



Le langage Python

Port-Au-Prince, Haïti

8 Mai - 11 Mai 2017

Etienne Trimaille

Présentation de Python

Qu'est-ce que Python ?

- Multi-usage (WEB, Gui, script, etc)
- POO
- Interpréte
- Centré sur la lecture et la productivité
- Grosse communauté

Quelques fonctionnalités

- Tout est objet
- Shell interactif
- Introspection
- Multi-plateforme
- Intégrations avec des autres langages (Jython)

Qui utilise Python ?

- GeoNode
- QGIS
- GDAL
- Google
- ...

Un éditeur pour Python ?

- Emacs
- Vim
- Bloc-Note
- PyCharm
- Eclipse (PyDev)

Versions

- Crée en 1989
- Python 1.0 en 1994
- Python 2.0 en 2000
- Python 3.0 en 2008
- Python 3.6 fin 2016
- Toujours et encore 2.7, l'adoption de Python 3 est longue !



Comment on l'utilise ?

Mode interactif

```
etienne@:~ $ echo 'Ceci est bash'  
Ceci est bash  
etienne@:~ $ python  
Python 2.7.13 (default, Dec 28 2016, 08:42:30)  
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)]  
on darwin  
Type "help", "copyright", "credits" or "license" for more  
information.  
>>> print 'Ceci est python'  
Ceci est python  
>>>
```

Mode script

```
etienne@:~/dev/script $ python ~/Desktop/code.py  
Hello CNIGS!
```

Syntaxe

Hello CNIGS

```
#/usr/bin/python
# coding=utf-8

print 'Hello CNIGS!'
```

Indentation

- La plupart des langages n'en tiennent pas compte
- Mais les humains SI et Python aussi !

Indentation

```
/* Bug en C */

if (boo)
    if (bar)
        baz(foo, bar);
else
    qux();
```

Indentation

```
/* Bug en C */

if (boo){
    if (bar) {
        baz(foo, bar);
}
else {
    qux();
}
```

Indentation

```
/* Bug en C */

if (boo)
if (bar)
baz(foo, bar);
else
qux();
```

Indentation

```
/* La bonne version, être explicite ! */

if (boo) {
    if (bar) {
        baz(foo, bar);
    }
    else {
        qux();
    }
}
```

Indentation

```
/* Python aime les indentations */

if boo:
    if bar:
        baz(foo, bar)
else:
    qux()
```

Commentaire

```
# Commentaire sur une ligne
```

```
"""Commentaire sur plusieurs lignes.  
C'est un exemple de commentaire sur  
plusieurs lignes.
```

```
"""
```



Types

Strings

```
# C'est une chaîne
name = 'C\'est Etienne'

# C'est aussi une chaîne
pays = "France"

# Une chaîne multi-ligne
formation = '''Le python c'est cool,
Django, c'est bien aussi.'''

# et encore une multi-ligne, en unicode
bio = u"""J'aime bien faire des sports extérieurs
comme du vélo ou de la randonnée."""
```

Nombres

```
# Integers
year = 2017
year = int("2017")

# Floating Point Numbers
pi = 3.141592
pi = float("3.141592")
```

Null

```
optional_data = None

not_a_number = float("nan")
from math import isnan
isnan(not_a_number)
>>> True
```

Tuples

```
liste_vide = ()  
liste_un_seul_item = ('hi', )  
liste = (1, 2, 3, 'hi !')
```

```
type(liste)  
>>> <type 'tuple'>
```

```
len(liste)  
>>> 4
```

```
liste[0]  
>>> 1
```

```
liste[0:2]  
>>> (1, 2)
```

```
liste[2:]  
>>> (3, 'hi !')
```

```
liste[:2]  
>>> (1, 2)
```

Listes

```
# List
nombres = []

# Ajout
liste.append(1)

# Extend
liste.extend([2, 3, 4])

# Equivalent à
nombres = [1, 2, 3, 4]
```

Dictionnaires

```
personnes = {}

# Getter/setter avec la clé
personnes['nom'] = 'etienne'

# Clé non mutable
personnes[42] = 'mon nombre favori'
personnes[(44.47, -73.21)] = 'coordinates'

# Problème
personnes[[44.47, -73.21]] = 'coordinates'
```

Booléen

```
# Un booleen
is_python = True

# Tout en python peut etre caste en boolean
is_python = bool('tout')

# Tout est equivalent a False
cest_faux = False or 0 or "" or {} or [] or None

# Tout est vrai
cest_vrai = True and 1 and "text" and 123 and {'a': 'b'} and ['c', 'd']
```



Opérateurs

Arithmétique

```
a = 10      # 10
1 += 1      # 11
1 -= 1      # 10

b = a + 1  # 11
c = a - 1  # 9

d = a * 2  # 20
e = a / 2  # 5
f = a % 3  # 1
g = a ** 2 # 100
```

Manipulation des chaînes

```
animaux = 'Chats ' + 'Chiens '
animaux += 'Lapins'

fruit = ', '.join(['Pomme', 'Banane', 'Orange'])

date = ' %d %s %d' % (8, 'Mai', 2017)

# Deprecated
nom = 'Prenom %(prenom)s et nom %(nom)s' % {'prenom':'etienne',
'nom':'trimaille'}

# Conseillé
nom = 'Prenom {prenom} et nom {nom}'.format(prenom='etienne',
nom='trimaille')
```

Opérateurs logiques

a **and** b

a **or** b

not a

(a **and** **not** (b **or** c))

Opérateurs logiques

```
a > b  
a >= b  
a < b  
a <= b  
a == b  
a != b  
a is b  
a is not b  
a in b
```

Contrôle des flux

Conditions

```
note = 13

if note >= 16:
    if note == 20:
        print u'Toutes mes félicitations'
    else:
        print u'Félicitations'
elif 14 <= note < 16:
    print u'Très bien'
elif 12 <= note < 14:
    print 'Bien'
else:
    print 'Peu mieux faire'
```

Boucle for avec une liste

```
for x in range(10):
    print x

fruits = ['pomme', 'banane']
for fruit in fruits:
    print 'Il y a une ' + fruit
```

Boucle for avec dictionnaire

```
states = {  
    u'Artibonite': u'Gonaïves',  
    u'Centre': u'Hinche',  
    u'Grande Anse': u'Jérémie',  
}  
  
for departement, capital in states.iteritems():  
    print '%s: %s' % (departement, capital)
```

Recherche d'un élément

```
fruits = ['pomme', 'banane']

# Solution simple
if 'banane' in fruits:
    print "Il y a des bananes"
else:
    print "Il n'y a pas de banane"

# Deuxième solution, plus complexe, le
for ... else
for fruit in fruits:
    if fruit == 'banane':
        print "Il y a des bananes"
        break
else:
    print "Il n'y a pas de banane"
```

Boucle while

```
x = 0

while x < 100:
    print x
    x += 1
```

Pas de boucle do ... while

```
faire()
while not condition_echec:
    faire()
```

