Algorithmique Appliquée

BTS SIO SISR

Bibliothèques Python

Yvo Solutions
Loïc Yvonnet

Plan

- Programmation modulaire
- Bibliothèque standard
- Focus sur les fichiers
- Gestionnaire de paquets
- Discussion sur les licences

Programmation modulaire

Taille d'une fonction

- Combien de **responsabilités** doit avoir une fonction ?
- 1 fonction ➡️ 1 responsabilité.
- En moyenne, une fonction doit faire entre **7 et 15 lignes**.
- Une fonction qui fait plus de 30 lignes doit être découpée en **plusieurs fonctions plus simples**.

Exemples de responsabilités

- Effectuer un calcul (ex : racine carrée).
- Rechercher une valeur (ex : recherche binaire)
- Appliquer une transformation (ex : nombre textuel vers numérique).
- Afficher un résultat (ex : une matrice).

Ne pas se répéter

- Lorsque l'on observe un **motif qui se répète** dans le code, il y a un problème.
- Ces répétitions sont le signe d'une **duplication de code**.
- A la place, il faut créer des fonctions et les appeler.
- Le processus de modification du code pour supprimer les duplications s'appelle la **refactorisation**.
- Il s'agit d'une bonne pratique du génie logiciel.

Notion de script modulaire

- On rassemble les fonctions de même nature dans un **script**.
- Par exemple, on peut avoir un script `racine.py` qui contient les fonctions `racine_carree` et `racine_cubique` .
- On pourrait avoir un autre script nommé `chaine_caracteres.py` qui contient des fonctions de manipulation de chaînes de caractères.

Module

- Un **module** est un fichier `.py` contenant des définitions et des déclarations Python.
- Le script `racine.py` est donc un module.

Module `racine.py`

racine.py - Visual Studio Code

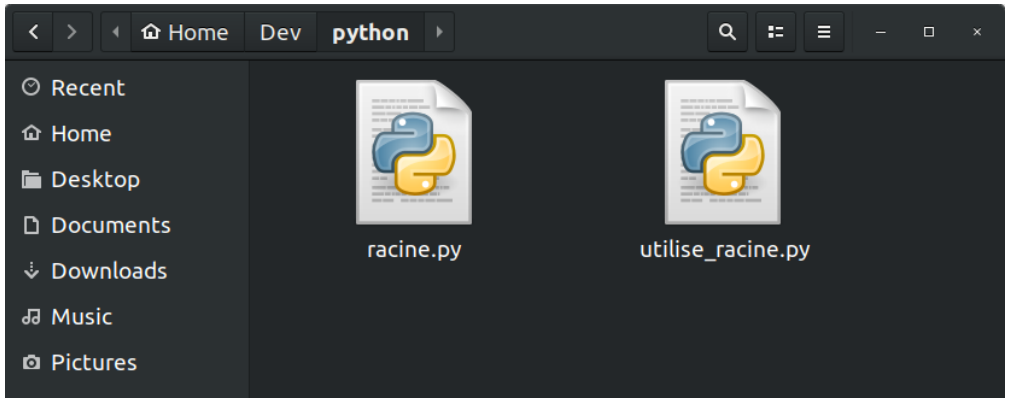
File Edit Selection View Go Run Terminal Help

racine.py

home > lyvonnet > Dev > python > racine.py > racine_carree

```python
def racine_carree(x, epsilon=0.000001):
    """Renvoie la racine carrée de x à epsilon près.

    Calcule la racine carrée d'un nombre x positif en employant
    l'algorithme de Newtown-Raphson.
    x - nombre flottant positif ou nul.
    epsilon - nombre flottant strictement positif.
    Retourne une valeur proche de la racine carrée de x, à plus
    ou moins epsilon près.
    """
    s = x / 2
    while abs(s ** 2 - x) >= epsilon:
        P = s ** 2 - x
        P_prime = 2 * s
        s = s - P / P_prime

    return s

def racine_cubique(x, epsilon=0.000001):
    """Renvoie la racine cubique de x à epsilon près.

    Calcule la racine cubique d'un nombre x en employant
    l'algorithme de Newtown-Raphson.
    x - nombre flottant.
    epsilon - nombre flottant strictement positif.
    Retourne une valeur proche de la racine cubique de x,
    à plus ou moins epsilon près.
    """
    positif = True
    if x < 0:
        positif = False
        x = -x
    s = x / 2
    while abs(s ** 3 - x) >= epsilon:
        P = s ** 3 - x
        P_prime = 3 * s ** 2
        s = s - P / P_prime
    if not positif:
        s = -s

    return s
```

feature/lecture_10 Python 3.9.7 64-bit ⊗ 0 ⚠ 0 Ln 1, Col 1 Spaces: 4 UTF-8 LF Python



