CS5710 :- Lab 3

Stephen Piddock

January 30, 2024

1 Learning outcomes

This lab session will introduce you to

- Python tuples, lists and dictionaries,
- object oriented programming,
- creating and customising your own classes.

2 Notes

- Complete the various sections described in this document. and have a look at the extensions.
- **Remember** the TAs and lecturers are here to help - if you are stuck then don't hesitate to ask questions.
- Read carefully the instructions, and ask for help if you feel lost.
- It is recommended that you store your programs in a folder hierarchy comprising of a single high-level folder, e.g., `CS5710Labs`, and one sub-folder for each lab session, e.g., `lab2`, `lab3`, etc.
- Unless stated otherwise, store your programs in the files called `ex<exercise_number>.py`. For example, the program for Exercise 1 should be stored in file `ex1.py`.
- If you work on a lab workstation, use the `Y:` folder to store your work as everything stored in the local folders is being erased on a daily basis.
- If your program gets stuck in an infinite loop, you can interrupt it with `CTRL-C` or click on the red box above the console window in Spyder.

3 Tuples and lists

1. Write a function `min_max()` that takes a tuple of numbers as input and returns a tuple consisting of the smallest and the largest numbers in the input tuple. Do this by iterating over the elements of the tuple using the `for`...`in` loop. Do not use the built-in `min()` and `max()` functions. For example, `min_max((29, 45, 6, 19))` should return `(6, 45)`.

2. Write a function `reverse()` that takes one argument that can be tuple as input, and returns its reversal. For example `reverse((3, 'hi', (5, 7), 8))` returns `(8, (5, 7), 'hi', 3)`.

3. Write a function `sum_mult()` that takes a tuple of numbers (either integers or floats) as input and returns a tuple consisting of their sum and product if the tuple is non-empty, and `None`, otherwise. For example, `sum_mult((1, 2, 3, 6))`, `sum_mult((10,))` and `sum_mult(())` should return `(12, 36)`, `(10, 10)`, and `None` respectively.

4. Write a function `filter_odd()` that takes a tuple as input, and returns a new tuple consisting of every other element of the input tuple starting from the first one. For example, `filter_odd(('I', 'love', 'Python', 'and', 'Java'))` should return `(('I', 'Python', 'Java'))`.

5. Write a function `apply_to_all()` that takes a tuple t and a function f as inputs, and returns a tuple, which is the result of applying the f to all elements of t. For example, `apply_to_all((1, -5, -6, 3), abs)` should return `(1, 5, 6, 3)`. Test `apply_to_all()` with both built-in/library functions (e.g., `abs()`, `int()`, `math.sqrt()`), and your own functions.

6. Repeat the above exercises but instead of using tuples, use lists. Also look up python documentation to see if you can find functions that do the above for you (there are some that are there and some that aren't). Try to use the formal Python documentation (https://docs.python.org/3.9/) rather than StackOverflow!

4 Dictionaries

4.1 Write a function `make_histogram()` that takes a string of letters as input argument and returns a dictionary mapping each letter to its frequency. For example, `make_histogram('parrot')` should return

`\{'p': 1, 'a': 1, 'r': 2, 'o': 1, 't': 1\}`

(not necessarily in this order). Test your implementation on several inputs, and make sure it works correctly.

4.2 Write a function `has_duplicates()` that takes a list of words as input and returns `True` if there is any element that appears more than once. Your implementation should traverse the input list and use a dictionary to store the words that have already been encountered until either a duplicate is found, or the end of the of the list is reached. Note that the words should be stored as *keys*. It does not matter what the values are (e.g., you can use `None` as the value for all keys). For example,
`has_duplicates(['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy',`
should return `True`, and `has_duplicates(['magic', 'tree', 'house'])` should return `False`.

4.3 Write a function `get_duplicates()` that takes a list of words as input and returns a dictionary mapping all duplicate words to their frequencies. For example,
`get_duplicates(['it', 'is', 'the', 'right', 'right', 'is', 'not', 'it', 'right']`
should return `{'it':2, 'is':2, 'right':3\}`.

4.4 Write a function `inverted_dict()` that takes a dictionary as input and returns a new dictionary, which is the inverted copy of its argument. Since there might be several keys mapped to the same value in the input dictionary, each value in the inverted copy should be a list of keys in the input dictionary. For example,
`inverted_dict(\{'p': 1, 'a': 1, 'r': 2, 'o': 1, 't': 1\})` should return
`\{1:['p', 'a', 'o', 't'], 2:['r']\}`. Start by implementing a simpler version assuming that each value in the input dictionary is unique, and once it works, extend it into a general solution.

4.5 Read the documentation of the dictionary method `setdefault()` (`https://docs.python.org/3/library/stdtypes.html`), and use it to write a more concise version of `inverted_dict()` from Exercise 4.4.

4.6 Re-implement Exercise 4.4, but instead of returning a new copy, mutate the input dictionary in place, and do not return any value. Name your function `invert_dict()`.

5 Object-Oriented Programming

Download the template code `lab3.py` from Moodle.

1. The `Coordinate` class implements an abstraction of a two-dimensional coordinate. The template code includes a *getter* and a *setter* methods – `getX()` and `setX()` – to respectively get and set the value of the `x` attribute. Add a getter and a setter methods (`getY()` and `setY()`) for the `y` attribute.

2. In the `if __name__ == '__main__:'` section of the template, you will find some code that instantiates `Coordinate`, invokes some methods on its instance and outputs its string representation. Create a few more `Coordinate` objects with various values of the `x` and `y` attributes, and use the setter and getter methods to modify and output their values.

3. Define a class `Circle` having two data attributes `centre` and `radius`, and add the following methods to its code:

   (a) A constructor `__init__()` that takes an instance of `Coordinate` and a number representing the circle radius as arguments, and initializes the attributes with the values passed. Note that since the `Coordinate` object is mutable, it is a good idea to have the `centre` attribute to store its copy rather than the alias passed to the constructor. The simplest way to create a copy of an object is to import the Python's `copy` library, and then use its `copy()` method with the object as argument as is shown in the code below:

```
import copy

c = Coordinate(3, 4)
c_copy = copy.copy(c)
c.setX(10) # modifies the object aliased by c,
                # but not by c_copy
print(c) # prints <10,4>
print(c_copy) # prints <3,4>
```

(b) A method `__str__()` that converts a `Circle` object to the following string representation:
`'Circle(<centre>, <radius>)'`.

(c) Getter methods `getCentre()` and `getRadius()`. Note that `getCentre()` should return a copy of the `centre` attribute to prevent it from being modified outside the class scope.

(d) Setter methods `setCentre()` and `setRadius()`. As per above, `setCentre()` should store a copy of the passed `Coordinate` object rather than its alias.

(e) A method `get_area()` that returns the area of the circle computed as πr^2 where r is the circle's radius. An approximation of π is available as `math.pi`. Do not forget to import the `math` module if you want to use it.

(f) A method `is_point_in()` that takes a point as an instance of `Coordinate` and returns `True` if the point lies inside the circle, and `False` otherwise.
Hint: use the `distance()` method of the `Coordinate` class.

(g) Add some code (after `if __name__ == '__main__:'`) that creates instances of relevant objects and tests the methods above.