Lecture 12 – EXAMPLES: Objects & Classes

https://cw.fel.cvut.cz/wiki/courses/be5b33prg/start

Michal Reinštein

Czech Technical University in Prague,
Faculty of Electrical Engineering, Dept. of Cybernetics,
Center for Machine Perception
http://cmp.felk.cvut.cz/~reinsmic/
reinstein.michal@fel.cvut.cz

EXAMPLES

```python
import datetime  # we will use this for date objects


class Person:

    def __init__(self, name, surname, birthdate, address, telephone, email):
        self.name = name
        self.surname = surname
        self.birthdate = birthdate

        self.address = address
        self.telephone = telephone
        self.email = email

    def age(self):
        today = datetime.date.today()
        age = today.year - self.birthdate.year

        if today < datetime.date(today.year, self.birthdate.month,
                                 self.birthdate.day):
            age -= 1

        return age


person = Person(
    "Jane",
    "Doe",
    datetime.date(1992, 3, 12),  # year, month, day
    "No. 12 Short Street, Greenville",
    "555 456 0987",
    "jane.doe@example.com"
)
```

```
In[3]: print(person.name)
Jane
In[4]: print(person.email)
jane.doe@example.com
In[5]: print(person.age())
25
```

Exercise 1

1. Explain what the following variables refer to, and their scope:
    1. `Person`
    2. `person`
    3. `surname`
    4. `self`
    5. `age` (the function name)
    6. `age` (the variable used inside the function)
    7. `self.email`
    8. `person.email`

Answer to exercise 1

1. 1. `Person` is a class defined in the global scope. It is a global variable.
   2. `person` is an instance of the `Person` class. It is also a global variable.
   3. `surname` is a parameter passed into the `__init__` method – it is a local variable in the scope if the `__init__` method.
   4. `self` is a parameter passed into each instance method of the class – it will be replaced by the instance object when the method is called on the object with the `.` operator. It is a new local variable inside the scope of each of the methods – it just always has the same value, and by convention it is always given the same name to reflect this.
   5. `age` is a method of the `Person` class. It is a local variable in the scope of the class.
   6. `age` (the variable used inside the function) is a local variable inside the scope of the `age` method.
   7. `self.email` isn't really a separate variable. It's an example of how we can refer to attributes and methods of an object using a variable which refers to the object, the `.` operator and the name of the attribute or method. We use the `self` variable to refer to an object inside one of the object's own methods – wherever the variable `self` is defined, we can use `self.email`, `self.age()`, etc..
   8. `person.email` is another example of the same thing. In the global scope, our person instance is referred to by the variable name `person`. Wherever `person` is defined, we can use `person.email`, `person.age()`, etc..

```python
import datetime  # we will use this for date objects


class Person:

    def __init__(self, name, surname, birthdate, address, telephone, email):
        self.name = name
        self.surname = surname
        self.birthdate = birthdate

        self.address = address
        self.telephone = telephone
        self.email = email

    def age(self):
        today = datetime.date.today()
        age = today.year - self.birthdate.year

        if today < datetime.date(today.year, self.birthdate.month,
                                 self.birthdate.day):
            age -= 1

        return age

```

Exercise 2

1. Rewrite the `Person` class so that a person's age is calculated for the first time when a new person instance is created, and recalculated (when it is requested) if the day has changed since the last time that it was calculated.

Answer to exercise 2

1. Here is an example program:

```
import datetime

class Person:

    def __init__(self, name, surname, birthdate, address, telephone, email):
        self.name = name
        self.surname = surname
        self.birthdate = birthdate

        self.address = address
        self.telephone = telephone
        self.email = email

        # This isn't strictly necessary, but it clearly introduces these attributes
        self._age = None
        self._age_last_recalculated = None

        self._recalculate_age()

    def _recalculate_age(self):
        today = datetime.date.today()
        age = today.year - self.birthdate.year

        if today < datetime.date(today.year, self.birthdate.month, self.birthdate.day):
            age -= 1

        self._age = age
        self._age_last_recalculated = today

    def age(self):
        if (datetime.date.today() > self._age_last_recalculated):
            self._recalculate_age()

        return self._age
```

