

Formation Python

École Secondaire Pierre-Dupuy

Olivier Chabot, 17 Novembre 2021

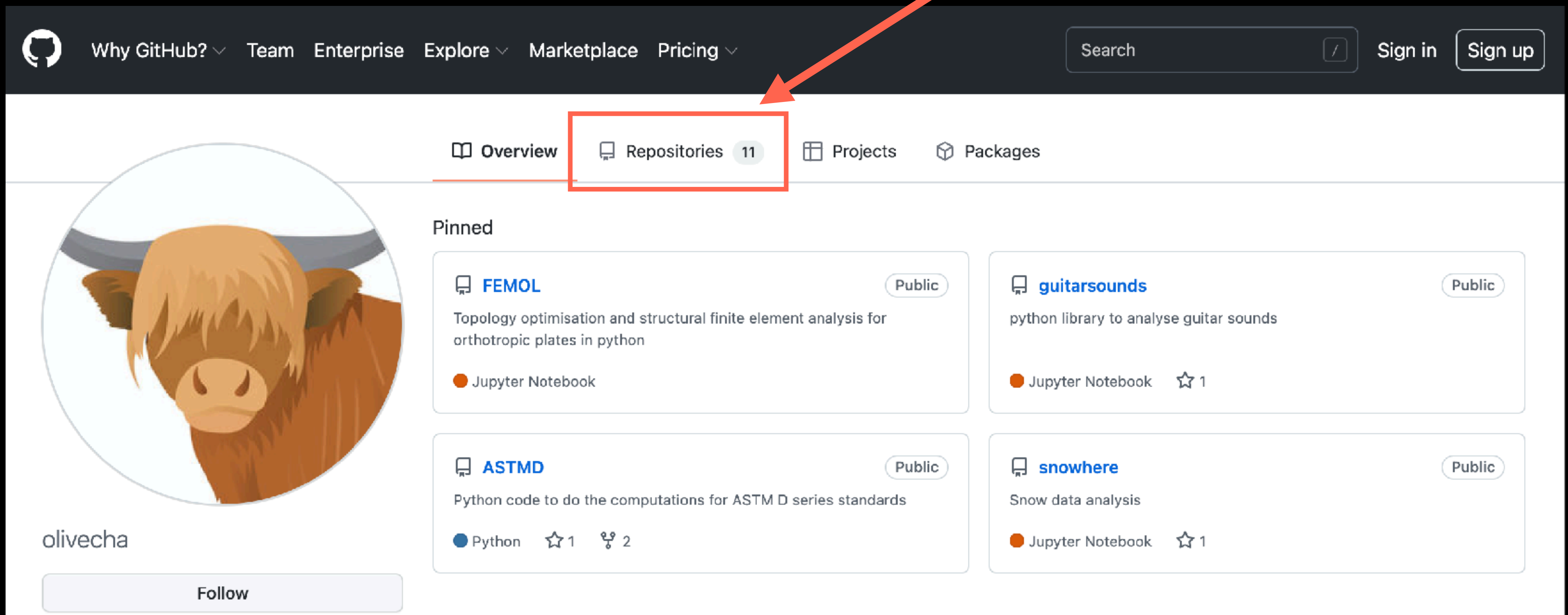
Plan de la formation

- Aller chercher le contenu du cours d'aujourd'hui
- Exercices pseudo code et boucles
- Retour sur le cours 4
- Les fonctions
- Approche pour créer vos propres programmes




Télécharger des fichiers de **github**

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




Why GitHub? ▾ Team Enterprise Explore ▾ Marketplace Pricing ▾ Search / Sign in Sign up


Overview Repositories 11 Projects Packages



olivecha
Follow

Pinned


 **FEMOL** Public


Topology optimisation and structural finite element analysis for orthotropic plates in python

