Programmation orientée objet

1 Vocabulaire de la programmation orientée objet

Nous avons déjà utilisé sans le savoir des objets.

Par exemple en Python lorsqu'on ajoute un nouvel élément à une liste donnée en utilisant la fonction `append()` on écrit par exemple à la console et dès que l'on a écrit `liste.` une fenêtre nous propose plusieurs fonctions possibles parmi lesquelles on retrouve `append()`

```
>>> liste = [1,2,3]

>>> liste.
```

append
clear
copy
count
extend
index
insert
pop

Si on écrit une fonction autre que celles proposées ci-dessus il y aura un message d'erreur

```
>>> liste = [1,2,3]

>>> liste.split(" ")
Traceback (most recent call last):
  File "<console>", line 1, in <module>
AttributeError: 'list' object has no attribute 'split'

>>> type(liste)
<class 'list'>
```

Nous voyons apparaître le mot 'object' dans le message d'erreur, de même lorsque nous demandons le type de la variable, nous voyons apparaître le mot 'class'

Nous avons aussi utilisé le module `turtle` qui contient plu-

sieurs **classes** et nous avons déjà utilisé l'une d'entre elles la classe `Turtle`

Qu'est ce qu'une **classe** ? qu'est ce qu'un **objet** ?

La **programmation orienté objet** (poo) est un **paradigme de programmation** (une méthode de programmation) nous aidant à la fois pour être un **programmeur-utilisateur** d'API (application programming interface), par exemple celle du module Pygame, ou du module Pyxel ou d'autres encore, mais aussi un **programmeur-concepteur** utilisant les méthodes de la programmation objet pour créer un code **structuré** (implémentation lisible) , **sûr** (sécurité) et **facile d'utilisation pour les autres**(interface compréhensible) .

Prenons un exemple de jeu écrit avec le module `pyxel` de Python visible ici `https://github.com/kitao/pyxel/blob/main/python/pyxel/examples/09_shooter.py`

Si vous avez joué à ce jeu vous avez identifié les principaux éléments du jeu et les règles du jeu (interactions entre les éléments).

(On ne montre ci après que le début des classes).

1. Un fond de jeu (`class Background`)

```
class Background:
    def __init__(self):
        self.stars = []
        for i in range(NUM_STARS):
            self.stars.append(
                (
                    pyxel.rndi(0, pyxel.width - 1),
                    pyxel.rndi(0, pyxel.height - 1),
                    pyxel.rndf(1, 2.5),
                )
            )

    def update(self):
        for i, (x, y, speed) in enumerate(self.stars):
            y += speed
            if y >= pyxel.height:
                y -= pyxel.height
            self.stars[i] = (x, y, speed)

    def draw(self):
        for x, y, speed in self.stars:
```

2. Vous (le joueur, `class Player`)

```
class Player:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.w = PLAYER_WIDTH
        self.h = PLAYER_HEIGHT
        self.is_alive = True

    def update(self):
        if pyxel.btn(pyxel.KEY_LEFT) or pyxel.btn(pyxel.GAMEPAD1_BUTTON_DPAD_LEFT):
            self.x -= PLAYER_SPEED
        if pyxel.btn(pyxel.KEY_RIGHT) or pyxel.btn(pyxel.GAMEPAD1_BUTTON_DPAD_RIGHT):
            self.x += PLAYER_SPEED
        if pyxel.btn(pyxel.KEY_UP) or pyxel.btn(pyxel.GAMEPAD1_BUTTON_DPAD_UP):
            self.y -= PLAYER_SPEED
        if pyxel.btn(pyxel.KEY_DOWN) or pyxel.btn(pyxel.GAMEPAD1_BUTTON_DPAD_DOWN):
            self.y += PLAYER_SPEED
        self.x = max(self.x, 0)
        self.x = min(self.x, pyxel.width - self.w)
        self.y = max(self.y, 0)
        self.y = min(self.y, pyxel.height - self.h)
```

3. Des ennemis, (`class Enemy`)

```
class Enemy:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.w = ENEMY_WIDTH
        self.h = ENEMY_HEIGHT
        self.dir = 1
        self.timer_offset = pyxel.rndi(0, 59)
        self.is_alive = True
        enemies.append(self)

    def update(self):
        if (pyxel.frame_count + self.timer_offset) % 60 < 30:
            self.x += ENEMY_SPEED
            self.dir = 1
        else:
            self.x -= ENEMY_SPEED
            self.dir = -1
        self.y += ENEMY_SPEED
        if self.y > pyxel.height - 1:
            self.is_alive = False
```

4. Des missiles (`class Bullet`)

```
class Bullet:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.w = BULLET_WIDTH
        self.h = BULLET_HEIGHT
        self.is_alive = True
        bullets.append(self)

    def update(self):
        self.y -= BULLET_SPEED
        if self.y + self.h - 1 < 0:
            self.is_alive = False

    def draw(self):
        pyxel.rect(self.x, self.y, self.w, self.h, BULLET_COLOR)
```