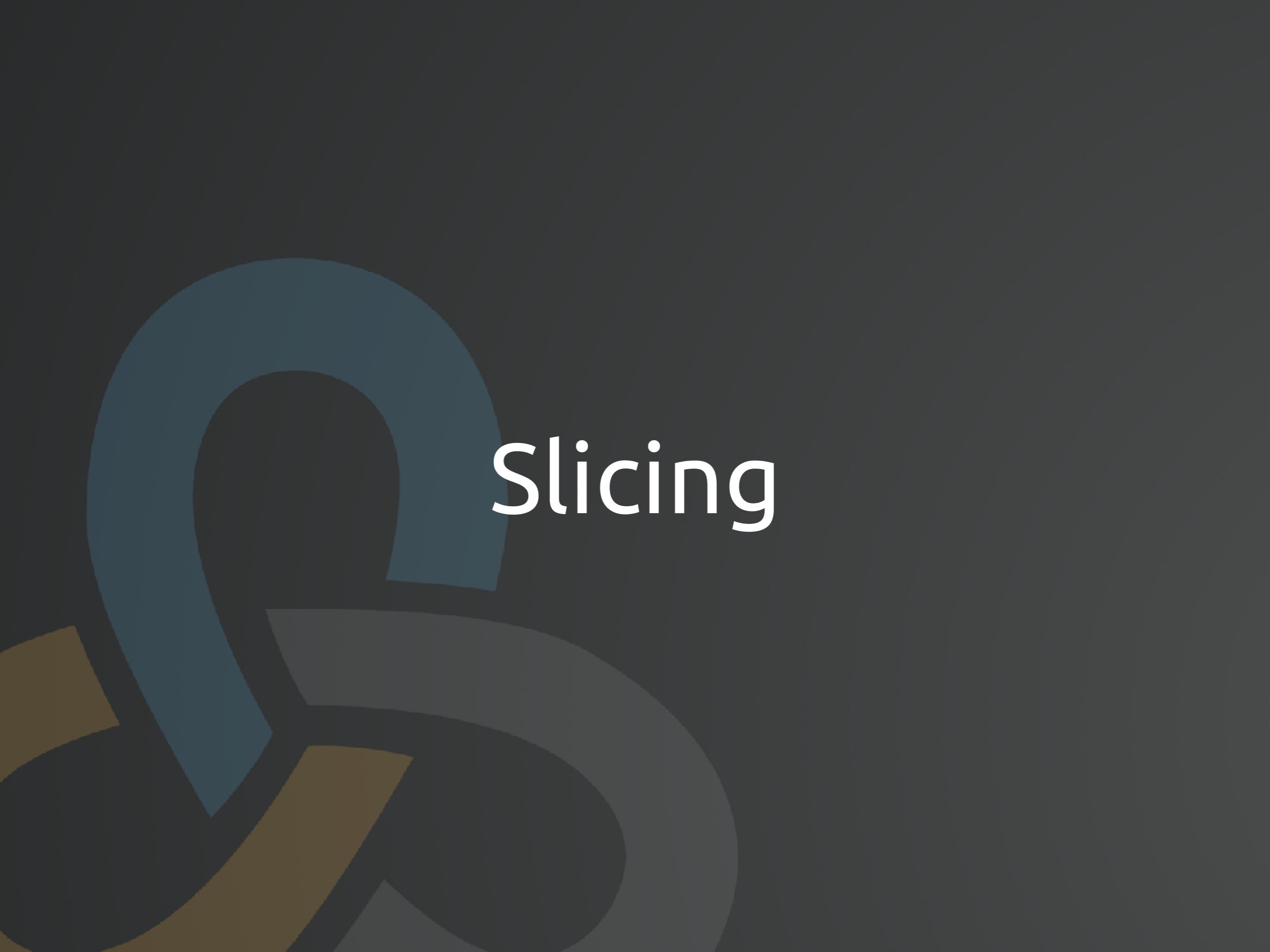
List Comprehensions

Remplacer une boucle for simplement

```
# Non pythonique
impair = []
for x in range(50):
    if x % 2:
        impair.append(x)
```

```
# Pythonique
impair = [ x for x in range(50) if x % 2 ]
```

Exercice



Slicing

Slicing

Sous-ensemble d'une liste

```
>>> alpha = "ABCDEFGHIJKLMNPQRSTUVWXYZ"  
>>> alpha[1]  
'B'  
>>> alpha[1:3]  
'BC'  
>>> alpha[-1]  
'Z'  
>>> alpha[-3:]  
'XYZ'  
>>> alpha[:6]  
'ABCDEF'
```

Fonctions

Fonctions

```
def bonjour():
    """Fonction pour dire bonjour."""
    print 'Bonjour CNIGS'
```