 Jupyter Notebook


 **guitarsounds** Public


python library to analyse guitar sounds

 Jupyter Notebook ☆ 1


 **ASTMD** Public

Python code to do the computations for ASTM D series standards

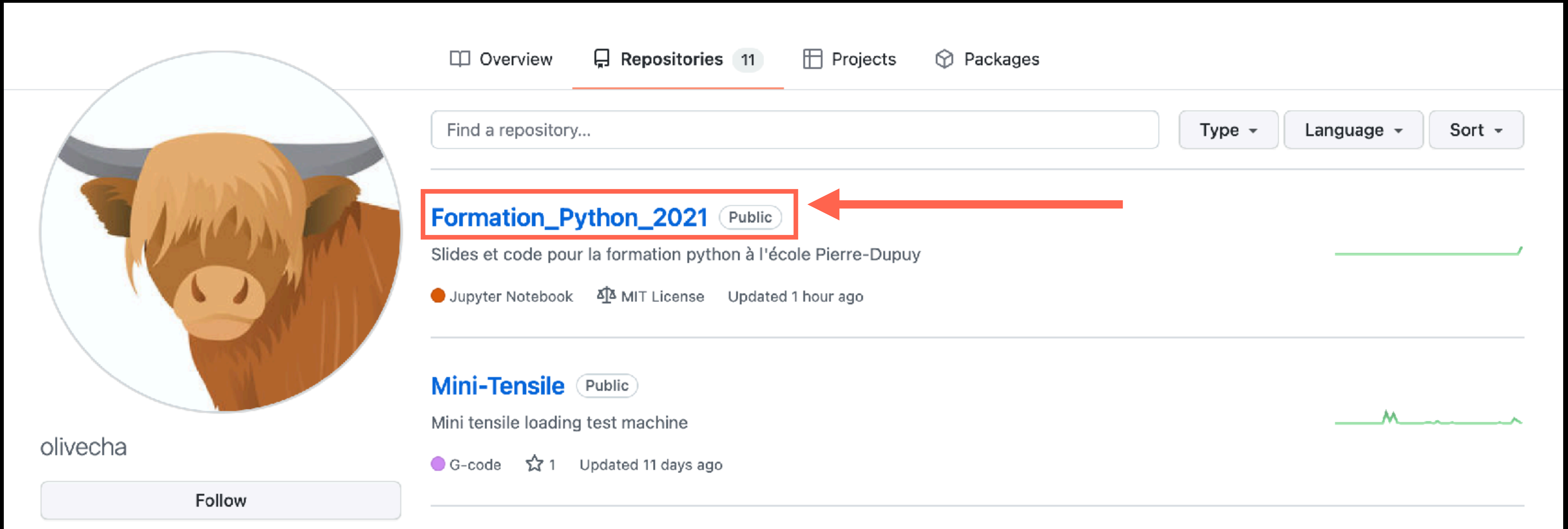
 Python ☆ 1 🍴 2

 **snowhere** Public

Snow data analysis

 Jupyter Notebook ☆ 1

Répertoire **github** de la formation



The screenshot shows the GitHub profile of a user named 'olivecha'. The profile picture is a cartoon illustration of a Highland cow. The user has 11 repositories. The 'Repositories' tab is selected. A search bar is present with the placeholder text 'Find a repository...'. Below the search bar, two repositories are listed. The first repository, 'Formation_Python_2021', is highlighted with a red box and a red arrow pointing to it. It is a public repository containing Jupyter Notebooks, licensed under MIT, and was updated 1 hour ago. The second repository, 'Mini-Tensile', is also public, contains G-code, has 1 star, and was updated 11 days ago. A 'Follow' button is located below the profile picture.