5. Les explosions (`class Blast`)

```
class Blast:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.radius = BLAST_START_RADIUS
        self.is_alive = True
        blasts.append(self)

    def update(self):
        self.radius += 1
        if self.radius > BLAST_END_RADIUS:
            self.is_alive = False

    def draw(self):
        pyxel.circ(self.x, self.y, self.radius, BLAST_COLOR_IN)
        pyxel.circb(self.x, self.y, self.radius, BLAST_COLOR_OUT)
```

Comme un metteur en scène de théâtre, le programmeur-concepteur va écrire une classe pour chaque personnage du jeu, et une classe appelée `App` pour Application (dans le langage de pyxel) pour écrire le code d'une seule partie du jeu dans laquelle il va utiliser toutes les autres classes.

(Convention : Le nom d'une classe commence toujours par une majuscule)

Vous avez observé **qu'à chaque fois que vous avez appuyé sur la barre espace, vous le joueur, représenté par une image de vaisseau spatial, avez lancé une missile**

Comment a-t-on programmé cela et dans quelle classe ?

Dans l'ordre, et en cascade :

1. On exécute **une** partie en appelant une fois le **constructeur** `App()` . Le constructeur construit en mémoire vive une **instance** (une réalisation) de la classe.

2. Au cours de la construction d'une partie on va créer le joueur en appelant une fois `Player()`

3. L'état du joueur va évoluer au cours de la partie. Il peut se déplacer, il peut mourir, donc on lui attribue des **caractéristiques** qui permettent de mémoriser son évolution. Ce sont les **attributs** de la classe `Player` (ou de l'objet construit désigné par self). Il y en a cinq, `x`,`y`, `w`, `h` et `is_alive`.

   Dans la classe `Player` le mot réservé **self** fait référence à l'objet construit.

   La syntaxe est `self.attribut` pour signifier l'attribut `attribut` de `self`.

```
class Player:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.w = PLAYER_WIDTH
        self.h = PLAYER_HEIGHT
        self.is_alive = True

    def update(self):
        if pyxel.btn(pyxel.KEY_LEFT) or pyxel.btn(pyxel.GAMEPAD1_BUTTON_DPAD_LEFT):
            self.x -= PLAYER_SPEED
        if pyxel.btn(pyxel.KEY_RIGHT) or pyxel.btn(pyxel.GAMEPAD1_BUTTON_DPAD_RIGHT):
            self.x += PLAYER_SPEED
        if pyxel.btn(pyxel.KEY_UP) or pyxel.btn(pyxel.GAMEPAD1_BUTTON_DPAD_UP):
            self.y -= PLAYER_SPEED
        if pyxel.btn(pyxel.KEY_DOWN) or pyxel.btn(pyxel.GAMEPAD1_BUTTON_DPAD_DOWN):
            self.y += PLAYER_SPEED
        self.x = max(self.x, 0)
        self.x = min(self.x, pyxel.width - self.w)
        self.y = max(self.y, 0)
        self.y = min(self.y, pyxel.height - self.h)
```

4. Pour faire évoluer le joueur on a les **méthodes** de la classe. Ce sont des fonctions.

   Une méthode de la classe `Player` est la méthode `update()`.

   Dans cette méthode on peut voir ces lignes de codes

```
if pyxel.btnp(pyxel.KEY_SPACE) or pyxel.btnp(pyxel.GAMEPAD1_BUTTON_A):
    Bullet(
        self.x + (PLAYER_WIDTH - BULLET_WIDTH) / 2, self.y - BULLET_HEIGHT / 2
    )
```

Ce qui signifie :

Si le joueur appuie sur la barre Espace alors une missile est créé (Bullet(...))

Enfin la classe `App()` a un attribut `player`, initialisé ainsi :

`self.player = Player(pyxel.width / 2, pyxel.height - 20)`

(et 30 fois par seconde la méthode `update` de la classe `Player` agit sur l'objet `self.player`.