Exercise 3

1. Explain the differences between the attributes `name`, `surname` and `profession`, and what values they can have in different instances of this class:

```
class Smith:
    surname = "Smith"
    profession = "smith"

    def __init__(self, name, profession=None):
        self.name = name
        if profession is not None:
            self.profession = profession
```

```
class Smith:
    surname = "Smith"
    profession = "smith"

    def __init__(self, name, profession=None):
        self.name = name
        if profession is not None:
            self.profession = profession
```

Answer to exercise 3

1. `name` is always an instance attribute which is set in the constructor, and each class instance can have a different name value. `surname` is always a class attribute, and cannot be overridden in the constructor – every instance will have a surname value of `Smith`. `profession` is a class attribute, but it can optionally be overridden by an instance attribute in the constructor. Each instance will have a profession value of `smith` unless the optional `surname` parameter is passed into the constructor with a different value.

Exercise 4

1. Create a class called `Numbers`, which has a single class attribute called `MULTIPLIER`, and a constructor which takes the parameters `x` and `y` (these should all be numbers).
    1. Write a method called `add` which returns the sum of the attributes `x` and `y`.
    2. Write a class method called `multiply`, which takes a single number parameter `a` and returns the product of `a` and `MULTIPLIER`.
    3. Write a static method called `subtract`, which takes two number parameters, `b` and `c`, and returns `b - c`.
    4. Write a method called `value` which returns a tuple containing the values of `x` and `y`. Make this method into a property, and write a setter and a deleter for manipulating the values of `x` and `y`.

Answer to exercise 4

1. Here is an example program:

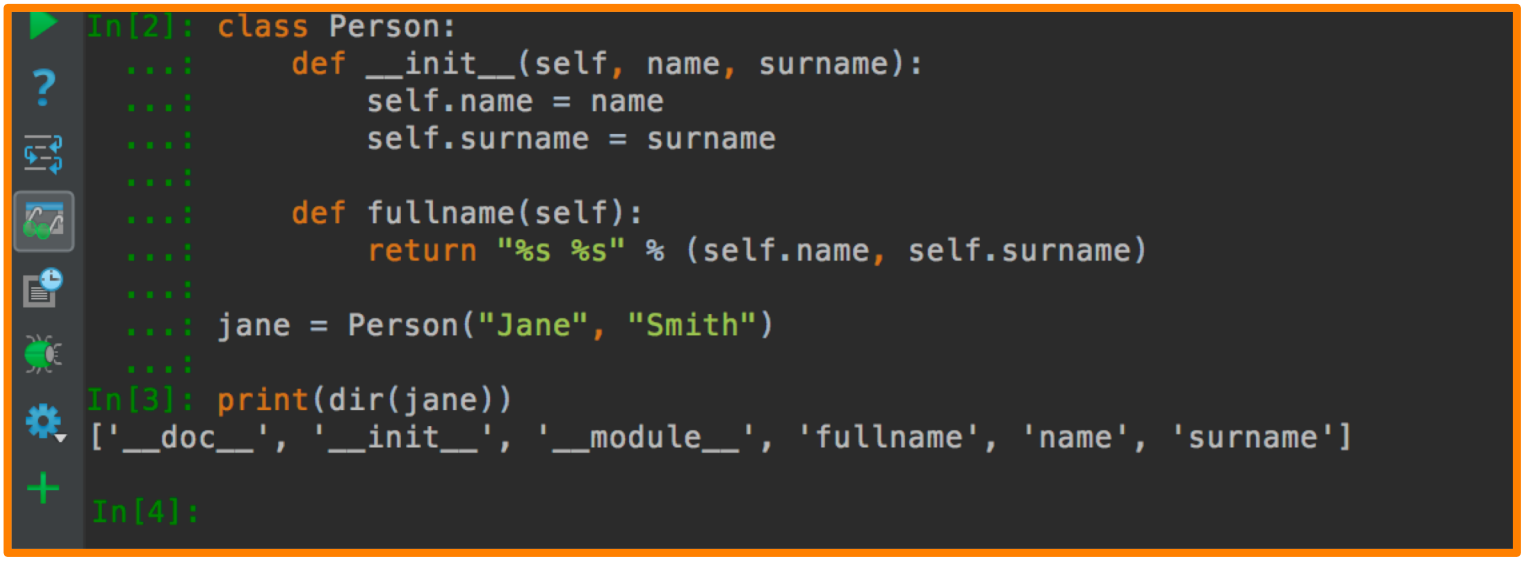
```
class Numbers:
    MULTIPLIER = 3.5

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def add(self):
        return self.x + self.y

    @classmethod
    def multiply(cls, a):
        return cls.MULTIPLIER * a

    @staticmethod
    def subtract(b, c):
        return b - c

    @property
    def value(self):
        return (self.x, self.y)

    @value.setter
    def value(self, xy_tuple):
        self.x, self.y = xy_tuple

    @value.deleter
    def value(self):
        del self.x
        del self.y
```

