

Python

Avancé

- Les modules
- Les packages
- Les exceptions
- Les objets
- Packages natifs
- POO

1. Python

Les modules

Les modules

- Espace de nom ?
 - C'est un lieu factice pour héberger un ensemble de fonction
- En Python, on l'utilise pour importer des librairies externes notamment, via différents mots clés

Les modules

- import
 - **import** math
- as
 - **import** math **as** m
 - Ex : m.sqrt(8)
- from
 - **from** math **import** sqrt
- *
 - **from** math **import** *

Les modules

```
1 # coding: utf-8
2 import malib
3
4 malib.hello()
```

```
1 # coding: utf-8
2
3 def hello():
4     print "Hello World !"
```

`__name__` => Variable spéciale de Python

Si le fichier exécuté est celui-ci, exécuter le code. Sinon, rien faire

```
1 # coding: utf-8
2 import malib
3
4 malib.hello()
```

```
1 # coding: utf-8
2
3 def hello():
4     print "Hello World !"
5
6 if __name__ == "__main__":
7     hello()
```

2. Python

Les packages

Les packages

- Un module contient des fonctions
- Un package contient des modules
 - Donc un package peut contenir une multitude de fonctions !
- Pour initialiser un package, on utilise un fichier `__init__.py` qui sera appelé lors de l'import d'un package.

Les packages - Mauvaise façon

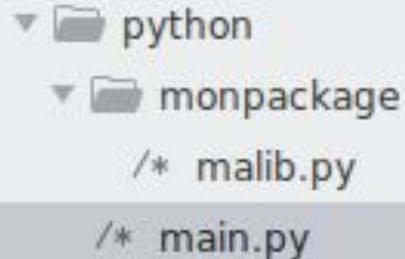
- Deux fichiers :

main.py

```
2
3 import monpackage.malib as malib
4
5 malib.hello()
6 malib.helloDear()
```

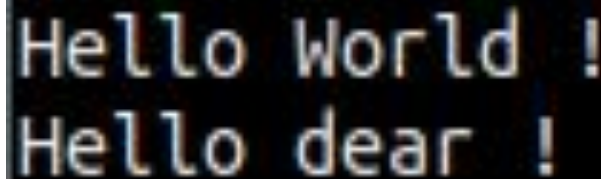
malib.py

```
2
3 def hello():
4     print "Hello World !"
5
6 def helloDear():
7     print "Hello dear !"
8
```



```
▼ python
  ▼ monpackage
    /* malib.py
    /* main.py
```

#~ python main.py



```
Hello World !
Hello dear !
```


Les packages - façon 1

python
└─ monpackage
 ├─ __init__.py
 ├─ malib.py
 ├─ simplymath.py
 └─ main.py

main.py

```
1 # coding: utf-8
2 from monpackage import *
3
4 malib.hello()
5 malib.helloDear()
6 simplymath.add(5, 4)
7 simplymath.minus(10, 3)
```

__init__.py

```
1 from monpackage import malib
2 from monpackage import simplymath
```

Le petit nouveau : `__init__.py`, c'est le sommaire de votre package.

`#~ python main.py`

```
Hello World !
Hello dear !
9
7
```

malib.py

```
1 # coding: utf-8
2
3 def hello():
4     print "Hello World !"
5
6 def helloDear():
7     print "Hello dear !"
8
```

simplymath.py

```
1 # coding: utf-8
2
3 def add(a, b):
4     print a + b
5
6 def minus(a, b):
7     print a - b
8
```

Les packages - façon 2 (+alias)

main.py

__init__.py

```
python
└─ monpackage
   ├── __init__.py
   ├── malib.py
   └── simplymath.py
   └─ main.py

1 # coding: utf-8
2 from monpackage import malib as mal, simplymath as sim
3
4 mal.hello()
5 mal.helloDear()
6 sim.add(5, 4)
7 sim.minus(10, 3)
```

malib.py

simplymath.py

#~ python main.py

```
Hello World !
Hello dear !
9
7
```

```
1 # coding: utf-8
2
3 def hello():
4     print "Hello World !"
5
6 def helloDear():
7     print "Hello dear !"
8
1 # coding: utf-8
2
3 def add(a, b):
4     print a + b
5
6 def minus(a, b):
7     print a - b
8
```

3. Python

Les exceptions

Les exceptions

- Tout comme dans d'autres langages, il existe un bloc Exception qui permet d'essayer du code, et s'il plante, on affiche une erreur. Général, c'est un "TRY...CATCH"
- En Python, c'est "TRY...EXCEPT"

Les exceptions

- La folie des mots-clés :
 - try...except -> Mots-clés de base
 - else -> Si ok, résultat voulu
 - finally -> Ok ou pas, j'y passe
 - pass -> On passe le except sans rien faire

Les exceptions

```
try:  
    10 / 0  
except:  
    print "divide by zero error"
```

```
divide by zero error
```

Les exceptions

```
nb1 = 10
nb2 = 2

try:
    result = nb1 / nb2
except NameError:
    print "Variable undefined"
else:
    print("%s %d" % ("result =>", result)) # shows "Result => 5"

nb3 = 10
result2 = None
try:
    result2 = nb3 / nb4
except NameError:
    print "Variable undefined"
finally:
    print("%s %s" % ("result =>", result2)) # shows "Result => None"
```

```
result => 5
Variable undefined
result => None
```

