https://goo.gl/8aM4s2

ITI1120 Review Session

Study Sheet

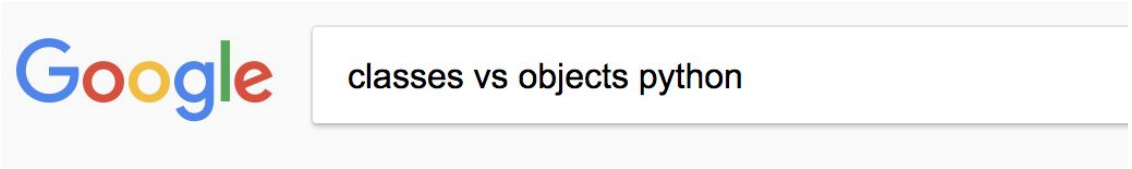
Classes and Objects

Author: Faizaan Chishtie

Classes and Objects

- Classes? Objects?
- What are classes?
- Why do we use classes?
- Syntax

Classes vs. Objects:



What is the difference between objects and classes in Python

5

These are two closely related terms in object oriented programming. The standard meaning is that an *object* is an instance of a *class*.

share improve this answer

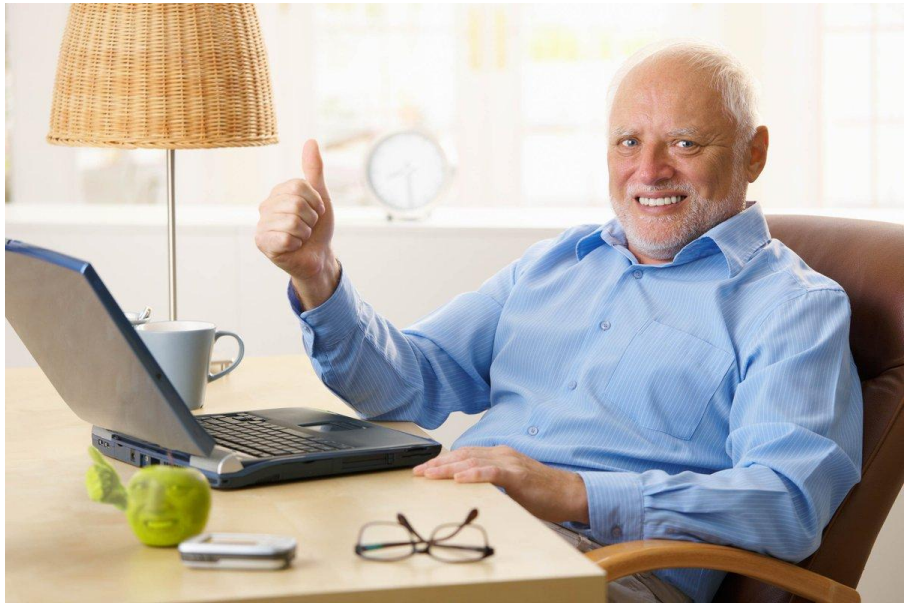
answered Sep 20 '11 at 10:52

David Heffernan

475k • 30 • 689 • 1075

add a comment

Classes vs. Objects:



What does that mean?

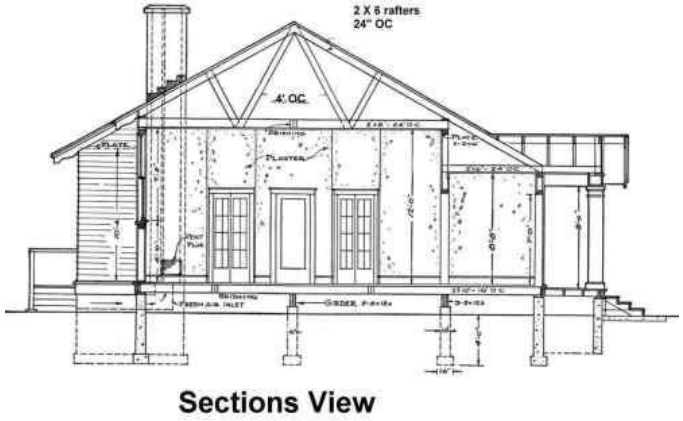
- Key term:
  - Instance

“The realization of something”

Classes vs. Objects:

In the same way that

Is an instance of



Classes vs. Objects:

Object Is an instance of a: **Class**

Classes and Objects

- Classes? Objects?
- What are classes?
- Why do we use classes?
- Syntax

What are classes?

9. Classes

Classes provide a means of bundling data and functionality together.

