Cours et objets du jour-21

Classes et objets

Python est un langage de programmation orienté objet. Tout dans Python est un objet, avec ses propriétés et ses méthodes. Un nombre, une chaîne, une liste, un dictionnaire, un tuple, un ens emble, etc. utilisé dans un programme est un objet d'une classe intégrée correspondante. Nous créons une classe pour créer un objet. Une classe est comme un constructeur d'objets ou un "pl an" pour créer des objets. Nous instancions une classe pour créer un objet. La classe définit les attributs et le comportement de l'objet, tandis que l'objet, en revanche, représente la classe.

Nous travaillons avec des cours et des objets dès le début de ce défi sans le savoir. Chaque é lément d'un programme Python est un objet d'une classe. Vérirons si tout dans Python est u ne classe:

```
asabeneh @ asabeneh: ~ $ python python 3.9.6 (par défaut, 28 juin 2021, 15:26:21) [Cl
ang 11.0.0 (Clang-1100.0.33.8)] sur Darwin Type "Help", "Copyright", "crédits" ou "Li
cence" pour plus d'informations.  >>> num = 10 >>> type (num) < class 'int' > >>> str
ing = 'string' >>> type (string) < class 'str' > >>> boooolean = true >>> < class 'bool' >
>>> lst = [] >>> type (lst) < class 'list' > >>> tpl = () >>> type (tpl) < class '> >>> set
1 = set () >>> type (set1) < classe 'set' > >>> dct = {} >>> type (dct) < classe 'dict' >
```

Créer une classe

Pour créer une classe, nous avons besoin de la classe de mots clés suivie du nom et du c
ôlon. Le nom de classe doit être Camelcase.

```
# Syntax Class ClassNam
e: le code va ici
```

Exemple:

```
Personne de classe: P
ass Print (personne)
```

<

```
  __main __. Objet personne à 0x10804e510 >
```

Créer un objet

Nous pouvons créer un objet en appelant la classe.

```
p = personne ()
Imprimer (P)
```

Constructeur de classe

Dans les exemples ci-dessus, nous avons créé un objet à partir de la classe de personne. Cependant,
Une classe sans constructeur n'est pas vraiment utile dans les applications réelles. Utilisons l
a fonction du constructeur pour rendre notre classe plus utile. Comme la fonction du constru
cteur dans Java ou JavaScript, Python a également une fonction de constructeur init () intégr
ée. La fonction du constructeur Init a un paramètre d'auto-paramètre qui est une référence à
l'instance actuelle des exemples de classe:

```
Personne de classe: def __init__ (self, nom): # self permet d'attacher le paramètre à
la classe self.name = name p = personne ('asabeneh') print (p.name) print (p)
```

```
# sortie asabeneh < __ Main __. Objet personne à 0x2abf46907e
80 >
```

Ajoutons plus de paramètres à la fonction du constructeur.

```
class Person:
```

```
Def __init __ (Self, FirstName, LastName, Age, Country, City): Self.FirstName = FirstNa
me self.lastname = LastName self.age = Age self.country = country self.city = City City




P = Person ('Asabeneh', 'YEAYEH', 250, 'Finland', 'Helsinki') Print (P.FirstName) Print (P.La
stName) Print (P.Age) Print (P.Country) Print (P.City)
```

```
# Sortie Asab
eneh YEAYE
H 250 Finlan
de Helsinki
```

Méthodes d'objet

Les objets peuvent avoir des méthodes. Les méthodes sont des fonctions qui appartiennent à l'objet.

Exemple:

```
class Person:
      def __init__(self, firstname, lastname, age, country,
city):
          self.firstname = firstname
          self.lastname = lastname
          self.age = age
          self.country = country
          self.city = city
      def person_info(self):
        return f'{self.firstname} {self.lastname} is
{self.age} years old. He lives in {self.city}, {self.country}'

p = Person('Asabeneh', 'Yetayeh', 250, 'Finland', 'Helsinki')
print(p.person_info())
```

```
# sortir
```

```
Asabeneh Yetayeh is 250 years old. He lives in Helsinki,
Finland
```

Méthodes par défaut d'objet

Parfois, vous voudrez peut-être avoir des valeurs par défaut pour vos méthodes d'objet. Si no us donnons des valeurs par défaut pour les paramètres du constructeur, nous pouvons éviter l es erreurs lorsque nous appelons ou instancions notre classe sans paramètres. Voyons à quoi ça ressemble:

Exemple:

```
class Person:
      def __init__(self, firstname='Asabeneh',
lastname='Yetayeh', age=250, country='Finland',
city='Helsinki'):
          self.firstname = firstname
          self.lastname = lastname
          self.age = age
          self.country = country
          self.city = city

      def person_info(self):
        return f'{self.firstname} {self.lastname} is
{self.age} years old. He lives in {self.city},
{self.country}.'

p1 = Person()
print(p1.person_info())
p2 = Person('John', 'Doe', 30, 'Nomanland', 'Noman city')
print(p2.person_info())
```

```
# sortir
Asabeneh Yetayeh a 250 ans. Il vit à Helsinki,
Finlande.
John Doe a 30 ans. Il vit à Noman City, Nomanland.
```

Méthode pour modifier les valeurs par défaut de classe

Dans l'exemple ci-dessous, la classe de personne, tous les paramètres du constructeur ont des valeurs par défaut. En plus de cela, nous avons un paramètre de compétences, auquel nous po uvons accéder à l'aide d'une méthode. Créons la méthode add_skill pour ajouter des compéte nces à la liste des compétences.

```
class Person:
```

```
      def __init__(self, firstname='Asabeneh',
lastname='Yetayeh', age=250, country='Finland',
city='Helsinki'):
          self.firstname = firstname
          self.lastname = lastname
          self.age = age
          self.country = country
          self.city = city
          self.skills = []

      def person_info(self):
        return f'{self.firstname} {self.lastname} is
{self.age} years old. He lives in {self.city},
{self.country}.'
      def add_skill(self, skill):
          self.skills.append(skill)

p1 = Person()
print(p1.person_info())
p1.add_skill('HTML')
p1.add_skill('CSS')
p1.add_skill('JavaScript')
p2 = Person('John', 'Doe', 30, 'Nomanland', 'Noman city')
print(p2.person_info())
print(p1.skills)
print(p2.skills)
```

sortir

```
Asabeneh Yetayeh a 250 ans. Il vit à Helsinki, en Finlande.

John Doe a 30 ans. Il vit à Noman City, Nomanland.
['Html', 'css', 'javascript']
[]
```

Héritage

En utilisant l'héritage, nous pouvons réutiliser le code de classe parent. L'héritage nous permet de définir une classe qui hérite de toutes les méthodes et propriétés de la classe parent. La clas se parent ou la classe Super ou Base est la classe qui donne toutes les méthodes et propriétés. La classe d'enfants est la classe qui hérite d'une autre classe ou d'une classe parent. Créons une classe étudiante en héritant de la classe de personne.

```
class Student(Person):
    pass


s1 = Student('Eyob', 'Yetayeh', 30, 'Finland', 'Helsinki')
s2 = Student('Lidiya', 'Teklemariam', 28, 'Finland', 'Espoo')
```

```
print (s1.person_info ()) s1.add_skill ('j
avascript')
s1.add_skill ('react') s1.add_skill ('
python') print (s1.skills)


print (s2.person_info ()) s2.add_skill ('organisatio
n') s2.add_skill ('marketing') s2.add_skill ('market
ing numérique'))

imprimer (s2.skills)
```

```
sortir
Eyob Yetayeh a 30 ans. Il vit à Helsinki, en Finlande.  ['Javascript', 'react', 'python'] lidiya te
klemariam a 28 ans. Il vit à Espoo, en Finlande.  [«Organisation», «marketing», «marketing
numérique»]
```

Nous n'avons pas appelé le constructeur init () dans la classe enfant. Si nous ne l'appelions pas, nous pouvons toujours accéder à toutes les propriétés du parent. Mais si nous appelons le const ructeur, nous pouvons accéder aux propriétés des parents en appelant *super*.

Nous pouvons ajouter une nouvelle méthode à l'enfant ou nous pouvons remplacer les métho des de classe parent en créant le même nom de méthode dans la classe enfant. Lorsque nous a jouterons la fonction init (), la classe infantile n'héritera plus de la fonction init () du parent.

