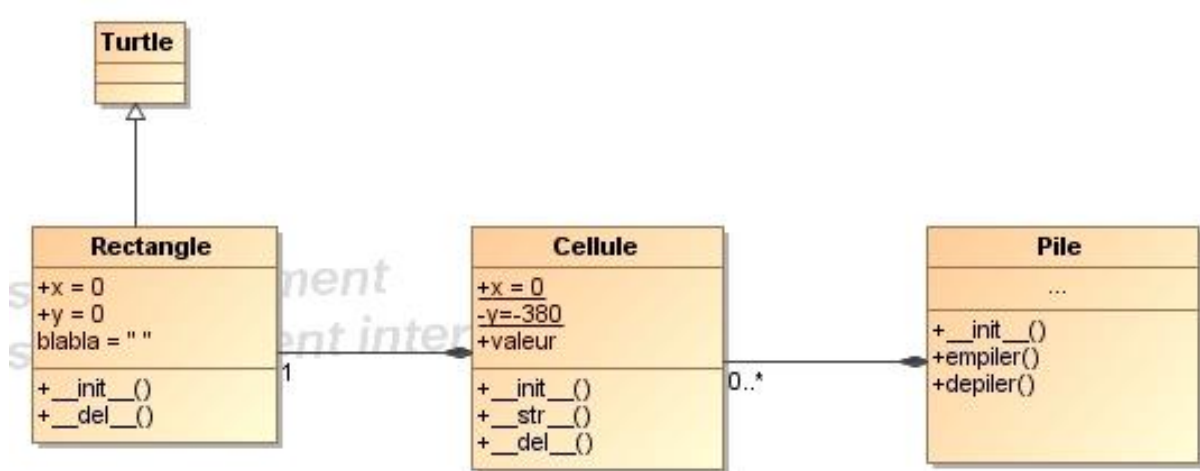


## TP – Pile et Files 2 - Pile dynamique

Objectif : représenter graphiquement à l'aide de la bibliothèque turtle une pile créer sous forme de liste chaînée.



On reprend le code d'une liste chaînée pour fabriquer une Pile. Fonctionnement FIFO on empile au sommet et on dépile le sommet de la pile.

### 1. Création de la pile sous forme de liste chaînée – reprendre le cours

#### 1.1. Implémentation de la cellule

```
# -*- coding: utf-8 -*-
"""
Pile dynamique et graphique
"""
from turtle import Screen, Turtle
import random
import sys # bibliothèque paramètres et fonctions propres des systèmes
```

```
class Cellule():
```

```
    x = 0
    y = -380
```

```
    def __init__(self, valeur, suivant=None):
```

A compléter : voir cours chainage des cellules

```
        self.emcombrement = sys.getsizeof(self) # taille de la cellule en octets
```

```
        self.__class__.x
```

```
        self.__class__.y
```

```
        #place pour le futur rectanglee
```

A compléter : à chaque cellule crée on déplacera le rectangle de 40 en hauteur

## TP – Pile et Files 2 - Pile dynamique

### 1.2. Implémentation de la Pile

Rappel cours :

Début de liste chaînée	<pre>class Liste:     #liste simplement chaînée     def __init__(self):         self.tete = None</pre>
<b>Insérer en tête</b>  « PUSH – insert(0,x) »	<pre>def inserer_en_tete(self,valeur):     # Créer une nouvelle cellule     # Y insérer la valeur     nouvelle_cellule =Cellule(valeur)     # Mettre la Cellule en tête de la liste     nouvelle_cellule.suivant = self.tete     # Faire pointer la liste sur la cellule     self.tete = nouvelle_cellule</pre>

Compléter le code suivant :

```
class Pile:

    def __init__(self):
        self.taille = 0 # nombre d'assiettes dans la pile
        self.sommet = 

    def empiler():
        self.sommet = Cellule()
        self.taille += 1

    def depiler(self):
        if self.taille > 0:
            val = 
            self.sommet = 
            self.taille 
            return val
        else :
            print("Pile vide")
```

