucles **for**

```
ma_liste = [1, 2, 3]
ma_liste.append(4)
print(ma_liste)
#>>> [1, 2, 3, 4]
```

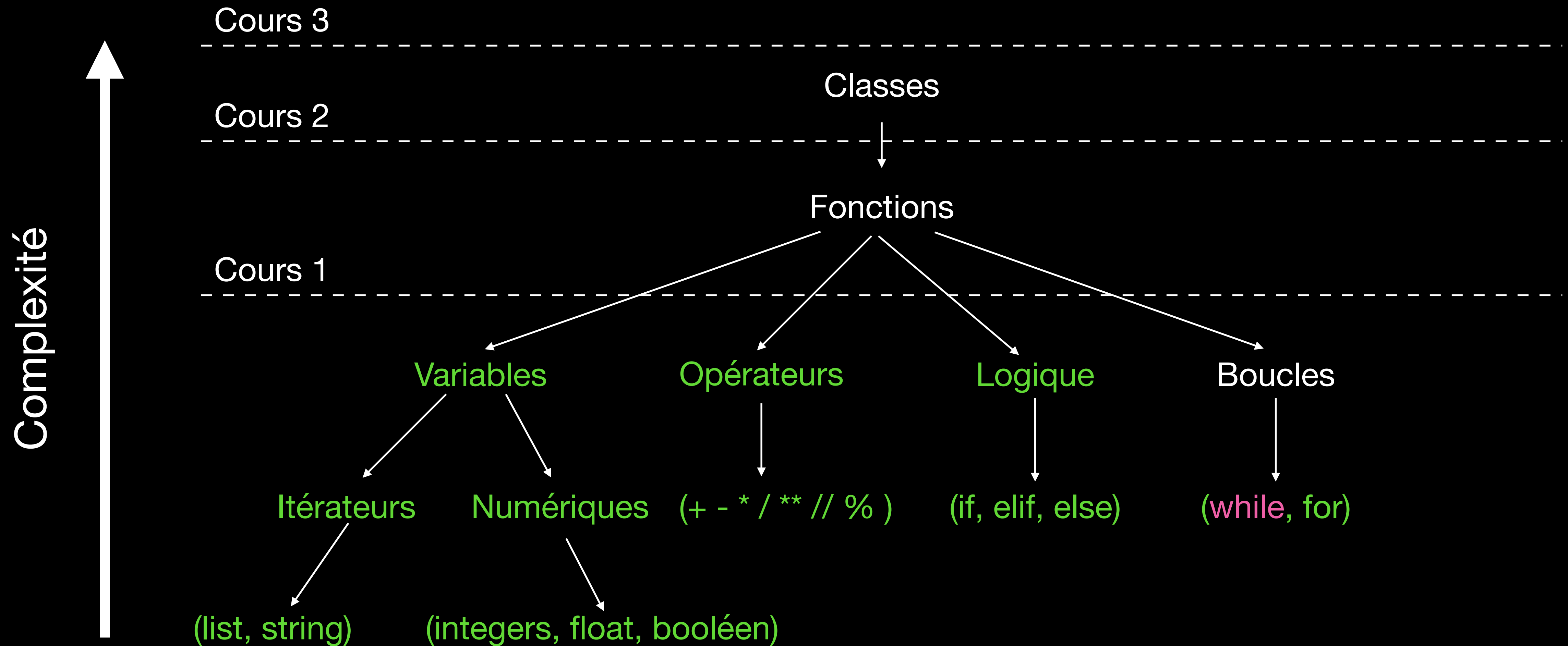
Exemple 2.9 : La méthode `.append()`

La méthode `.append()` dans les boucles `for`

```
# Une liste sur laquelle on itere
ma_liste = ['item1', 'item2', 'item3']
# Une nouvelle liste vide que l'on va remplir
ma_nouvelle_liste = []

# Pour chaque item dans ma liste:
for chaque_item in ma_liste:
    # On change chaque item
    nouvel_item = operation(chaque_item)
    # On ajoute chaque item changé dans la nouvelle
    # Liste
    ma_nouvelle_liste.append(nouvel_item)
```


Bientôt la maîtrise de tout les blocs de base !



