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

Tue Herlau
tuhe@dtu.dk

28 January, 2022

Contents

1 Installation guide

- **Simplest approach: Just follow the two installation videos on DTU Learn and skip this section**

1.1 Installation of toolbox

- Run `python --version` from the command line to check your python version. You need version 3.8 or later for this course[1].

[1]The pickle module and other stuff was updated between 3.6 and 3.8.

  - If you don't have python version 3.8 or later then (i) uninstall your existing version of python to avoid conflicts (ii) install python from `https://www.python.org/`

- Install Pycharm community edition from `https://www.jetbrains.com/pycharm/`
- In the console, go to a base directory such as your `Documents` folder
- Obtain toolbox code from DTU gitlab using the command

```
git clone https://gitlab.gbar.dtu.dk/02465material/02465students.git
```

- Change folder `cd 02465students`
- Install all python package dependencies using

```
pip install -r requirements.txt
```

  - Many of the packages are only used later in the course, so if this command does not completely work it is okay to continue with the guide and fix issues as they crop up
  - Some features of e.g. openai gym use external dependencies; these may or may not work out of the box depending on your platform. I maintain instructions in the `02465students` repository at `https://gitlab.gbar.dtu.dk/02465material/02465students`

- Start pycharm. In the create new project dialog, (i) select system interpreter (ii) select the `02465students` folder as location for the project so that the `irlc` package is recognized. Please see the video if this is not entirely clear.

1.2 Installation of Unitgrade

Unitgrade is the testing and evaluation framework used in the course. It is required to hand in the projects. If you followed the guide in section 1.1, unitgrade was automatically installed when you ran `pip install -r requirements.txt`. You can also install it manually

```
pip install unitgrade
```

If you at some point need to upgrade to the latest version you can run

```
pip install unitgrade --upgrade --no-cache
```

More information, including privacy information and source code, can be found at `https://lab.compute.dtu.dk/tuhe/unitgrade`

1.3 Exercise Format

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

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

```
# irlc/ex00/fruit_homework.py
    print("add(2,5) function should return 7, and it returned", add(2, 5))
```

The bottom line is that whenever you see the line `## TODO: n lines missing.` you need to write some code and remove the exception so the function can run.

The code will always raise an error when code is missing so it is clear what code needs to be changed, and what does not need to be changed. This is in accordance with the age-old machine learning maxim: "*If it compiles it's good, if it runs it's perfect!*". You are obviously allowed to change the surrounding code, add new functions, and so on but it should not be necessary.

The exercises will require reading of the surrounding code, comments, etc.[2]. The exercises are therefore in equal parts about understanding what functionality is already implemented, what is missing, and finally solving the problem. I encourage you to use the debugger when you try to understand the data structure, just like you would in any realistic workflow.

When you are happy with your solution, run the file[3]:

```
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
```

2 Self-test/homework example (`fruit_homework.py`)

This is an example of a simple homework problem. The exercises 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

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

[3] See video link above for how to do this in Pycharm

at [Her21, chapter 1]; if you find the exercises that involve data structures such as lists or dictionaries hard look at [Her21, chapter 2], and if you are not familiar with classes or packages look at [Her21, 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`.

2.1 Datatypes, lists and dictionarires

Problem 1 Warmup; a function to add numbers

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

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

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

2.1.1 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:

```
# 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']
```

2.1.2 Mean of a die

A convenient way to represent a die is using a dictionary, where the key are the sides $x_i \in \{1, \ldots, 6\}$, and the values is the probability $p_i = p(x_i)$ (see [Her21, 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:

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

2.1.3 Fruits ordered

Let's suppose we want to build a fruit-shop. A neat way to represent an order to the fruit-shop is as a dictionary, where the keys are the names of the fruit, and the values are the quantity to buy. As an example:

```
# fruit_homework.py
    order = {'apples': 1.0,
             'oranges': 3.0}
    print("The different fruits in the fruit-order is", fruits_ordered(order))
```

In this problem, we will simply make a function that extracts the fruits the customer is interested in.