Fichier `racine.py`

Explorateur de fichiers

Utilisation d'un module

- On emploie le mot clé `import` pour utiliser une bibliothèque de fonctions.
- Par exemple, avec le module `racine.py` :

```
import racine

cinq = racine.racine_carree(25)
print(cinq)

trois = racine.racine_cubique(27)
print(trois)
```

Autre exemple avec `math`

```
import math

print(math.cos(0))
```

```
1.0
```

Intérêt du préfixe

- Lorsque l'on importe un module, on doit employer `module.f()` pour appeler la fonction `f` définie dans `module.py`.
- On appelle ce préfixe une **qualification complète** (fully qualified 🇬🇧).
- L'objectif est d'éviter la **collision de nom**.
- En effet, plusieurs modules peuvent définir une fonction `f`.

Importer une fonction spécifique

```
from racine import racine_carree

cinq = racine_carree(25)
print(cinq)
```

```
5.000000000016778
```

⚠️ Les autres fonctions sont invisibles

```
from racine import racine_carree

trois = racine_cubique(27)
print(trois)
```

```
NameError: name 'racine_cubique' is not defined
```

⚠️ Le module aussi est invisible

```
from racine import racine_carree

trois = racine.racine_cubique(27)
print(trois)
```

⬇️

```
NameError: name 'racine' is not defined
```

Tout importer ? 💀

```
from racine import *

cinq = racine_carree(25)
print(cinq)

trois = racine_cubique(27)
print(trois)
```

⬇️

```
5.000000000016778
3.000000000000002
```

Inconvénients de tout importer

- On perd l'avantage de la **qualification complète**.
- On peut donc avoir des **collisions de noms**.
- Le code est également plus **difficle à comprendre** car on ne sait pas d'où viennent les fonctions utilisées.

Alias

```python
import racine as rcn

cinq = rcn.racine_carree(25)
print(cinq)

trois = rcn.racine_cubique(27)
print(trois)
```

```
5.000000000016778
3.000000000000002
```

Importer une liste d'objets

```
from math import cos, sin, pi

print(cos(0))
print(cos(pi))
print(sin(0))
print(sin(pi))
```

```
1.0
-1.0
0.0
1.2246467991473532e-16
```

La fonction principale (1/2)

- Lorsque l'on exécute un script en ligne de commande, l'interpréteur assigne la chaîne de caractère `"__main__"` à la variable globale `__name__` .
- Cela permet de distinguer le cas où un script est importé avec `import`, du cas où un script est exécuté indépendemment.

La fonction principale (2/2)

```
def main():
    # On peut par exemple tester le bon fonctionnement de
    # racine_carree et racine_cubique ici.
    pass

if __name__ == "__main__":
    # Exécuté uniquement si le script est lancé en ligne de commande
    main()
```

Tour d'horizon de la bibliothèque standard Python

Bibliothèque standard

- Le **langage python** est constitué de l'ensemble des mots clés tels que `while`, `for`, `if`, de sa grammaire et de quelques fonctions de base comme `max` et `range`.
- Pour effectuer des opérations plus avancées, on doit utiliser des modules de la **bibliothèque standard**.
- Par exemple : le module `math`.

Services offerts (1/4)

- Collections supplémentaires.
- Hiérarchie d'exceptions.
- Traitement de texte.
- Fonctions mathématiques.
- Module de programmation fonctionnelle.

Services offerts (2/4)

- Système de fichiers.
- Persistence dans une base de données.
- Algorithmes de compression.
- Algorithmes de cryptographie.
- Services de gestion de système d'exploitation.

Services offerts (3/4)

- Concurrence et parallélisation (multithreading).
- Réseau et communication inter-processus.
- Protocoles Internet.
- Parsers (HTML, XML, JSON, etc.).
- Services multimedia.

Services offerts (4/4)

- Internationalisation (i.e. plusieurs langues).
- Interface Graphique avec Tk.
- Outils de développement et de débogage.
- Gestionnaire de paquets.
- Interpréteur Python (AST, Tokenizer, etc.).