What are classes?

Imagine you wanted to create a program that modified the position of a Tesla in 2D space:

Data:

x_coordinate
y_coordinate

Functions:

move_tesla()
will_crash() #checks if given Tesla will crash into a truck.

Let’s create a class named Tesla

Tesla

```
class Tesla:
    #dont use this
    area to declare
    variables
    def __init__(self, x,y):
        self.x_coordinate = x
        self.y_coordinate = y

    def move_tesla(self, x_move, y_move):
        “”” (number, number) -> None
            Changes Tesla’s x and y coordinate
        “””
        self.x_coordinate = x_move
        self.y_coordinate = y_move
```

This is the way we initialize classes.
Key words: “__init__” and “self”

This is a method! May change data inside an object.

NOTE

Notice that unlike functions we can modify variables inside of the class.

NOTE

Notice that UNLIKE functions we can MODIFY variables INSIDE of the class.

NOTE

Notice that **UNLIKE** functions we can **MODIFY** variables **INSIDE** of the class.

NOTE

Notice that **UNLIKE** functions we can **MODIFY** variables **INSIDE** of the class.

Key Terms

"__init__"

- "When a class defines an __init__() method, class instantiation automatically invokes __init__() for the newly-created class instance."
- Init is called every time you call a class:

  Using our Tesla example:

  modelS = Tesla(3,4)

This line of code creates a new Tesla object called "modelS" with: x_coordinate = 3, and y_coordinate = 4

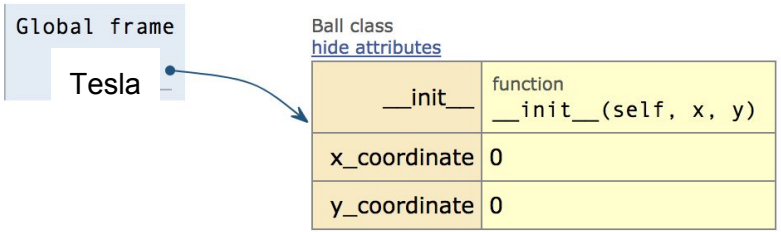
```
def __init__(self, x,y):
        self.x_coordinate = x
        self.y_coordinate = y
```

Let's see VISUALIZED VERSION

self <u>should</u> be written in the parameters

__init__(self,x,y)

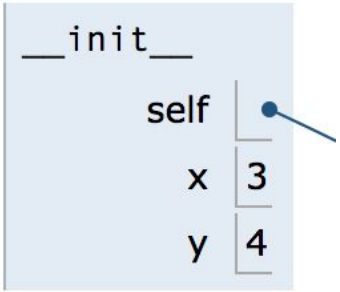
Class



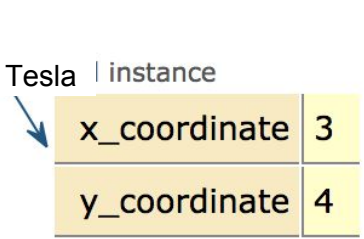
Instantiate class with variable name “Tesla”

```
9 modelS = Tesla (3,4)
```

The parameters x and y in the init function before being assigned

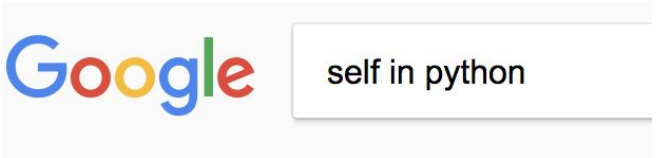


Instantiated modelS at (3,4)



Key Terms

“self”



self is a key word in Python that we use to **reference** the object we are **instantiating**.

What is the purpose of the `self` word in Python?

`self` is a reference to an object. It's very close to the concept of `this` in many C-style languages. Check out this code:

Methods

“A method is a function that takes a class instance as its first parameter.”

- Don’t forget your “self”
- This distinguishes a function from a method.