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

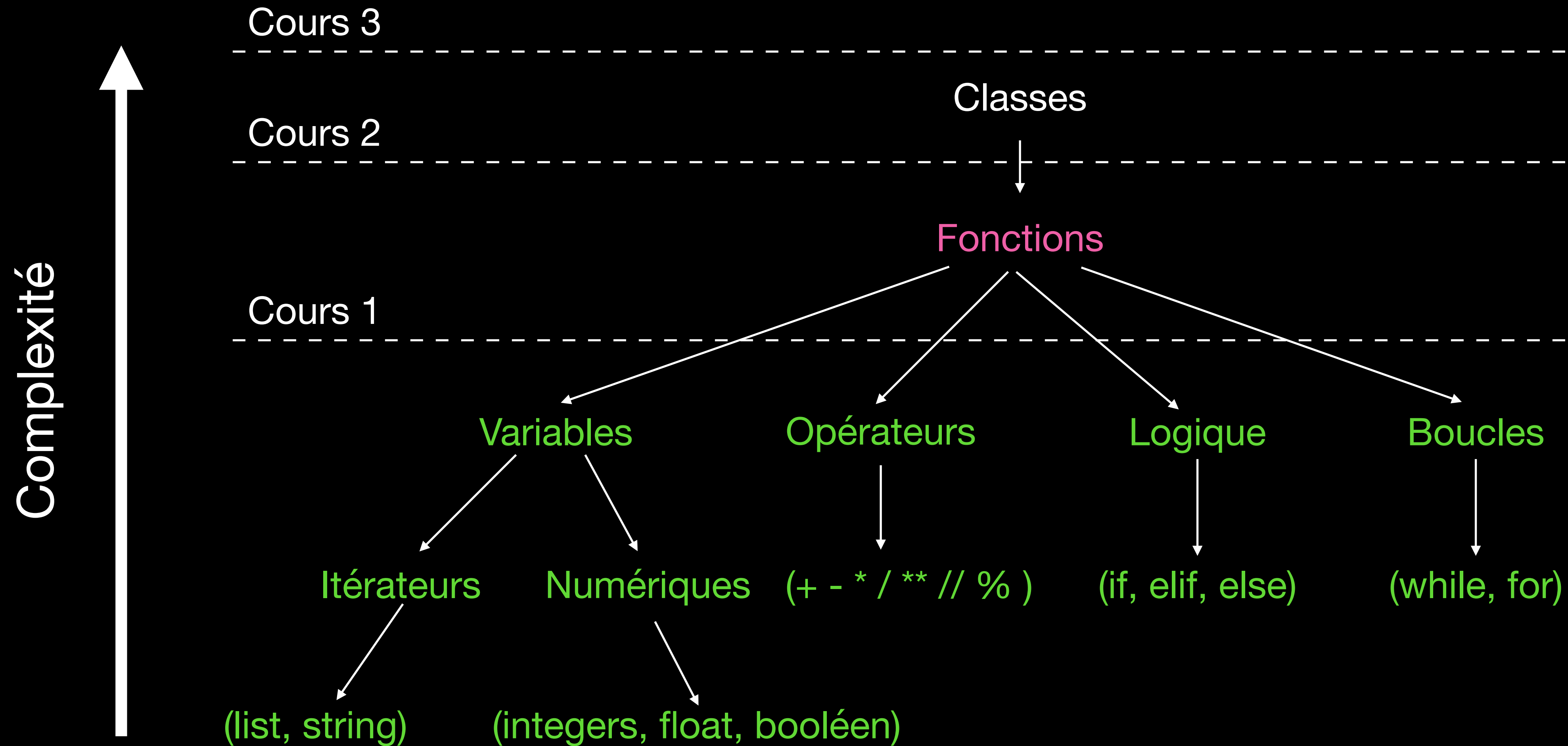
https://github.com/olivecha/Formation_

Use Git or checkout with SVN using the web URL.

Open with GitHub Desktop

Download ZIP

Rappel des séances précédentes



Les opérateurs booléens

Permettent de comparer des variables

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

Les différents opérateurs

Permettent des transformer des variables

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

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

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 *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 boucle For

Faire quelque chose pour chaque variable dans un itérateur

```
iterator = [1, 2, 3] # a list
new_list = [] # an empty list
for each_thing in iterator:
    # do something with each thing
    each_thing = each_thing + 1
    # append in a list
    new_list.append(each_thing)

# >> new_list = [2, 3, 4]
```

