Introduction to Python — Exercise 2

Rebecca Breu

February 2011

Data Types II

Exercise 1 The most frequent letters in the English alphabet are E, T, A, O, I, N, S, H, R, D. Write a function which checks whether a string consists only of those letters.

1.1 How can you make the function independent of capitalisation?

1.2 How can you get the letters of a given string that are not in the list of most frequent strings?

Exercise 2 (Morse code) Write a function which translates strings into Morse code. The translation should be independent of capitalisation.

A	· –	J	· – – –	S	· · ·
B	– · · ·	K	– · –	T	–
C	– · – ·	L	· – · ·	U	· · –
D	– · ·	M	– –	V	· · · –
E	·	N	– ·	W	· – –
F	· · – ·	O	– – –	X	– · · –
G	– – ·	P	· – – ·	Y	– · – –
H	· · · ·	Q	– – · –	Z	– – · ·
I	· ·	R	· – ·		

Object Oriented Programming

Exercise 3 (Point Class)

3.1 Implement the Point class from the lecture. Implement a human-readable print output and ensure that two points are recognised as equal when their x and y values match.

3.2 Implement the + and − operator for two points:

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + \begin{pmatrix} x_2 \\ y_1 2 \end{pmatrix} = \begin{pmatrix} x_1 + x_2 \\ y_1 + y_2 \end{pmatrix}, \qquad \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} * \begin{pmatrix} x_2 \\ y_1 2 \end{pmatrix} = (x_1 * x_2) + (y_1 * y_2)$$

The following should be possible:

```
>>> print Point(1, 2) + Point(4, 2)
(5, 4)
>>> print Point(1, 2) * Point(4, 2)
8
```

3.3 Implement the norm of a point with the help of the above scalar product:

$$\|p\| = \sqrt{p * p}$$

Exercise 4 (Bank account)

4.1 Implement a bank account class with attributes for the account number, the account balance and the name of the account holder. The account must not be overdrawn; ensure that any access to the account balance does not result in a negative account balance.

4.2 Additionally to the account data, add an attribute for the interest rate, which is the same for all accounts. Implement a methods which informs the user about the current interest rate, and another method which calculates the interest for the current account balance.

Python's Standard Library

Exercise 5 (Password generator) Write a program which generates a password consisting of eight characters. *Hint:* Look at `string.letters` and `string.digits`!

Exercise 6 Write a program which renames all files in a directory with extension `htm`, so that the new file extension is `html`.

Exercise 7 (CSV) Given the following file:

```
"Mr. Spock","Vulcan"
"Freddie ""Nightmare"" Krueger, Jr.","Elm Street 42,
Springfield"
```

(The linebreak in the second data set is on purpose!). The data sets consist of two fields: name and address. Read all data sets and display them.

Hint: You don't want to parse the data manually. (Why?)

Exercise 8 (Fun with regular expressions) Write a program which removes all html tags in a file. (html tags are enclosed in angled brackets, e.g. `<body>`, `</body>`.)

Exercise 9 (XML RPC) An XML RPC server runs on address `http://grow.zam.kfa−juelich.de`:8000. Create an XML RPC client and use its methods `listMethods` and methodHelp to find out what methods the server provides and how to use them.

Additional Exercises

Exercise 10 (UNO) Implement a simple UNO game without action cards. Proceed by the following steps:

10.1 (Basics) Suggestion for structuring:

- Class `Card` with attributes `suit` and `rank`, and a method `match(self, other)` which checks if a card can be played on another card.
- Class for a shuffled card deck which inherits from `list`:

```
class DeckUNO(list):
    RANK = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
    SUIT = ["Red", "Yellow", "Green", "Blue"]

    def __init__(self):
        cards = []

        for suit in self.SUIT:
            for rank in self.RANK:
                cards.append(Card(suit, rank))

        random.shuffle(cards)

        list.__init__(self, cards)
```

- Class `Player`. The init method takes a card deck object as parameter, of which six cards are removed and saved as the player's hand of cards; more parameters (player name) are possible. Additionally the `Player` needs methods to execute a move. If you want to have different types of players (computer player, human player), those methods differ. In this case you could inherit the classes for the different flavours of players from on basic `Player` class.

Let the program run until one player has won or no more new cards can be drawn from the playing stack. For this, exceptions might be useful, which can inherit from the basic `Exception`:

```
class WinException(Exception):
    pass

class StackEmptyException(Exception):
    pass
```

10.2 (Action cards) Want to do more? Implement action cards (like skip, draw, reverse).

10.3 (For experts) You can create the sorted list of cards much easier. You can use a list comprehension, and the `itertools` module could be of use...

Exercise 11 (For experts) The lecture showed how to use classes as a way of a flexible structure. Enhance on the principle and write a class `Bunch` whose `init` method accepts arbitrary keyword parameters and saves them to according attributes:

```
point = Bunch(x=2, y=3)
print point.x, point.y

person = Bunch(givenname="Homer", familyname="Simpson", phone=123456)
print person.givenname, person.familyname, person.phone
```

Not enough? More exercises:

http://www.pythonchallenge.com