Practical 4 – [Lists]

Now that you know how to use Git and GitHub, please use them consistently for all your work. Each time you finish a part of the practical, commit it. Push once or twice per practical - and definitely when you finish and/or it's time to leave. We will remind you every now and then, but from now on, it should just be how you normally work. When you're about to make a change, like refactoring, commit first so you can track the differences and you can go back to a previous version if you need to.

Please remember, if you do not finish a practical during class time, it's not over… you need to complete it during the week so that you're all finished by the *start* of the next practical. If you do not understand anything, bring those questions to your tutor the following week or ask us for help online.

Warm-Up

```
numbers = [3, 1, 4, 1, 5, 9, 2]
```

First, write down your answers to these questions without running the code, then use Python to see if you were correct. What values do the following expressions have?

```
numbers[0]
numbers[-1]
numbers[3]
numbers[:-1]
numbers[3:4]
5 in numbers
7 in numbers
"3" in numbers
numbers + [6, 5, 3]
```

Write Python expressions (in the IDE) to achieve the following:

1. Change the first element of `numbers` to "ten"
2. Change the last element of `numbers` to 1
3. Get all the elements from `numbers` except the first two
4. Check if 9 is an element of `numbers`

Walkthrough Example

Calculating a List of Cumulative Totals

(Read the whole question before starting the work!)

Accountant Annie wants you to write a program to calculate the monthly cumulative totals for incomes. The program should ask for the number of monthly incomes to enter, then get and store them in a list. When the incomes have been entered, the program should display a list of the month's income next to the cumulative income at that point - for each month. Here's some sample output:

```
How many months? 5
Enter income for month 1: 120
Enter income for month 2: 245.4
Enter income for month 3: 900
Enter income for month 4: 1205.56
Enter income for month 5: 12.35
```

```
Income Report
-------------
Month  1 - Income: $     120.00 Total: $     120.00
Month  2 - Income: $     245.40 Total: $     365.40
Month  3 - Income: $     900.00 Total: $    1265.40
Month  4 - Income: $    1205.56 Total: $    2470.96
Month  5 - Income: $      12.35 Total: $    2483.31
```

Have a think about how to do this before reading on…

We need a **list** to store the incomes. How do you add values to a list?
We need a counter variable (int) to store the month number (remember that list indexes start at 0 but we want to print from 1).
How many loops will we need? What kind of loops?
We need a cumulative total to update as we loop through the list to display the incomes.
And lastly we need to format the output nicely, which we can use the string format() method for.

Things to do:

1. Copy the starter code from the CP1404 Practicals repo and commit it to your own prac repo: https://github.com/CP1404/Practicals/blob/master/prac_04/total_income.py
2. Study it to see how this code answers those questions so far.
3. Change the line that gets the income input so that it uses str.format() instead of string concatenation.
4. We have two variables that sound similar: **incomes** and **months**. They're both plural so they sound like they're both lists. incomes is a list of incomes, so we might assume that months is a list of months, but it's actually a scalar value that stores the number of months – an int not a list.
   Refactor the months **variable to a better name**. DO NOT just change it in 3 places or use find and replace... but use refactoring in PyCharm by clicking anywhere on the variable and pressing Shift+F6 (or use a menu). Then rename it.
5. Run the program again with some sample data and make sure it's still working well.
6. Now, let's refactor (move) the report printing into its own function. Select those 6 lines and turn them into a new function with a good name.
7. Test again and make sure it's all good.

Intermediate Exercises

Here are some small problems to give you more practice working with lists.
The **solutions** for these are at the bottom of this document so you can make sure you're on track.

1. Write a program that prompts the user for 5 numbers and then stores each of these in a list called numbers. The program should then output various interesting things, as in the output below (green represents user input).
   Note that you can use the **functions** `min`, `max`, `sum` and `len`, and you can use the `append` **method** to add a number to a list.

```
Number: 5
Number: 20
Number: 1
Number: 2
Number: 3
The first number is 5
The last number is 3
The smallest number is 1
The largest number is 20
The average of the numbers is 6.2
```

2. Woefully inadequate security checker…
   Copy the following list of usernames into a new Python file:
   ```
   usernames = ['jimbo', 'giltson98', 'derekf', 'WhatSup', 'NicolEye', 'swei45',
   'BaseInterpreterInterface', 'BaseStdIn', 'Command', 'ExecState', 'InteractiveConsole',
   'InterpreterInterface', 'StartServer', 'bob']
   ```

   Ask the user for their username. If the username is in the above list of authorised users, print “Access granted”, otherwise print “Access denied”.

3. **List comprehensions**
   Download https://github.com/CP1404/Practicals/blob/master/prac_04/list_comprehensions.py
   and see how the first two list comprehensions work.
   Complete the TODOs, creating two more list comprehensions to:
   a. create a list of integers from a list of strings
   b. create a list of all of the full_names in lowercase

Do-from-scratch Exercises

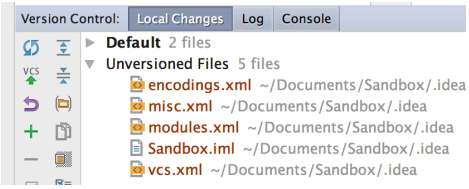
1. **“Quick Pick” Lottery Ticket Generator**
   Write a program that asks the user how many “quick picks” they wish to generate. The program then generates that many lines of output. Each line consists of 6 random numbers between 1 and 45. These values should be stored as CONSTANTS.
   Each line should not contain any repeated number. Make sure your numbers are unique.
   Each line of numbers should be displayed in sorted (ascending) order.
   Note: Python's random function has a **sample** method, which returns a selection from a list. This is a nice way to solve this problem... but it's too easy :) Do not use it for this program.