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

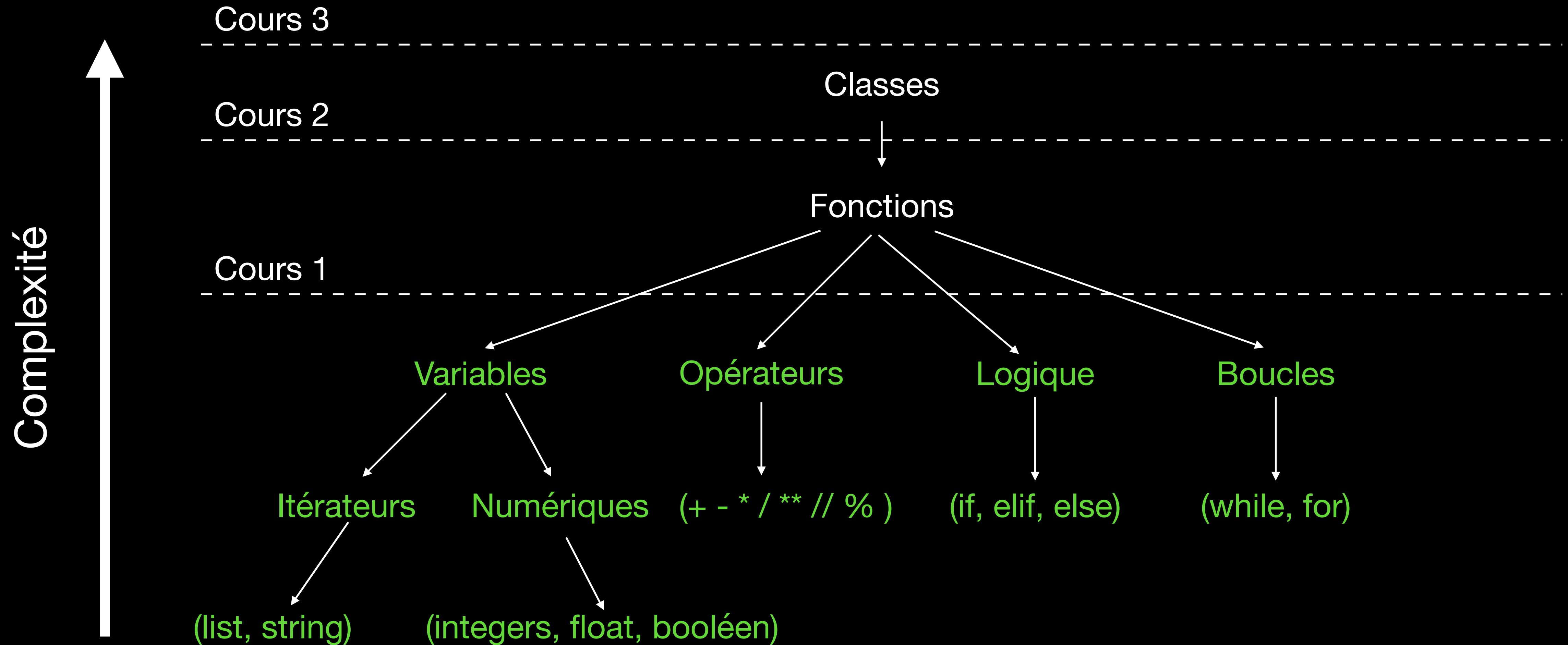
Structure de la boucle while

```
# Pendant que la condition est vraie
while condition is True:
    # Faire quelque chose à chaque boucle
    do_something()
    # Regarder si quelque chose s'est changé
    # Changer la valeur de la condition dans la boucle
    if something_hapened:
        # Lorsque condition = False on sort de la boucle
        condition=False

# Ajouter un statement à exécuter à la fin du while
else:
    do_something_once_at_the_end()
```

