La programmation orienté objet

I.Notion de Class, Instance et Méthode

1. Class

   Les class sont des objets qui nous permettent de crée nos propre type

   Et pour definir une class on utilise Class suivue du nom de la class

*Exemple

```
class Phrase:
    ma_phrase="je suis data ingenieur"
```

Une class a son propre espace de nommage et pour acceder a l'espace de nommage on utilise nom de la class point **dict**.
Et Vars () permet d’accéder au dictionnaire de l’espace de nommage

*Exemple

```
Phrase.__dict__
vars(Phrase)
```

2. Instance d'une class

   A chaque fois qu’on appelle une class il va créer de nouvelle instance. Et il ya une relation entre la class et son instance : c’est une relation d’héritage.Les class et les instances sont des Objets mutables.

   *Exemple d'instance

```
p=Phrase()
```

3. Les méthodes

   Les méthodes sont des fonctions définie a l'intérieur des class . il défini le comportement des instances.

   *Exemple

```
s="je suis data ingenieur"
class Phrase
    def initia(self,ma_phrase):
        self.ma_phrase=ma_phrase
p=Phrase()
p.initia(s)
```

II.Notion de Méthode Spécial

Une méthode spécial nous permet de créer nos propre class. Il commence par __ et se termine par __.Avec les méthodes special on peut implémenter plusieurs opérations sur notre class.
On distingue plusieurs méthodes spécial parmis lesquels on peut citer:
def **init**():Pour initialiser les instances
Def **contrains**(self, mots sur lequel on fait le tes): Pour vérifier un test d'appartenance. Il va retourner un booléen. True si mot se trouve dedans et false s'il ne se trouve pas.
def **len**(): pour savoir la longueur de l'objet
def **str**().

*Exemple

```
class Phrase:
    def __init__(self):
        self.ma_phrase=ma_phrase
        self.mots=ma_phrase.split
    def __len__(self):
        return len(self.mots)
p=Phrase("je suis data ingenieur")
len(p)
    def __contrain__(self,mot):
        return mots in self.mots
"mocc" in p
"data" in p
```

III.Arbre d'héritage

une class qui se comporte comme une autre class
Une class peut hériter d’un autre class
Pour montrer qu’une class hérité un Aure class on le met () pendant la créations de la class.
Isinstance () permet de vérifier si votre Object directement d’une class ou du superclasse
Quand on hérite une class on hérite aussi tous
ses méthodes.

*Exemple

```
s="je suis data ingénieur"
class Phrase:
    def __init__(self):
        self.ma_phrase=ma_phrase
        self.mots=ma_phrase.split
    def __len__(self):
        return len(self.mots)
    def __contrain__(self,mot):
        return mots in self.mots
p=Phrase("je suis data ingénieur")
len(p)
"mocc" in p
"data" in p
```

```
class Newphrase(Phrase):
    pass
p_no=Newphrase(s)
isinstance(p_no,Phrase)
isinstance(p_no,Newphrase)
```