`self.player.update()`

2 Courbes de poursuite mutuelles

Du module `turtle` on importe la classe `Turtle` et on créé trois instances de cette classe

```
from turtle import Turtle
tortue_1 = Turtle()
tortue_2 = Turtle()
tortue_3 = Turtle()
```

Toutes les méthodes de la tortue sont documentées sur le site officiel de Python à l'adresse ci-dessous

`https://docs.python.org/fr/3/library/turtle.html#overview-o`

Par exemple la méthode `position()` ou encore la méthode `towards()` sont définies ci-dessous dans la documentation officielle de Python, où `turtle` est une instance de la classe `Turtle`

Connaître l'état de la tortue

turtle.**position**()
turtle.**pos**()

Renvoie la position actuelle de la tortue (x,y) (en tant qu'un vecteur Vec2d).

```
>>> turtle.pos()
(440.00,-0.00)
```

turtle.**towards**(*x*, *y=None*)

Paramètres:
- **x** -- un nombre, ou une paire / un vecteur de nombres, ou une instance de tortue
- **y** -- un nombre si *x* est un nombre, sinon `None`

Renvoie l'angle entre l'orientation d'origine et la ligne formée de la position de la tortue à la position spécifiée par (x,y), le vecteur ou l'autre tortue. L'orientation d'origine dépend du mode — "standard"/"world" ou "logo".

```
>>> turtle.goto(10, 10)
>>> turtle.towards(0,0)
225.0
```

En utilisant la méthode `goto()` on va placer les trois tortues en trois points de coordonnées (-200,-150), (200,-150) et (0,250)

```
from turtle import Turtle
SEUIL = 10
#on créé trois instances  de la classe Turtle
#mémorisées par des variables globales
tortue_1 = Turtle()
tortue_2 = Turtle()
tortue_3 = Turtle()

#on place les tortues
tortue_1.penup()
tortue_1.goto(-200,-150)
tortue_1.color("red")
tortue_1.pendown()

tortue_2.penup()
tortue_2.goto(200,-150)
tortue_2.color("green")
```

```
tortue_2.pendown()

tortue_3.penup()
tortue_3.goto(0,250)
tortue_3.color("blue")
tortue_3.pendown()
```

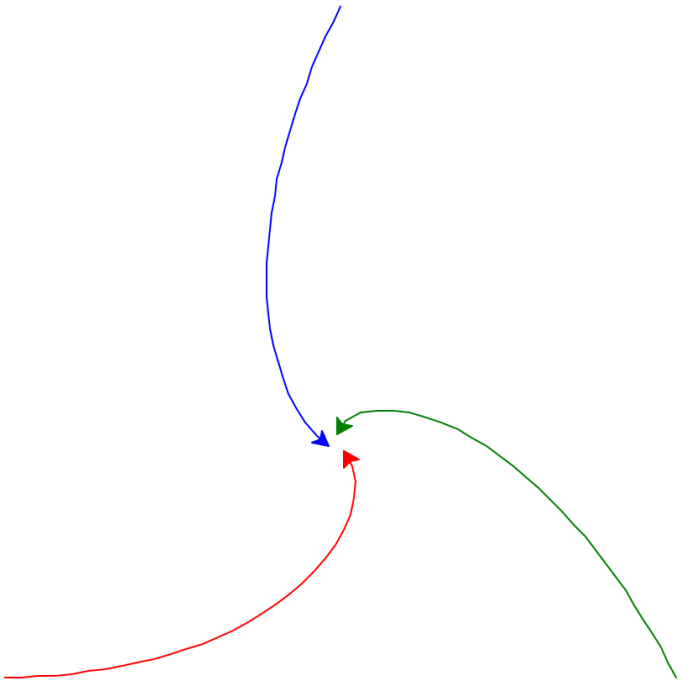
Enfin on va faire avancer la tortue `tortue_3`, dans la direction de la tortue `tortue_1` , de 10 pas, puis la tortue `tortue_1` dans la direction de la tortue `tortue_2` , de 10 pas, puis la tortue `tortue_2`, dans la direction de la tortue `tortue_3` , de 10 pas, et **ceci tant que chaque tortue et celle qu'elle vise sont éloignées de plus de 10 pas**

`tortue_1.distance(tortue_2.position())` retourne la distance entre les tortues `tortue_1` et `tortue_2` en pas

```
from turtle import Turtle
SEUIL = 10
#on créé trois instances  de la classe Turtle
#mémorisées par des variables globales
tortue_1 = Turtle()
tortue_2 = Turtle()
tortue_3 = Turtle()

#on place les tortues
tortue_1.penup()
tortue_1.goto(-200,-150)
tortue_1.color("red")
tortue_1.pendown()

tortue_2.penup()
tortue_2.goto(200,-150)
```

```
tortue_2.color("green")
tortue_2.pendown()

tortue_3.penup()
tortue_3.goto(0,250)
tortue_3.color("blue")
tortue_3.pendown()

#création des courbes de poursuite
while tortue_1.distance(tortue_2.position()) > SEUI
      tortue_2.distance(tortue_3.position()) > SEUI
      tortue_3.distance(tortue_1.position()) > SEUI
 # tortue_3 avance de 10 pas
 #dans la direction de tortue_1
    tortue_3.setheading(\
    tortue_3.towards(tortue_1.position()))
    tortue_3.forward(10)

 # tortue_2 avance de 10 pas
 #dans la direction de tortue_3
    tortue_2.setheading(\
    tortue_2.towards(tortue_3.position()))
    tortue_2.forward(10)

 #tortue_1 avance de 10 pas
 #dans la direction de tortue_2
    tortue_1.setheading(\
    tortue_1.towards(tortue_2.position()))
    tortue_1.forward(10)
```