Byte

Exemple de collections supplémentaires

```
octets = b"1 2 3"
print(octets)

chaine = octets.decode()
print(chaine)

liste = octets.split()
print(liste)
```

⬇️

```
b'1 2 3'
1 2 3
[b'1', b'2', b'3']
```

Expression régulière

Exemple de traitement de texte

```
import re

chaine = "si ton tonton tond ton tonton, ton tonton sera tondu"
expression = r"\bton[a-z]*"
resultat = re.findall(expression, chaine)
print(resultat)
```

```
['ton', 'tonton', 'tond', 'ton',
 'tonton', 'ton', 'tonton', 'tondu']
```

Nombres complexes

Exemple de fonctions mathématiques

```
import cmath

c = complex(cmath.e, 0.0) # e
print(c)
print(cmath.log(c))       # 1

c = complex(cmath.e, 1.0) # e + i
print(c)
print(cmath.log(c))
```

```
(2.718281828459045+0j)
(1+0j)
(2.718281828459045+1j)
(1.0634640055214861+0.35251342177761 9j)
```

SQLite - Exemple de persistence (1/2)

```
import sqlite3

# Création d'une base de données en mémoire
connection = sqlite3.connect(':memory:')
curseur = connection.cursor()

# Création d'une table
curseur.execute("""CREATE TABLE personnage
                   (nom, prenom, age)""")

# Insertion d'une ligne dans la table
curseur.execute("""INSERT INTO personnage
                    VALUES ('Tyrion', 'Lannister', 27)""")

# Commit la transaction actuelle
connection.commit()

# Suite sur la diapositive suivante...
```

SQLite - Exemple de persistence (2/2)

```
# Parcourt des personnages dans la BD
personnages = curseur.execute("SELECT * FROM personnage")
for perso in personnages:
    print(perso)

# Fermeture de la connection (important)
connection.close()
```

```
('Tyrion', 'Lannister', 27)
```

ZLib

Exemple d'algorithmes de compression

```
import zlib

chaine = "Lorem Ipsum blabla"
octets = chaine.encode()

compresse = zlib.compress(octets, zlib.Z_BEST_COMPRESSION)
print(compresse[:11])

decompresse = zlib.decompress(compresse)
print(decompresse[:11])
```

```
b'x\xda\xf3\xc9/J\xcdU\xf0,('
b'Lorem Ipsum'
```

Hash

Exemple d'algorithmes de cryptographie

```python
import hashlib

hash = hashlib.sha512(b"Personne ne doit savoir...")
hexa = hash.hexdigest()
print(hexa[:15])
```

```
c8c43d2d251edb0
```

Lancement d'une commande système

Exemple de services de SE

```
import os

if os.name == "posix":
    os.system("ls")
else:
    os.system("dir")
```

```
assets   cours   environment.yml   LICENSE          node_modules
bin      dist    includes          marp.config.js   now.json
```

Lancement d'un fil d'exécution

Exemple de concurrence

```
from threading import Thread

def bonjour():
    print("bonjour depuis le thread")

thread = Thread(None, bonjour)
thread.start()
thread.join()
```

```
bonjour depuis le thread
```

Echange TCP - Exemple réseau (1/2)

Serveur socket TCP

```python
import socket

# Lie une socket sur l'IP 127.0.0.1 et le port 8080
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('127.0.0.1', 8080))
s.listen(1)
connexion, _ = s.accept()

# Attend un message
while True:
    message = connexion.recv(1024)
    if not message: break
    connexion.sendall(message)

# Ferme la socket et la connexion (important)
connexion.close()
s.close()
```

Echange TCP - Exemple réseau (2/2)

Client socket TCP

```python
import socket

# Ouvre une socket sur l'IP 127.0.0.1 et le port 8080
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('127.0.0.1', 8080))

# Envoi un message
s.sendall(b'Bonjour')

# Attend une réponse
reponse = s.recv(1024)

# Ferme la socket (important)
s.close()

print(reponse)
```

Requête HTTP - Exemple de Protocoles Internet

```python
import http

# Créé une connexion HTTP
connexion = http.client.HTTPSConnection('yvo.solutions')

# Envoie une requête GET pour récupérer la page principale
connexion.request("GET", "/")

# Récupère et affiche la réponse
reponse = connexion.getresponse()
print(reponse.status, reponse.reason)
print(reponse.read()[:15])

# Ferme la connexion HTTP
connexion.close()
```

