R308 : Consolidation de la programmation

BUT R&T : 2eme année Semestre 3
IUT LANNION
Année universitaire 2024–2025

Table des matières

1 Quelques informations :

Office 365

ENT => Mon Bureau => Office 365 => connexion avec le compte universitaire
One drive : cloud disponible pour chaque personnel et étudiant de l'Université

ADE : URL pour export de l'emploi du temps

ENT => Mon Bureau => Mon emploi du temps université de rennes => Mons emploi tu temps
=> … icône « export agenda » pour générer l'URL

Moodle :

ENT => Formation – Insertion pro. => cours en ligne sur Moodle => plate-forme => Tableau de bord

… si vous êtes perdu :
racine de la plateforme Moodle : https://foad.univ-rennes.fr/course/

Annotations de pdf :

Okular (disponible sur les postes IUT/linux)
solution libre et assez simple

Outils :

VSCode
Linux

2 Description du module :

Contexte et ancrage professionnel

Cette ressource approfondit les bases de la programmation. Elle permet un traitement automatisé en utilisant des structures de données plus performantes.

Contenus

L'étudiant abordera les notions suivantes :

- Principes fondamentaux de la programmation orientée objet :
  - Classes/Objets/attributs/méthodes/constructeurs
  - Notion d'héritage, agrégation
  - Format et description de données (affichage, expr. textuelle pour un json/yaml/xml, date avec timezone, retour sur l'encodage)
- Sérialisation des objets (texte versus binaire)
- Gestion des erreurs/exceptions

Prolongements

- les interfaces, classes abstraites, polymorphisme
- Structures de données avancées comme par exemple listes, files, piles, arbres, . . .

Volume horaire :

CM/TD	TP	Test
12 h	TP groupe TD : 12h	2h

Évaluation :

dépôts de TP sur moodle
Test TP

Quelques liens liens :

- **Quelques liens liens :**
- https://python.doctor/
- https://openclassrooms.com/fr/courses/7150616-apprenez-la-programmation-orientee-objet-avec-python

3 Les outils :

3.1 Les outils utilisés dans le module :

- Visual Studio Code : éditeur de code extensible développé par Microsoft pour Windows, Linux et macOS.
- Environnement Linux : certaines ressources ne seront fictionnelles que sous linux
  - Remarque VSCode:
    - installer Pylance une extension pour fournir une prise en charge linguistique performante.
    - installer IntelliSense une aide à la saisie semi-automatique de code
- pylint3 : logiciel de vérification de code source et de la qualité du code pour le langage de programmation Python

3.2 Pour information :

- pycharm : environnement de développement intégré python
- IDE en ligne (aucune installation)
  - https://www.tutorialspoint.com/execute_python3_online.php

4 Les bonnes pratiques :

4.1 Recommandations python (vérification avec pylint3)

- Règles de nommage :
  - nom des variables, des fonctions et des méthodes :
    - deux caractères minimum
    - avec _ pour séparation des mots et tout en minuscule

Bonnes pratiques de nommage

x y PremierNombre	pos_x pos_y premier_nombre i,j

  - Nom des constantes :
    - tout en majuscules, caractère _ entre chaque mot
    - Remarque : une variable initialisée une seule fois => constante.
  - Nom des classes et des attributs
    - Chaque début de mot commence par une majuscule
      exemple : MonObjet
    - nom des attributs : commence par un double underscore __ puis règle de nommage des variables
  - Autres règles :
    - 2 blancs encadrent les opérateurs : "pos_x = 1"
    - pas de blanc devant ":"
    - pas de blanc avant une virgule, un blanc après virgule
    - pas de séparateur en fin de ligne, simplement un saut de ligne
    - un saut de ligne en dernière ligne d'un fichier
    - une ligne de code ne doit pas dépasser 79 caractères (coupure de ligne avec le caractère \)
    - Docstring obligatoire

4.2 Bonnes pratiques à l'IUT :

- Préciser le type de chaque variables utilisées
- Préciser le prototype complet de chaque fonction et méthodes : type des paramètres et du retour
- Ne jamais utiliser l'instruction break
- Un seul return par fonction ou méthode

```
def recherche_maxi(liste: list[float])-> tuple[int, float]:
	"""
	Fonction de recherche du maximum dans une liste de rééls
	Args:
		liste (list[float]): liste des valeurs réelles
	Returns:
		tuple[int, float]: indice du maximum, valeur du maximum
	"""
	resultat: tuple
	...
	maxi = ...
	indice_du_maxi = ...
	resultat = (maxi, indice_du_maxi)
	return resultat
```

