Python course documentation

Release 0.0

Python course teachers

November 21, 2016

Contents

1 Installation 1

2 Exercises day 1 3

3 Exercises day 2 9

CHAPTER 1

Installation

You need a recent version of Python 3 installed as well as a couple of packages that we will use in the course (see below). At the very least, version 3.3. Version 3.5 is recommended. You can check the version you have installed with:

```
$ python3 --version
```

1.1 Option 1: Anaconda (good for beginners)

Go to https://www.continuum.io/downloads and download the Python 3.5 version for your system.

1.1.1 Windows

Install Python 3 using all of the defaults for installation but make sure to check "Make Anaconda the default Python".

1.1.2 Mac OS X

Install Python 3 using the defaults for installation.

1.1.3 Linux

In your terminal run the installer that you just downloaded, e.g.:

```
$ bash Anaconda3-4.0.0-Linux-x86_64.sh
```

If you answer "yes" to the question "Do you wish the installer to prepend the Anaconda3 install location to PATH in your /home/user/.bashrc ?" then this will make the Anaconda distribution the default Python. You can always undo this by editing your .bashrc. Otherwise you go with the defaults.

1.2 Option 2: Virtual Environments (Linux or Mac OS X)

For this make sure that you have virtualenv installed. This also assumes that Python is installed on the system. See also http://docs.python-guide.org/en/latest/dev/virtualenvs/.

Go to https://github.com/uit-no/python-course and download the repository as zip file (click on "Download ZIP" on the right).

Then extract the zip file and inside python-course-master you can install all requirements using:

```
$ python3 -m venv venv
$ source venv/bin/activate
$ pip install -r requirements_course.txt
```

We recommend this approach to seasoned users. This because a possible pitfall with this approach is that some of the pip packages need C compilation, which depends on a number of system packages that pip can't install. If the "pip install" step fails with compilation errors, then you likely don't have some required C libraries installed on your system.

1.3 Option 3: Vagrant

If you are familiar with Vagrant, then a Linux virtual machine with everything you need is just a "git clone" and a "vagrant up" away.

First, download and unpack or clone the course repository from GitHub. Then, open a terminal, cd to the directory in which you have unpacked or cloned the repository, and run:

```
$ vagrant up
```

In a couple of minutes, you'll have a VM that has everything installed. Log in to the machine and look around like so:

```
$ vagrant ssh
```

Then, once you are logged in:

```
$ cd /vagrant
$ ls
```

One thing you will for sure like to do is to start jupyter, which we use in the course for some presentations and exercises:

```
$ cd /vagrant
$ ./run_jupyter.sh
```

Now just point your browser to http://localhost:8888.

CHAPTER 2

Exercises day 1

We will work on these exercises during the course. Do not worry about them before the course.

Below you find the source code for the exercises. Copy and paste them into a file called e.g. `basics.py`. It is OK to copy all or just one or two exercises to start with. We give you the tests and you need to code the functions to make the tests pass. It is OK to work in pairs and it is OK to use Google and Stack Overflow.

You can run the tests like this:

```
$ py.test -s -vv basics.py
```

You can also run a single test, e.g.:

```
$ py.test -s -vv -k test_reverse_list basics.py
```

2.1 Basics

Make the following tests green:

```
def reverse_list(l):
    """
    Reverses order of elements in list l.
    """
    return None


def test_reverse_list():
    assert reverse_list([1, 2, 3, 4, 5]) == [5, 4, 3, 2, 1]


# ----------------------------------------------------------------------------

def reverse_string(s):
    """
    Reverses order of characters in string s.
    """
    return None


def test_reverse_string():
    assert reverse_string("foobar") == "raboof"

