Python basics V: Classes

Classes or declaring new types

Classes are blueprints for the creation of new types. Think of strings, lists, and dictionaries. They all are types that have both variables and functions (methods) that you access using "." operator. Using the `class` keyword you can define a completely new type.

```
# use the type function to get a better
# idea of this:
test_list = []
type(test_list)

#output
<type 'list'>

type(3)

#output
<type 'int'>
```

```
class NewType:
    pass # contains nothing

new_type = NewType() # create new
instance of NewType

type(new_type)

#output
<type 'instance'>
```

class variables (or attributes)

Classes can have both variables and functions that can be used from instances of new classes. In python these are called class attributes

```
class MyClass:
    variable = "i am a class variable"

class_instance_1 = MyClass()
print(class_instance_1.variable) # can access class variables with
"." operator

#output
i am a class variable

class_instance_1.variable = "can alter the value"
print(class_instance_1.variable)

#output
can alter the value
```

Each instance has its own variables

```
class MyClass:
    variable = "i am a class variable"

class_instance_1.variable = "can alter the value"
print(class_instance_1.variable)

#output
can alter the value

class_instance_2 = MyClass()
print(class_instance_2.variable)

#output
i am a class variable
```

Class methods

As mentioned earlier classes can also have functions which are called class methods.

```
class MyClass:
    def func():
        print("I am a function")

class_instance_1 = MyClass()
class_instance_1.func()

#output
I am a function
```

The point of classes

```
# a simple example of data organization
# We have 3 employees with lots of associated data.

# without classes:
first_name = ["John", "Joe", "David"]
last_name = ["Anderson", "Davidson", "Johnson"]
pay = [50000, 20000, 100000]
employee_level = [2, 1, 4]
emails = ["john@noname.com", "joe@noname.com", "dave@noname.com"]

# Now suppose we have an automated raise system that gives people a 10%
raise
# each year and we want to notify them

def give_raise(first_name, last_name, pay, email):
    message  = "congrats " + first_name + " " + last_name + " you have "
    message += "been given a yearly raise of " + str(pay*.10)
    send_email(email, message)

# and now we need to go through all employees

for i in range(0, len(emails)):
    give_raise(first_name[i], last_name[i], pay[i], emails[i])
    pay[i] += pay[i]*.10
```

The point of classes

```
class Employee:
    def __init__(self, first_name, last_name, pay, level, email):
        self.first_name = first_name
        self.last_name = last_name
        self.pay = pay
        self.level = level
        self.email = email

# now we can have employee objects the store all related information in
one place

employee_1 = Employee("John", "Anderson", 50000, 2, "john@noname.com")
employee_2 = Employee("Joe", "Davidson", 20000, 1, "joe@noname.com")
employee_3 = Employee("David", "Johnson", 100000, 4, "dave@noname.com")

def give_raise(employee):
    message  = "congrats " + employee.first_name + " " +
employee.last_name + " you have "
    message += "been given a yearly raise of " + str(employee.pay*.10)
    send_email(employee.email, message)
    employee.pay += employee.pay*.10
```

Magic class methods

Python has special class methods that control specific behavior of the class.

```
class SpecialNumber:
    def __init__(self, num):
        self.num = num

# suppose you wanted to add these classes together, specifically
their interal num variable

num_1 = SpecialNumber(1)
num_2 = SpecialNumber(2)
num_1 + num_2
File "test.py", line 11, in <module>
    num_3 = num_1 + num_2
TypeError: unsupported operand type(s) for +: 'instance' and
'instance'
```

Magic class methods

```
class SpecialNumber:
    def __init__(self, num):
        self.num = num

    # tells python how to deal with two SpecialNumber objects being
added
    def __add__(self, other):
        return SpecialNumber(self.num + other.num)

num_1 = SpecialNumber(1)
num_2 = SpecialNumber(2)
num_3 = num_1 + num_2

print(num_3)

#output
<__main__.SpecialNumber instance at 0x100f704d0>

print(num_3.num)

#output
3
```