On observe alors trois **courbes de poursuite**



3 Définition d'une classe Vecteur

On a pu un jour **définir** nos propres fonctions, maintenant on va pouvoir **définir** notre propre type plus précisément une classe

Ci après nous allons créer la classe `Vecteur` (à deux dimensions) qui existe déjà en Python sous le nom de classe Vecteur2D, dont voici un extrait de la documentation

class `turtle.`**`Vec2D`**(*x*, *y*)

Une classe de vecteur bidimensionnel, utilisée en tant que classe auxiliaire pour implémenter les graphiques *turtle*. Peut être utile pour les programmes graphiques faits avec *turtle*. Dérivé des *n*–uplets, donc un vecteur est un *n*–uplet !

Permet (pour les vecteurs *a*, *b* et le nombre *k*) :

- `a + b` addition de vecteurs
- `a - b` soustraction de deux vecteurs
- `a * b` produit scalaire
- `k * a` et `a * k` multiplication avec un scalaire
- `abs(a)` valeur absolue de a
- `a.rotate(angle)` rotation

On va **définir** ce qu'est un vecteur de deux manières :

1. Par ses **attributs** :

   Un vecteur a deux attributs : une abscisse et une ordonnée

2. Par ses **méthodes**

   Etant donné un vecteur on peut le multiplier par un réel pour obtenir un nouveau vecteur, on peut l'ajouter avec un autre vecteur pour obtenir un autre vecteur etc...

```
#Définition
class Vecteur:
    # Constructeur
    def __init__(self,abscisse,ordonnee):
        self.x = abscisse
        self.y = ordonnee

    #Méthodes
    def mult_reel(self,k):
        return Vecteur(k*self.x,k*self.y)

    def somme(self,other):
        return Vecteur(self.x+other.x,self.y+other.

#Exécution
v1 = Vecteur(2,-1)
v2 = Vecteur(3,4)
v3 = v1.somme(v2)
```

Remarques

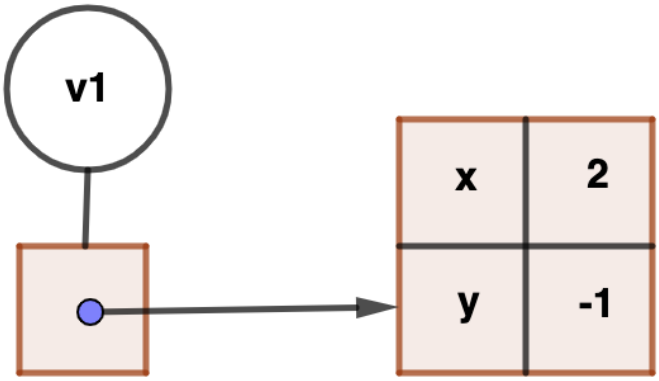
1. Une classe est définie avec le mot réservé `class` qui englobe toutes les méthodes
2. Par convention le nom d'une classe commence par une lettre majuscule

3. Une méthode joue un rôle important c'est la méthode `__init__()` qui construit l'objet dénommé par `self` en mémoire

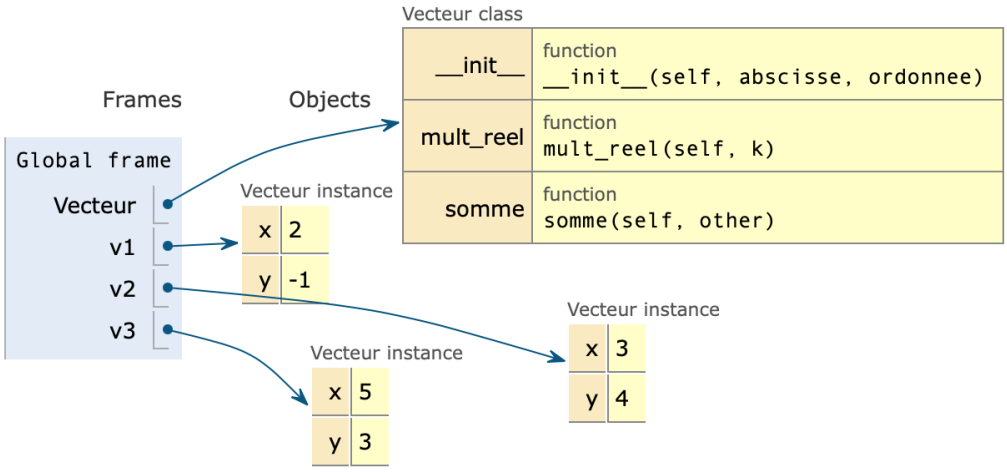
   L'instruction `v1 = Vecteur(2,-1)` créé en mémoire une variable `v1` de type Vecteur et dont la valeur est une référence en mémoire où se trouvent les valeurs 2 et -1