- **Exemple 4 :** La boucle while

Les blocs de construction d'un programme



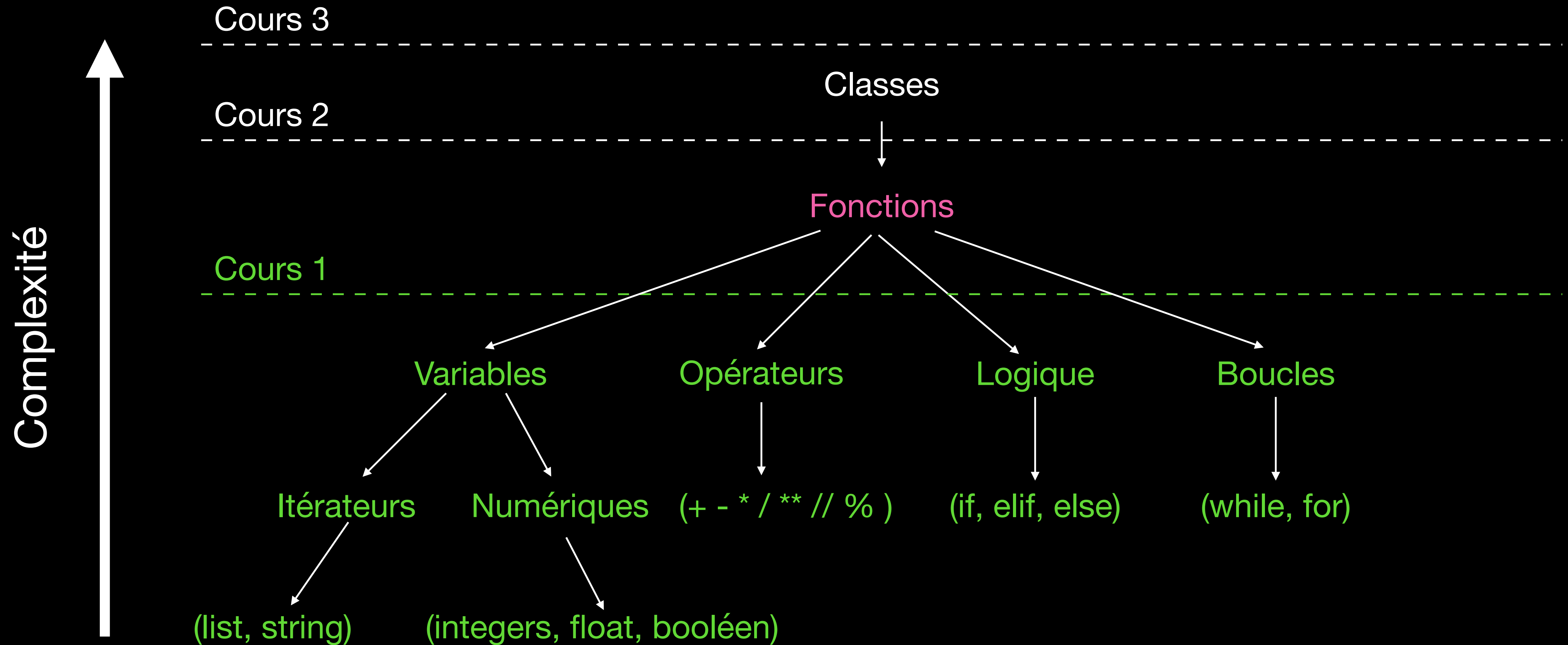
Votre premier Jeu : devine mon nombre

- 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)

Les blocs de construction d'un programme



Les fonctions

- Fonctions que vous connaissez déjà ou presque :