   Your code should produce output that matches this sample output (except the numbers are random):

   ```
   How many quick picks? 5
    9 10 11 32 38 44
    2  9 12 14 28 30
    5 10 16 22 35 42
    1  2 16 22 37 40
    1 17 20 23 25 27
   ```

.gitignore

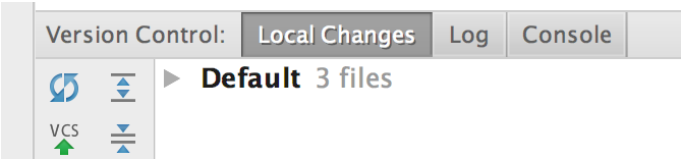
Before we're done, let's learn one more Git thing, ignoring files.
You will have files in your project that you don't want stored in your repo (like PyCharm metadata files), you can just choose not to add them (as we've done until now) but they do show up as "unversioned files", which is kind of a warning... We'd prefer this to show us files we probably should add.



The solution is to add a file called **`.gitignore`** to your repository. Note the exact spelling, including the dot at the start. On Unix-like systems (including Mac), the dot makes a file/folder hidden.
This is just a plain text file that stores the names of any files or folders you want Git not to track and not to warn you about.

Do this now – create the file, and let PyCharm add it to Git.
Then enter one line (the trailing slash means it will match a directory but not a file with that name):
`.idea/`

.idea is the directory (folder) that PyCharm stores its project metadata in.
Now look at that Version Control tool window... problem solved!



Commit.
Note: if you have already committed your .idea folder to your repository, PyCharm does not provide a way to stop tracking this. You have to use the Git command line (e.g. git bash, or Mac Terminal) and run the command:

```
git rm -r --cached .idea
```

This removes (rm) recursively (-r) the .idea folder, but only from the index, not the local disk (--cached).

Practice & Extension Work

We know from many years of teaching programming, that you need to spend lots of time writing programs, so here's some to write. This final part of practicals will usually be for you to do outside of practical time.
Use these exercises as normal practice and as ways to learn new things.

1. Download the scores reading program and its data file from
   https://github.com/CP1404/Practicals/blob/master/prac_04/scores.py
   https://github.com/CP1404/Practicals/blob/master/prac_04/scores.csv
   The data file stores the scores for each subject for 10 people.
   This code reads the lines into lists, saves the first line as a list of subject codes and converts the rest of the lines from a list of strings into a list of numbers, which it then prints with the maximum value for that subject. Nice… Except, it's broken! It reads the lists per user not per subject, so the results are incorrect.
   a. Save the CSV file to your project folder and copy the code into new file and run it to see how it currently works.
   b. Can you fix it so that it prints the list of (10) scores per subject with the maximum per subject?
   c. When you've done that, also print the minimum and average for each subject.
   d. Then print the results in a nicely-formatted table.

2. Extend the first intermediate exercise so that the user can enter any number of numbers until a number less than zero is entered. Adjust the prompt so that it prints like "Number 1: " then "Number 2: "

3. Write a program that asks the user for an indefinite set of strings - until an empty string is entered - then prints all of the strings that were repeated.
   E.g. if the user entered: "hello", "world is good", "hello", "john", "good"… the program would print "Strings repeated: hello".
   If no repeated strings are entered, the program should print "No repeated strings entered".

4. Memberwise Addition…
   In Python, the + operator concatenates lists, so [1, 2, 3] + [4, 5, 6] = [1, 2, 3, 4, 5, 6].
   What if we want to add the elements together instead?
   Write a function, `memberwiseAddition`, that takes two lists, and returns the list that contains the sum of elements that are in the same index in the two lists. For example:
   memberwiseAddition([1, 2, 3], [4, 5, 6]) would return [5, 7, 9]
   memberwiseAddition([1, 2, 3], [1, 2, 3, 4]) would return [2, 4, 6, 4]

Solutions to Selected Exercises

Remember we have many solutions available in the solutions branch of the Practicals repository: https://github.com/CP1404/Practicals/tree/solutions

Remember also that the solutions are there to help you learn ***after*** you have done your best to complete the tasks on your own and using the teaching resources we have provided.

1.

```
numbers = []
for i in range(5):
    number = int(input("Number: "))
    numbers.append(number)

print("The first number is", numbers[0])
print("The last number is", numbers[-1])
print("The smallest number is", min(numbers))
print("The largest number is", max(numbers))
print("The average of the numbers is", sum(numbers) / len(numbers))
```

2.

```
usernames = ['jimbo', 'giltson98', 'derekf', 'WhatSup', 'NicolEye',
'swei45''BaseInterpreterInterface', 'BaseStdIn', 'Command', 'ExecState',
'InteractiveConsole', 'InterpreterInterface', 'StartServer', 'bob']
username = input("Enter username:" )
if username in usernames:
    print("Access granted")
else:
    print("Access denied")
```