Python Classes and Objects

https://www.w3schools.com/python/python_classes.asp

Python — это объектно-ориентированный язык программирования.
Почти все в Python является объектом со своими свойствами и методами.
Класс подобен конструктору объектов или «чертежу» для создания объектов.t

---

Чтобы создать класс, используйте ключевое слово `class`:

Создайте класс с именем MyClass со свойством с именем x:

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

Теперь мы можем использовать класс с именем MyClass для создания объектов:

Создайте объект с именем p1 и выведите значение x:

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

Функция __init__()

Приведенные выше примеры представляют собой классы и объекты в их простейшей форме и не очень полезны в реальных приложениях.
Чтобы понять значение классов, мы должны понять встроенную функцию __init__().
Все классы имеют функцию __init__(), которая всегда выполняется при инициализации класса.
Используйте функцию __init__() для присвоения значений свойствам объектаили других операций, которые необходимо выполнить при создании объекта:

Создание класса с именем Person, используйте функцию __init__() для присвоения значений имени и возрасту:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
p1 = Person("John", 36)
print(p1.name)
print(p1.age)
```

Функция `__init__()` вызывается автоматически каждый раз, когда класс используется для создания нового объекта.

Методы объекта

Объекты также могут содержать методы. **Методы в объектах — это функции, принадлежащие объекту.**

Создадим метод в классе Person:

Вставьте функцию, которая печатает приветствие, и выполните ее для объекта 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()

Hello my name is John
```

Параметр `self` является ссылкой на текущий экземпляр класса и используется для доступа к переменным, принадлежащим классу.

The self Parameter

Параметр `self` является ссылкой на текущий экземпляр класса и используется для доступа к переменным, принадлежащим классу.

Его не обязательно называть `self`, вы можете называть его как хотите, но он должен быть первым параметром любой функции в классе:

Используйте слова *mysillyobject* и *abc* вместо *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()

# or

class Person:
  def __init__(name, age):
    name.age = age
  def myfunc(abc):
    print("Hello my age is " + str(abc.age))
p1 = Person(36)
p1.myfunc()
```

```
Hello my age is 36
```

Modify Object Properties

You can modify properties on objects like this: Set the age of p1 to 40:

```
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.age = 40
p1.myfunc()
print(p1.age)

Hello my name is John
40
```

Delete Object Properties

You can delete properties on objects by using the `del` keyword:

```
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)
del p1.age
if hasattr(p1, 'age'):
        print(p1.age)
else:
        print("Sorry, the property of this object has been removed!")

Sorry, the property of this object has been removed!
```

Check, if attribute exists in class:

```
if hasattr(a, 'property'):
    a.property
```

Delete Objects

You can delete objects by using the `del` keyword:

```
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)
del p1
# print(p1)
try:
    p1
except NameError:
    p1_exists = False
    print("This object does not exist(")
else:
    p1_exists = True

This object does not exist(
```

Check, if object exists:

```
# print(p1)
try:
    p1
except NameError:
    p1_exists = False
    print("This object does not exist(")
else:
    p1_exists = True
```

It's was **EAFP** style, "**E**asier to **A**sk **F**orgiveness than **P**ermission"

The pass Statement

`class` definitions cannot be empty, but if you for some reason have a `class` definition with no content, put in the `pass` statement to avoid getting an error.

```
class Person:
  pass
# having an empty class definition like this, would raise an error without the pass statement
```

Example:

```
class Tire:
    """
    Tire represents a tire that would be used with an automobile.
    """

    def __init__(self, tire_type, width, ratio, diameter, brand='', construction='R'):
        self.tire_type = tire_type
        self.width = width
        self.ratio = ratio
        self.diameter = diameter
        self.brand = brand
        self.construction = construction


```