   Cette référence est représentée par une flèche dans le dessin ci-dessous

   La référence de la variable `v1`



Voici une visualisation avec Python Tutor du programme ci-dessus



4 Implémentation d'une liste chainée

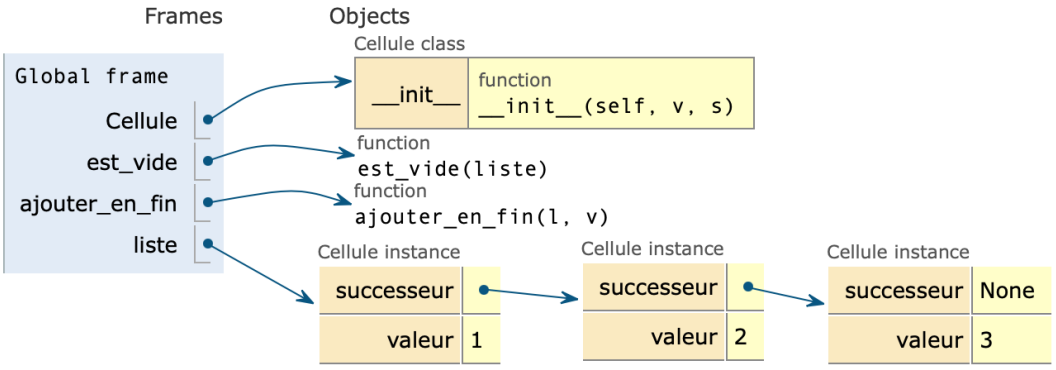
Comme on l'a souligné dans la section précédente, **à la création d'un objet, est associée à la variable qui désigne cet objet une référence, l'adresse mémoire où est enregistré cet objet** :

Donc si on veut créer une liste d'objets, il faut dans un premier temps créer une classe par exemple nommée `Cellule` contenant deux attributs :

1. Un attribut contenant un nombre, ou une chaîne de caractères, ou plus généralement un autre objet.
2. Un attribut contenant une **référence** d'un autre objet de type `Cellule` ou contenant `None` qui dans le langage Python représente l'ensemble vide.

Une liste ne sera alors qu'une référence sur la première cellule de la liste.

Voici une image d'une liste chaînée. C'est une structure naturellement **récursive**




```
class Cellule:
    """
    définit un élément d'une liste chainée"""
    def __init__(self,v,s=None):
        self.valeur = v
```

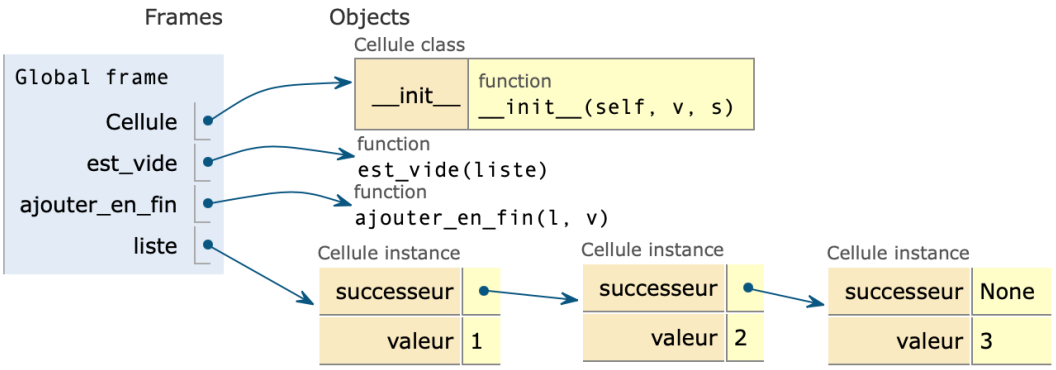
```
        self.successeur = s

# fonctions

def est_vide(liste:Cellule):
    return liste is None

def ajouter(liste:Cellule,val:int):
    if liste is None:
        return Cellule(val)
    return Cellule(val,liste.successeur)

def ajouter_en_fin(liste:Cellule,val:int):
    """
    fonction récursive
    """
    if liste.successeur is None:
        liste.successeur =  Cellule(val,None)
    else:
        ajouter_en_fin(liste.successeur,val)

liste = Cellule(1,Cellule(2,None))
ajoute_en_fin(liste,3)
```

Voici la documentation Python à propos de `is` et `is not`

6.10.3. Comparaisons d'identifiants

Les opérateurs `is` et `is not` testent l'égalité des identifiants des objets : `x is y` est vrai si et seulement si *x* et *y* sont le même objet. L'identifiant d'un objet est déterminé en utilisant la fonction `id()`. `x is not y` renvoie le résultat contraire de l'égalité des identifiants [4].