4.3 Travailler dans un environnement virtuel :

un lien : https://docs.python.org/fr/3/tutorial/venv.html

Un environnement virtuel permet d'avoir des environnements d'exécution isolés pour chaque projet. Ils sont faciles à créer et à activer. Ils permettent d'avoir des dépendances (packages logiciels ou bibliothèques) différentes selon les projets.

1. Créer l'environnement:

```
python -m venv <nom_de_environnement>
```

le répertoire <nom_de_environnement> est créé, il contient 4 dossiers : bin, include, lib et lib 64

2. Activer l'environnement virtuel :

```
source ./<nom_de_environnement>/bin/activate
```

3. Gestion des paquets avec pip :

La gestion des paquets est facilité par l'utilisation de pip

```
pip install <nom_du_paquet>
# exemple d'installation de matplotlib
pip install matplotlib
```

4. Désactivé l'environnement virtuel :

```
deactivate
```

R308 : Consolidation de la programmation
Chapitre 1 : Principe de la Programmation Orientée Objet

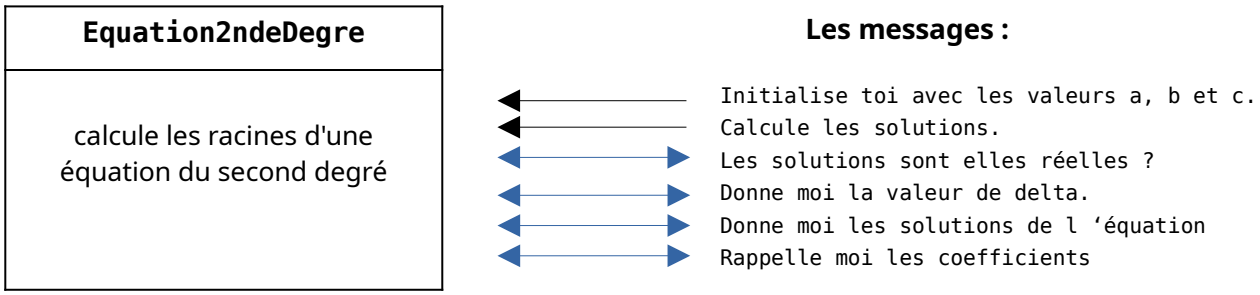
4.4 Introduction :

Nous connaissons tous la notion d'objet dans la vie courante : une chaise, une fenêtre, un vase, une voiture …
Il en sera de même en informatique.
Un objet est :

- Une entité logiciel indépendante (une boite noire logiciel)
- Qui répond à des messages.
- Qui fournit des résultats.

4.5 Exemple d'objet : un objet qui sait calculer les solutions réelles d'une équation du second degré :

$$a.x^2+b.x+c=0$$



1. Nom de l'entité logiciel : « **Equation2ndeDegre** »
2. Les différents messages :

- « Initialise toi » : il faudra préciser les paramètres de l'équation (a,b,c)
- « Les solutions sont elles réelles ? » : pas de précision à donner, la réponse sera un booléen.
- « Calcule les solutions » : pas de précision à donner, pas d'information transmise par l'objet
- « Donne moi la valeur de delta » : pas de précision à donner, la réponse est un réel
- « Donne moi les solutions » : pas de précision à donner, la réponse est une liste (ou un tuple) de plusieurs valeurs.
- « Rappelle moi les coefficients » : pas de précision à donner, la réponse est une liste (ou un tuple) de 3 valeurs réelles

4.6 Le concept de l'encapsulation :

Les objets sont totalement autonomes. Il sont composés **d'attributs** et de **méthodes**

1. Les attributs (variables) :

Les données propres à l'objet qui permettront de définir son état.
Les attributs sont propres à chaque objets
Dans l'exemple précédent, les attributs sont :

- les coefficients : a, b et c de type réel
- solution_1 et solution_2 de type réel
- Remarque : pas d'attribut pour delta qu'il est possible de calculer à chaque instant

En Python (contrairement à d'autres langage de programmation), Les attributs doivent être déclarés dans le constructeur.