Create a class called `Numbers`, which has a single class attribute called `MULTIPLIER`, and a constructor which takes the parameters `x` and `y` (these should all be numbers).

1. Write a method called `add` which returns the sum of the attributes `x` and `y`.
2. Write a class method called `multiply`, which takes a single number parameter `a` and returns the product of `a` and `MULTIPLIER`.
3. Write a static method called `subtract`, which takes two number parameters, `b` and `c`, and returns `b - c`.
4. Write a method called `value` which returns a tuple containing the values of `x` and `y`. Make this method into a property, and write a setter and a deleter for manipulating the values of `x` and `y`.

Exercise 5

1. Create an instance of the `Person` class from example 2. Use the `dir` function on the instance. Then use the `dir` function on the class.
    1. What happens if you call the `__str__` method on the instance? Verify that you get the same result if you call the `str` function with the instance as a parameter.
    2. What is the type of the instance?
    3. What is the type of the class?
    4. Write a function which prints out the names and values of all the custom attributes of any object that is passed in as a parameter.



```
In[2]: class Person:
   ...:     def __init__(self, name, surname):
   ...:         self.name = name
   ...:         self.surname = surname
   ...:
   ...:     def fullname(self):
   ...:         return "%s %s" % (self.name, self.surname)
   ...:
   ...: jane = Person("Jane", "Smith")
   ...:
In[3]: print(dir(jane))
['__doc__', '__init__', '__module__', 'fullname', 'name', 'surname']

In[4]:
```

- Use function **dir** for inspecting objects: output list of the attributes and methods

Answer to exercise 5

1. 1. You should see something like `'<__main__.Person object at 0x7fcb233301d0>'`.
   2. `<class '__main__.Person'>` – `__main__` is Python's name for the program you are executing.
   3. `<class 'type'>` – any class has the type `type`.
   4. Here is an example program:

```
def print_object_attrs(any_object):
    for k, v in any_object.__dict__.items():
        print("%s: %s" % (k, v))
```

Exercise 6

1. Write a class for creating completely generic objects: its `__init__` function should accept any number of keyword parameters, and set them on the object as attributes with the keys as names. Write a `__str__` method for the class – the string it returns should include the name of the class and the values of all the object's custom instance attributes.

Answer to exercise 6

1. Here is an example program:

```
class AnyClass:
    def __init__(self, **kwargs):
        for k, v in kwargs.items():
            setattr(self, k, v)

    def __str__(self):
        attrs = ["%s=%s" % (k, v) for (k, v) in self.__dict__.items()]
        classname = self.__class__.__name__
        return "%s: %s" % (classname, " ".join(attrs))
```

REFERENCES

This lecture re-uses selected parts of the OPEN BOOK PROJECT

Learning with Python 3 (RLE)

http://openbookproject.net/thinkcs/python/english3e/index.html

available under GNU Free Documentation License Version 1.3)

- Version date: October 2012
- by Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers (based on 2nd edition by Jeffrey Elkner, Allen B. Downey, and Chris Meyers)
- Source repository is at https://code.launchpad.net/~thinkcspy-rle-team/thinkcspy/thinkcspy3-rle
- For offline use, download a zip file of the html or a pdf version from http://www.ict.ru.ac.za/Resources/cspw/thinkcspy3/

This lecture re-uses selected parts of the PYTHON TEXTBOOK

Object-Oriented Programming in Python

http://python-textbok.readthedocs.io/en/1.0/Classes.html#

(released under CC BY-SA 4.0 licence Revision 8e685e710775)