Voici une visualisation avec Python Tutor du programme ci-dessus



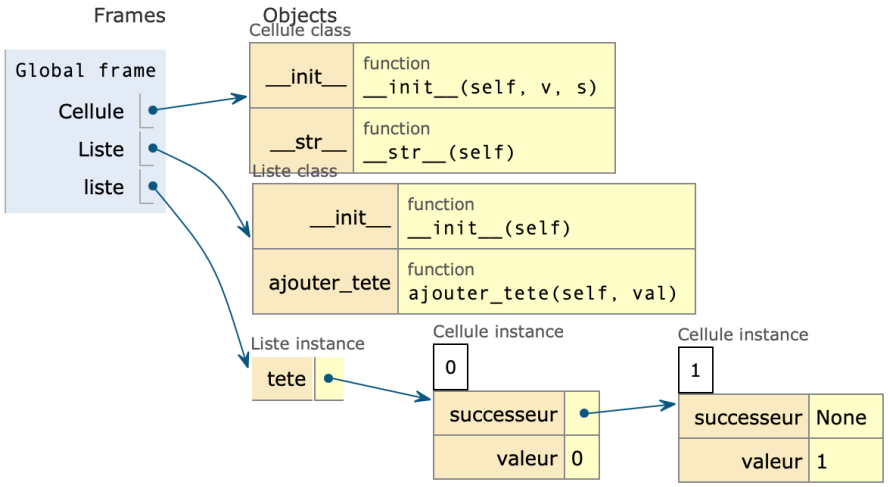
5 Encapsulation : Une classe Liste

On veut **envelopper, encapsuler** la liste de telle sorte que l'utilisateur d'une liste le fasse à travers une interface **sans savoir comment est implémenté la liste**

La classe `Liste` a un attribut `tete`, initialisé à `None` puis contiendra une référence sur la première cellule de la liste si elle n'est pas vide.

```
class Cellule:

    définit un élément d'une liste chainée"""
    def __init__(self,v,s):
        self.valeur = v
        self.successeur = s

    def __str__(self):
        return str(self.valeur)

class Liste:

    def __init__(self):
        self.tete = None
```

```
    def est_vide(self):
        return self.tete is None

    def ajouter(self,val):
        self.tete = Cellule(val,self.tete)

if __name__ == "__main__":
    liste = Liste()
    liste.ajouter(1)
    liste.ajouter(0)
```



A retenir :

La programmation objet est un **paradigme de programmation** permettant de :

1. créer des nouveaux **types** autre que `int,float,tuple, list,str` par exemple `Vecteur`
2. Une `classe` nommée `Vecteur` par exemple peut être vue comme un ensemble de fonctions dont l'une le constructeur `__init__(self,abscisse,ordonnee)` crée un **objet** de type `Vecteur` en mémoire ayant des **attributs** (caractéristiques) et sur lequel peuvent agir d'autres fonctions de la classe appelées **méthodes**
3. Pour créer un objet à l'image de la classe on affecte une variable au résultat de l'exécution du constructeur, par exemple :
   (a) (sans paramètre)
   ```
   tortue_1 = Turtle()
   ```
   (b) (avec paramètres)
   ```
   vecteur_1 = Vecteur(1,2)
   ```
4. Dans la classe on fait référence à l'objet construit par le mot réservé **self**.
5. Une fois la construction initiale faite l'objet peut évoluer sous l'action des méthodes, par xemple
   ```
   vecteur_1 = Vecteur(1,2)
   vecteur_1 = vecteur_1.mult(2)
   ```

6 Exercices

Ex 1

Vérifier votre compréhension du vocabulaire de la programmation objet en répondant aux questions au sujet de la classe suivante `Point`

```
class Point:
    def __init__(self,x,y):
        self.abscisse = x
        self.ordonnee = y

    def sont_alignes(self,point1,point2):
    """
    renvoie Vrai si le point est aligné
    avec les points point1
    et point2 de type Point
    """
        pass

    def distance(self,point1):
    """
    renvoie la distance euclidienne
    entre self et point1
    """
    pass

    #main
    o = ....
    i = ...
    j = ...
```

1. Compléter la phrase suivante :

   "Les trois objets `o`, `i` et `j` sont construits suivant le modèle de la classe `Point`.

   On dit qu'ils sont des ... de la classe `Point`."