Important:

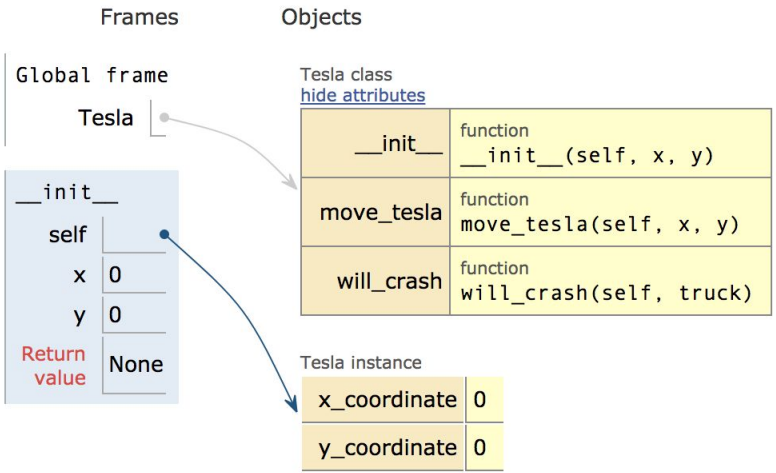
- Methods can modify the data inside of the class by using:
  - self.variablename = new stuff

```
def move_tesla(self, x_move, y_move):
    [redacted]
    self.x_coordinate = x_move
    self.y_coordinate = y_move
```

What do you think this does?

Let’s see this in the visualizer. (cleaner)

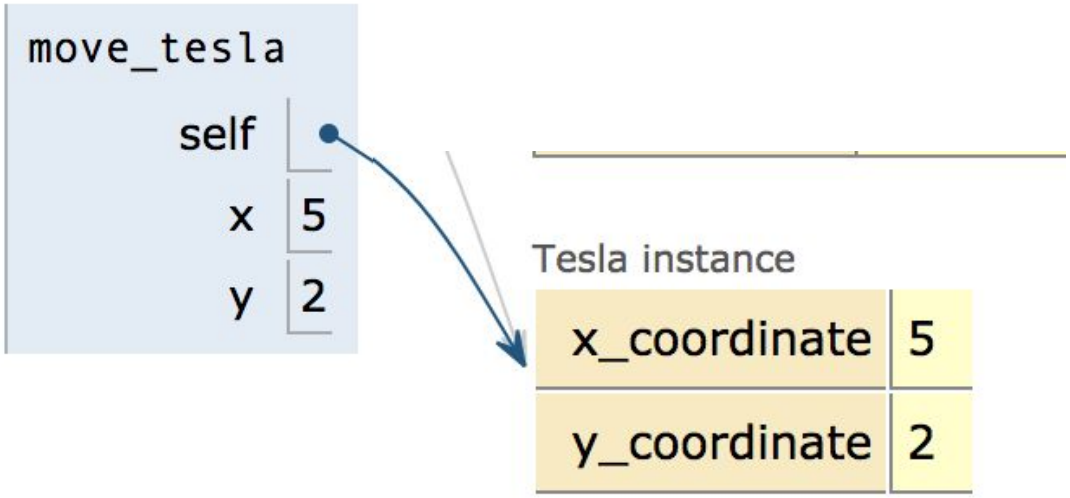
- I cleaned the code up a bit
  - Do not need to declare the data variables outside of the __init__ function.
  - Gives us the same result




```
1  class Tesla:
2      def __init__(self,x=0,y=0):
3          self.x_coordinate = x
4          self.y_coordinate = y
5
6      def move_tesla(self,x,y):
7          self.x_coordinate = x
8          self.y_coordinate = y
```

Let's see this in the visualizer II. (method)

```
modelS = Tesla()
modelS.move_tesla(5,2) #What will this do?
```



- Updates the Tesla's x_coordinate and y_coordinate!

Exquisite news!

Let’s have some fun with methods!

- Imagine the 2D plane we use is a city grid!
- New ModelS
- If any type of truck is within 2 units (x or y) of our car, we WILL crash.
  - Implies <= 2
- Let’s write a method that tells us whether or not we will crash into a truck!
  - Based on our (x,y) coordinates and the truck’s (x,y) coordinates.

Let’s have more fun with methods!