Problem 4 Fruits ordered

Complete the `fruits_ordered`-function. It takes an order dictionary of the above form as input, and return a list of the names of the fruit the customer has ordered.

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

```
The different fruits in the fruit-order is ['apples', 'oranges']
```

2.2 Classes

Next, lets build a small class that 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:

```
# 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 5 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, look at the comments. If the class-syntax itself is mysterious I recommend reading [Her21, chapter 3]; nearly all problems in this course will require modifying functions in classes.

2.2.1 Class inheritance

The main obstacle in understanding class inheritance is understanding which situations it is useful in in the first place. I would therefore recommend reading [Her21, chapter 3] for an overview if you are new to inheritance.

In this problem, we will use class inheritance to build an online fruit-shop class that inherits from the basic fruit shop, but adds a method which can compute the total cost of an order (the dictionary contains fruits and quantity as before) using the price dictionary. The online fruit shop can be used as follows:

```
# fruit_homework.py
    price_of_fruits = {'apples': 2, 'oranges': 1, 'pears': 1.5, 'mellon': 10}
    shopA = OnlineFruitShop('shopA', price_of_fruits)
    print("The price of the given order in shopA is", shopA.price_of_order(order))
```

Problem 6 The online fruit shop

Complete the `price_of_order` function. You need to use the `prices` variable in the `BasicFruitShop`. If this kind of code is new to you, you should read the notes or follow an online tutorial on classes.

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

```
The price of the given order in shopA is 5.0
```

Don't try to guess your way through this example. It is important to have a reasonable understanding of class inheritance in order to be able to read the code in the course – and you can get such an understanding in very little time!

2.2.2 Using classes

The last example will consider a combination of classes and datastructures. You have to complete the function `shop_smart`. It is given a list of `OnlineFruitShop` instances and

an order, and return the shop from the list which has the lowest cost of the order. An example:

```
# fruit_homework.py
    shopB = OnlineFruitShop('shopB', {'apples': 1.0, 'oranges': 5.0})

    shops = [shopA, shopB]
    print("For the order", order, " the best shop is", shop_smart(order, shops).name)
    order = {'apples': 3.0}  # test with a new order.
    print("For the order", order, " the best shop is", shop_smart(order, shops).name)
```

Problem 7 Using a class

Complete the `shop_smarter` method. It is a good idea to use the `price_of_order` method, since it will tell you the cost of an order in a given shop.

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

```
For the order {'apples': 1.0, 'oranges': 3.0}  the best shop is shopA
For the order {'apples': 3.0}  the best shop is shopB
```

When you solve the problem, you may run into a problem of finding the index corresponding to the lowest value in a list. Stackexchange or the lecture notes can help you with that problem.

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 be more optimized for debugging and helping you solve problems than codejudge.

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. This section will show both approaches using the fruit-homework as an example.

3.1 Running tests in pycharm

This is the recommended way to run the script. In pycharm, simply right-click on the file `irlc/project0/fruit_project.py` and use the `"Run"` or `"Debug"` option. This will run the tests like a normal test-script. You will not be able to see your total number of points, but it will be easier to navigate the expected output and so on. You can look at pycharms documentation for information about how to run individual tests and other common features.

3.2 Running tests in command line

You can also run the script from the command line. This will show the number of points as well as errors, but is in my opinion a bit less useful. To run it in the command line, go to the `02465students` directory. Then run the command

```
python -m irlc.project0.fruit_project
```

I.e., you simply run it like you would run any other file in a package. This will run all tests and show the expected number of points. You can use the `--help` command line argument to see how you can run individual tests.

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

To create your hand-in, you have to use the `fruit_project_grade.py`-file. This file contains the same tests, but without any risk of accidental changes. You can run this file by right-clicking it in pycharm 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_70_of_70.token
```

The numbers are the points you obtained. You should hand in this file without modifications. The file contains a copy of the source code you have written. Note the points are only indicative of how well you did.

References

[Her21] Tue Herlau. Sequential decision making. (See **02465_Notes.pdf**), 2021.