⬇️

```
200 OK
b'<!doctype html>'
```

Désérialisation JSON - Exemple de Parsers

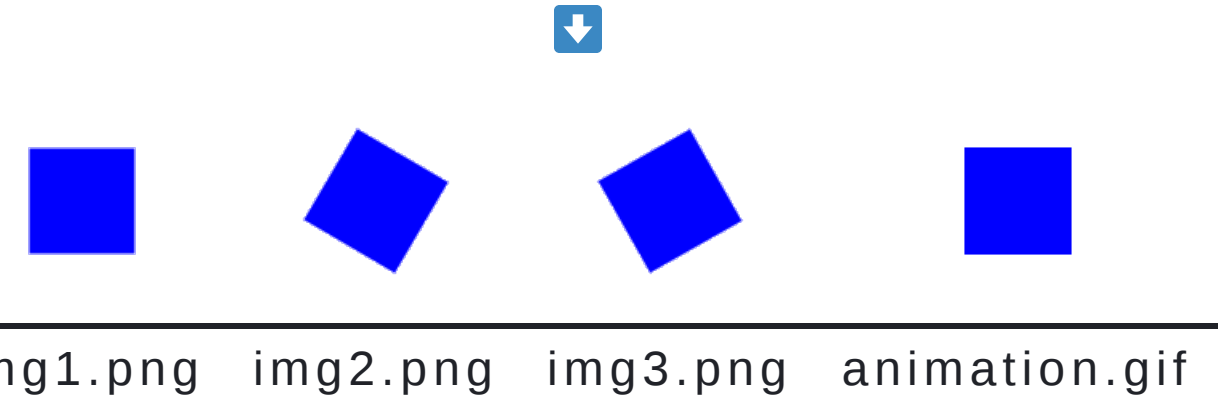
```python
import json

# Désérialise une chaîne représentant un objet JSON
chaine1 = """{ "nom": "Tyrion", "prenom": "Lannister", "age": 27 }"""
dico = json.loads(chaine1)

# Applique une modification
dico["adresse"] = "Lannisport"

# Sérialise en chaine représentant un object JSON
chaine2 = json.dumps(dico, sort_keys=True, indent=4)
print(chaine2)
```

```
{
    "adresse": "Lannisport",
    "age": 27,
    "nom": "Tyrion",
    "prenom": "Lannister"
}
```

Création d'un gif animé

Exemple de Services multimedia

```
import imageio

fichiers = ["img1.png", "img2.png", "img3.png"]
animation = "animation.gif"
images = [imageio.imread(fichier) for fichier in fichiers]
imageio.mimsave(animation, images, fps=3)
```



Affichage de l'AST - Exemple d'interpréteur Python

```python
import ast

noeud = ast.parse("[1, 2, 3]", mode='eval')
arbre = ast.dump(noeud, indent=4)
print(arbre)
```

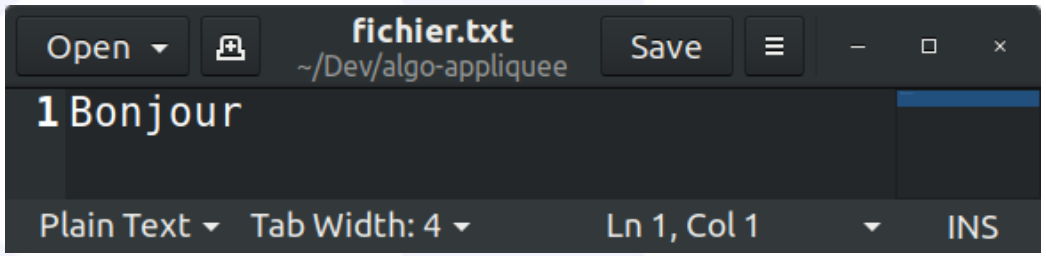
```
Expression(
    body=List(
        elts=[
            Constant(value=1),
            Constant(value=2),
            Constant(value=3)],
        ctx=Load()))
```

Focus sur les fichiers

Fichiers

- Tous les principaux systèmes d'exploitation offre un **système de fichiers**.
- Cela permet de **sauvegarder** des données.
- Python offre de nombreux services pour **manipuler les fichiers**.

