COMP11124 Object Oriented Programming

Introduction to Python Programming II

Learning Outcomes

The practical lab exercises in this document continue the introduction to programming using the Python programming language. After completing this week's class, you will be able to build more complex programs that include conditionals, loops, and lists and can handle user input.

Topics Covered

Comparisons and Conditionals, Lists, Loops, Input,

Getting Started Task: To retain all the code that you write and be able to work through the exercises in this lab, ***create a new Python file called:*** **`lab_week_2.py`** like how we did it last week. You can include the code of this week here and should continuously execute it to see the output. If you want to see the value of a variable, please *print* it using the **`print()`** function, as I have omitted that in the sample code.

1. Comparisons and Conditionals

In Python, comparison operators are used to compare values, and conditionals help control the flow of a program based on these comparisons.

Exercise 1: Comparison Operators

Comparisons in Python simply compare two or more values. A comparison always returns a Boolean value, meaning it can either be True or False.

For example, consider the following code, which is not a comparison but stores a Boolean.

```python
is_true = True
```

Here we create a new variable and assign it the value: **`True`** (Note: this is without quotes as it is not a *String* but a Boolean). You could also assign it the value **`False`**.

A comparison in Python is very similar to us evaluating a statement such as: ***Is 5 larger than 4?*** The answer would be yes (which is equivalent to **`True`**) for our program. We can then turn this comparison into code and evaluate the output as follows:

```python
is_true = 5 > 4
```

Where we are simply instructing the Python interpreter to store the output of the comparison whether 5 is larger than 4. This will return **`True`**.

Next to checking whether values are larger and smaller using the following operators: < (less than), > (greater than) we can also create a comparison using the following:

- == (equal)
- != (not equal
- <= (less than or equal to)
- >= (greater than or equal to)

Task: Follow this link and click on the *Try It* button for each of the comparison operators. Change the values for each x variable to make the comparisons false.
After this, you should be able to build your comparisons successfully based on variables.

Exercise 2: Logical Operators

Rather than using only a single comparison to check whether something is True or False, there are cases where you would want to combine them. For example, you want to check whether someone is between the ages of 20 and 30. Using logical operators, you can combine those comparisons and evaluate them in one go. In our example we would therefore check: **Is the age OVER 20 and UNDER 30.**

Python has three logical operators:
- The **and** operator returns **True** if both operands are True.
- The **or** operator returns **True** if at least one operand is True.
- The **not-operator** returns **True** if the operand is False and vice versa.

So for our example above we could use the *and* keyword to build our comparison:
```
age = 25

is_in_age_range = age > 20 and age < 30
```
where age takes the value of the age that we want to check.

Task: Follow this link and look through the examples in the Try It sections.

Exercise 3: if – Conditionals

Now that we have learned how we can make comparisons, we can start creating instructions that execute certain lines of code *when a comparison is true*.
For example, in plain English, we can say that: *If someone is over 18, they are an adult.*
We can then turn this into Python code using a simple if-conditional:
```
age = 19
age_group = "child"

if age > 18:
    age_group = "adult"

print(f"The age group is {age_group}")
```

In the first two lines, we assign an **age** and a default **age_group**. In the middle of the code lines, we have our **if** statement. It simply checks whether the age is over 18 and if it is (e.g. the comparison is True) it changes the **age_group** to adult in the indented code block. Indented code blocks (meaning the code is moved 4 spaces (or 1 tab) to the right) signal that they are related to the previous – non-indented code block.
For your code, if you write indented code, use the *tab* symbol to *indent* the code.

Task: Run this program twice, once with the given age and once when you change the age to something below 18. Observe the output and you will see that depending on the age, the **age_group** changes.

Exercise 4: if – else Conditionals

The standard *if* conditional only allows us for one choice, e.g. *if the comparison is True do something.* However, many times we want to choose one option out of two. This is where the *if-else* conditional comes in. Let's say we have a statement in plain English, and we want to check whether it is a windy day. *"If the wind speed is over 10mph it is a windy day, otherwise it is a calm day"*. Turning this statement into code would look as follows:

```python
wind_speed = 30

if wind_speed < 10:
    print("It is a calm day")
else:
    print("It is a windy day")
```

First, we set an arbitrary wind speed, then in the last 4 lines we check whether *the wind speed is less than 10*, and if this is true, we print that it is a calm day.
The other option is indicated by the **`else`** statement and covers all other cases. For us, this means that when the wind speed is *not* smaller than 10, it is windy.
Note that *only one* option will be executed, either the one following "**`if`**" or the one following "**`else`**".

