La **classe** est un concept de base de la **programmation orientée objet**. En langage Python, vous pouvez très bien écrire un script sans définir de classes (c'est ce que nous avons fait jusqu'à présent). Cependant, vous manipulez forcement des **objets (ou instances de classes)** : une variable entière est une instance de la classe `int`, une chaîne de caractères est une instance de la classe `str`... Le module `Tkinter` (que nous aborderons dans le chapitre 7) sert à créer des interfaces graphiques : ce module fournit une bibliothèque de classes. Il est donc important d'étudier les classes notamment pour bien comprendre comment les utiliser (pour un débutant, c'est déroutant !). Ce chapitre est difficile : je vous conseille de relire ce que l'on a vu dans le chapitre 1.

Définition d'une classe

Voici un exemple de script qui utilise une classe définie par l'utilisateur. Nous allons commencer par créer le fichier source `CompteBancaire.py` (on parlera par la suite du module `CompteBancaire`) : Ouvrir IDLE : Démarrer → Programmes → Python → IDLE (Python GUI) File → New Window Copier puis coller le code source ci-dessous :

```
# CompteBancaire.py

# définition de la classe Compte
class Compte:
   """Un exemple de classe :
   gestion d'un compte bancaire"""

   # définition de la méthode spéciale __init__ (constructeur)
   def __init__(self,soldeInitial):
      """Initialisation du compte avec la valeur soldeInitial."""
      # assignation de l'attribut d'instance solde
      self.solde = float(soldeInitial)

   # définition de la méthode NouveauSolde()
   def NouveauSolde(self,somme):
      """Nouveau solde de compte avec la valeur somme."""
      self.solde = float(somme)

   # définition de la méthode Solde()
   def Solde(self):
      """Retourne le solde."""
      return self.solde

   # définition de la méthode Credit()
   def Credit(self,somme):
      """Crédite le compte de la valeur somme. Retourne le solde."""
      self.solde += somme
```

```
        return self.solde

    # définition de la méthode Debit()
    def Debit(self,somme):
        """Débite le compte de la valeur somme. Retourne le solde."""
        self.solde -= somme
        return self.solde

    # définition de la méthode spéciale __add__ (surcharge de l'opérateur +)
    def __add__(self,somme):
        """x.__add__(somme) <=> x+somme
        Crédite le compte de la valeur somme.
        Affiche 'Nouveau solde : somme'"""
        self.solde += somme
        print("Nouveau solde : {:+.2f} euros".format(self.solde))
    # définition de la méthode spéciale __sub__ (surcharge de l'opérateur -)
    def __sub__(self,somme):
        """x.__sub_(somme) <=> x-somme
        Débite le compte de la valeur somme.
        Affiche 'Nouveau solde : somme'"""
        self.solde -= somme
        print("Nouveau solde : {:+.2f} euros".format(self.solde))

if __name__ == '__main__':
    # Ce bloc d'instructions est exécuté si le module est lancé en tant que programme autonom
    # Instanciation de l'objet cb1 de la classe Compte
    cb1 = Compte(1000)
    # formatage des données pour afficher deux chiffres après la virgule et le signe
    print("{:+.2f}".format(cb1.Solde()))
    print("{:+.2f}".format(cb1.Credit(200)))
    print("{:+.2f}".format(cb1.Debit(50.23)))
    print("{:+.2f}".format(cb1.Solde()))
    cb1.NouveauSolde(5100)
    print("{:+.2f}".format(cb1.Solde()))
    cb1+253.2
    cb1-1000
    cb1-cb1.Solde()
```

File → Save As **Chez vous :** Répertoire : C:\PythonXX Nom du fichier : CompteBancaire.py **Au Lycée des Flandres** Répertoire : H:\Travail\NSI\Python\Cours Nom du fichier : CompteBancaire.py Puis exécuter le module : Run → Run Module (ou touche F5) :

```
>>>
+1000.00
+1200.00
+1149.77
```

```
+1149.77
+5100.00
Nouveau solde : +5353.20 euros
Nouveau solde : +4353.20 euros
Nouveau solde : +0.00 euros
>>>
```

L'instruction `class` Pour définir la nouvelle classe `Compte`, on utilise l'instruction `class`. Une classe possède des fonctions que l'on appelle **méthodes** et des données que l'on appelle **attributs**. Une méthode se définit de la même manière qu'une fonction (on commence par l'instruction `def`). Une méthode possède au moins un paramètre qui s'appelle `self`. Le paramètre `self` désigne toutes les instances qui seront créées par cette classe. 7 méthodes sont ainsi définies. Un seul attribut d'instance est utilisé dans cette classe : `solde` (à ne pas confondre avec la méthode `Solde()`).

La méthode constructeur La méthode constructeur (méthode spéciale `__init__()`) est exécutée automatiquement lorsque l'on instancie (crée) un nouvel objet de la classe.