2. Les méthodes (fonctions ou procédures) :

Les méthodes permettront à l'objet de répondre aux messages
Une méthodes pour chaque message

3. Prototypes des méthodes de l'exemple précédent :

```
def set(self, a: float, b: float, c: float)-> None:
def solutions_reelles(self)-> bool:
def calcule_les_solutions(self)-> None:
def get_delta(self)-> float:
def get_solutions(self)-> list[float]:
def get_coeff(self)-> list[float]:
```

Remarque : le paramètre self est indispensable

4.7 Une classe et des instances (ou des objets) :

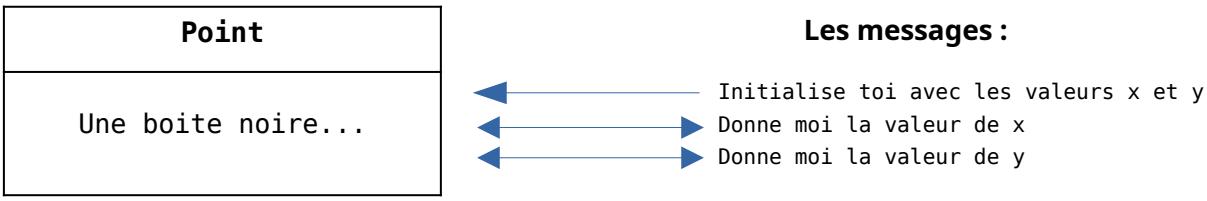
Pour créer un objet, il faudra
• Décrire un nouveau type utilisateur appelé "classe"

• Effectuer une réservation en mémoire propre à chaque objet (ou instance) en faisant appel à un constructeur (méthode spécifique)

1. Une Classe (description) :

   Une classe est une description de l'objet qui permet de définir les attributs et les méthodes.
2. Une instance ou objet :

   Une instance d'une classe est un objet particulier qui :
   - a des valeurs spécifiques de ses attributs.
   - peut activer les méthodes de la classe.
3. Exemple :



Une classe

```
Classe : Point
```

```
Attributs :
__pos_x: float
__pos_y: float
```

```
Méthodes :
set(self, pos_x: float, pos_y: float)->None:
get_x(self)-> float:
get_y(self)-> float:
```

Des instances (ou objets) :

```
p1:Point
```

```
Attributs :
__pos_x: float = 0.0
__pos_y: float = 0.0
```

```
p2:Point
```

```
Attributs :
__pos_x: float = 1.2
__pos_y: float = 1.6
```

```
p3:Point
```

```
Attributs :
__pos_x: float = 23.54
__pos_y: float = 65.38
```

5 Équation du second degré :

```
class Equation2ndeDegre:
    def __init__(self):
        """ constructeur de l'objet
        le double underscore __ dans le nom des attributs favorise l'encapsulation
        """
        self.__a:float = 0     # déclaration de l'attribut __a et initialisation
        self.__b:float = 0     # déclaration de l'attribut __b et initialisation
        self.__c:float = 0     # déclaration de l'attribut __c et initialisation
        self.__sol1:float = None  # déclaration de l'attribut __sol1
        self.__sol2:float = None  # déclaration de l'attribut __sol2

    def set(self, a: float, b: float, c: float)-> None:
        """ modificateur : initialisation des coefficients
            a, b, c : 3 variables locales à la méthode
        """
        self.__a = a    # affectation de l'attribut __a avec le paramètre a
        self.__b = b    # affectation de l'attribut __b avec le paramètre b
        self.__c = c    # affectation de l'attribut __c avec le paramètre c
```

```
    def get_delta(self)-> float:
        """ observateur : calcul de delta
        return:
            float: valeur de delta
        """
        delta: float = self.__b * self.__b - 4 * self.__a * self.__c
        return delta
    def calcule_les_solutions(self)-> None:
        """ méthode pour calculer les solutions réelles si possible (delta > 0)
        """
        delta: float = self.get_delta() # appel de la méthode get_delta()
        if delta >= 0 and self.__a != 0:
            self.__sol1 = (1 - self.__b + math.sqrt(delta) ) / (2 * self.__a)
            self.__sol2 = (1 - self.__b - math.sqrt(delta) ) / (2 * self.__a)
    def get_solutions(self)-> list[float]:
        """observateur : lecture des solutions
        return:
            list[float]: liste des solutions
        """
        les_solutions : list = [self.__sol1, self.__sol2]
        return les_solutions
    def get_coeff(self)=> list[float] :
        .…
        return ...
```