## TP – Pile et Files 2 - Pile dynamique

### 1.3. Fin du programme utilisation de Turtle pour la fenêtre de dessin

- Instancier un objet Pile1 de type Pile.
- Empiler et dépiler plusieurs valeurs
- Ajouter le code ci-dessus pour utiliser une fenêtre de Turtle.

```
dessin = Screen()  
# Taille et position de la fenêtre Turtle  
dessin.setup(width =400, height=800,startx = None, starty = None)  
dessin.colormode(255)  
dessin.bgcolor("white")  
  
dessin.title("Empiler _ Dépiler ")
```

```
Pile1 = Pile()  
for i in "La petite maison "  
    Pile1.empiler(i)  
Pile1.depiler()  
Pile1.depiler()  
Pile1.depiler()
```

```
dessin.exitonclick()
```

Vérifier que votre pile fonctionne comme prévue. Dans le cas contraire trouver votre erreur.

## 2. Création de la classe Rectangle

### 2.1. Définition des attributs de la classe rectangle

Pour dessiner Le rectangle contenu dans chaque Cellule on implémente une classe avec le nom Rectangle. Cette dernière hérite de Turtle.

Compléter le début du code de cette classe :

```
class Rectangle(Turtle):
```

```
    def __init__(self,x=0,y=0,blabla=""):  
        Turtle.__init__(self)
```

```
        -  
        -  
        -
```

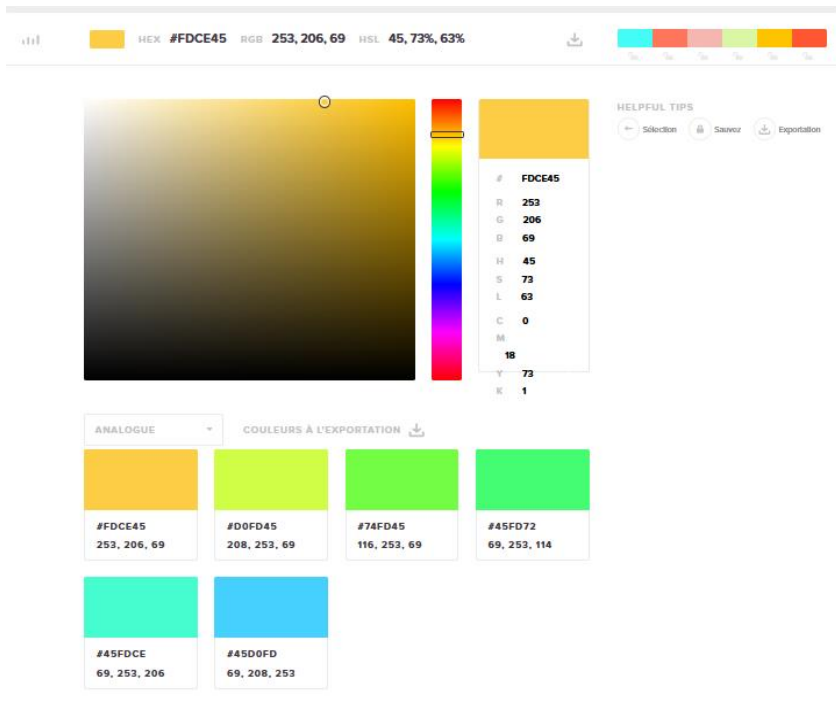
## TP – Pile et Files 2 - Pile dynamique

### 2.2. Rappels sur les dictionnaires : Création d'un dictionnaire de couleurs

A la suite sur la même indentation on désire obtenir un panel de couleurs « flashie » pour remplir chaque rectangle aléatoirement

Pour obtenir un panel de couleurs « flashie » on utilise le site

<https://htmlcolorcodes.com/fr/selecteur-de-couleur/>



Puis aller dans couleurs à l'exportation pour obtenir les couleurs RGB

EXPORT ×

### Analogous colors

HEX	RGB	HTML
#fdce45	rgb(253, 206, 69)	style="color:#fdce45;"
#d0fd45	rgb(208, 253, 69)	style="color:#d0fd45;"
#74fd45	rgb(116, 253, 69)	style="color:#74fd45;"
#45fd72	rgb(69, 253, 114)	style="color:#45fd72;"
#45fdce	rgb(69, 253, 206)	style="color:#45fdce;"
#45d0fd	rgb(69, 208, 253)	style="color:#45d0fd;"

