Programmer en Python
Licence 2, Mathématiques
V. Monbet
TD 3. Vers la programmation orientée objet

1 Classe complexe

Un complexe z est composé d'une partie réelle x et d'une partie imaginaire y, telles que $z = x + iy$. Nous proposons ici d'implémenter une classe `Complex`.

Commencez par définir le i complexe.

```
>>> i=complex(0,1)
>>> print(i**2)
```

1. Définir une classe `Complex` et la commenter.
2. Si cela n'a pas déjà été fait, ajouter un constructeur puis
   - ajouter deux attributs x et y ;
   - ajouter au constructeur la possibilité de spécifier les valeurs de x et y.
3. Créer un objet `c1`, instance de la classe `Complex`.
4. Affecter la valeur 3 à l'attribut x et 4 à l'attribut y de l'objet `c1`.
5. Les fonctions peuvent utiliser des objets comme paramètres.
   Définir une fonction `afficheComplex` qui affiche les valeurs du complexe passé en paramètre.
   Testez la avec l'objet `c1`.
6. Créer une méthode (fonction dans la classe) `show` qui affiche les attributs de la classe `Complex`.
   Testez la avec l'objet `c1`.
7. Ajouter les méthodes permettant de passer d'une représentation en coordonnées cartésiennes à une représentation en coordonnées polaires et inversement.
8. Ajouter une méthode `showPolaire` qui affiche les valeurs du complexe passé en paramètre sous sa forme $\rho e^{i\phi}$.
9. Créer une fonction `compareComplex` qui renvoie vrai si chaque attribut de deux objets sont identiques.
   Créer un objet `c2` de type `Complex`, tester `compareComplex`.
10. Affecter la valeur 3 à l'attribut x et 4 à l'attribut y de l'objet `c2`
    Ajouter une méthode clone qui recopie un objet de type `Complex`.
    Effectuer les manipulations suivantes :

```
>>> print (c1 == c2)
>>> c1=c2; print (c1 == c2)
>>> c3=c1.clone(); print (c1 == c3)
```

2 Compte Bancaire

Définissez une classe `CompteBancaire`, qui permette d'instancier des objets tels que `compte1`, `compte2`, etc.

1. Le constructeur de cette classe initialisera deux attributs `nom` et `solde`, avec les valeurs par défaut "Dupont" et 1000.
2. Trois autres méthodes seront définies :
   (a) `depot(somme)` permettra d'ajouter une certaine somme au solde.
   (b) `retrait(somme)` permettra de retirer une certaine somme du solde
   (c) `affiche()` permettra d'afficher le nom du titulaire et le solde de son compte.

Exemples d'utilisation de cette classe :

```
compte1 = CompteBancaire("Machin" , 800)
compte1.depot(350)
compte1.retrait(200)
compte1.affiche( )
compte2 = CompteBancaire( )
compte2.depot(50)
compte2.affiche( )
```

3 Régression linéaire

Nous allons construire une classe qui définit un modèle de régression linéaire dans le cas univarié.

La régression linéaire simple consiste à modéliser l'évolution d'une quantité Y en fonction d'une quantité X par une relation linéaire ; ie

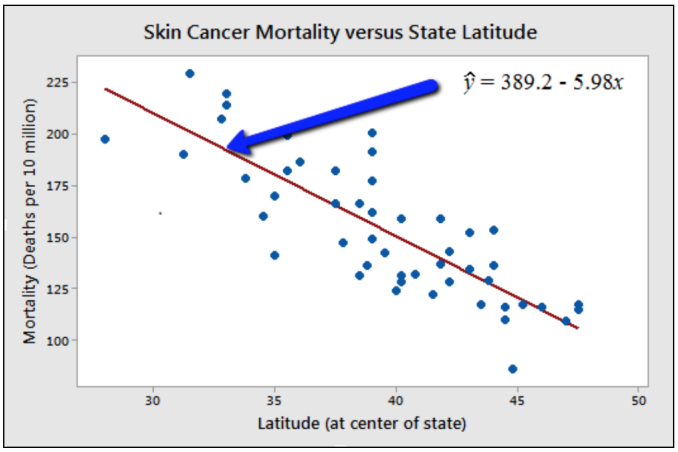
$$Y = aX + b + \epsilon$$

où ϵ représente une erreur de modèle.
En pratique, on dispose de n observations $\{(x_1, y_1), \cdots, (x_n, y_n)\}$ et on doit estimer la pente a et l'intercept b. Ceci est réalisé par la méthode des moindres carrées. C'est à dire qu'on cherche les valeurs $\hat{a}$ et $\hat{b}$ qui réalisent le minimum de

$$\sum_{i=1}^{n} (y_i - ax_i - b)^2$$

.

Par exemple, considérons des données de nombre de cancer de la peau en fonction de la latitude du centre de l'état aux US.



Cette classe sera décrite par les attributs `donnees`, `intercept` et `pente`. Les méthodes à implémenter sont :
- `fit` pour ajuster le modèle par la méthode des moindres carrés
- `get_param` qui renvoie les paramètres (intercept et pente)
- `predict` pour prédire la valeur de y pour un (ou plusieurs) nouvel(eaux) x
- `plotReg` pour tracer les données et la droite ajustée.
- `plotRes` pour tracer les résidus (ou erreurs).

Implémenter la classe `RegLin` et la tester sur les données de cancer (`Skin_cancer.dat`).

Une version multivarié de cette classe est disponible dans le package sklearn.

```
os.chdir('/users/valerie/Dropbox/ENSEIGNEMENT/PYTHON')
skin_cancer = open('skin_cancer.dat', 'r')
lignes = skin_cancer.readlines()
skin_cancer.close()

tab = []
for chn in lignes:
    tab.append(chn.split(' ,'))
state = []
latitude = np.zeros(len(tab)-1)
mortalite = np.zeros(len(tab)-1)
for k in range(len(tab)-1):
    state.append(tab[k+1][0])
    latitude[k] = float(tab[k+1][1])
    mortalite[k] = float(tab[k+1][2])

x = latitude
y = mortalite
```

```
from sklearn import linear_model

lr = linear_model.LinearRegression()
lr.fit(np.reshape(x,[len(y),1]),np.reshape(y,[len(y),1]))
lr.coef_
lr.intercept_
```