Task: Run this program twice, once with the given wind speed and once when you change the speed to something below 10. Observe the output and you will see that depending on the wind speed, different parts will be executed.

Exercise 5: if – elif - else Conditionals

In some occasions, two choices following the if-else are not enough. There are cases where we have more than two options and we can also reflect this in Python code.

Let's say we are building a grading system and want to automatically give feedback based on a student's mark. The following rules would apply:

- 0-50% -> Fail
- 50-60% -> Passed.
- 60-70% -> Good Pass
- 70-80% -> Excellent Pass

We can reflect this in Python by putting an **`elif`** block in between the **`if`** and **`else`** blocks. The **`elif`** (else-if) block behaves like an *if* conditional and simply checks whether another condition is true.

```python
grade = 55

if grade < 50:
    print("You failed")
elif grade < 60:
    print("You passed")
elif grade < 70:
    print("You got a good pass")
else:
    print("You got an excellent pass")
```

Here, in the first line, we start by assigning a value to the variable grade and then we have our conditional.

The first **`if`** simply checks whether our grade is below 50, if it is true, the next indented line which tells us that we have failed, will be executed.

But if we have a grade larger than 50%, the first **`elif`** code block will be executed. Here, the check is simply whether our grade is below 60% (as we have already covered the case of a grade being below 50%) and if this is **`true,`** then we will have a message telling us we have passed. The next **`elif`** block checks whether our grade is below 70% in a similar fashion and executes the print statement with a "good pass" if our mark is between 60 and 70%. If the mark is now above 70% (which is our **`else`** statement) we will obtain an *excellent pass*.

Exercise 6: Summary Tasks

Task: Look through the examples and exercises here to understand how to use them in more detail: https://www.w3schools.com/python/python_conditions.asp

Task Compare Temperatures:

- Create two variables, **`temperature1`** and **`temperature2`**, and assign different values to them.
- Use an **`if`** statement to check if the temperatures are equal and print a corresponding message.
- Use an **`else`** statement to print a message if the temperatures are not equal.
- Run the code and see if your statement has been built correctly.

2. Python Lists

In Python, a list is a versatile and mutable (changeable) data structure that can store a collection of items. Lists are defined by enclosing elements in square brackets [] and are indexed starting from 0. They can contain elements of different data types.

Exercise 1: Creating a list

Creating a list is similar to creating a variable and assigning it a value.
Look at the following examples to see how we can create lists. One important thing to note is that when you create a list, you need to start and end with square brackets. Each item in the list should be separated by a comma and a space.

```python


integer_list = [1, 2, 3, 4, 5]
string_list = ["apple", "banana", "orange", "grape"]
empty_list = []
list_with_different_types = [1, "two", 3.0, True]

person_1_age = 20
person_2_age = 30
# creating a list based on variables
age_list = [person_1_age, person_2_age]
```

```
list_within_a_list = [["red", "green", "blue"], ["yellow", "orange", "purple"]]
```

The last list (within a list) is somewhat special in that the list consists of two sub-lists with three items each.

Task: Create a list in the variable **`city_list`**. It should contain the following items in this order: "Glasgow", "London", "Edinburgh”.

Exercise 2: Accessing a list

You can utilise the whole list or individual items in the list by using the list name followed by square brackets containing the index of the item you want to use. For example, if you wanted to use the first item in the **`string_list`**, you would use the following code: **`string_list[0]`** which would yield the output "apple".
List items are indexed starting at 0, so the first item in the list has an index of 0, the second item has an index of 1, and so on.
If you try to access an item in a list that does not exist (for example, if you try to access the 4th item in a list with only 3 items), you will get an error.

```
second_item = string_list[1] # will store "banana" in second_item
```

In addition to accessing individual items in a list, you can also access a range of items in a list. This is called slicing.
To perform slicing, you need to specify the index of the first item you want to access, followed by a colon, followed by the index of the last item you want to access plus one.
Example:

```
string_list[0:2] # will return ["apple", "banana"]
```

will return the first two items in the list, "apple" and "banana".