Méthode des parents dominants

```
classe étudiante (personne): def __init__ (self, firstName = 'asabeneh', lastname = 'yetaye
h', = 250, country = 'finland', ville = 'helsiki', genre = 'male'): Self.gende = Gender Gend
er Gender} Super () .__ Init __ (FirstName, LastName, Age, Country, City) Def Person_i
nfo (self): Gender = 'He' If self.gender == 'male' else 'Elle' renvoie f '{self.firstname} {se
lf.lastname} est {self.age} ans. {Gender} vit dans {self.city}, {self.country}. '
```

```
S1 = étudiant ('eyob', 'yetayeh', 30, 'Finland', 'Helsinki', 'mâle') s2 = étudiant ('lidiya', 'teklema
riam', 28, 'finland', 'Espoo', 'féminin') print (s1.person_info ()) s1.add_skill (javascript ')') ')
```

```
s1.add_skill ('react') s1.add_skill
('python')
imprimer (s1.skills)

print (s2.person_info ()) s2.add_skill ('organisatio
n') s2.add_skill ('marketing') s2.add_skill ('market
ing numérique'))

imprimer (s2.skills)

Eyob Yetayeh a 30 ans. Il vit à Helsinki, en Finlande.  ['Javascript', 'react', 'python'] lidiya te
klemariam a 28 ans. Elle vit à Espoo, en Finlande.  [«Organisation», «marketing», «marketi
ng numérique»]
```

Nous pouvons utiliser une fonction intégrée Super () ou la personne du nom de parent pour hér iter automatiquement des méthodes et des propriétés de son parent. Dans l'exemple ci-dessus, n ous l'emportons sur la méthode parent. La méthode de l'enfant a une caractéristique différente, elle peut identifier, si le sexe est un homme ou une femme et attribuer le pronom approprié (il / elle).

Maintenant, vous êtes entièrement chargé d'une super puissance de programmation. Fait es maintenant quelques exercices pour votre cerveau et vos muscles.

Exercices: Jour 21

Exercices: niveau 1

1. Python a le module appelé *statistics* et nous pouvons utiliser ce module pour faire tous les calculs statistiques. Cependant, pour apprendre à faire de la fonction et à réutiliser la fo nction, essayons de développer un programme, qui calcule la mesure de la tendance centra le d'un échantillon (moyenne, médiane, mode) et la mesure de la variabilité (plage, varian ce, écart-type). En plus de ces mesures, trouvez la distribution Min, Max, Count, centile et fréquence de l'échantillon. Vous pouvez créer une classe appelée statistique et créer toutes les fonctions qui effectuent des calculs statistiques comme méthodes pour la classe de stat istiques. Vérifiez la sortie ci-dessous.

```
Âges = [31, 26, 34, 37, 27, 26, 32, 32, 26, 27, 27, 24, 32, 33, 27, 25, 26, 38, 37, 31, 34, 24, 33, 29, 26]

print ('count:', data.count ()) # 25 print ('sum:', data.sum ()) # 744 print ('min:', data.min ()) # 24 print ('max:', data.max ()) # 38 print ('plage:', data.range () # 14 print ('mean. print ('mode:' , data.mode ()) # {'mode': 26, 'count': 5} print ('standard écart:', data.std ()) # 4.2 print ('varian ce:', data.var ()) # 17.5 print ('Distribution de fréquence:', data.freq_dist ()) # [(20.0, 26), (16.0 , 27), (12.0, 32). (8.0, 37), (8.0, 34), (8.0, 33), (8.0, 31), (8.0, 24), (4.0, 38), (4.0, 29), (4.0, 25) ]
```

```
# you output should look like this  print(data.describe())  Count: 25  Sum:  744  Min:  24  M ax: 38 Range: 14 Mean: 30 Median: 29 Mode: (26, 5) Variance: 17.5 Standard Devia tion:  4.2  Frequency Distribution: [(20.0, 26), (16.0, 27), (12.0, 32),  (8.0, 37), (8.0, 34), (8. 0, 33), (8.0, 31), (8.0, 24), (4.0, 38), (4.0, 29), (4.0, 25)]
```

Exercices: niveau 2

1. Créez une classe appelée PersonAccount. Il a FirstName, LastName, Revenmes, dépenses les propriétés et il a Total_income, Total_Expense, Account_info, ADD_INCOE, ADD_EXPENSE et Account_Balance. Les revenus sont un ensemble de revenus et sa description. Il en va de même pour les dépenses.