## You should use Python 2.x in the course. It is always good to know differences between versions 2.x and 3.x.

## Python has its own official code-style called pep8. You should follow pep8 guidelines through the course. It is recommended to install the pep8 package from pip and run it over you code to check its codestyle. <https://www.python.org/dev/peps/pep-0008/>

## REQUIRED

**Exercise 1-1.**Set up Python interpreter and pip

Useful videos:

* How to install python on Windows 7 [http://www.youtube.com/watch?v=L5t5U0XnSew](http://www.youtube.com/watch?v=L5t5U0XnSew" \t "[object Object])
* How to install pip on Windows [http://www.youtube.com/watch?v=zPMr0lEMqpo](http://www.youtube.com/watch?v=zPMr0lEMqpo" \t "[object Object])

Don’t forget to check your installation by executing:

pip install pep8

**Exercise 1-2.** Setup virtualenv ([https://pypi.python.org/pypi/virtualenv](https://pypi.python.org/pypi/virtualenv" \t "[object Object])) and virtualenvwrapper ([http://virtualenvwrapper.readthedocs.org/en/latest/index.html](http://virtualenvwrapper.readthedocs.org/en/latest/index.html" \t "[object Object]))

Note: If you are using Windows please consider this project

[https://github.com/davidmarble/virtualenvwrapper-win/](https://github.com/davidmarble/virtualenvwrapper-win/" \t "[object Object])

pip install virtualenv

pip install virtualenvwrapper

You can read more about this here: [http://docs.python-guide.org/en/latest/dev/virtualenvs/](http://docs.python-guide.org/en/latest/dev/virtualenvs/" \t "[object Object])

**Exercise 1-3.** Create a new Python virtual environment and use it during the course

**Exercise 1-4.** Read the **pep8** documentation. Install the pep8 package on your virtual environment

**Exercise 1-5.** Start the Python interpreter, write a simple **Hello World** program

ADDITIONAL

**Exercise 1a-1.** Write a simple game **guess the number**. The program will randomly choose a number within the specified range and will wait for the user to input a number. If the number is smaller or greater than that chosen by the program, the system tells that to the user. If the guessed number is correct, then the game is over and the program displays the number of iterations taken.

**Exercise 1a-2.** Write a program that generates and draws a simple labyrinth in ASCII mode. The labyrinth should consist of 6 different tiles (3x3 symbols) represented as string as follows

NO = '###

      ###

      ###' # No path

LR = '###

      ###' # Path from left to right

LT = '# #

        #

      ###' # Path from left to top

BR = '###

      #

      # #' # Path from bottom to right

LB = '###

        #

      # #' # Path from left to bottom

TR = '# #

      #

      ###' # Path from top to right

Write a function that draws any given combination of tiles as 2-dimensional array. For example

render([

    [NO, BR, LR, LB, NO],

    [LR, LT, NO, TR, LR]

])

Should be rendered as

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*       \*\*\*\*

\*\*\*\* \*\*\*\*\* \*\*\*\*

\*\*\*\* \*\*\*\*\* \*\*\*\*

     \*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Exercise 1a-3.** Run the pep8 checker to be sure that your programs satisfy the style guide.

## NEED TO KNOW

1. What is Python and what versions we should use and why?

2. How to install Python on your system.

3. What is a virtual environment and why do we need it?

4. What is pep8, and why it is recommended to follow it?

5. How to write a simple program and run it from the command line.

## REQUIRED

**Exercise 2-1.** Write a recursive function that computes factorial

**Exercise 2-2.** Write a function computing the average from arbitrary numbers of an argument. For example: avg(4, 5, 2, 3)

**Exercise 2-3.** Write a function adding two numbers in the following manner: add(1)(2) == 3

**Exercise 2-4.** Write a decorator that computes the execution time of the decorated function

**Exercise 2-5.** Write a function that computes a large factorial using multiple threads

## ADDITIONAL

**Exercise 2a-1.** Write a bubble-sort function

**Exercise 2a-1.** Write a decorator class **profile** that will compute the function’s execution time, number of calls, and call the hierarchy of functions decorated with **profile**

**Exercise 2a-3.** Make the decorator profile thread-safe

**Exercise 2a-4.** Measure the execution time of the multi-threaded program from exercise 2-5 with**profile** decorator. Compare this execution time with the execution time of a single-threaded large factorial function. Explain the difference.