Les exceptions

```
nb3 = 10
result2 = None
try:
    result2 = nb3 / nb4
except NameError:
    pass
finally:
    print("%s %s" % ("result =>", result2)) # shows "Result => None"
```

```
result => None
```


4. Python

Les objets

Les objets

- En Programmation Orientée Objet, on parle de classes et d'objets.
 - Avant de rentrer dans la réelle POO, un petit rappel sur la manipulation des objets est de mise en Python

Les objets

```
# Une string est un objet !
string = "louder"
print string
print string.upper() #.lower() exists too... :-) etc.

name = "Paul"
course = "English"

print "Hello {0} ! How are you ? \
I hope you will like this {1} course !".format(name, course)

string = "Hello {name} ! How are you ? \
I hope you will like this {course} \
course !".format(name="Florian", course="Python")

print string
```

```
louder
LOUDER
Hello Paul ! How are you ? I hope you will like this English course !
Hello Florian ! How are you ? I hope you will like this Python course !
```

Les objets - liste, tuple et dictionnaire

- `list()` ou bien `[]` créé une liste vide
 - Une liste, c'est un tableau. Il a donc le même comportement que dans d'autres langages
 - Avec `enumerate`, on a vu une liste
- `monTuple = (1, 4, 12, 6)` créé un tuple
 - Un tuple, est une liste qui ne bougera pas.
 - Aucune suppression, aucun ajout, aucune modification. C'est un peu une "constante" sous forme de tableau.

Les objets - liste, tuple et dictionnaire

```
def someCalc(a, b):  
    return a+b, a-b  
  
print someCalc(4, 1)
```

```
def someCalc(a, b):  
    return a+b, a-b  
  
add, minus = someCalc(4, 1)  
  
print add  
print minus
```

```
fdoyen@f  
(5, 3)  
fdoyen@f  
5  
3  
fdoyen@f
```

Les objets - liste, tuple et dictionnaire

- Sur bien des points, un dictionnaire ressemble à une liste. Ce n'est finalement qu'une liste ne fonctionnant pas avec des indices mais avec des clés

```
dictionary = {  
    "fruits" : ["Orange", "Apple"],  
    "cars" : ["BMW", "Peugeot", "Kia"]  
}  
  
print dictionary["fruits"] # shows ["Orange", "Apple"]
```

5. Python

Les packages natifs

Les packages natifs

```
import os, math

cwd = os.getcwd()

print cwd # Shows for example : /home/userdir/subdir/python
print math.sqrt(16) # Shows 4.0|
```

<https://packaging.python.org/tutorials/installing-packages/>

6. Python

POO

Les classes

```
from monpackage import malib
from monpackage import simplymath
from monpackage.User import User
```

```
# coding: utf-8
```

```
from monpackage import *
```

```
florian = User("Florian", "Doyen", 30, "Formateur")
print florian.getJob()
florian.setJob("Dev. Web")
print florian.getJob()
```

```
class User:
```

```
    def __init__(self, firstname, lastname, age, job):
        self.firstname = firstname
        self.lastname = lastname
        self.age = age
        self.job = job
```

```
    def getJob(self):
        return self.job
```

```
    def setJob(self, job):
        self.job = job
```

```
Formateur
Dev. Web
```

Formateur
30
Dev. Web

Les classes - Héritage

```
main.py x
# coding: utf-8
```

```
from monpackage import *

florian = Man("Florian", "Doyen", 30, "Formateur")
print florian.getJob()
print florian.getAge()
florian.setJob("Dev. Web")
print florian.getJob()
```

```
User.py x
```

```
1 class Human:
2     def __init__(self, firstname, lastname, age):
3         self.firstname = firstname
4         self.lastname = lastname
5         self.age = age
6
7     def getAge(self):
8         return self.age
9
10    def setAge(self, age):
11        self.age = age
```

```
__init__.py x
from monpackage import malib
from monpackage import simplymath
from monpackage.User import Human
from monpackage.Man import Man
```

```
Man.py x
```

```
1 from monpackage import Human
2
3 class Man(Human):
4     def __init__(self, firstname, lastname, age, job):
5         self.job = job
6         Human.__init__(self, firstname, lastname, age)
7
8     def getJob(self):
9         return self.job
10
11    def setJob(self, job):
12        self.job = job
```

Les classes - Rappel sur isinstance()

`isinstance(obj, class)`

Exemple : `isinstance(florian, Human) # true`

Exemple : `isinstance(florian, Man) # true`

Eh oui ! florian est un objet de type “Man” mais il hérite de “Human”, donc il est aussi un “Human”

Les classes - héritage multiple

- Class Enfant(Parent1):
- Class Enfant(Parent1, Parent2):

Les classes - Exception personnalisée

- On fait hériter notre classe de “Exception”
- On crée deux méthodes :
 - `__init__` qui stock le message
 - Init sera un setteur
 - `__str__` qui affichera le message
 - Str sera un getteur
 - On peut également ajouter un fichier, une ligne, afin de préciser l'erreur
- Ensuite, il suffira d'appeler cette classe dans le `try...except` au moment souhaité