Créer enfin un dictionnaire « col » dans lequel les clés vont de 1 à 5 et les valeurs sont les 3 upplets RGB

Ajouter avec la même indentation ce code à la suite dans le constructeur

```
#Implémentation de la variable "couleur" choisie aléatoirement dans le dictionnaire de couleurs "col"
```

## TP – Pile et Files 2 - Pile dynamique

### 2.3. Dessin du rectangle

La classe rectangle hérite de Turtle ce qui implique qu'elle peut utiliser toutes les méthodes de Turtle. Comme ces méthodes sont disponibles au sein de la classe Rectangle pour les appliquer à elle-même il suffit de les appeler précédées de **self**-----.

On adapte la méthode de traçage déjà vue dans le TP sur la POO. Compléter le code ci-dessous et le mettez-le à la suite du constructeur avec la même indentation

```
#dessin du rectangle Largeur 300 Hauteur 40
L = ?
H = ?
self.up()
self.speed(10)
self.hideturtle()
self.goto( ? , ?)
self.color( ? )
self.fillcolor( ? )
self.up()
self.goto(self.x-L/2 , self.y-H/2 )
self.down()
self.hideturtle()
self.begin_fill()
for _ in range(2) :
    self.forward(L)
    self.left(90)
    self.forward(H)
    self.left(90)
self.end_fill()
```

On en profite pour écrire par-dessus chaque Rectangle les informations de la cellule correspondante.

```
# Ecriture dans le rectangle des information de la cellule correspondante
self.goto(self.x,self.y-10)
self.color("black")
self.fillcolor("black")
self.write(self.blabla, move=False, align="center", font=("Arial", 16, "bold"))
```

### 2.4. Gestion du dessin du rectangle

Une fois la classe rectangle existante il faut

Ajouter dans cellule un rectangle :

```
class Cellule():
    x = 0
    y = -380

    def __init__(self, valeur, suivant=None):
        A compléter : voir cours chainage des cellules

        self.emcombrement = sys.getsizeof(self)
        self.__class__.x
        self.__class__.y
        #place pour le futeur rectangle
        self.r = Rectangle(? , ? , ? (valeur) +" id : "+ str(id(self)))
```

Le dessin du rectangle reste dans la fenêtre si on ne prend pas soin de l'effacer au bon moment. Dans la classe Cellule il faut ajouter un « destructeur » qui est activé quand la Cellule est supprimée. Turtle permet d'effacer un objet avec la méthode clear(). Compléter à ajouter le code suivant :

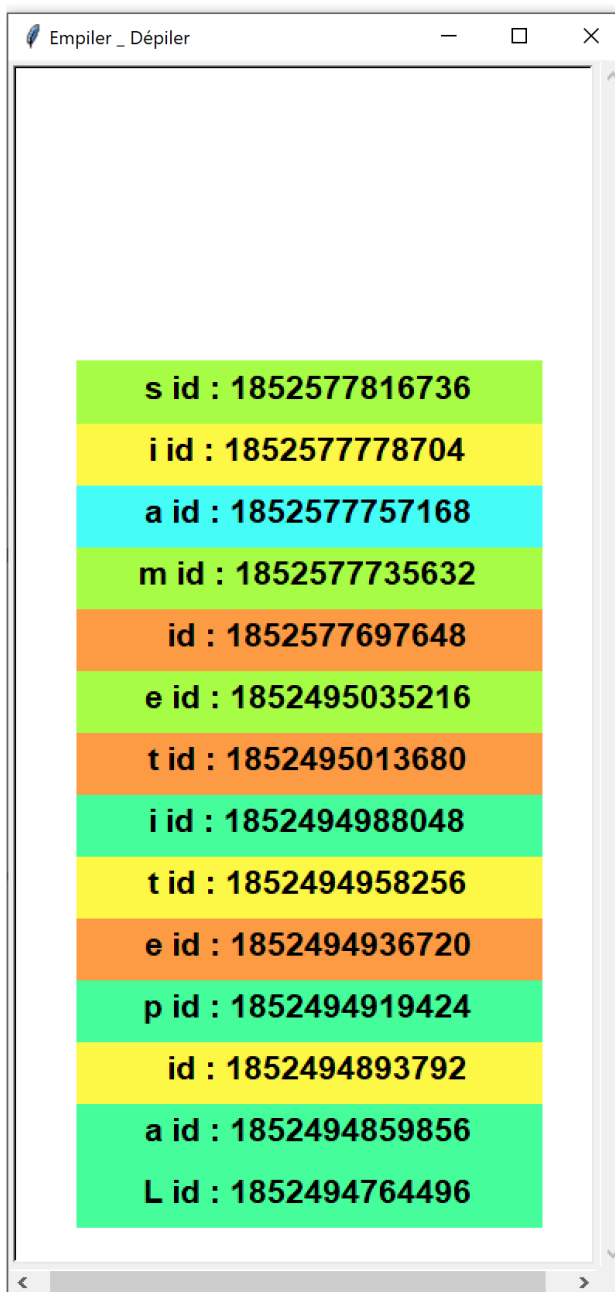
```
def __del__(self):
    self.r.?
    print("destructeur Cellule")
```