Fonctions

```
def ajouter(x, y):
    """Ajouter deux nombres."""
    return x + y

def crier(phrase='hi'):
    """Fonction pour crier une phrase."""
    print phrase.upper()

def discuter(texte, personnage='Moi'):
    """Fonction pour ajouter le nom d'un personnage."""
    print '%s: %s' % (personnage, texte)
```

Fonctions, args et kwargs

```
def une_fonction(*args, **kwargs):
    for arg in args:
        print arg
    for key, value in kwargs.iteritems():
        print '%s -> %s' % (key, value)

une_fonction(1, 2, 3, text='Ma phrase')
```

Fibonacci

```
def fibonacci(n):
    """Retourne Fibonacci jusqu'à N."""
    results = []
    a, b = 0, 1
    while a < n:
        results.append(a)
        a, b = b, a + b
    return a
```

Plusieurs retours

```
def decomposer(entier, divise_par):
    """Retourne la partie entiere et le reste de d'une division."""
    partie_entiere = entier / divise_par
    reste = entier % divise_par
    return partie_entiere, reste
```

Portée des variables et références

Portée des variables

```
def print_a():
    """Fonction chargée d'afficher la variable a."""
    print("La variable a = {}".format(a))

>>> print_a()
La variable a = 5.
>>> a = 8
>>> print_a()
La variable a = 8.
```

Portée des variables

```
fruits = ['pomme', 'banane', 'coco']

if 'pastèque' in fruits:
    pastèque_trouve = True
else:
    pastèque_trouve = False

print pastèque_trouve
```

Passage par référence

```
ma_liste1 = [1, 2, 3]
ma_liste2 = ma_liste1
ma_liste2.append(4)

print ma_liste2
print ma_liste1
```

Imports

- Réutilisation du code, isolation
- Import dans le namespace courant

```
import math  
print math.pi
```