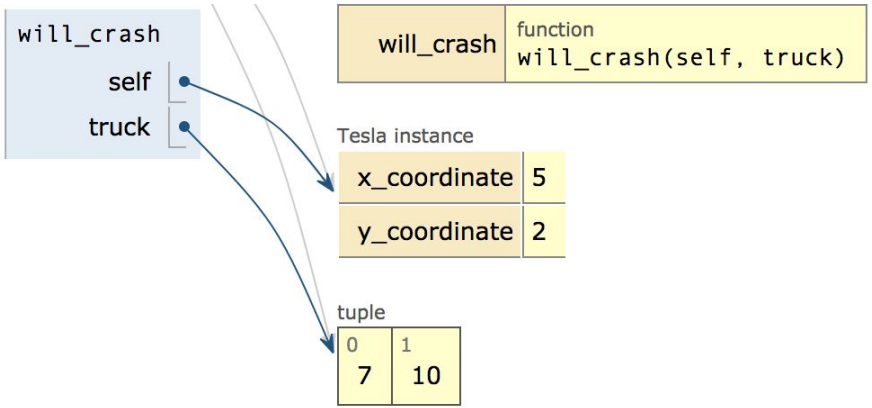
```
10      def will_crash(self, truck):
11          """ (tuple) --> boolean
12              if the truck is 2 units away from tesla they will crash
13          """
14          x_dist = self.x_coordinate - truck[0] #checks distance
15          y_dist = self.y_coordinate - truck[1]
16          x_dist = abs(x_dist) #takes absolute value
17          y_dist = abs(y_dist)
18          case_x = x_dist <= 2 #cases for crash
19          case_y = y_dist <= 2
20          if case_x and case_y:
21              return True
22          return False
```

Let’s have even more fun with methods by watching what happens when we put them through the visualizer!

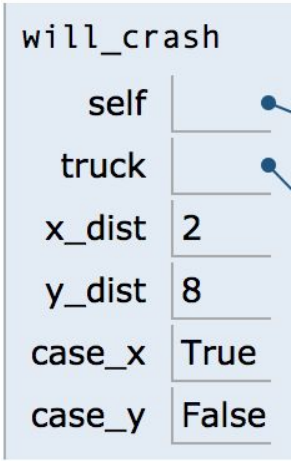
```
24  modelS = Tesla()
25  modelS.move_tesla(5,2)
26  truck = (7,10) #creates new tuple called truck
27  print(modelS.will_crash(truck)) #what will this print?
```

Let's visualize!

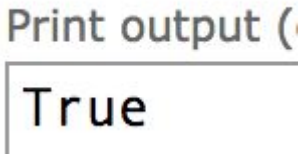
Calling the will_crash function with truck:



After going through will_crash code:



So it prints:



A method is able to operate on data that is contained within the class (remembering that an object is an instance of a class - the class is the definition, the object is an instance of that data).

Classes and Objects

- Classes? Objects?
- What are classes?
- Why do we use classes?
- Syntax

Why do we use classes?

- They make our code cleaner.

Classes and Objects

- Classes? Objects?
- What are classes?
- Why do we use classes?
- Syntax

Syntax

Let’s do this on the whiteboard together!

APPENDIX

Stuff you should know (Vida’s email)

arithmetic expressions in python (including +,-,*, /, exponentiation **, integer division //, mod %, ..)
- (compound) Boolean expressions (i.e. Boolean expressions with and, or, not)
- data types in Python
- type conversion
- function design
- function calls
- docstrings
- print vs return
- if statements
- for loops
- while loops

strings (including + operator being concatenation on strings)
- lists and tuples and operations on them including slicing (in ALL
forms), looping over elements of the list, looping over indices in the
list
- 2D lists (i.e. matrices)
- dictionaries and sets
- try/except
- file opening
- order of execution
- mutable vs immutable objects (i.e. variables that refer to immutable
objects like strings, numbers and tuples vs variables that refer to
mutable objects like lists, 2D lists and objects)
- copy vs aliasing (what is happening in the memory)
- counting how many times something is printed

The following functions/methods and operators appear in many of questions. Make sure you know everything we have seen about these functions.

print
range
.append (from list module)
len
'in' operator
'is' operator
'==' operator
'!=' operator

The following functions appear/methods in at least one function. Make sure you know what these functions do.

input
list
set
abs
open
sort
super
strip (from str module)
striplines (from str module)
lower (from str module)

- LOGIC/ALGORITHMS/SOLVING COMPUTATIONAL PROBLEMS (understanding what a program/function does, writing programs etc) on all of the above data types: lists, 2D lists, strings, objects etc ...
- sorting -- what it it used for and why; sorting algorithms eg. selection sort and merge sort (general idea on how they work and the number of operations it does roughly) and Python's built in sorting functions: sort and sorted. You do not need to memorize any sorting algorithms.
- linear search and binary search
- number of operations executed by presented programs/solutions and their efficacy (exact number of operations and couple of questions on big O)

- objects: variables referring to objects; accessing data fields and methods of objects via dot operator; designing methods that belongs to a class; overriding and extending methods, including those inherited from Python's object class such as __init__, __repr__, __str__, __eq__; creating and calling constructors; inheritance
- recursion including tracing recursive function calls and height/depth of recursion (i.e. number of functions on the stack) ...