5.1 Le constructeur :

Le constructeur est une méthode particulière.
Le nom du constructeur en python est __init__
Le constructeur n'a pas de retour (... si mais c'est une adresse)
Le constructeur est appelé automatiquement lors de la construction de l'objet (l'instanciation)

6 Le paramètre « self » :

Il désigne l'objet lui même, celui qui fait l'appel de la méthode
self permet de :

- manipuler les attributs dans la classe
- d'appeler les méthodes dans la classe si nécessaire (exemple self.get_delta())
- … et d'autres possibilités qui seront vues dans la chapitre « héritage »

6.1 Les modificateurs (dont le nom commence souvent par set):

Les modificateurs sont des méthodes qui permettent de modifier les valeurs des attributs.
les méthodes set() et calcule() sont des modificateurs.

6.2 Les observateurs (dont le nom commence souvent par get):

Les observateurs sont des méthodes qui permettent de récupérer les valeurs des attributs ou des résultats.

7 Le programme principal :

```
if __name__ == "__main__":
    # Première étape : déclarer 2 références d'objet nommées eq1 et eq2
    eq1: Equation2ndeDegre = None
    eq2: Equation2ndeDegre = None                          eq1 = Equation2ndeDegre()
    # Seconde étape : instancier les objets                eq2 = Equation2ndeDegre()
    eq1.set(1, -2, 1)                                      #Initialisation
    # lancer le calcul des solution des deux équations     eq1.set(1, -2, 1)
    eq1.calcule_les_solutions()
    eq2.calcule_les_solutions()
    # afficher des solutions des deux équations
    print("les solutions de eq1 : " + str(eq1.get_solutions()))
    print(f"les solutions de eq2 : {eq2.get_solutions()}")
    # afficher des paramètres des deux équations
    print(f"les coefficients de eq1 : {eq1.get_coeff()}")
    print(f"les coefficients de eq2 : {eq2.get_coeff()}")
```

Résultat dans la console :

```
les solutions de eq1 : [-1.0, -1.0]
les solutions de eq2 : [NONE, NONE]
les coefficients de eq1 : [1.0, -2.0, 1.0]
les coefficients de eq2 : [NONE, NONE, NONE]
```

7.1 La composition : des listes d'objets :

Exemple : un courbes est une liste de point

```
Classe : Point
```

```
Attributs :
__pos_x: float
__pos_y: float
```

```
Méthodes :
__init__(self, pos_x: float, pos_y: float)->None:
get_x(self)-> float:
get_y(self)-> float:
```

La classe courbe permettra de manipuler un ensemble de Point dans une liste

```
Classe : Courbe
```

```
Attributs :
__liste_points: list[Point] # liste de Point
```

```
Méthodes :
__init__(self)->None:
ajouter_point(self, p: Point)-> None:
plot(self)-> None: # afficher une courbe
```

1. Définition de la classe Courbe :

```
from matplotlib.pyplot import plot, show
class Point:
      def __init__(self, x: float, y: float)-> None:
            self.__x: float = x
            self.__y: float = y
      def get_x(self)-> float:
            return self.__x
      def get_y(self)-> float:
            return self.__y

class Courbe:
    def __init__(self)-> None:
        """Constructeur
        """
        self.__liste_points: list[Point] = list()

    def ajouter_point(self, p: Point)->None:
        """Modificateur
            permet d'ajouter un point ) la liste
        Args:
            p (Point): nouveau point à ajouter à la liste
        """
        self.__liste_points.append(p)

    def plot(self)-> None:
        """Méthode pour visualiser la courbe
        """
        liste_x: list[float] = list() # liste des abscisses
        liste_y: list[float] = list() # liste des ordonnées
        for p in self.__liste_points:
            liste_x.append(p.get_x())
            liste_y.append(p.get_y())
        plot(liste_x, liste_y) # appel de la méthode plot de la classe matplotlib
        show() # affichage de la courbe
```