La boucle while

Faire quelque chose tant qu'une condition est vraie

```
# Définir une variable avant la boucle
my_count = 0
# Pendant que ma variable respecte une condition
while my_count < 10:
    # Faire quelque chose
    print(my_count)
    # CHANGER LA VARIABLE (sinon boucle infinie)
    my_count = my_count + 1
```


Les fonctions de base python

Faites par les pros juste pour vous

```
def randint(self, a, b):  
    """Return random integer in range [a, b],  
    """  
  
    return self.randrange(a, b+1)
```

```
def randrange(self, start, stop=None, step=1, _int=int):  
    """Choose a random item from range(start, stop[, step]).  
  
    This fixes the problem with randint() which includes the  
    endpoint; in Python this is usually not what you want.  
  
    """  
  
    # This code is a bit messy to make it fast for the  
    # common case while still doing adequate error checking.  
    istart = _int(start)  
    if istart != start:  
        raise ValueError("non-integer arg 1 for randrange()")  
    if stop is None:  
        if istart > 0:  
            return self._randbelow(istart)  
        raise ValueError("empty range for randrange()")
```

```
# stop argument supplied.  
istop = _int(stop)  
if istop != stop:  
    raise ValueError("non-integer stop for randrange()")  
width = istop - istart  
if step == 1 and width > 0:  
    return istart + self._randbelow(width)  
if step == 1:  
    raise ValueError("empty range for randrange() (%d,%d, %d)" % (istart, istop, width))  
  
# Non-unit step argument supplied.  
istep = _int(step)  
if istep != step:  
    raise ValueError("non-integer step for randrange()")  
if istep > 0:  
    n = (width + istep - 1) // istep  
elif istep < 0:  
    n = (width + istep + 1) // istep  
else:  
    raise ValueError("zero step for randrange()")  
  
if n <= 0:  
    raise ValueError("empty range for randrange()")  
  
return istart + istep*self._randbelow(n)
```

Suite de la fonction randint

Just for fun

```
def _randbelow(self, n, int=int, maxsize=1<<BPF, type=type,
               Method=_MethodType, BuiltinMethod=_BuiltinMethodType):
    "Return a random int in the range [0,n).  Raises ValueError if n==0."

    random = self.random
    getrandbits = self.getrandbits
    # Only call self.getrandbits if the original random() builtin method
    # has not been overridden or if a new getrandbits() was supplied.
    if type(random) is BuiltinMethod or type(getrandbits) is Method:
        k = n.bit_length() # don't use (n-1) here because n can be 1
        r = getrandbits(k) # 0 <= r < 2**k
        while r >= n:
            r = getrandbits(k)
        return r
```


Suite de la fonction randint

De moins en moins fun

```
def getrandbits(self, k):  
    """getrandbits(k) -> x.  Generates an int with k random bits."""  
    if k <= 0:  
        raise ValueError('number of bits must be greater than zero')  
    if k != int(k):  
        raise TypeError('number of bits should be an integer')  
    numbytes = (k + 7) // 8                                # bits / 8 and rounded up  
    x = int.from_bytes(_urandom(numbytes), 'big')  
    return x >> (numbytes * 8 - k)                        # trim excess bits
```