Fonction	Arguments	Ce qui se passe	Output
<code>print()</code>	<code>(arg1, arg2, ...)</code>	Les arguments sont imprimés dans la console séparés par des espaces	<code>None</code>
<code>type()</code>	<code>(arg)</code>	Retourne le type de l'argument	<code>object</code>
<code>int()</code>	<code>(arg)</code>	Transforme l'argument en nombre entier si possible	<code>int</code>
<code>help()</code>	<code>(fonction)</code>	Imprime le paragraphe d'aide lié à la fonction	<code>None</code>
<code>str()</code>	<code>(arg)</code>	Transforme l'argument en chaine de caractère si possible	<code>string</code>
<code>input()</code>	<code>(string)</code>	Affiche la chaine de caractère en argument et retourne le input de l'utilisateur	<code>string</code>
<code>len()</code>	<code>(iterable)</code>	Calcule la longueur d'un itérable et renvoie un nombre entier correspondant	<code>int</code>
<code>random.randint()</code>	<code>(lower, upper)</code>	Génère un nombre entier au hasard entre lower et upper	<code>int</code>

Syntaxe d'une fonction

je **DÉFINIS** une **FONCTION** prenant des **ARGUMENTS** :

```
def fonction(arguments):  
    """  
    Fonction qui change pas arguments  
    """  
    sortie = arguments  
    return sortie
```

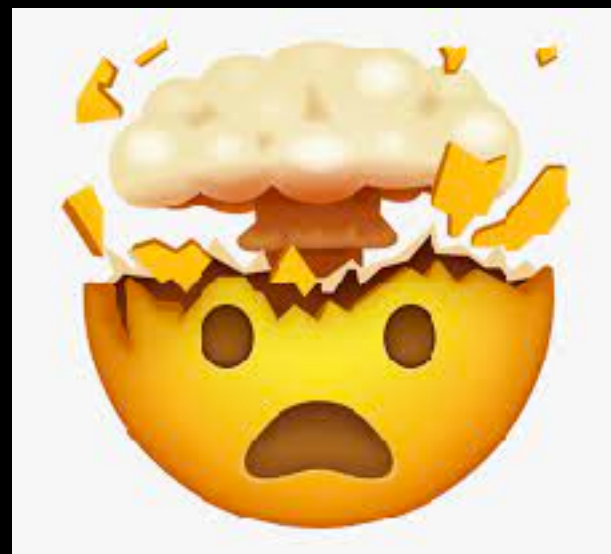
la **FONCTION** agit sur les **ARGUMENTS** et **RETOURNE** quelque chose

Utilisation d'une fonction

j' **APPELLE** ma **FONCTION** avec **()**

```
fonction('a')  
>> 'a'
```

Les fonctions sont aussi des variables :
(tout est une variable dans python)



```
f = fonction  
f('a')  
>> 'a'
```

La portée d'une fonction (scope)

Les fonctions c'est comme Vegas :

Ce qui se passe dans la fonction reste dans la fonction

```
def fonction_1(a, b)
    # La variable 'c' est dans
    # le scope global
    return a + b + c
```

```
def fonction_2(a, b)
    # La variable 'c' appartient seulement
    # à la fonction
    c = 3
    return a + b + c
```

```
c = 4
ma_fonction_1(1, 1)
>> 6
ma_fonction_2(1, 1)
>> 5
print(c)
>> 4
```

- **Exemple 5 : Les fonctions**

Les arguments mots clés

- Les arguments mot clefs permettent de d'avoir une valeur par défaut et d'éviter devoir se rappeler de l'ordre des arguments dans la fonction

```
def salutation(phrase = 'Bonjour', nom=''):
    """
    Fonction qui salue quelqu'un
    """
    print(phrase + ' ' + nom + ' !')
```

```
salutation()
>> 'Bonjour !'
salutation(nom='Maxime')
>> 'Bonjour Maxime !'
salutation(nom='Karine', phrase='Salut')
>> 'Salut Karine !'
```