```

```
24
25  # ------------------------------------------------------------------------
26
27  def get_word_lengths(s):
28      """
29      Returns a list of integers representing
30      the word lengths in string s.
31      """
32      return None
33
34
35  def test_get_word_lengths():
36      text = "Three tomatoes are walking down the street"
37      assert get_word_lengths(text) == [5, 8, 3, 7, 4, 3, 6]
38
39
40  # ------------------------------------------------------------------------
41
42  def find_longest_word(s):
43      """
44      Returns the longest word in string s.
45      In case there are several, return the first.
46      """
47      return None
48
49
50  def test_find_longest_word():
51      text = "Three tomatoes are walking down the street"
52      assert find_longest_word(text) == "tomatoes"
53      text = "foo foo1 foo2 foo3"
54      assert find_longest_word(text) == "foo1"
55
56
57  # ------------------------------------------------------------------------
58
59  def remove_substring(substring, string):
60      """
61      Returns string with all occurrences of substring removed.
62      """
63      return None
64
65
66  def test_remove_substring():
67      assert remove_substring("don't", "I don't like cake") == "I  like cake"
68      assert remove_substring("bada", "bada-bing-bada-bing") == "-bing--bing"
69
70
71  # ------------------------------------------------------------------------
72
73  def read_column(file_name, column_number):
74      """
75      Reads column column_number from file file_name
76      and returns the values as floats in a list.
77      """
78      return None
79
80
81  def test_read_column():
```

```
    import tempfile
    import os

    text = """1   0.1  0.001
2   0.2  0.002
3   0.3  0.003
4   0.4  0.004
5   0.5  0.005
6   0.6  0.006"""

    # we save this text to a temporary file
    file_name = tempfile.mkstemp()[1]
    with open(file_name, 'w') as f:
        f.write(text)

    # and now we pass the file name to the function which will read the column
    assert read_column(file_name, 2) == [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]

    # we remove the temporary file
    os.unlink(file_name)


# --------------------------------------------------------------------------------

def histogram(l):
    """
    Converts a list of tuples into a simple string histogram.
    """
    return None


def test_histogram():
    assert histogram([('a', 2), ('b', 5), ('c', 1)]) == """a: ##
b: #####
c: #"""


# --------------------------------------------------------------------------------

def character_statistics(text):
    """
    Reads text from file file_name, then
    lowercases the text, and then returns
    a list of tuples (character, occurence)
    sorted by occurence with most frequent
    appearing first.
    Use the isalpha() method to figure out
    whether the character is in the alphabet.
    """
    return None


def test_character_statistics():

    text = """
To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
```

```
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them? To die: to sleep;
No more; and by a sleep to say we end
The heart-ache and the thousand natural shocks
That flesh is heir to, 'tis a consummation
Devoutly to be wish'd. To die, to sleep;
To sleep: perchance to dream: ay, there's the rub;
For in that sleep of death what dreams may come
When we have shuffled off this mortal coil,
Must give us pause: there's the respect
That makes calamity of so long life;
For who would bear the whips and scorns of time,
The oppressor's wrong, the proud man's contumely,
The pangs of despised love, the law's delay,
The insolence of office and the spurns
That patient merit of the unworthy takes,
When he himself might his quietus make
With a bare bodkin? who would fardels bear,
To grunt and sweat under a weary life,
But that the dread of something after death,
The undiscover'd country from whose bourn
No traveller returns, puzzzles the will
And makes us rather bear those ills we have
Than fly to others that we know not of?
Thus conscience does make cowards of us all;
And thus the native hue of resolution
Is sicklied o'er with the pale cast of thought,
And enterprises of great pith and moment
With this regard their currents turn awry,
And lose the name of action.--Soft you now!
The fair Ophelia! Nymph, in thy orisons
Be all my sins remember'd."""

    assert character_statistics(text) == [('e', 146), ('t', 120), ('o', 99), ('s', 88), ('a', 87),
                                          ('h', 79), ('r', 71), ('n', 70), ('i', 57), ('l', 44),
                                          ('d', 43), ('u', 41), ('f', 36), ('m', 32), ('w', 29),
                                          ('p', 24), ('c', 23), ('y', 18), ('b', 17), ('g', 14),
                                          ('k', 10), ('v', 8), ('z', 3), ('q', 2)]


# ----------------------------------------------------------------------

def simple_zip(l1, l2):
    """
    Implement a simple zip function.
    Do not use the built-in zip().
    """
    return None


def test_simple_zip():

    assert simple_zip(['a', 'b', 'c'], [1, 2, 3]) == [('a', 1), ('b', 2), ('c', 3)]


# ----------------------------------------------------------------------

