Python orienté objet

Classes / objets Python

Python est un langage de programmation orienté objet.

Presque tout en Python est un objet, avec ses propriétés et ses méthodes.

Une classe est comme un constructeur d'objets, ou un "plan" pour créer des objets.

Créer une classe

Pour créer une classe, utilisez le mot-clé `class`:

Exemple

Créez une classe nommée MyClass, avec une propriété nommée x:

```
class MyClass:
  x = 5
```

Créer un objet

Nous pouvons maintenant utiliser la classe nommée MyClass pour créer des objets:

Exemple

Créez un objet nommé p1 et affichez la valeur de x:

```
p1 = MyClass()
print(p1.x)
```

La fonction __init __ ()

Les exemples ci-dessus sont des classes et des objets dans leur forme la plus simple et ne sont pas vraiment utiles dans les applications réelles.

Pour comprendre la signification des classes, nous devons comprendre la fonction intégrée __init __ ().

Toutes les classes ont une fonction appelée __init __ (), qui est toujours exécutée lorsque la classe est lancée.

Utilisez la fonction __init __ () pour attribuer des valeurs aux propriétés de l'objet ou à d'autres opérations nécessaires lors de la création de l'objet:

Exemple

Créez une classe nommée Person, utilisez la fonction __init __ () pour attribuer des valeurs pour le nom et l'âge:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)
```

```
print(p1.name)
print(p1.age)
```

Remarque: La `__init__()`fonction est appelée automatiquement chaque fois que la classe est utilisée pour créer un nouvel objet.

Méthodes d'objets

Les objets peuvent également contenir des méthodes. Les méthodes dans les objets sont des fonctions qui appartiennent à l'objet.

Créons une méthode dans la classe Person:

Exemple

Insérez une fonction qui imprime un message d'accueil et exécutez-la sur l'objet p1:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Remarque: Le `self`paramètre est une référence à l'instance actuelle de la classe et est utilisé pour accéder aux variables appartenant à la classe.

Le paramètre de soi

Le `self`paramètre est une référence à l'instance actuelle de la classe et est utilisé pour accéder aux variables appartenant à la classe.

Il n'a pas besoin d'être nommé `self`, vous pouvez l'appeler comme vous le souhaitez, mais il doit être le premier paramètre de toute fonction de la classe:

Exemple

Utilisez les mots *mysillyobject* et *abc* au lieu de *self* :

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Modifier les propriétés de l'objet

Vous pouvez modifier les propriétés d'objets comme celui-ci:

Exemple

Définissez l'âge de p1 sur 40:

```
p1.age = 40
```

Supprimer les propriétés de l'objet

Vous pouvez supprimer des propriétés sur des objets en utilisant le `del`mot - clé:

Exemple

Supprimez la propriété age de l'objet p1:

```
del p1.age
```

Supprimer des objets

Vous pouvez supprimer des objets en utilisant le `del`mot - clé:

Exemple

Supprimez l'objet p1:

```
del p1
```

La déclaration de réussite

`class`les définitions ne peuvent pas être vides, mais si, pour une raison quelconque, vous avez une `class`définition sans contenu, insérez l' `pass`instruction pour éviter d'obtenir une erreur.

Exemple

```
class Person:
  pass
```