Ouverture et fermeture




```
# Ouvre le fichier
fichier = open("fichier.txt")

# Ferme le fichier (important)
fichier.close()
```

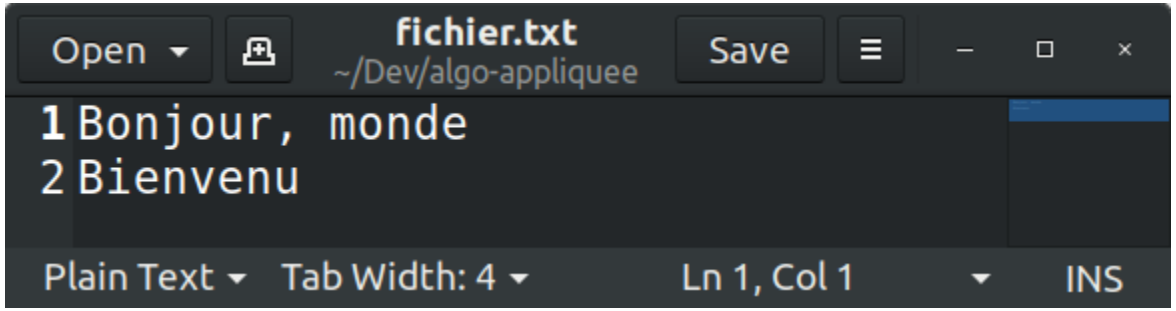
Lecture

```
fichier = open("fichier.txt")

for ligne in fichier:
    print(ligne)

fichier.close()
```

```
Bonjour
```

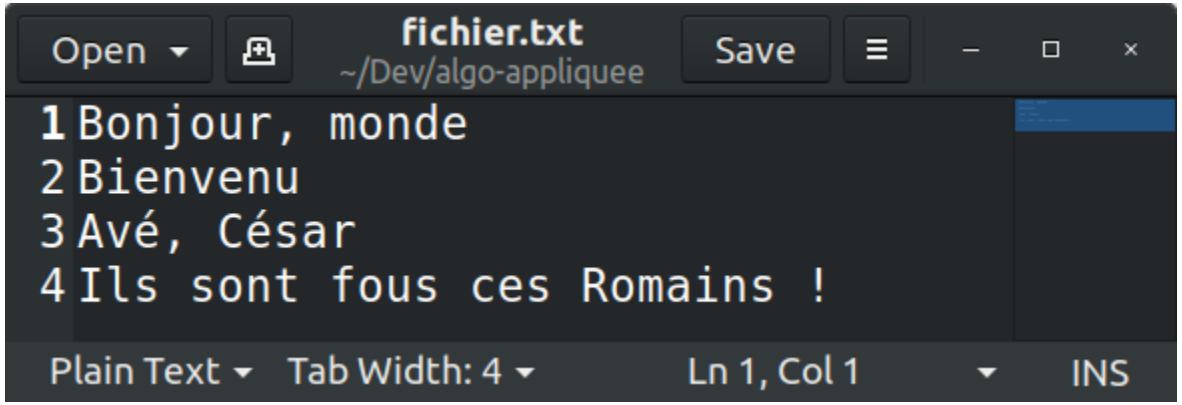
Ecriture

```
fichier = open("fichier.txt", "w")

fichier.write("Bonjour, monde\n")
fichier.write("Bienvenu\n")

fichier.close()
```



Ajout à la fin

```
fichier = open("fichier.txt", "a")

fichier.write("Avé, César\n")
fichier.write("Ils sont fous ces Romains !\n")

fichier.close()
```



Buffer et `flush`

- Un appel à `write` ne produit *pas* une écriture immédiate sur disque.
- Cette manière de fonctionner serait **trop lente** en pratique.
- A la place, il existe des **caches intermédiaires**.
- Pour demander une écriture sur disque, on doit appeler `flush`.

Garantir la fermeture avec `with`

- Pour éviter les **fuites de ressources**, on peut utiliser `with`.
- A la fin du bloc `with`, le fichier est **automatiquement fermé**.
- Il est conseillé de toujours utiliser `with` lorsqu'une action de fin est obligatoire.

```
with open("fichier.txt") as fichier:
    for ligne in fichier:
        print(ligne)
```

TP : Initiation aux fichiers

TP : Initiation aux fichiers

Lien vers le sujet de TP.