Les modules

Une façon de garder l'ordre dans les fonctions

- Un module contient des fonctions auxquelles on peut accéder en *l'important*

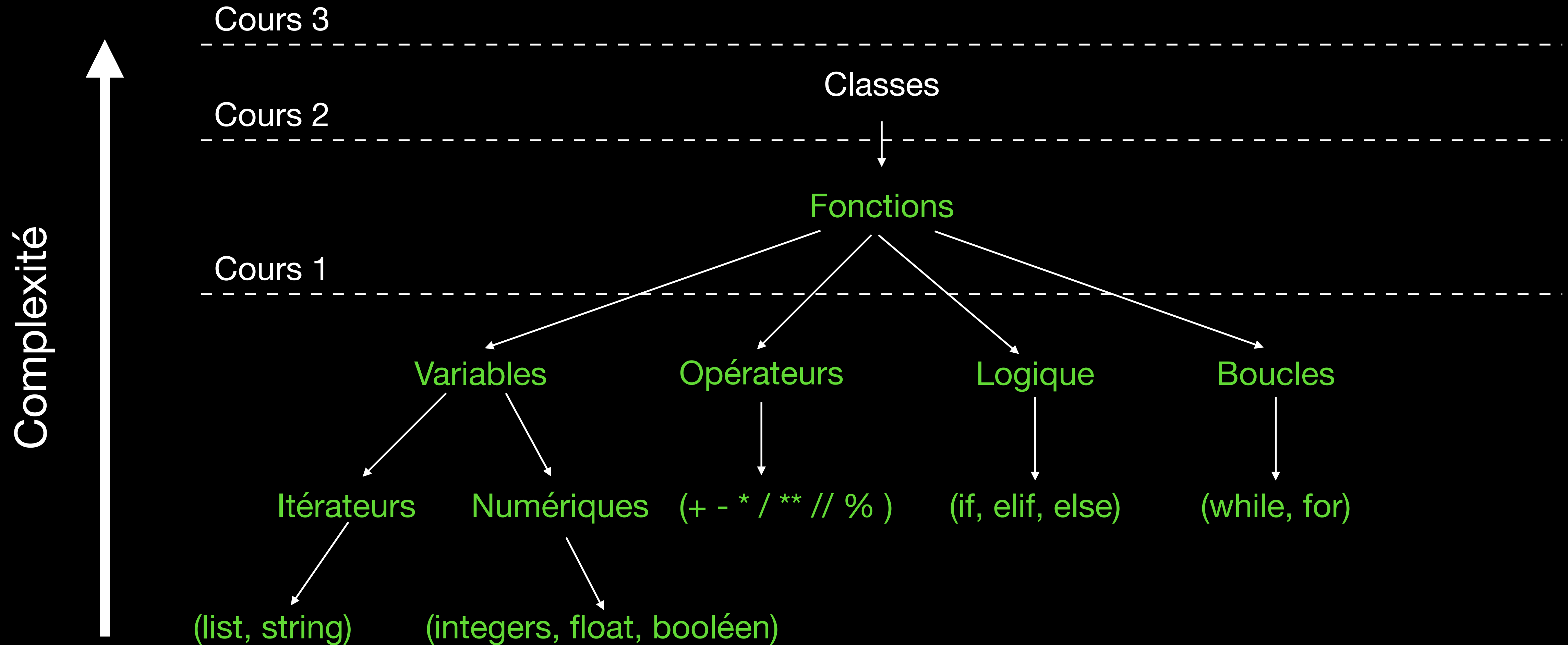
```
import random  
type(random)  
>> module
```

```
import random  
type(random.randint)  
>> method
```

```
from random import randint  
type(randint)  
>> method
```

Ici `method` est une fonction

Les blocs de construction d'un programme



Jeu 2 : Le bonhomme pendu

Utiliser des fonctions pour mieux concevoir le jeu