Autres méthodes spéciales Deux autres méthodes spéciales sont utilisées : `__add__()` et `__sub__()`. L'écriture `cb1+253.2` est équivalente à `cb1.__add__(253.2)` L'opérateur + représente ici une addition sur le solde. L'écriture `cb1-1000` est équivalente à `cb1.__sub__(1000)` L'opérateur - représente ici une soustraction sur le solde.

```
>>> cb1.NouveauSolde(500)
>>> print(cb1.Solde())
500.0
>>> cb1.__add__(1000)
Nouveau solde : +1500.00 euros
>>> cb1+2000   # cette écriture est très pratique !
Nouveau solde : +3500.00 euros
>>>
```

L'instruction `if __name__ == '__main__':` Ici, le module est exécuté en tant que programme principal : les instructions qui suivent sont donc exécutées. Dans le cas où ce module est importé dans un autre programme, cette partie du code est sans effet.

Documentation La fonction `help()` est très utile. On comprend ici l'intérêt de bien documenter ses programmes avec les `"""docstrings"""` :

```
>>> help(cb1)
Help on instance of Compte in module __main__:
```

```
class Compte
 |  Un exemple de classe :
 |  gestion d'un compte bancaire
 |
 |  Methods defined here:
 |
 |  Credit(self, somme)
 |      Crédite le compte de la valeur somme. Retourne le solde.
 |
 |  Debit(self, somme)
 |      Débite le compte de la valeur somme. Retourne le solde.
 |
 |  NouveauSolde(self, somme)
 |      Nouveau solde de compte avec la valeur somme.
 |
 |  Solde(self)
 |      Retourne le solde.
 |
 |  __add__(self, somme)
 |      x.__add__(somme) <=> x+somme
 |      Crédite le compte de la valeur somme.
 |      Affiche 'Nouveau solde : somme'
 |
 |  __init__(self, soldeInitial)
 |      Initialisation du compte avec la valeur soldeInitial.
 |
 |  __sub__(self, somme)
 |      x.__sub_(somme) <=> x-somme
 |      Débite le compte de la valeur somme.
 |      Affiche 'Nouveau solde : somme'
>>>
```

La fonction `dir()` retourne la liste des méthodes et attributs :

```
>>> dir(cb1)
['Credit', 'Debit', 'NouveauSolde', 'Solde', '__add__',
'__doc__', '__init__', '__module__', '__sub__', 'solde']
```

Remarques On peut instancier plusieurs objets d'une même classe :

```
>>> cb2 = Compte(10000)
>>> print(cb2.Credit(500))
10500.0
>>> cb3 = Compte(6000) # et encore un !
>>> cb3-500
Nouveau solde : +5500.00 euros
>>>
```

Même si cela n'est pas recommandable, vous pouvez accéder directement aux attributs d'instance :

```
>>> print(cb2.solde)
10500.0
>>> cb2.solde = 5000.0 # assignation de l'attribut d'instance solde
>>> print(cb2.solde)
5000.0
>>> print(cb2.Solde())
5000.0
>>>
```

Notez que des méthodes spéciales sont prévues pour personnaliser le mode d'accès aux attributs : `__setattr__()`, `__getattr__()`...

Importation d'un module défini par l'utilisateur

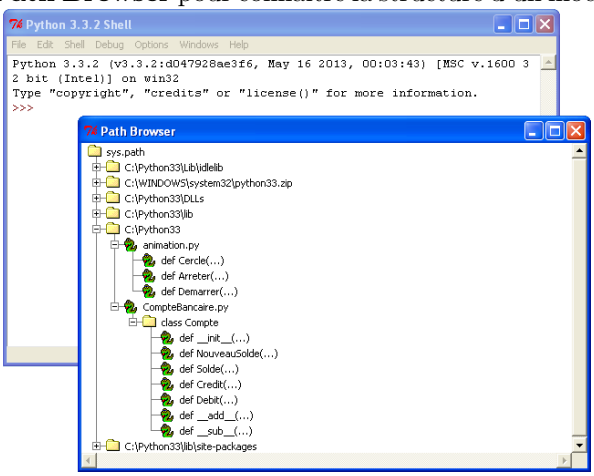
L'importation de notre module personnel `CompteBancaire` se fait de la même façon que pour les modules de base (`math`, `random`, `time`...). Redémarrer l'interpréteur intéractif (Python Shell) : Shell → Restart Shell

```
>>> import CompteBancaire
>>> cb = CompteBancaire.Compte(1500)
>>> print(cb.Debit(200))
1300.0
>>> print(cb.solde)
1300.0
>>> cb+2000
Nouveau solde : +3300.00 euros
>>>
```

Le module `CompteBancaire` est importé par le programme principal (ici l'interpréteur intéractif). La partie du code qui suit l'instruction `if __name__ == '__main__':` est ignorée (il n'y a donc pas d'affichage après l'instruction `import CompteBancaire`).