Introduction aux paquets

Gestionnaire de paquets `pip`

Au-delà de la bibliothèque standard

- La bibliothèque standard Python **ne couvre pas tous les besoins**.
- La bibliothèque standard n'a **pas vocation** à couvrir tous les besoins.
- Il existe un **écosystème riche** de **bibliothèques tierces**.
- Comment utiliser ces bibliothèques tierces ?

Gestionnaire de paquets

- Les différentes bibliothèques sont distribuées sous forme de **paquets**.
- L'**installation d'un paquet** permet l'utilisation de la biobliothèque.
- On peut alors `import`er les modules de la bibliothèque tierce.
- L'installation se fait via un **gestionnaire de paquets**.

`pip` et `conda`

- Il existe différents gestionnaires de paquets pour Python.
- `pip` et `conda` sont les plus utilisés.
- Nous allons utiliser `pip` dans ce cours.

Syntaxe

```
# Installe paquet
python3.9 -m pip install paquet

# Met à jour paquet
python3.9 -m pip install -U paquet
```

Quelques exemples (1/2)

- **NumPy** : tableau à N dimensions et algèbre linéaire.
- **MatPlotLib** : affichage de graphes 2D et 3D.
- **Panda** : tableur et analyse statistique.
- **SimPy** : résolution d'équations symboliques.
- **scikit-image** : traitement d'images.

Quelques exemples (2/2)

- **graph-tool** : analyse de graphes.
- **Django** : bibliothèque web pour faire un site web.
- **FastAPI** : bibliothèqe pour écrire des web services.
- **FastAI** : réseaux de neurones profonds.
- **TensorFlow** : apprentissage par machine.

Discussion sur les licences

Logiciel Gratuit

- Vous avez le droit d'installer et d'utiliser gratuitement le logiciel.
- Vous n'avez pas forcément accès au code source.
- Le logiciel reste soumis aux droits d'auteur.
- Par défaut, vous n'avez notamment pas le droit de redistribuer le logiciel.

Logiciel Open Source

- **Open Source** veut simplement dire que le code source est à disposition pour **être lu**.
- Il existe de nombreuses licences Open Source **plus ou moins permissives**.
- **Vous ne pouvez *pas* faire ce que vous voulez**.
- Un logiciel propriétaire et payant peut être Open Source.

Logiciel Libre

- Le logiciel **Libre** est obligatoirement **Open Source**.
- Tout le monde a le droit de lire, modifier et redistribuer des logiciels libres.
- Les modifications d'un logiciel libre doivent être mises à disposition de toute la communauté.
- Si on utilise le code source d'un logiciel libre, notre logiciel devient de facto libre également.
- Cela ne signifie pas que le logiciel est **libre de droits**.

GPL

- Célèbre licence de **Richard Stallman** et de la **Free Software Foundation**.
- Licence utilisée pour le **projet GNU** (GCC, GTK, etc.) et le **kernel Linux**.
- C'est *la* licence libre **la plus restrictive**.
- Si vous utilisez du code sous GPL, votre code est sous GPL.
- Toujours consulter le service juridique d'une entreprise avant d'envisager utiliser du code sous GPL.

LGPL

- C'est la *Lesser* GPL.
- Cette licence est beaucoup moins restrictive.
- Du code propriétaire est en droit d'utiliser du code sous LGPL.
- C'est notamment le cas des programmes en espace utilisateur sous GNU/Linux.

MIT

- Initialement une alternative à la GPL.
- Issue de la célèbre université de Boston.
- Utilisation possible sans problème dans des projets industriels.

Les autres...

BSD, Apache, ...

- **A chaque fois** que vous voulez utiliser une nouvelle bibliothèque, **regardez la licence associée**.
- S'il n'y a pas de licence, cherchez une **alternative**.
- **Lisez les restrictions** associées à cette licence.
- Si vous pensez que les restrictions sont trop contraignantes, cherchez une autre bibliothèque.
- **Demandez à votre manager** ou **au service juridique** si vous avez le droit d'utiliser cette licence.

TP : Courbes et traitement d'images

TP : Courbes et traitement d'images

Lien vers le sujet de TP.

Programmation Orientée Object

Optionnel (hors programme)

Classe

- Une **classe** regroupe données et fonctions agissant sur ces données.
- Les données s'appellent **données membres**.
- Les fonctions s'appellent **méthodes**.
- **Encapsulation** : certaines données membres et méthodes sont *privées*.

Devoir à la Maison 05

DM : Plus de modules

Lien vers le sujet de DM.