```

```python
def password_good(password):
    """
    Implement a function that tests if the input string is a "good" password.

    A "good" password should:

    - Be at least 8 characters long
    - Contain at least one upper-case letter [A-Z]
    - Contain at least one lower-case letter [a-z]
    - Contain at least one digit [0-9]
    - Contain at least one special character [#%&]

    The function should return True if the password is good, False otherwise.
    """
    return None


def test_password_good():

    good_passwords = ['Aa0#abcd', 'Zz9&0000', 'ABrt#&%aabb00']

    for pw in good_passwords:
        assert password_good(pw)

    bad_passwords = ['Aa0#', 'Zz9&000', 'ABrtaabb00', 'rt#&%aabb00',
                     'AB#&%001', 'ABrt#&%aabb']

    for pw in bad_passwords:
        assert not password_good(pw)


# ----------------------------------------------------------------------------

def generate_password():
    """
    Write a function that generates a random "good" password. The generated
    password should return True if checked by password_good.

    For easy to remember strong passwords see: https://xkcd.com/936/
    """
    return None


def test_generate_password():

    # generate list of 10 random passwords
    pw_list = []
    for _ in range(10):
        pw_list.append(generate_password())

    # passwords should be random, test for duplicates
    assert len(pw_list) == len(set(pw_list))

    # test all passwords in list
    for pw in pw_list:
        assert password_good(pw)
```

CHAPTER 3

Exercises day 2

3.1 Control Structures

Make the following tests green:

```
def simple_generator():
    """
    Also yield 'cow' and 'mouse'.
    """
    yield 'horse'


def test_simple_generator():
    assert list(simple_generator()) == ['horse', 'cow', 'mouse']


# --------------------------------------------------------------------


def simple_range(limit):
    """Yield numbers from 0 up to but not including limit.
    You can use a normal while loop."""
    pass


def test_simple_range():
    assert list(simple_range(0)) == []
    assert list(simple_range(3)) == [0, 1, 2]


# --------------------------------------------------------------------


def word_lengths(words):
    """
    Return a list of the length of each word.
    (Use len(word).)
    """
    pass


def test_word_lengths():
    words = ['lorem', 'ipsum', 'python', 'sit', 'amet']
```

```
    lengths = [5, 5, 6, 3, 4]
    assert word_lengths(words) == lengths


# ----------------------------------------------------------------------


def simple_filter(f, l):
    """
    Implement a simple filter function.
    Do not use the built-in filter().
    """
    return None


def test_simple_filter():

    def greater_than_ten(n):
        return n > 10

    assert simple_filter(greater_than_ten, [1, 20, 5, 13, 7, 25]) == [20, 13, 25]


# ----------------------------------------------------------------------


def simple_map(f, l):
    """
    Implement a simple map function.
    Do not use the built-in map().
    """
    return None


def test_simple_map():

    def square_me(x):
        return x*x

    assert simple_map(square_me, [1, 2, 3, 4, 5]) == [1, 4, 9, 16, 25]
```

3.2 Classes

Make the following tests green:

```
def make_dog_class():
    """
    Make a class `Dog` that satisfies the following conditions:
    * a dog has an attribute `happiness` which is initially set to 100, and
      which is decremented by 1 when time advances.
    * when a dog meets another dog, both dogs' happiness is reset to 100.
    * when a dog meets a fish, the dog feeds and the fish dies.
      * Note: "when a dog meets a fish" need not have the same effect as "when
        a fish meets a dog" - but extra kudos to you if you can make it so.
    """

