IUT2 Université Grenoble Alpes
Département STID

# Programmation Objet

R3.VCOD.10

Martin LENTSCHAT

martin.lentschat@univ-grenoble-alpes.fr

# Les principales notions de la POO

- **Classe** (avec ses **Attributs** et **Méthodes**) et **Objet/Instance**
- Envoi de messages
- **Encapsulation**
- Héritage
- Polymorphisme
- Associations entre classes
- Agrégation
- Composition

# Objet / Instance d'une classe

- Quelques fonctions utiles :
  - *type* pour renvoyer le type d'un objet
  - *isinstance* pour vérifier qu'un objet a le type (càd la classe) attendu.

```
import personne

p1 = personne.Personne("Durand", "Paul", "Grenoble")
p2 = personne.Personne("Dupont", "Jeannine", "SMH")

print(type(p1))
print(isinstance(p2,personne.Personne))
```

main.py

```
<class 'personne.Personne'>
True
```

À l'exécution

IUT2 Université Grenoble Alpes
Département STID

# Objet / Instance d'une classe

- Autres méthodes et fonctions utiles :
  - Lister les propriétés (i.e. méthodes et attributs) d'un objet:

    **dir(obj)**

```
etu1 = Etudiant('Curie', 'Marie')
etu2 = Etudiant('Einstein', 'Albert')
print(dir(etu1))
print(etu1.__dict__)
```

```
['__class__', '__del__', '__delattr__', '__dict__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__gt__', '__hash__', '__init__', '__init_subclass__', '__le__',
'__lt__', '__module__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '__weakref__', 'first_name', 'get_first_name',
'get_last_name', 'last_name', 'notes', 'set_first_name',
'set_last_name', 'to_string']
```

  - Récupérer une instance d'objet en dictionnaire

    **obj.__dict__**

```
{'first_name': 'Marie', 'last_name': 'Curie', 'notes': []}
```

IUT2 Université Grenoble Alpes
Département STID

# Attributs d'instance / Attributs de classe

**Il existe en réalité 2 types d'attributs :**

- **Les attributs d'instance :** (cf. exemples précédents)
  - Sont définis dans le constructeur
  - Chaque instance possède ses propres attributs d'instance.
  - On y accède via le nom de l'objet.

- **Les attributs de classe :** (plus rarement)
  - Sont définis dans la classe
  - Ils sont partagés par toutes les instances de la classe.
  - On y accède via la nom de la classe.
  - On peut définir des **méthodes de classe** pour accéder ou modifier les attributs de classe

# Attributs de classe / Méthode de classe

**Attribut de classe** pour la classe Etudiant

**Méthode de classe** pour accéder / modifier les attributs de classe

```python
class Etudiant:

    moyenne_age : int

    def __init__(self, nom : str, prenom : str, age : int):
        self.firstname = prenom
        self.lastname = nom
        self.age = age


    def get_age(self):
        return self.age

    def set_moyenne_age(ma : float):
        Etudiant.moyenne_age = ma
```

etudiant.py

**Appel à la méthode de classe** (simplifiée ☺)

**Appel à l'attribut de classe**

```python
import etudiant

p1 = etudiant.Etudiant("Durand", "Paul", 18)
p2 = etudiant.Etudiant("Dupont", "Jeannine", 22)

etudiant.Etudiant.set_moyenne_age((p1.get_age()+p2.get_age())/2)

print("Age de p1: ", p1.get_age())
print("Age de p2: ", p2.get_age())
print("Moyenne d'âge issue de  p1: ", p1.moyenne_age)
print("Moyenne d'âge issue de p2: ", p2.moyenne_age)
```

main.py

```
Age de p1:  18
Age de p2:  22
Moyenne d'âge issue de  p1:  20.0
Moyenne d'âge issue de p2:  20.0
```

À l'exécution

IUT2 Université Grenoble Alpes
Département STID

# Exercice

1. Définir une classe *Etudiant*, ayant des attributs d'objet first_*name*, last_*name* et *age*, un *constructeur* et les méthodes *get* et *set* nécessaires

2. Définir un attribut de classe *list_inscrits*, avec des méthodes permettant d'ajouter ou enlever des étudiants.
   a) Comment automatiser la mise à jour de cette liste ?

3. Définir un attribut de classe *age_moy*, et une méthode permettant de le calculer pour tout les étudiants inscrits
   a) Comment garder automatiquement cette valeur à jour ?

4. Définir un attribut de classe *age_min*, comment contrôler que les nouveaux étudiants remplissent cette condition ?
   a) Comment appliquer cette condition aux étudiants inscrits existants si la valeur vient à changer ?

IUT2 Université Grenoble Alpes
Département STID

# Encapsulation

- L'encapsulation permet de contrôler l'accès en lecture/écriture des attributs via des méthodes
- Python est un langage 'ouvert', on fait confiance à l'utilisateur
- Si on veut rendre un attribut ou une méthodes privé, on débute par des doubles '_'
  - Accessible uniquement dans la définition de la classe
  - La lecture/écriture se fait par *set* et *get*

```
class Animal:
    def __init__(self):
        self.__age = 0

    def vieillir(self):
        self.__age = self.__age + 1

    def get_age(self):
        return self.__age

animal = Animal()
animal.vieillir()
print(animal.get_age())
print(animal.__age)
```

```
1
```

```
AttributeError: 'Animal' object has no attribute '__age'
```