Installation, python self-test and the fruit-project (Complete before lecture 1)

Tue Herlau
tuhe@dtu.dk

31 January, 2025

Contents

1 Installing the toolbox

The first step is to install the course toolbox. You can find the instructions here:

- `https://www2.compute.dtu.dk/courses/02465/exercises/ex00.html`

1.1 Exercise Format

The programming exercises takes the form of writing missing code. An example is given in `irlc/ex00/fruit_homework.py`:

```python
def add(a, b):
    """  This function should return the sum of a and b.
    I.e. print(add(2,3)) should print '5'. """
    # TODO: 1 lines missing.
    raise NotImplementedError("Implement function body")
```

The code raises `Exception` to make it clear code is missing[1]. To solve the exercise, you should remove the exception and insert a solution which makes the code work – in this case `return a + b` will do the trick.

[1]**Note:** Some have pointed out that the `## n lines missing`-part can be distracting since their solution might be longer. The goal is not to solve the problem in n-lines – I just included the number because I thought it could be helpful in some situations.

The scripts you edit will typically also call the code you write for easy verification. For instance, near the bottom the script the `add`-function is tested as follows:

```python
# fruit_homework.py
print("add(2,5) function should return 7, and it returned", add(2, 5))
```

As a rule, the exercises will require that you read the surrounding code, comments, etc., just like any realistic workflow[2]. The course website contains documentation about the functions we will use for each week:
https://www2.compute.dtu.dk/courses/02465/exercises/index.html
When you are happy with your solution run the file to verify your solution:

```
irlc/ex00/fruit_homework.py
```

If you implemented the `add` function correctly it should produce the output:

```
add(2,5) function should return 7, and it returned 7
```

If you are not sure how to run a file in VS code please see the course website:
https://www2.compute.dtu.dk/courses/02465/exercises/ex00.html

2 Example exercise (`fruit_homework.py`)

This is an example of a simple homework/project problem. It will showcase some of the features of python we will use, and allow you to get an idea if there are things you need to brush up on. If you find the exercises generally hard to get started with look at [Her25, chapter 1]; if you find the exercises that involve data structures such as lists or dictionaries hard look at [Her25, chapter 2], and if you are not familiar with classes or packages look at [Her25, chapter 3].

The name of the file you should edit is given in the section title, and it can be found in the directory labeled by the the current week. In this case you should edit `02465students/irlc/ex00/fruit_homework.py`.

Adding numbers

Problem 1 Warmup; a function to add numbers

Complete the `add`-function in `fruit_homework.py`. Run the script and verify it gives you the right output.

[2]Tip: In VS Code, use `ctrl+<click>` on variables and names to go to code definition

Info: Once you are done, you should get the following output:

```
add(2,5) function should return 7, and it returned 7
```

Misterfy

Problem 2 Lists and strings

This function will take a list of animal names as input, and then return a new of animal names where the names have been given the prefix `"mr"`. An example of how the function is used:

```python
# fruit_homework.py
animals = ["cat", "giraffe", "wolf"]
print("The nice animals are", misterfy(animals))
```

Complete the `misterfy`-function. Verify you get the right output.

Info: Once you are done, you should get the following output:

```
The nice animals are ['mr cat', 'mr giraffe', 'mr wolf']
```

Mean of a die

A convenient way to represent a die is using a dictionary, where the key are the sides $x_i \in \{1, \dots, 6\}$, and the values is the probability $p_i = p(x_i)$ (see [Her25, chapter 2]). Your job is to build a function which takes such a dictionary as input, and computes the mean value of the distribution the dictionary represent, i.e.

$$\text{mean value} = \sum_{i=1}^{6} x_i p(x_i)$$

The way we use the function in python is as follows:

```python
# fruit_homework.py
"""
This problem represents the probabilities of a loaded die as a dictionary such
↪ that
```

```python
    > p(roll=3) = p_dict[3] = 0.15.
    """
    p_die = {1: 0.20,
             2: 0.10,
             3: 0.15,
             4: 0.05,
             5: 0.10,
             6: 0.40}
    print("Mean roll of die, sum_{i=1}^6 i * p(i) =", mean_value(p_die))
```

Problem 3 Mean of a die

Complete the `mean_value`-function. Verify you get the right output.

Info: Once you are done, you should get the following output:

```
Mean roll of die, sum_{i=1}^6 i * p(i) = 3.95
```

If you are stuck, insert a breakpoint in the `mean_value` function (see video instructions for how) and work out a solution in the command line. You will need a `for` loop over the dictionary.

A Fruitshop-class

In this exercise you will build a simple class which represents a fruit shop. The fruit shop will have a name and a dictionary which represent the cost of each fruit. You job is to implement the cost-function, which, given the name a of a fruit as a string, does a look-up in the fruit-price dictionary and return the cost of the given fruit. You don't need to do any sort of error handling. The following code should work:

```python
# fruit_homework.py
price1 = {"apple": 4, "pear": 8, 'orange': 10}
shop1 = BasicFruitShop("Alis Funky Fruits", price1)

price2 = {'banana': 9, "apple": 5, "pear": 7, 'orange': 11}
shop2 = BasicFruitShop("Hansen Fruit Emporium", price2)

fruit = "apple"
print("The cost of", fruit, "in", shop1.name, "is", shop1.cost(fruit))
print("The cost of", fruit, "in", shop2.name, "is", shop2.cost(fruit))
```

Problem 4 Basic fruit shop

Complete the `cost`-function in the `BasicShop`.

Info: Once you are done, you should get the following output:

```
The cost of apple in Alis Funky Fruits is 4
The cost of apple in Hansen Fruit Emporium is 5
```

If you are stuck you should read [Her25, chapter 3] which discuss classes. I will review classes in the second half of the first lecture. Nearly all problems in this course will require modifying functions in classes.

3 Unitgrading your first homework

This course will use an automatic framework, unitgrade, for handing in project solutions. Unitgrade is build on top of pythons unittest framework, which is the industry-standard way of testing and verifying python code. This means it will work well with the VS Code debugger which I recommend you use to your advantage!

It is up to you if you want to use unitgrade to debug the code, or only want to use unitgrade to create a handin. To run the tests in VS Code, you can click on the flask-icon in the left bar and select the test you wish to run as shown in this video:
`https://www2.compute.dtu.dk/courses/02465/exercises/ex00.html`

3.1 Creating your hand-in `.token`-file

To create your hand-in, you have to use the `fruit_project_grade.py`-file. This file will run the same tests as mentioned in the previous section. You can run this file by right-clicing on it in VS code and selecting `Run`, or by using the command

```
python -m irlc.project0.fruit_project_grade
```

Either way, you will notice the script produces a file called

```
02465students/irlc/project0/FruitReport_handin_40_of_40.token
```

This file contains the outcome of your tests as well as a copy of the code you have written for verification. You should hand in this file on DTU Learn *without* modifications. The file name contains the number of points you obtain from the local tests. After handin I will use the `token`-file to run additional tests on your code, however, my experience is that if your local tests work the code is correct – and thus you can generally trust the score in the filename!

References

[Her25] Tue Herlau. Sequential decision making. (Freely available online), 2025.