2. Compléter les trois instructions du `main` pour que `o`, `i` et `j` sont construits suivant le modèle de la classe `Point` de telle sorte que le point `o` ait pour coordonnées (0,0), `i` (1,0) et `j` (0,1)
3. Quels sont les **attributs** des objets de cette classe ?
4. Compléter les **méthode** `sont_alignes` et `distance`

Ex 2

Implémenter une classe `Intervalle` qui correspond à l'**interface** suivante

```
Intervalle(gauche,droit)
# crée un objet de type Intervalle ayant pour attri
# deux objets gauche et droit de type float,
# les extrémités de l'intervalle

i.contient(j)
# Est ce que l'intervalle i contient
# le nombre j?

i.intersecte(j)
# Est ce que l'intervalle i "coupe"
# l'intervalle j ?

__str__()
# "[ gauche, droit] "
```

Ex 3

1. Compléter la classe `Rectangle` ci-dessous

```
class Rectangle:
    def __init__(self,x, y, w, h):
    """
    un objet de type Rectangle est défini par:
    -son centre de coordonnées (_x,_y)
    -la largeur du rectangle _w
    _la hauteur du rectangle _h
    """
        self._x = x
        ..............

    def aire(self):
    """
    renvoie l'aire de self
    """
        pass

    def perimetre(self):
    """
    renvoie le périmètre de self
    """
        pass

    def intersecte(self,other):
    """
    renvoie Vrai si l'intersection
    de self et de other est non vide
    """
        pass
```

2. Réfléchir à une représentation des objets de type `Rectangle`

Ex 4

1. Créer deux tortues tortue_1 et tortue_2
2. Laisser tortue_1 au centre du repère
3. Placer tortue_2 au point (200,0)
4. Répéter jusqu'à ce que les deux tortues soient "proches"
   Faites avancer tortue_1 vers le Nord de 10 pas puis faites avancer tortue_2 dans la direction de tortue_1
5. Décrire ce que vous observez.
6. Quel est le rôle de l'Informatique relativement aux Mathématiques dans ce genre de problème ?

Ex 5

1. Recopier et exécuter le programme suivant.

```
ANGLE = 4
NB_FOIS = 50
t = Turtle()
t.dot(5)
t.goto(-200,0)
t.color('red')
t.dot(10)
t.color('black')
t.goto(200,0)
t.color('blue')
t.dot(10)
t.color('black')
t.setheading(90)
```

```
for j in range(NB_FOIS):
    t.left(ANGLE)
    t.forward(10)
```

A l'exécution on observe une tortue partant du point (200,0) (point bleu) en regardant vers le Nord et en avançant 50 fois de 10 pixels tout en déviant de 4 degrés vers la gauche

2. Modifier les valeurs de `ANGLE` et `NB_FOIS` de telle sorte que la tortue arrive sur le point rouge en (-200,0)
3. On a construit ainsi un demi-cercle de centre (0,0) et de rayon 200. Pouvait-on prévoir les valeurs de `ANGLE` et `NB_FOIS` par un calcul ?
4. Créer une deuxième tortue `t2` , placée au centre du demi-cercle. Puis construire une courbe de poursuite où :
   (a) `t` avance indéfiniment de 10 pas en déviant de `ANGLE` vers la gauche
   (b) `t2` avance de 10 pas en regardant dans la direction de `t`

Ex 6

1. L'utilisateur entre un entier n entre 3 et 6 compris

   Créer n instances de la classe Turtle (et les insérer dans une liste)
2. Placer les n instances aux sommets d'un polygone régulier de n côtés
3. Créer n courbes de poursuites à la manière de ce qui a été fait précédemment

Ex 7

1. Chercher dans votre ordinateur le fichier `turtle.py` du module turtle

```
class Vec2D(tuple):
    """A 2 dimensional vector class, used as a helper class
    for implementing turtle graphics.
    May be useful for turtle graphics programs also.
    Derived from tuple, so a vector is a tuple!

    Provides (for a, b vectors, k number):
       a+b vector addition
       a-b vector subtraction
       a*b inner product
       k*a and a*k multiplication with scalar
       |a| absolute value of a
       a.rotate(angle) rotation
    """
```

   Quel est le rôle de la méthode `__new__()` ?

2. Ouvrir une console et entrer les commandes suivantes pour construire le vecteur `v1` d'abscisse -1 et d'ordonnée 2.5

```
>>> from turtle import Vec2D
>>> v1 = .....
```

3. Construire le vecteur `v2` d'abscisse 3.5 et d'ordonnée 2

4. En utilisant la méthode `__add__()` à la console calculer les coordonnées du vecteur `v3` somme de `v1` et de `v2`

```
>>> v3 = .....
```

```
def __add__(self, other):
    return Vec2D(self[0]+other[0], self[1]+other[1])
def __mul__(self, other):
    if isinstance(other, Vec2D):
        return self[0]*other[0]+self[1]*other[1]
    return Vec2D(self[0]*other, self[1]*other)
def __rmul__(self, other):
    if isinstance(other, int) or isinstance(other, float):
        return Vec2D(self[0]*other, self[1]*other)
def __sub__(self, other):
    return Vec2D(self[0]-other[0], self[1]-other[1])
def __neg__(self):
    return Vec2D(-self[0], -self[1])
def __abs__(self):
    return (self[0]**2 + self[1]**2)**0.5
```