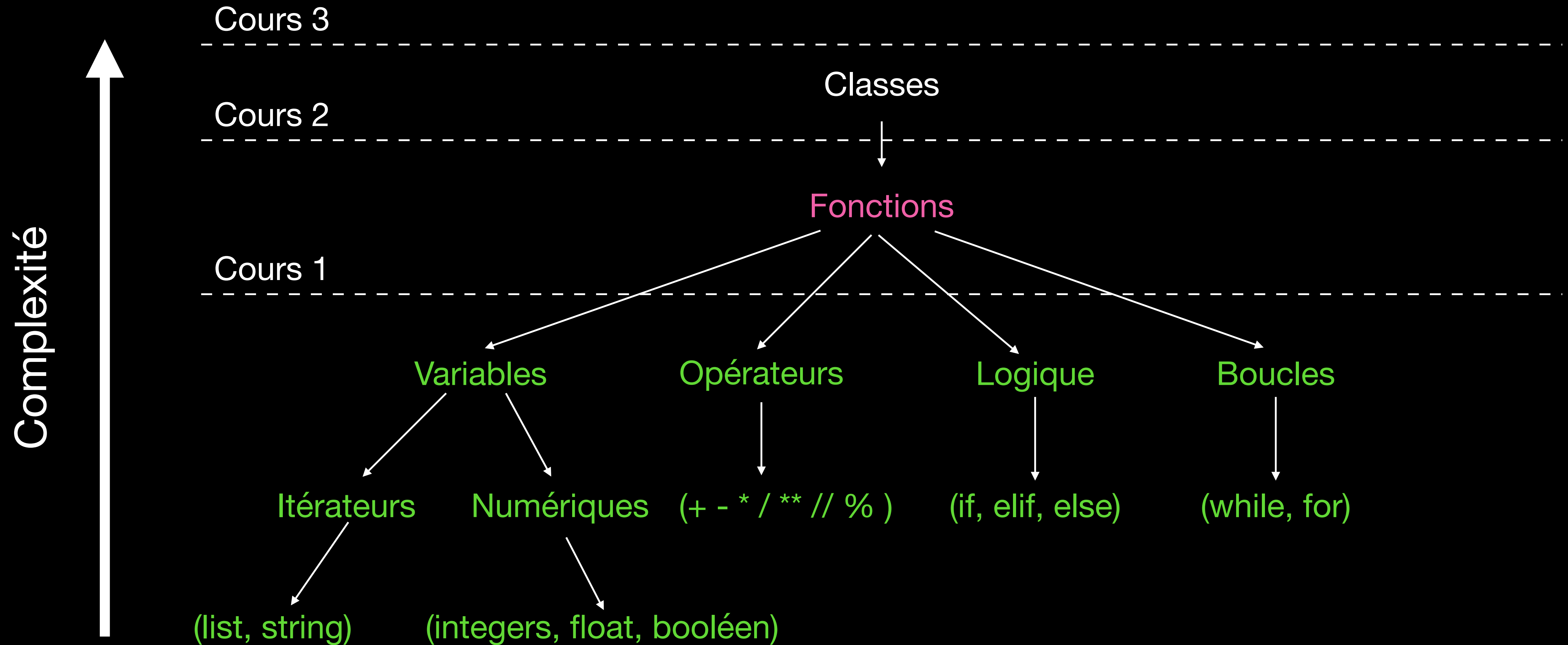
We reached the end



Les fonctions de base que vous connaissez

Description	Exemple
Génère un nombre entier aléatoire dans un certain interval	<pre>print(random.randint(10, 20)) # >> 13</pre>
Affiche un message à l'utilisateur et enregistre l'input dans une variable	<pre>nom_utilisateur = input('Quel est votre nom ?') print('Votre nom est : ', nom_utilisateur)</pre> <p>Quel est votre nom ? Olivier Votre nom est : Olivier</p>
Transforme un autre type en int, si les types sont compatibles	<pre>type(int('1'))</pre> <p>int</p>
Affiche l'aide d'une fonction	<pre># La fonction help() peut aider si jamais on oublie help(random.randint)</pre> <p>Help on method randint in module random:</p> <p>randint(a, b) method of random.Random instance Return random integer in range [a, b], including both end points.</p>

Les fonctions : compartimenter le code



Syntaxe d'une fonction

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

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

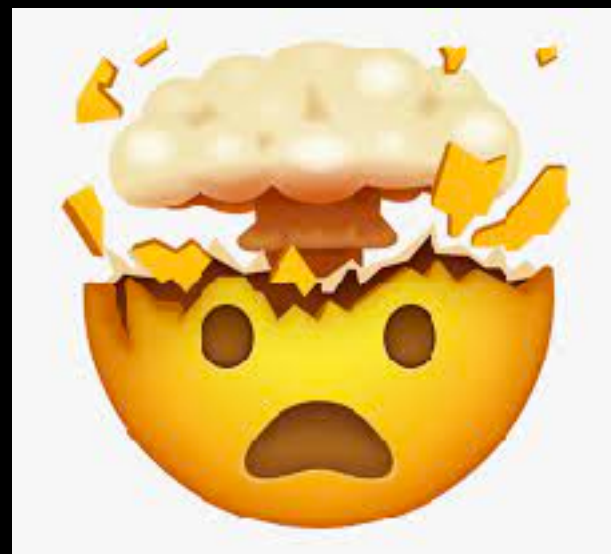
la **FONCTION** agit sur les **ARGUMENTS** et **RETOURNE** quelque chose