You can also use negative numbers to access items in a list. Negative numbers count backwards from the end of the list. For example, **`string_list[-1]`** will return the last item in the list, "grape".

```
last_item = string_list[-1] # will store "grape" in last_item
```

Once you have accessed a list item you can do various things with it, similar to what you can do with a variable. For example, you can assign it to a new variable, or you can use it in a print statement.

Task: Print the third item in the **`city_list`** list. Then print the last two items in the **`city_list`** list as well using slicing.

Exercise 3: Modifying a list

You can modify a list by assigning a new value to an item in the list. For example, if you wanted to change the first item in the list to "pear", you would use the following code:

```
string_list[0] = "pear"
```

The syntax is similar to assigning a new value to a variable but instead of the variable name, you use the syntax that we have just covered in the previous section.

Additionally, you can also add items to a list by using the **`append()`** function. For example, if you wanted to add the item "orange" to the end of the list, you would use the following code:

```
string_list.append("orange")
```

The **`append()`** function adds the item to the end of the list and you will simply need to specify the item you want to add in between the brackets.

Task: Add the item "Manchester" to the end of the **`city_list`** list. Then change the second item in the **`city_list`** list to "Birmingham".

Exercise 4: Summary Task

Now that you understand how to work with lists, try the following. If you are stuck or unsure how something works, please don't hesitate to look into the documentation here https://www.w3schools.com/python/python_lists.asp (whose reading is recommended to understand more about lists).

Task Create, Access and Modify Lists:

- Write Python code to create a list named **`colours`** containing the names of three colours as strings.
- Print the entire list.
- Access the second element of the **`colours`** list and print it.
- Modify the first element of the list to a new colour of your choice.
- Print the modified list.
- Check and print the length of the **`colours`** list using the **`len()`** method. This is similar to using the similar to using the **`type()`** method from before.
- Use a conditional to check if "red" is in the **`colours`** list. If yes, print that "Red is in the list".
- Use slicing to create a new list named **`selected_colours`** containing the second and third elements from the **`colours`** list.
- Print the **`selected_colours`** list.

3. Python Loops

Loops in Python allow you to repeatedly execute a block of code. Two common types of loops are for loops, which can iterate over a sequence (such as a list), and **`while`** loops, which will repeat as long as a condition that was specified in the beginning is true.
To find out more about loops, read the following links

- https://www.w3schools.com/python/python_while_loops.asp
- https://www.w3schools.com/python/python_for_loops.asp

and work through the next few exercises.

Exercise 1: While Loops

While loops are a type of loop that will continue to run as long as a certain condition is true. For example, if you wanted to print the numbers 0 through 4, you could use a while loop like this:

```
i = 0
while i < 5:
    print(i)
    i += 1
```

You may be wondering what the **+=** operator in the above example is? This is simply a shortcut for writing **`i = i + 1`** and is a shortened assignment operator. More examples of this are shown here: https://www.w3schools.com/python/python_operators.asp/

The **`while`** loop uses a comparison as introduced previously. In this case, the comparison is i < 5. As long as this comparison is true, the loop will continue to run. The loop will stop running when the comparison becomes false. In this case, the comparison will become false when i is equal to 5 because 5 is not less than 5. The loop will run 5 times because it will run when i is equal to 0, 1, 2, 3, and 4.

Note: Be careful when using while loops! If the comparison never becomes false, the loop will run forever. This is called an infinite loop. If you accidentally create an infinite loop, you can stop it by pressing the stop button in the toolbar at the top of the screen in VSCode or pressing Ctrl+C to stop the program.

Exercise 2: For Loops

For loops on the other hand are a type of loop that will iterate over a sequence of items. For example, if you wanted to print all items in our **`string_list`** list, you could use a for loop like this:

```python
for fruit in string_list:
    print(fruit)
```

The **`for`** loop uses the variable fruit to represent each item in the list. This variable can be named anything you want, but it is common to use a variable name that is related to the list. It only exists within the loop, so you can use the same variable name for other things outside of the loop (but it is good practice to use a different variable name to avoid confusion).

The loop will run once for each item in the list, and the variable fruit will represent the current item in the list. In this case, the loop will run 4 times, if there are 4 items in the list, printing each item in the list (as it is stored in the variable fruit).