On en profite aussi pour faire afficher dans la console d'autres renseignements.

```
def __str__(self):
    return(f"(Cellule emcombrement : {self.emcombrement}, valeur : {self.valeur} suivant : {self.suivant} adresse : {id(self)} y : {self.__class__.y}")
```

### 3. Programme principal

Compléter le code pour obtenir :



A partir de la chaîne de caractères : « La petite maison » ( dans la prairie ).

Les couleurs seront nécessairement différentes.

## TP – Pile et Files 2 - Pile dynamique

### 4. Adaptation à une file

Une file est une structure de données abstraite.

On reprend l'idée de la file d'attente et on précise les opérations permises :

- On peut enfiler un élément (une personne arrive en queue de file)
- On peut défiler un élément (la personne en début de file sort de la file).
- On peut savoir si la file est vide ou non.

On peut trouver une implémentation sous la forme d'une liste doublement chaînée :

```
class Maillon:

    def __init__(self, valeur, precedent=None, suivant=None):
        self.valeur = valeur
        self.precedent = precedent
        self.suivant = suivant

class File:

    def __init__(self):
        self.longueur = 0
        self.debut = None
        self.fin = None

    def enfiler(self, valeur):
        if self.longueur == 0:
            self.debut = self.fin = Maillon(valeur)
        else:
            self.fin = Maillon(valeur, self.fin)
            self.fin.precedent.suivant = self.fin
        self.longueur += 1

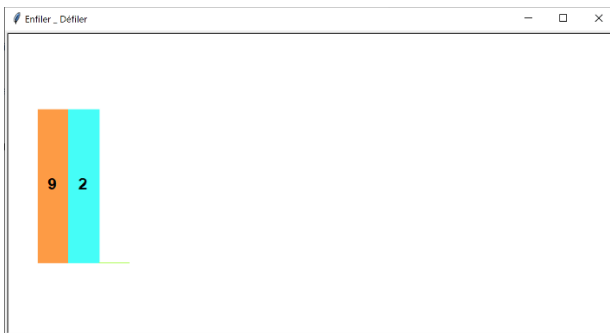
    def defiler(self):
        if self.longueur > 0:
            valeur = self.debut.valeur
            if self.longueur > 1:
                self.debut = self.debut.suivant
                self.debut.precedent = None
            else:
                self.debut = self.fin = None
            self.longueur -= 1
        return valeur
```



## TP – Pile et Files 2 - Pile dynamique

```
def estVide(self):  
    return self.longueur == 0  
  
def __str__(self):  
  
    ch = "\nEtat de la file:\n"  
    maillon = self.debut  
    while maillon != None:  
        ch += str(maillon.valeur) + " "  
        maillon = maillon.suivant  
    return ch  
  
q = File()  
q.enfiler(9)  
q.enfiler(2)  
q.enfiler(5)  
  
print(q)  
  
q.defiler()  
q.enfiler(7)  
  
print("La file est-elle vide: ", q.estVide())  
print(q)  
print("Longueur de la file:", q.longueur)
```

Transformer ce code avec la méthode précédente pour obtenir une représentation graphique du type :



Puis