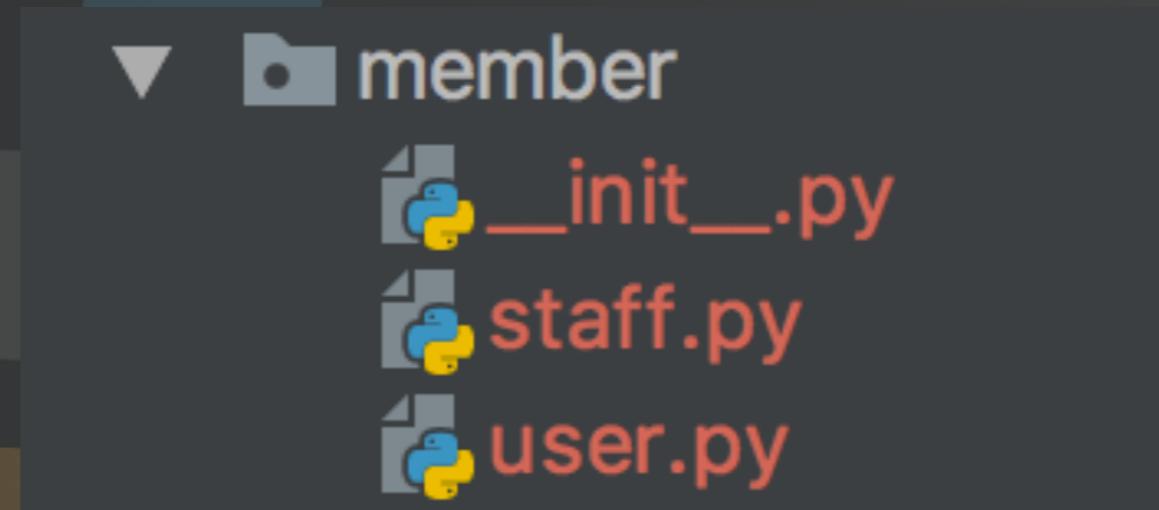
```
from math import pi  
print pi
```

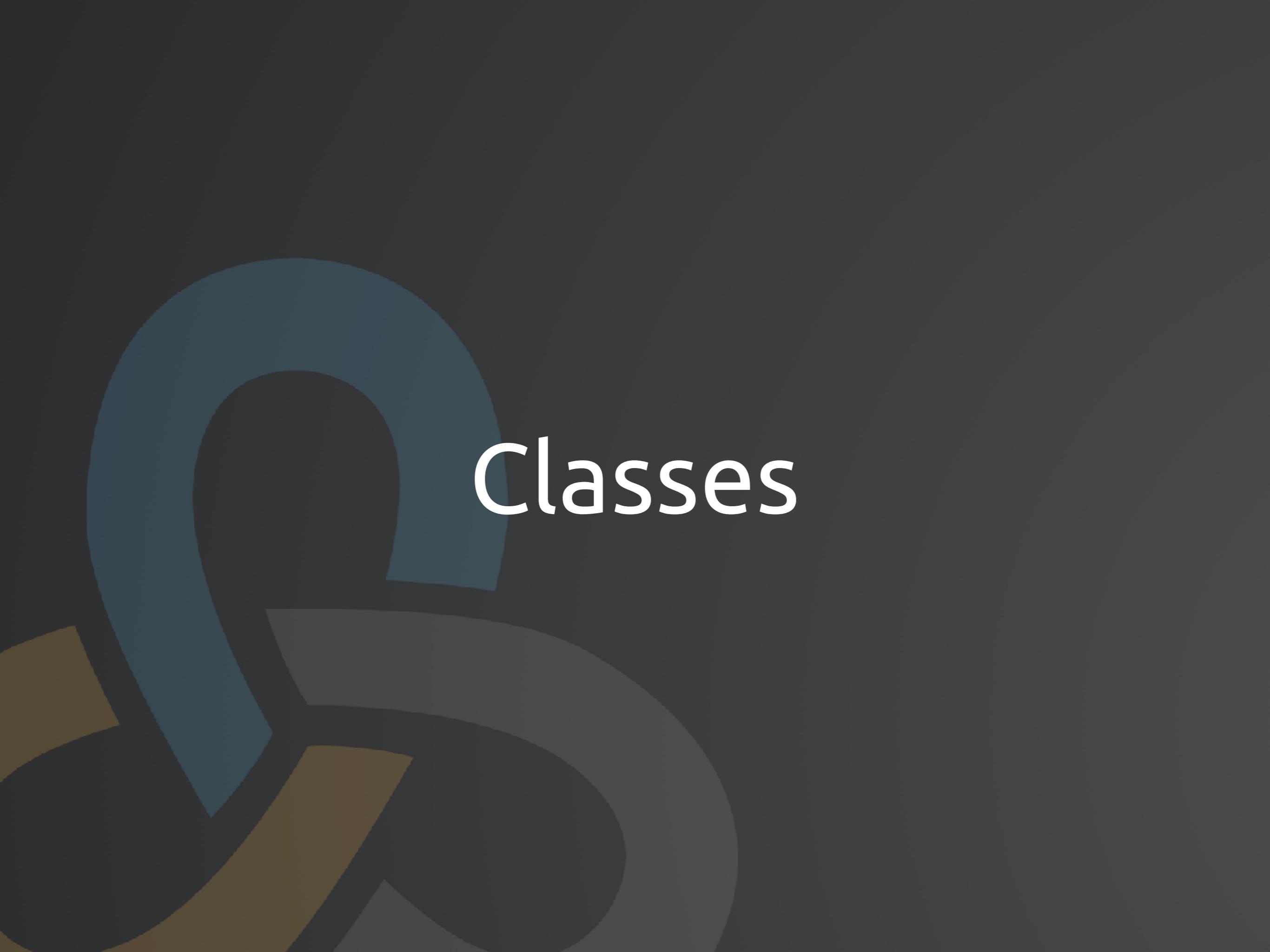
```
from math import pi as pi_number  
print pi_number
```

```
from math import * # NO NO
```