Modules de classes, modules de fonctions

IDLE dispose de l'outil **Path Browser** pour connaître la structure d'un module.



File → Path Browser Un module peut contenir des classes (`CompteBancaire`), des fonctions (`animation`) ou des fonctions et des classes.

Autre exemple Le module standard `math` est un module de fonctions. Pour s'en convaincre :

```
>>> import math
>>> help(math)
Help on built-in module math:

NAME
    math

FILE
    (built-in)

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.

    acosh(...)
        acosh(x)
```

```
        Return the hyperbolic arc cosine (measured in radians) of x.
...
...

    trunc(...)
        trunc(x:Real) -> Integral

        Truncates x to the nearest Integral toward 0. Uses the __trunc__ magic method.

DATA
    e = 2.718281828459045
    pi = 3.141592653589793
```

Le module `math` a également deux données (`pi` et `e`) :

```
>>> print(math.e)        # donnée e du module math (nombre d'Euler)
2.71828182846
>>> print(math.pi)     # donnée pi du module math (nombre pi)
3.14159265359
>>> print(math.sin(math.pi/4.0))  # fonction sin() du module math (sinus)
0.707106781187
>>> print(math.sqrt(2.0))  # fonction sqrt() du module math (racine carrée)
1.41421356237
>>> print(math.exp(-3.0))  # fonction exp() du module math (exponentielle)
0.0497870683679
>>> print(math.log(math.e))    # fonction log() du module math (logarithme népérien)
1.0
```

Héritage de classes Si vous êtes familiarisé avec le module `Tkinter`, voici un exemple qui montre toute la puissance des classes et de la notion d'héritage :


```
# -*- coding: utf-8 -*-
from tkinter import *

class EnterMessage(Frame):
    """ Classe EnterMessage (Frame de saisie du message)
    Cette classe dérive de la classe Tkinter.Frame"""
    def __init__(self,master=None):
        """Initialisation : création d'un widget Frame"""
```

```
        Frame.__init__(self,master,bg='navy')
        self.pack(padx=10,pady=10)
        self.CreateWidgets()

    def CreateWidgets(self):
        """ Création des widgets Entry et Button dans le widget Frame"""
        self.NouveauMessage = StringVar()
        Entry(master=self,textvariable= self.NouveauMessage).pack(side=LEFT,padx=10,pady=10)
        Button(master=self,text="Nouveau",fg='navy',command=self.Nouveau).pack(padx=10,pady=

    def Nouveau(self):
        """ Création d'une instance de la classe NewPostIt """
        if self.NouveauMessage.get() != "":
            NewPostIt(master=self.master,message=self.NouveauMessage.get())
            self.NouveauMessage.set("")


class NewPostIt(Frame):
    """ Classe post-it (Frame Post-It)
    Cette classe dérive de la classe Tkinter.Frame"""
    def __init__(self,master=None,message=None):
        """Initialisation : création d'un widget Frame"""
        Frame.__init__(self,master,bg="maroon")
        self.pack(side=LEFT,padx=10,pady=10)
        self.CreateWidgets(message)

    def CreateWidgets(self,message):
        """ Création des widgets Label et Button dans le widget Frame"""
        Label(master=self, text = message,fg ='maroon',bg ='white').pack(padx=10,pady=10)
        Button(master=self, text = "Effacer",fg='navy',command=self.destroy).pack(padx=10,pa


if __name__ == '__main__':
    # création de la fenêtre principale
    Mafenetre = Tk()
    Mafenetre.title('Post-it')
    Mafenetre['bg']='bisque'

    # Création d'une instance de la classe EnterMessage
    EnterMessage(master=Mafenetre)

    Mafenetre.mainloop()
```

Exercices

Exercice 6.1 * On s'intéresse au module CompteBancaire défini plus haut. Définir une nouvelle méthode `Decouvert()` de la classe `Compte`, qui affiche selon le cas **Solde positif** ou **Solde négatif** :

```
>>> CompteFabrice = Compte(2000)
>>> CompteFabrice-3000
Nouveau solde : -1000.00 euros
>>> CompteFabrice.Decouvert()
Solde négatif
>>> CompteFabrice+10000
Nouveau solde : +9000.00 euros
>>> CompteFabrice.Decouvert()
Solde positif
>>>
```

Exercice 6.2 ** On s'intéresse au module CompteBancaire défini plus haut. Définir une nouvelle méthode `Suivi()` qui affiche l'historique du solde. On pourra définir un nouvel attribut de type `list` :

```
>>> CompteMuriel = Compte(5000)
>>> CompteMuriel-1000
Nouveau solde : +4000.00 euros
>>> CompteMuriel+5000
Nouveau solde : +9000.00 euros
>>> CompteMuriel.Suivi()
[5000.0, 4000.0, 9000.0]
>>>
```

Webographie

- Documentation sur les classes
- Documentation sur les objets