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

Comme une aide mémoire

- 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*

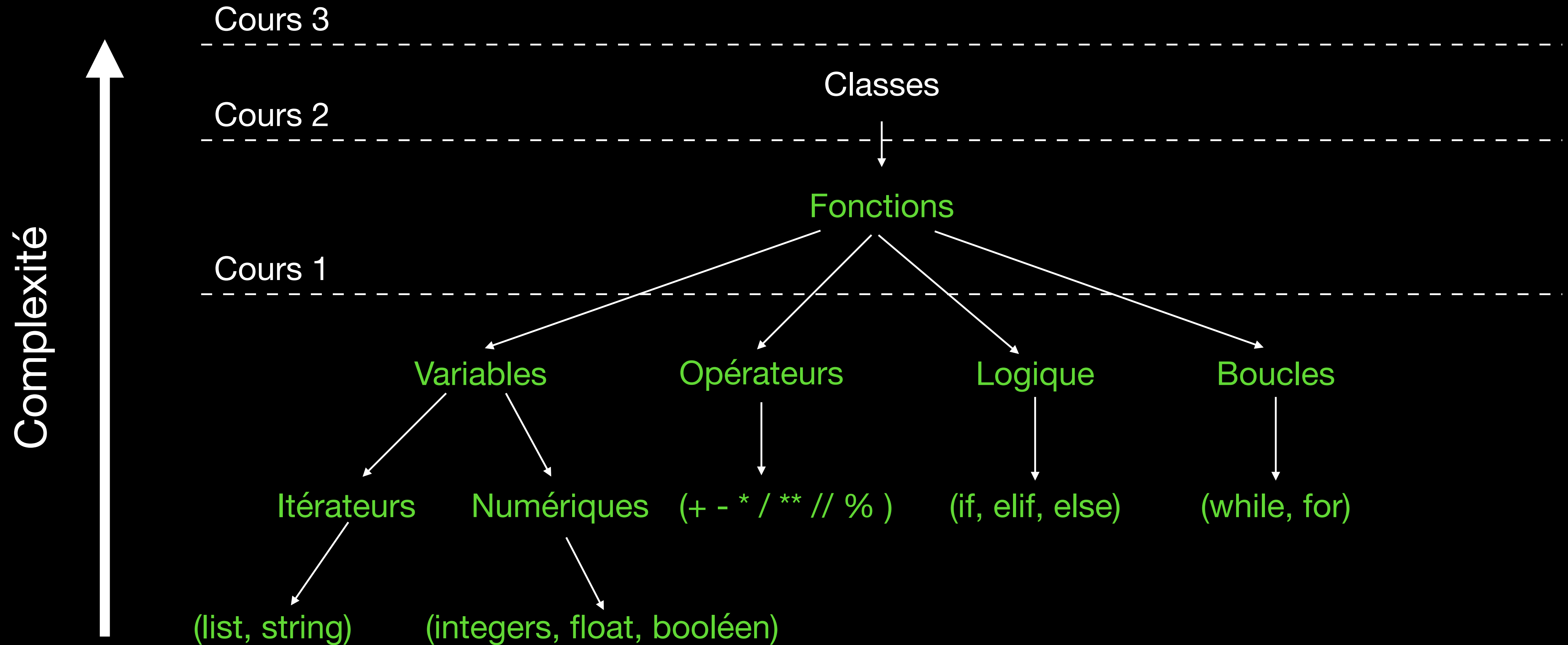
```
import random  
type(random)  
>> module
```

```
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