## NEED TO KNOW

1. What is kwargs and why do we need them?

2. What is decorator?

3. How to write your own decorator as a function and a class.

4. How to start a new thread and attach it to the main thread.

5. What is GIL (Global Interpreter Lock)?

## REQUIRED

**Exercise 3-1.** Write a program that computes the number of occurrences of each symbol in a string. The string is provided by the program.

**Exercise 3-2.** Write a program that produces Fibonacci numbers and is based on generators. The user should input a number that limits the number of results.

## ADDITIONAL

**Exercise 3a-1.** Write a program that computes the result of a simple arithmetic expression, containing addition, subtraction, division and multiplication. Usage of the eval function is forbidden. Add flows for exceptions of user input and arithmetical errors.

**Exercise 3a-2.** Write a program that generates a maze using the tiles from **Exercise 1a-2**. Please note that all paths should be correctly connected to each other.

## NEED TO KNOW

1. The differences between list, tuple, and set.

2. What is a dictionary?

3. Which types are mutable, and which are not?

4. What does “tuple unpacking” mean?

5. What is list comprehension and dictionary comprehension?

6. What’s the result of the expression **[1, 2, 3][0 or 2]** and why?

7. What does a generator return? What are the pros and cons of using generators?

8. How are exceptions handled? How to raise an exception?

## REQUIRED

**Exercise 4-1.** Modify Exercise 3-1 to extract all program logic in a separate module.

**Exercise 4-2.** Modify Exercise 4-1 to define the string from command line.

**Exercise 4-3.** Write a program that computes the statistics of a file’s content: most frequent word, number of words, and number of characters.

**Exercise 4-4.** Write a couple of unit tests for **Exercise 4-1**.

## ADDITIONAL

**Exercise 4a-1.** Write unit test for **Exercise 4-1**. Mocks for user input are required.

**Exercise 4a-2.** Write unit test for **Exercise 4-2.** Imagine that the file system is unreachable. Don’t forget about negative test cases.

## NEED TO KNOW

1. How to open and how to close a file.

2. What does the keyword **with** mean?

3. What is a module?

4. How to write your own module?

5. How to write and run unit tests?

6. How to mock user input and file system calls?

## REQUIRED

**Exercise 5-1.** Write a class representing two-dimensional vector (with x and y coordinates), which supports the following operations:

1. Addition of two vectors: **v1 + v2**

2. Subtraction of two vectors: **v1 – v2**

3. Dot product of two vectors:

    a. Using operator **v1 \* v2**

    b. Using method **v1.dot(v2)**

    c. Using class method **Vector2d.dot(v1, v2)**

4. Vector length property:

    a. **v.length**

5. String representation: **str(v)**

**Exercise 5-2.** Write classes of the following heroes: Warrior, Mage, and Archer. All classes have health points (HP) and magic energy (mana). There are three different types of magic: Fire, Ice and Arcane; Fire magic has the spell **fireball**, Ice magic has the spell freeze and Arcane magic has the spell **black hole**. The Warrior is furious and can use Fire magic as well as his hammer; The Archer is calm and can use Ice magic in addition to his arrows; The Mage can use all types of magic. Each magic spell costs an arbitrary amount of magic energy and can be cast only if there is enough mana. Use magic mixins to reduce repetitive code.

## ADDITIONAL

**Exercise 5a-1.** Create an arena for the mighty heroes from **Exercise 5-2.** The game starts with hero class selection. Then the game will generate a random monster. The monster has health points and can attack the hero, subtracting some amount of HP. The monster can have (or not) an affix that reduces damage taken from fire, ice or arcane magic. The main point of the game is to stay alive (hero HP > 0) and defeat the enemy. An example combat log is shown here:

> Please select a hero:

> warrior

> The game is begun

> Warrior. HP: 200, Mana: 100

> You see the monster. He’s horrifying and dangerous.

> Warrior. HP: 200, Mana: 100

> Monster. HP: 100

> Please select an action:

> hammer\_attack

> Hit! Monster. HP: 70

> Monster strikes you. Warrior. HP: 210

> Please select an action

> hero info

> Warrior. HP: 210, Mana: 90

> Please select an action

> fireball

> Hit! Monster defeated!

> You win!

## NEED TO KNOW

1. Inheritance in Python.

2. What does **mixin** mean?

3. Difference between new and old style classes.

4. How to define a class?

5. What is the difference between class attribute and instance attribute?

6. What does **magic methods** mean?

7. How to check if an object is an instance of a particular type?

8. What does **duck-typing** mean?

9. What is public, private and protected attributes in Python?

10. What does **name mangling** mean?