5. Comment calculer cette somme à la console de manière plus intuitive ?
6. Calculer à la console la norme du vecteur `v1` de deux manières différentes.
7. Comprendre le code de la méthode `rotate()` de la classe Vec2D

```
    def rotate(self, angle):
        """rotate self counterclockwise by angle
        """
        perp = Vec2D(-self[1], self[0])
        angle = angle * math.pi / 180.0
        c, s = math.cos(angle), math.sin(angle)
        return Vec2D(self[0]*c+perp[0]*s, self[1]*c+perp[1]*s)
```

Ex 8

1. Quelle est la différence entre une classe et un objet ?
2. Quelle est la différence entre un attribut et une méthode ?

Ex 9

Ouvrir une console et faire (à condition que `pygame` soit installé

```
>>> from pygame.math import Vector2
```

1. Puis entrer à la console

   ```
   >>> v1 = Vector2(5, 10)
   ```

   puis rappeler v1 vous devez observer ceci

   ```
   >>> v1 = Vector2(5, 10)
   >>> v1
   <Vector2(5, 10)>
   ```

   Qu'allez vous observer si vous entrez

   ```
   >>> v1 = Vector2(5)
   ```

2. Qu'allez vous observer si vous entrez

```
>>> v3 = v1 + v2
```

3. Comment appelle-t-on `x` et `y` par rapport à `v1` lorsqu'on écrit `v1.x` et `v1.y`
4. Maintenant entrer `v1 = Vector2(1,2)` et `v2 = Vector2(4,6)`
   La documentation Pygame dit que la méthode `distance_to()` de la classe Vector2 "calculates the Euclidean distance to a given vector"
   Appliquer cette méthode avec v1 et v2 et que pensez vous du résultat ?
5. Quelle est la méthode pour calculer la norme du vecteur v1 ?
6. Quelle est la méthode pour calculer le produit scalaire de v1 et v2 ?

Ex 10

On a vu que l'échange de valeurs entre deux variables ne marche pas si on n'a pas les adresses mémoires de ces variables.

Une façon de faire est de mémoriser ces adresses en créant un type "enveloppé"

```
class Integer:
    def __init__(self,e):
        self.entier = e

def echanger(x,y):
    temp = y.entier
    y.entier = x.entier
    x.entier = temp

x = Integer(2)
```

```
y = Integer(3)
echanger(x,y)
```

Essayer.

Quel est le rapport avec la fonction `echanger(tab:list,i:int,j:int)` vue l'année dernière ?

```
def echanger(tab:list,i:int,j:int):
    temp = tab[j]
    tab[j] = tab[i]
    tab[i] = temp
```

Ex 11

1. Comment caractériserez vous un élève du Lycée ? (Définir des attributs)
2. Définir un constructeur possible de la classe Eleve
3. Quelles méthodes ajouterez vous à cette classe ?

Ex 12

Définir une méthode `longueur()` pour la classe Liste de deux manières, itérative et récursive

Ex 13

Définir la méthode __str()__ pour la classe Liste

Ex 14

Définir la méthode spécifique __len()__ pour la classe Liste qui retourne la longueur d'une liste l, de telle sorte que l'appel de cette méthode se fait ainsi `len(l)` comme pour les listes en Python

Ex 15

Ecrire une fonction **itérative** `ajouter_en_tete(liste,val)` où la valeur val est insérée à la fin de la liste.

Ex 16

Quelle est la différence entre les méthodes `__str()__` et `__repr()__` ?

Ex 17

Définir une fonction `inv_liste(l)` qui renvoie l'inverse de la liste chaînée l avec une complexité linéaire.

Ex 18

En s'inspirant du TP sur la classe Date, définir une classe `Fraction` pour représenter un nombre rationnel.

Cette classe a deux attributs `num` et `denum` qui sont des entiers et désignent respectivement le numérateur et le dénominateur de la fraction. On demande que le dénominateur soit strictement positif.

1. Ecrire le constructeur de cette classe.Le constructeur doit lever une `ValueError` si le dénominateur n'est pas strictement positif.
2. Ecrire une méthode `somme` qui a pour paramètre un autre objet de type `Fraction` et qui renvoie un nouvel objet de type `Fraction`, résultat de la somme de `self` et de l'autre fraction.
3. Ecrire une méthode `__str__` qui renvoie une chaîne de caractères de la forme "7/4" ou simplement "7" si le dénominateur vaut 1

Ex 19

Ex 1 Asie 1 - Bac 2024

Ex 20

Ex 1 Centres Etrangers Afrique 2 - Bac 2024