21.1_ENCAPSULER DES ATTRIBUTS
TOUT EST OBJET

L’encapsulation :

Nous avons vue précédemment que l’un des intérêts de la programmation orientée objet est de pouvoir utiliser les objets sans se préoccuper de leur fonctionnement interne.

Problématique :

Si, lors de l’usage d’un objet par un programme, celui-ci passe au constructeur un argument d’un type inapproprié, ce qui n’est nullement interdit, le programme va renvoyer une erreur.
Mais cela peut être contrôler facilement en modifiant le constructeur.

Exemple : Pour que notre formulaire puisse gérer une suite de lettres pour l’année de naissance on va procéder comme suit :

```
class formulaire:
    def __init__(self, nom, prenom, naissance):
        self.nom = nom.upper()
        self.prenom = prenom.upper()
        na = str(naissance)
        if na.isnumeric():
            self.naissance = int(na)
        else:
            self.naissance = 1900
    def age(self):
        return 2020 - self.naissance
    def majeur(self):
        return self.age() >= 18
    def memeFamille(self, formulaire):
        return self.nom == formulaire.nom
```

```
jd = formulaire('Doe', 'John', 2005)
jb = formulaire('Noob', 'John', '2004')
ad = formulaire('doe', 'Alice', 'yolo')

print(jd.age())
print(jb.age())
print(ad.age())
```

```
15
16
120
```

Problème :

Cependant, en l’état rien n’interdit au programmeur de modifier la valeur de l’attribut naissance sans passer par le constructeur.

```
ad = formulaire('doe', 'Alice', 'yolo')
ad.naissance = 'yolo'
print(ad.age())
```

Solution :

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-786dcdcc34b3> in <module>
      1 ad = formulaire('doe', 'Alice', 'yolo')
      2 ad.naissance = 'yolo'
----> 3 print(ad.age())

<ipython-input-5-1eb9ca708b91> in age(self)
      9             self.naissance = 1900
     10     def age(self):
---> 11         return 2020 - self.naissance
     12     def majeur(self):
     13         return self.age() >= 18

TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

l’encapsulation

L’idée général est d’éviter les effets de bord et les modifications non contrôlées en restreignant les accès en lecture et surtout en écriture des différent attribut.

Pour mettre en œuvre l’encapsulation on utilise :

- _get_naissance() pour définir comment se fait toute lecture d’un attribut donné.
- _set_naissance() pour définir comment se fait toute modification d’un atrribut donné.
- La fonction property().

L’encapsulation

```python
class formulaire:
    def __init__(self, nom, prenom, naissance):
        self.nom = str(nom).upper()
        self.prenom = str(prenom).upper()
        self.naissance = naissance
    def _set_naissance(self, naissance):
        na = str(naissance)
        if na.isnumeric():
            self._naissance = int(na)
        else:
            self._naissance = 1900
    def _get_naissance(self):
        return self._naissance
    naissance = property(_get_naissance, _set_naissance)
    def age(self):
        return 2020 - self.naissance
    def majeur(self):
        return self.age() >= 18
    def memeFamille(self, formulaire):
        return self.nom == formulaire.nom
```

```python
ad = formulaire('doe', 'Alice', 'yolo')
print(ad.age())
ad.naissance = 'yolo'
print(ad.age())
ad.naissance = '1990'
print(ad.age())
```

```
120
120
30
```

Exercice :

1) Modifier _get_naissance() de sort qu’il provoque un affichage dans la console avant de renvoyer la valeur. L’appel au calcul de l’âge provoque-t-il un affichage supplémentaire ? Pourquoi ?
2) Modifier _set_naissance() de sorte qu’il gère aussi le cas ou la date de naissance est donnée comme une liste.
3) Utiliser des propriété similaire pour encapsuler le nom et le prénom.