Task: Create a for loop that prints each item in the **`city_list`** list.

Exercise 3: Loop Keywords: Range, break and continue

The **`range()`** function is a built-in function in Python that allows you to generate a series of numbers within a given range. For example, **`range(0, 5`**) generates the numbers 0, 1, 2, 3, and 4. You can use the **`range()`** function to generate a series of numbers to use in a for loop.
Here are some examples of using the **`range()`** function:

```python
range(0, 5) # will return [0, 1, 2, 3, 4]
range(5) # will return [0, 1, 2, 3, 4]
range(0, 5, 2) # will return [0, 2, 4] because the third parameter is the step size
range(5, 0, -1) # will return [5, 4, 3, 2, 1]
```

But why do we need the **`range()`** function? Well, **`the range()`** function is useful when you want to perform a task a certain number of times. For example, if you wanted to print the numbers 0 through 4, you could use a for loop like this:

```python
for i in range(5):
    print(i)
```

Here **i** is the variable that represents the current number in the range which is printed.

But what if you want to stop a loop before it has finished all of its iterations? For example, if you wanted to print the numbers 0 through 4, but stop the loop when **i** is equal to 2? For this, we have the **break** keyword.

The **break** keyword will stop the loop from running any more iterations. You simply write the **break** keyword inside the loop where you want to stop it (often inside an **if** statement).

```
for i in range(5):
    if i == 2:
        break
    print(i)
```

Task: Modify the above code to print the numbers 0 through 4, but stop the loop when **i** is equal to 3.

Another loop keyword is the **continue** keyword. The **continue** keyword will stop the current iteration of the loop and continue to the next iteration, e.g. skipping the rest of the code in the loop for the current iteration.
Let us say we want to continue counting from 0 to 4, but skip the number 2. We can do this by using the **continue** keyword inside an if statement like this:

```
for i in range(5):
    if i == 2:
        continue
    print(i)
```

Exercise 4: Summary Tasks

Now that you understand loops, let us write some code examples in your Python file to test your understanding. The following are several tasks that can be completed independently but you should try to do them all.

Task – Even Numbers

1. Create a list called **numbers** which contains the integers from 1 to 10.
2. Use a **for** loop to iterate through the list and only print the *even* numbers. (Hint: use the modulo % operator)

Task – Sum of Squares

1. Create a variable **sum_of_squares** and initialize it to 0.
2. Use a **for** loop to iterate through the numbers from 1 to 5 (inclusive) using the **range()** function.
3. Add the square of each number to **sum_of_squares.**
4. Print the final value of **sum_of_squares**. (Hint: if you do it correctly, the result should be 55)

Task – Countdown:

1. Create a variable **countdown** and initialize it to 10.
2. Use a **while** loop to print a countdown from the value of **countdown** to 1.
3. After the countdown, print "Liftoff!"

4. Obtaining User Input

A lot of times you want to not only write programs that work with pre-set values and variables but rather use interactions with the users to achieve the outcome you planned. One way to do this is to obtain the input text that the user enters.

In Python, the **`input()`** function is used to take input from the user. It pauses the program and waits for the user to enter some text. The entered text is treated as a string. More information on the **`input()`** function can be found here: https://www.w3schools.com/python/ref_func_input.asp

Some examples of how **`input()`** can be used are found below:

Example: Entering a basic string

```
user_input = input("Enter something: ")
print("You entered:", user_input)
```

Example: Entering a number

```
age = input("How old are you? ")

# Convert the age to an integer
age = int(age)

next_year_age = age + 1
print("Next year, you'll be", next_year_age, "years old.")
```

Task: User Input and Conditional Statements

Write a code that takes the user's age as input. If the age is less than 18, print "You are a minor." If the age is between 18 and 65 (inclusive), print "You are an adult." If the age is greater than 65, print "You are a senior citizen."

Task: Temperature Converter

Modify the temperature converter from last's week lab class to also take a user input. The user should be able to enter a value in degrees Celsius and your converter should convert this to Fahrenheit and Kelvin. **As an extra task**, (if you are finished before the class is over or want to practice a bit more Python) you should allow the user of the program to enter what temperature they want to convert (C, K or F) and then print out the conversions. Use conditionals for this.