```

```
    # The classes Pet and Fish are taken from the talk, with the addition of
    # '_advance_time_individual' in Pet.
    class Pet:
        population = set()

        def __init__(self, name):
            self.name = name
            self.hunger = 0
            self.age = 0
            self.pets_met = set()
            self.__class__.population.add(self)

        def die(self):
            print("{} dies :(".format(self.name))
            self.__class__.population.remove(self)

        def is_alive(self):
            return self in self.__class__.population

        @classmethod
        def advance_time(cls):
            for pet in cls.population:
                pet._advance_time_individual()

        def _advance_time_individual(self):
            # the leading _ in an attribute name is a convention that indicates
            # to users of a class that "this is an attribute that is used
            # internally, I probably shouldn't call it myself"
            self.age += 1
            self.hunger += 1

        def feed(self):
            self.hunger = 0

        def meet(self, other_pet):
            print("{} meets {}".format(self.name, other_pet.name))
            self.pets_met.add(other_pet)
            other_pet.pets_met.add(self)

        def print_stats(self):
            print("{o.name}, age {o.age}, hunger {o.hunger}, met {n} others".
                  format(o = self, n = len(self.pets_met)))


    class Fish(Pet):
        def __init__(self, name, size):
            self.size = size
            super().__init__(name)

        def meet(self, other_fish):
            super().meet(other_fish)
            if not isinstance(other_fish, Fish):
                return
            if self.size > other_fish.size:
                self.feed()
                other_fish.die()
            elif self.size < other_fish.size:
                other_fish.feed()
```

```
                self.die()


    Dog = None # make Dog class here

    return Pet, Fish, Dog


def test_dog_class():
    Pet, Fish, Dog = make_dog_class()
    assert type(Dog) == type

    attila = Dog("Attila")
    assert hasattr(attila, "happiness")
    assert attila.happiness == 100

    tamerlan = Dog("Tamerlan")

    Pet.advance_time()
    assert attila.happiness == tamerlan.happiness == 99

    attila.meet(tamerlan)
    assert attila.happiness == tamerlan.happiness == 100
    assert attila in tamerlan.pets_met
    assert tamerlan in attila.pets_met

    steve = Fish("Steve", 1)

    assert attila.hunger > 0
    attila.meet(steve)
    assert attila.hunger == 0
    assert not steve.is_alive()


################################################################################

def define_hungry():
    """
    Copy your classes from the first exercise, and make the following happen:
    * all pets have an `is_hungry()` method which returns True if the animal is
      hungry, and False if not. In general, pets are considered to be hungry
      when their hunger is > 50. Dogs, however, are considered to be hungry
      when their hunger is > 10.
    * there is a classmethod `Pet.get_hungry_pets()` which returns the set of
      pets that are currently hungry.
    """
    Pet = Fish = Dog = None

    return Pet, Fish, Dog


def test_define_hungry():
    Pet, Fish, Dog = define_hungry()

    p = Pet("p")
    f = Fish("f", 1)
    d = Dog("d")
    assert isinstance(Pet.get_hungry_pets(), set)
```

```
    for x, h in [(p, 51), (f, 51), (d, 11)]:
        assert len(Pet.get_hungry_pets()) == 0
        assert not x.is_hungry()
        x.hunger = h
        assert x.is_hungry()
        assert Pet.get_hungry_pets() == {x}
        x.hunger = 0
```

3.3 Containers

We revisit the "character statistics" exercise from yesterday. Implement a solution using *collections.Counter*:

```
def character_statistics(text):
    """
    Reads text from file file_name, then lowercases the text, and then returns
    a list of tuples (character, occurence) sorted by occurence with most
    frequent appearing first.

    You can use the isalpha() method to figure out whether the character is in
    the alphabet.

    Use collections.Counter for counting.
    """
    return None


def test_character_statistics():

    text = """
To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them? To die: to sleep;
No more; and by a sleep to say we end
The heart-ache and the thousand natural shocks
That flesh is heir to, 'tis a consummation
Devoutly to be wish'd. To die, to sleep;
To sleep: perchance to dream: ay, there's the rub;
For in that sleep of death what dreams may come
When we have shuffled off this mortal coil,
Must give us pause: there's the respect
That makes calamity of so long life;
For who would bear the whips and scorns of time,
The oppressor's wrong, the proud man's contumely,
The pangs of despised love, the law's delay,
The insolence of office and the spurns
That patient merit of the unworthy takes,
When he himself might his quietus make
With a bare bodkin? who would fardels bear,
To grunt and sweat under a weary life,
But that the dread of something after death,
The undiscover'd country from whose bourn
No traveller returns, puzzzles the will
And makes us rather bear those ills we have
```

```
Than fly to others that we know not of?
Thus conscience does make cowards of us all;
And thus the native hue of resolution
Is sicklied o'er with the pale cast of thought,
And enterprises of great pith and moment
With this regard their currents turn awry,
And lose the name of action.--Soft you now!
The fair Ophelia! Nymph, in thy orisons
Be all my sins remember'd."""

    assert character_statistics(text) == [('e', 146), ('t', 120), ('o', 99), ('s', 88), ('a', 87),
                                          ('h', 79), ('r', 71), ('n', 70), ('i', 57), ('l', 44),
                                          ('d', 43), ('u', 41), ('f', 36), ('m', 32), ('w', 29),
                                          ('p', 24), ('c', 23), ('y', 18), ('b', 17), ('g', 14),
                                          ('k', 10), ('v', 8), ('z', 3), ('q', 2)]
```