Votre propre package

- Un package est un dossier, avec un `__init__.py`
- Les fichiers Python dans un package sont des modules





Classes

Déclaration de classe

```
class User(object):

    is_staff = False

    def __init__(self, name='Anonymous'):
        self._name = name
        super(User, self).__init__()

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, name):
        self._name = name

    def is_authorized(self):
        return self.is_staff
```

Héritage d'une classe

```
from personne import User

class SuperUser(User):
    is_staff = True

    def __init__(self, name, number):
        self.staff_number = number
        super(SuperUser, self).__init__(name)
```

```
admin = SuperUser('etienne')
print admin.name
print admin._name
print admin.isAuthorized()
```

Visibilité des attributs

- Pas d'attributs/méthodes réellement privées
 - Notation avec préfix underscore _ (ou 2) par convention
- Fonctions spéciales commencent et terminent avec __ (deux underscores)
 - `__init__`, `__doc__`, `__name__`, `__str__`

Les exceptions

```
a = 1
b = 2

def division(a, b):
    """Divise deux nombres."""
    print a / b

division(a, b)
```

Les exceptions

```
a = 1
b = 2

def division(a, b):
    """Divise deux nombres."""
    try:
        print a / b
    except ZeroDivisionError:
        print 'Division impossible par 0'
    except Exception:
        print 'Erreur lors de la division'
    else:
        print 'Une division possible sans probleme'
    finally:
        print 'Une autre division a me faire faire ?'
```

Les fonctions spéciales

Pour inspecter un objet, obtenir des informations

`__doc__`

`__type__`

`__name__`

`__class__`

Package management

Python :

```
import sys  
print sys.path
```

Bash :

```
pip install Django  
pip freeze  
pip show Django
```

Un mini serveur web

```
#!/usr/bin/python

from wsgiref import simple_server

def hello(environ, start_response):
    status = '200 OK'
    headers = [('Content-Type', 'text/plain')]
    start_response(status, headers)
    return 'Hello world'

if __name__ == '__main__':
    host, port = '127.0.0.1', 8080
    httpd = simple_server.make_server(host, port, hello)
    try:
        print "Open http://%s:%s" % (host, port)
        httpd.serve_forever()
    except KeyboardInterrupt:
        pass
```

La règle PEP8

- 79 caractères maximum
- Ligne vide en fin de fichier
- Pas d'espace en fin de ligne
- Espace entre les opérateurs
- CamelCase sur les classes
- `snake_case` sur les fonctions
- <https://www.python.org/dev/peps/pep-0008/>

Test Driven Development

- UnitTest, NoseTests ...
- <https://docs.python.org/2/library/unittest.html>

Les bonnes pratiques

- PEP8
- Flake8
- Landscape : <https://landscape.io>

Python Bonus

- La fonction enumerate

```
users = ['Tom', 'James', 'John']

for i, user in enumerate(users):
    print '%s -> %s' % (i, user)
```

- La fonction zip

```
list_1 = [1, 2, 3, 4, 5]
list_2 = ['a', 'b', 'c', 'd', 'e']

for number, letter in zip(list_1, list_2):
    print '%s : %s' % (number, letter)
```

Python Avancé

- Les décorateurs

```
from functools import wraps

def my_decorator(f):
    @wraps(f)
    def wrapped(*args, **kwargs):
        print 'Before decorated function'
        r = f(*args, **kwargs)
        print 'After decorated function'
        return r
    return wrapped

@my_decorator
def my_function(myarg):
    print "my function is called with " + myarg
    return "return value of my function"

r = my_function('Hi')
print r
```

Python Avancé

- Lambda
- Générateurs
- Coroutines

Ressources

- <https://openclassrooms.com/courses/apprenez-a-programmer-en-python?status=published>