Unit 10 Exercises – Python data structures, performing calculations and plotting

Aims and objectives

- Familiarise yourself with Jupyter if you haven't used it in the last semester
- Refresh Python fundamentals, such as lists, tuples, dictionaries
- Completing a small task of basic calculations and printing a table of information generated
- Create a basic plot with Matplotlib

This week's exercises

These exercises involve refreshing Python data structures such as lists, dictionaries and doing some basic calculations and plotting with Matplotlib.

Task 0 – Jupyter (for those who have not used it before)

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

To use Jupyter, please follow this guide. To run Jupyter on your own notebook, follow this video.

Task 1 – Re-read Chapter 9 and 11 and 20 (tuples, lists, dictionaries)

Read Chapter 9 and Chapter 11 and Chapter 20. Work through the examples with the Python interpreter, and try some similar examples of your own.

Task 2 – List manipulations

Write the following functions in Python. If you can, try to present multiple implementation for the same problem.

1. `flatten(a)`: Write a function that flattens a list (optional: show multiple implementations) – Hint: this is a recursive function!

   ```
   Input: [[[1, 2, 3], [4, 5, 6], [7], [8, 9], 10]]
   Output: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
   ```

2. `identical(a, b)`: Write a python program to check whether two lists are circularly identical (a circular list is a list where we consider the first element as next of last element, forming a "circle").

   ```
   identical([10, 10, 0, 10], [10, 10, 10, 0]) -> True
   identical([10, 10, 5, 10], [10, 10, 10, 0]) -> False
   ```

3. `chunks(list, size)`: Split list to chunks with this function taking the list and the size for the chunks required.

```
Input: [1,2,3,5,1,3,4,12,3,5,3,2,3,4,5], 6
Output: [1, 2, 3, 5, 1, 3], [4, 12, 3, 5, 3, 2], [3, 4, 5]
```

Task 3 – Tuple manipulations

Write the following functions in Python. If you can, try to present multiple implementations for the same problem.

1. `sortbyfloat(a)`: Write a Python function to sort a list by the tuples' second, float element. Hint: try using `lambda` functions for key to `sorted()`.

```
Input: [('item1', 12.20), ('item2', 15.10), ('item3',
24.5)]
Output: [('item3', 24.5), ('item2', 15.10), ('item1',
12.20)]
```

2. `addbyelement(t1, t2)`: Write a function that adds two tuples by elements. Hint: try using functions such as `map()`, `sum()` and `zip()`.

```
Input: (1,2,3), (1,2,3)
Output: (2,4,6)

Input: (1,5,1), (1,-4,3)
Output: (2,1,4)
```

Task 4 – Dictionary manipulations

Write the following functions in Python. If you can, try to present multiple implementation for the same problem.

1. `applyfunondict(a, myfun)`: Write a function that applies a function on the values of a dictionary (e.g., calculates power of each value). Make sure that your solution works on nested dictionaries. Hint: this is a recursive function.

   Let's create this function that we would like to apply on the values of the dictionary:

```
def myfun(elem):
     return elem**2
```

```
Input: {'apple': 6, 'banana': {'hello': 5, 'world': 4}}
Output: {'apple': 36, 'banana': {'hello': 25, 'world':
16}}
```

2. `comparedicts(dict1, dict2)`: Write a function that compares two dictionaries by both their keys and values and return a boolean (without using `cmp()` or `==`).

```
Input: {'a': 1, 'b': 2}, {'a': 1, 'b': 2}
Output: True

Input: {'a': 1, 'b': 3}, {'a': 1, 'b': 2}
Output: False
```

3. `differences(dict1, dict2)`: Write a function that displays the differences between two dictionaries by returning a set of tuples.

```
Input: {'a': 1, 'b': 3}, {'a': 1, 'b': 2}
Output: {('b', 3), ('b', 2)}

Input: {'a': 1, 'b': 3}, {'a': 1, 'b': 3}
Output: set()
```

Task 5 – Organising a concert

Suppose you are putting together a concert and need to figure out how much to charge for a ticket. Your total expenses are £8000 (band and venue). The venue can seat at most 2000 people and you have determined through market research that the number of tickets you are likely to sell is related to the ticket's selling price by the following relationship:

```
sales = 2500 - 80 * price
```

According to this relationship, if you give away tickets for free, you will overfill the venue. On the other hand, if you charge too much, you won't be able to sell tickets at all. Your total income from the ticket sales will be `sales * price` and your profit will be this amount minus your expenses which are £8000.

A concert

Write a program to determine the most profitable ticket price, by creating a function that prints a table for each possible ticket price from £0 up to the `maxPrice` parameter that is given to this function as an argument. Name your function as `profitTable(maxPrice)`

When calling `profitTable(30)`, we should get a result like below:

```
Price      Income     Profit
-----      ------     ------
£ 1         2000       -6000
£ 2         4000       -4000
              ...
£ 22       16280        8280
£ 23       15180        7180
```

Our program so far has only considered whole pound tickets. Now modify the program to increment the prices with 50 cents in each iteration instead. Does our new function find a better ticket price?

Just by looking at these tables, it is hard to see the relationship and trends between the different parameters. Therefore, let's use `matplotlib` to plot the results. For this, you will have to create a function (call it `profitList()`) that returns the data for our plot (this will be two lists – one with prices and one with profits – think about how to return two lists from one function). Once you have this function, you can use matplotlib to plot a simple line plot, as shown here.

You should get a graph similar to the following: