Workshop 8 and 9: Strings, Dictionaries and Sets

Exercise 1 – Concept Revision in the Python Shell

Let's revise the concepts from this module by typing some simple statements into the **Python Shell**. Feel free to deviate from the workshop and experiment!

1. `pun = 'Eating a clock is very time consuming!!!'`

2. `pun[23:]`

3. `pun = pun.rstrip('!')`
4. `pun.count('i')`
5. `punWords = pun.lower().split()`

These statements manipulate a string in various ways.

6. `scores = {'Mary': [22, 23, 36], 'Fred': [20, 21, 29]}`

This creates a dictionary with two items. The items have keys of "Mary" and "Fred", and the values are both lists containing three integers (test scores out of 25, 25 and 50).

7. `scores['Mary']`
8. `scores['Fred'][1]`

These statements demonstrate how to refer to an item in a dictionary via its key. Since the value is a list, you can then refer to an item in the list via its index.

9. `scoreKeys = scores.keys()`

*This statement creates a live list of the keys in the "`scores`" dictionary. Check what it contains now by typing "`scoreKeys`", and then check it again after statement **10**.*

10. `scores['Jane'] = [21, 24, 30]`

11. `scores['Fred'][2] = 39`

These statements demonstrate adding a new item to the dictionary, and changing the value of an item in one of the lists inside the dictionary.

12. `HDs = [name for name in scores if sum(scores[name]) >= 80]`

This statement uses list comprehension to loop through the "`scores`" dictionary, summing the list of test scores for each person and adding their name to the "`HDs`" list if is at least 80.

13. `set1 = {4, 2, 6, 3, 6, 2, 7}`

14. `set2 = set(9, 1, 0, 2, 4, 1, 4, 6)`

15. `set2 = set([9, 1, 0, 2, 4, 1, 4, 6])`

Notice how sets automatically elminate duplicate values, and how the "`set()`" function only accepts one parameter (hence the need to pass data as a list, or use "{" and "}" instead).

16. `set1 | set2`
17. `set1 & set2`

These statements show the union and intersection of the sets – try some other operations!

Exercise 2 – Password Checker

Below is a copy of the password rules from ECU's password changing page:

- A maximum of 16 characters
- A minimum of 8 characters
- A minimum of 1 upper case character
- A minimum of 1 lower case character
- A minimum of 1 number
- A minimum of 1 special character: '~!#$%^*()_+-={}|[]\:<>?,./

Create a new file in the **Python Editor** and copy-paste the following code into it:

Python

```
def checkPassword(pword):
    return True

password = input('Enter your password: ')
if checkPassword(password):
    print('Your password is valid.')
else:
    print('Your password is not valid.')
```

As you can see, this code defines a "`checkPassword()`" function which receives a parameter named "`pword`" and at the moment simply returns `True`. The program then prompts for a password, and proceeds to an "if" statement which calls the "`checkPassword()`" function to display an appropriate message.

Your task is to **write the code of the "`checkPassword()`" function** so that it implements the password rules listed above and returns True if the password is valid. Read through the following hints and then try to implement the pseudocode on the next page to complete the function.

The easiest way to tackle this is to set *boolean variables for each piece of criteria* we need to check and set them to be *False*. These variables will indicate whether or not we have met each piece of criteria as we examine the password.

Next, *check each piece of criteria* and if it is met set the corresponding variable to *True*. At the end of the function, return *True if all of the boolean variables are True*, otherwise return False.

The length criteria can be checked using the "`len()`" function. The uppercase, lowercase and number criteria can be checked by looping through each character of the password and using string testing methods.

Checking for a special character is best achieved by creating a string containing all of the special characters (add a *backslash before the quote mark and the backslash characters in the string* so that they don't get misinterpreted) and checking if the character is "`in`" it:

Python

```
specialChars = '\'~!#$%^*()_+-={}|[]\\:<>?,./'
```

Here is pseudocode demonstrating this approach:

Pseudocode

```
Create a "shortEnough" variable and set it to False
Create a "longEnough" variable and set it to False
Create a "hasUpper" variable and set it to False
Create a "hasLower" variable and set it to False
Create a "hasNumber" variable and set it to False
Create a "hasSpecial" variable and set it to False
Create "specialChars" string containing the special characters

If password length is <= 16:
    Set "shortEnough" to True

If password length is >= 8:
    Set "longEnough" to True

For each character in the password:
    If it is uppercase:
        Set "hasUpper" to True

    If it is lowercase:
        Set "hasLower" to True

    If it is a digit:
        Set "hasNumber" to True

    If it is in the "specialChars" string:
        Set "hasSpecial" to True

If all boolean variables are True:
    Return True
Else:
    Return False
```

Write the code to implement this pseudocode in the "`checkPassword()`" function.

Exercise 3 – Enhancing the Password Checker

There are a number of ways that the "`checkPassword()`" function could be improved. Let's focus on making it more efficient. Try implementing the following changes:

- Immediately return False if the password is too long or too short.
- Only check for a certain piece of criteria if it has not already been met: Change "`if char.isupper()`" to "`if not hasUpper and char.isupper()`" so that the code can simply move on to the next check if the variable is already set to True.
- Stop checking characters if all criteria has been met: Move the line of code that checks if all of the boolean variables are True into the end of the loop body and return True right away if they are. After the loop, return False without checking the variables – the only way to get that far in the function is if the criteria was not met.

Exercise 4 – Username Generator

Create a new file in the **Python Editor** and copy-paste the following code into it:

```
students = {60254: 'John Smith', 60255: 'Gert Du-Cart', 60256: 'Sun Po',
            60257: 'Bert Woods', 60258: 'Andrew Butters', 60259: 'Betty Ho'}
```
Python

This simply creates a dictionary named "students" which contains 6 items. The keys are student numbers, and the values are their names. Your task is to write a program that will **generate usernames** from the student names / numbers provided, implementing the following rules:

- All characters in a username should be in *lowercase*.
- *Dashes* ("**-**") in a student's name should not appear their username.
- Usernames consist of the *first letter of the student's first name*, followed by the *first 4 letters of their surname* (e.g. "John Smith" would become "jsmit"). We will assume that a student's name only consists of a first name and surname.
- Usernames must have a *minimum length of 5 characters*. If a student's surname is less than 4 characters long, include *add 0s (zeroes) to the end* (e.g. "Sun Po" becomes "spo0").

Your program should **create a dictionary** named "`usernames`", with an item for each student. The keys should be their student number, and the values should be their username, e.g.:

```
usernames = {60256: 'spo0', 60257: 'bwood', 60258: 'abutt',
             60259: 'bho0', 60254: 'jsmit', 60255: 'gduca'}
```

(Since dictionaries are not ordered, the order that the items appear in may be confusing)

Once your program has generated all of the usernames and added them to the "`usernames`" dictionary, **loop through the dictionary to display them**:

```
60256 spo00
60257 bwood
60258 abutt
60259 bho00
60254 jsmit
60255 gduca
```

This is the sort of programming task where planning your solution using pseudocode is extremely helpful and can save you a lot time. Spend a few minutes thinking about what you need to do and **write a solution in pseudocode** before writing the Python code. Here are some hints to think about:

- You'll need to loop through the "`students`" dictionary and have access to both the student number and name – the "`items()`" dictionary method may be helpful here.
- Use string methods such as "`lower()`", "`replace()`" and "`split()`" on the names to sanitise and separate them into something you can use to generate usernames.
- Use the "`ljust()`" string method to add 0s to the end of usernames if they are too short. You don't even need to check the length of the username in advance – read the documentation to see how it works, and test the method in the Python Shell if needed.
- Create the empty "`usernames`" dictionary before the loop, and add usernames to it at the end of the loop body (once you've ensured that they are long enough).

Exercise 5 – Enhancing the Username Generator

Enhance the username generator to incorporate one additional requirement:

- Usernames must be *unique* – if a username would be identical to a previous username, resolve it by *adding a number to the end*. Start with 1, and increment the number until a unique username is generated (e.g. "jsmit1", or "jsmit2" if there is already a "jsmit1"...)

Since it takes a few steps to implement this requirement, writing a function that does it can help to keep the program nice and clear. I've provided pseudocode below for one approach I came up with, but there are many other ways you could tackle the problem.

Before you start, here's a new dictionary of students to use in your program – this one will result in multiple usernames being the same – there will be 3 "jsmit" usernames and 2 "spo00" usernames:

Python

```
students = {60254: 'John Smith', 60255: 'Gert Hans-Dyer', 60256: 'Sun Po',
            60257: 'Bort Woods', 60258: 'Andrew Butters', 60259: 'Betty Ho',
            60260: 'Jonah Smithers', 60261: 'Sha Po', 60262: 'Jane Smitt'}
```

The first step in ensuring there are no duplicate usernames is to detect them. This part is easy – check if a username you have generated is "`in`" the "`values()`" of the "`usernames`" dictionary. If it is, pass the relevant data to the function that will resolve the issue and store the returned result as the new username (which you then add to the "`usernames`" dictionary).

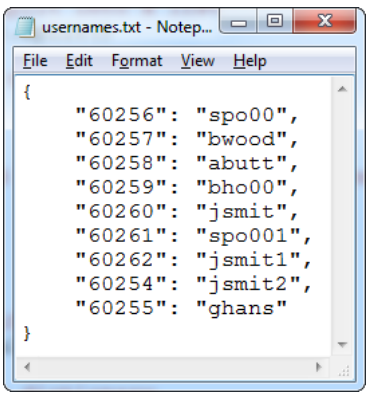
Once you have your code detecting duplicate usernames, work on the function to fix them:

Pseudocode

```
"resolveDuplicate" function
Receives "username" (string) and "usernames" (list) parameters

    Set "counter" to 0
    Endless Loop
        Add 1 to "counter"
        Set "newUsername" to "username" + "counter"
        If "newUsername" is not in "usernames"
            Return newUsername
```

The second enhancement is to make the username generator **write all usernames to a text file** named "usernames.txt", *in JSON format*.



To do this, import the "json" module, open/create "usernames.txt" in write mode, then use "`json.dump()`" to write the data. Remember to close the file after dumping the data to it.

See Reading 7.1 for more information about using the JSON module.