

# COMP11124 Object Oriented Programming

## Introduction to Python Programming I

---

### Learning Outcomes

The practical lab exercises in this document serve as an introduction to Python. After completing this week's class, you should be able to implement your first own Python procedural programs, understand how to use the Python syntax.

### Topics Covered

Visual Studio Code, Running Python Code, Variables, Operators, Print, Casting, Strings, f-Strings

---

## 1. Setting Up a Code Editor and Running Python Scripts

### Exercise 0: Install Python

We first need to install the Python interpreter before we start with any programming in Python.

**Task:** Download and install Python from the following link: <https://www.python.org/downloads/>

### Exercise 1: Setting Up Visual Studio Code Editor

To write any Python code we will first need a code editor. In this module, we will use the Visual Studio Code (VS Code) editor developed by Microsoft. This code editor does not only support Python, but it also supports reading and editing a range of other file types and programming languages.


**Task:** Download and install VS Code on your computer. You can find the link to download it here: <https://code.visualstudio.com/>. It would also be beneficial if you have a look around on the linked site, as it also contains information about other features of this editor.

**Note:** If you are using a university lab computer, you can skip this step as the editor may already be installed.

Once you have installed it, you can now start the editor.


**Task:** Start the editor and open the file for this week available on aula (*lab\_week1.py*). VS Code will ask you if you want to install the **Python extension** upon the first start up, which you should confirm and install.

If you look around in this editor, you will see the Python version that is currently installed on the bottom right-hand corner:



Ln 79, Col 44 Spaces: 4 UTF-8 LF Python 3.11.3

You may find that there is some code already in this file. You can run the file in VSCode by clicking the “Play” button at the top right hand corner.

**Task:** Click the play button  at the top right hand corner and look at the output of the script. Change the value of the variable “*name*” and run the script again. Does the output change?

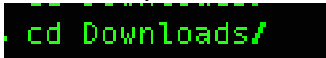
What we have done here is running the script using the editor. However, there is also an option to run the script via the command line.

## Exercise 2: Running a Script Using the Command Line

We can use the command line to run our Python scripts as well. To do so we need to open the command line tool (which is CMD on Windows and Terminal on Mac).

**Task:** Open the command line tool on your computer. Navigate to the directory where you have saved the *lab\_week\_1.py* file **in the command line**. The easiest way to do is to type the *cd* command and then follow this up with the directory where the file is saved in.

Example:

For example, I have saved my file in the *Downloads* directory. Therefore, after opening the command line I would type: 

Once we have changed the directory successfully, we want to type the command to run the file. It is: ***python lab\_week\_1.py***

You should now see the same output as when you have run the file via VSCode.

You should now understand that there are multiple ways how you can run Python files, both ways are valid and usually give the same output.

## 2. Python Introduction

### Exercise 1: Variables and Types

A variable is a named location in memory used to store data. You can assign a value to a variable using the assignment operator `=`. Variables are fundamental in programming and play a key role in writing dynamic and flexible code. Variables allow you to work with data by giving it a name.

Python supports various data types, including integers, floats, and strings. In this exercise you will practice working with different data types and find out how 1) we can use the `type()` function to check the data type of a variable, 2) we can convert (cast) between different data types using the `int()`, `float()`, and `str()` functions, and 3) we can use the `print()` function to print out the data type of a variable. A variable is a named location in memory used to store data.

Below is some example code that shows you how variables in Python are used. Here we create variables and assign them certain values.

```
# Create a variable 'x' and assign it the value 10
x = 10

# Create a variable 'name' and assign it a string
name = "Alice"

# Create a variable 'pi' and assign it a float value
pi = 3.14159
```

**Task:** Create a new Python file called *lab\_week1\_part2.py*. To do this click on File -> New File in VS code and store the file using this name. This should also automatically open the file.

**Task:** Copy the following code into your new Python file:

```
var_1 = True    # Type =
var_2 = 1       # Type =
var_3 = 3.14159 # Type =
var_4 = "Hello World" # Type =
```

Then, look at the value of each variable and try to guess what type they are. Put your guess behind the "equals =" sign into the code. If you are unsure, use the following link to help you:

[https://www.w3schools.com/python/python\\_datatypes.asp](https://www.w3schools.com/python/python_datatypes.asp)

**Task:** To now double check whether your guesses were correct we can use Python's built in `type()` function and print out the result.

Use the following syntax:

```
print(type(var_1))
```

and print out the type for each variable (var\_1 to var\_4). Does this align with your initial guesses?

The print function is very helpful for finding out what values are stored in variables and what type they are. There are better ways to do this, but for now, this is a good way to check your work.

At last, Python also allows us to convert values from one type to another. This is called casting. We can cast a value to a string by using the `str` function like this: `str(5)`.

Other functions that we can use are:

- int() to convert to an integer
- float() to convert to a floating point number
- bool() to convert to a Boolean value

#### Task:

1. Create a variable called my\_int and assign it the value 5.
2. Create a variable called my\_float and assign it the value 5.5.
3. Create a variable called my\_bool and assign it the value True.
4. Print the value of each variable.
5. Cast my\_int to a float and store the result in a variable called my\_int\_float.
6. Cast my\_float to an integer and store the result in a variable called my\_float\_int.
7. Cast my\_bool to an integer and store the result in a variable called my\_bool\_int.
8. Print the value of each variable.

Then observe the output of each variable. Does this align with what you thought the variables will be after casting? To find out some more information, it is recommended to check out this link:

[https://www.w3schools.com/python/python\\_casting.asp](https://www.w3schools.com/python/python_casting.asp)

### Exercise 2: Arithmetic Operators

Python arithmetic operators, which allow you to perform mathematical operations on numeric values. Understanding and using these operators is fundamental as they make up the base of a lot of programs.

Python provides several arithmetic operators for performing basic mathematical operations:

- **Addition (+):** Adds two values.
- **Subtraction (-):** Subtracts the right operand from the left operand.
- **Multiplication (\*):** Multiplies two values.
- **Division (/):** Divides the left operand by the right operand.
- **Floor Division (//):** Performs floor division, returning the quotient as an integer.
- **Modulus (%):** Returns the remainder of the division.
- **Exponentiation (\*\*):** Raises the left operand to the power of the right operand.

**Task:** Copy and paste the following code into your Python file, try to understand it, and then run it:

```
#Addition
result_addition = 10 + 5
print("Addition:", result_addition)

#Subtraction
result_subtraction = 20 - 8
print("Subtraction:", result_subtraction)

#Multiplication
result_multiplication = 6 * 4
print("Multiplication:", result_multiplication)

#Division
result_division = 15 / 3
print("Division:", result_division)
```

```
#Floor Division
result_floor_division = 17 // 4
print("Floor Division:", result_floor_division)

#Modulus
result_modulus = 17 % 4
print("Modulus:", result_modulus)

#Exponentiation
result_exponentiation = 2 ** 3
print("Exponentiation:", result_exponentiation)
```

You will also see that we are using the *print* function not only to print out one value, but several ones.

Now, let us put the use of those operators and variables into practices by performing two common kinds of calculations.

### Task – Calculating the Average:

1. Create two variables, *num1* and *num2*, with any numeric values.
2. Calculate their average using the arithmetic operators and assign it to a variable called *average*.
3. Print a message that displays the values of *num1*, *num2*, and the calculated average.

### Task - Calculate the Area of a Rectangle:

1. Create two variables, *length* and *width*, representing the dimensions of a rectangle.
2. Calculate the area of the rectangle using the arithmetic operators and assign it to a variable called *area*.
3. Print a message that displays the values of *length*, *width*, and the calculated area.

### Exercise 3: Strings and f-Strings

So far you will have used the String data type to store sequences of characters in variables. However, Python offers us also extra functionality that we can use with strings. Each variable that contains a string is an Object (more on that in later classes) and has built in methods (that are actions that it can perform).

You can find a list of methods that you can perform on strings in this documentation:

[https://www.w3schools.com/python/python\\_ref\\_string.asp](https://www.w3schools.com/python/python_ref_string.asp)

As an aspiring programmer we do not expect you to know everything about a programming language. One very important soft-skill is to be able to use documentations (that exist for every programming language). You need to be able to find a way to look for a solution, even if you have not done it before. In that regard, the next task requires you to use the link above and modify string-variables to do the following:

### Task – Modify Strings

1. Create a new variable called *my\_string* and store the following in it: *"This class covers OOP."*

2. Print the *my\_string* variable
3. Using the documentation in the link above, look through the examples and do the following:
  - a. Create a new variable *my\_uppercase\_string* that stores the *my\_string* converted to uppercase
  - b. Create a new variable *my\_lowercase\_string* that stores the *my\_string* converted to lowercase
  - c. Create a new variable *my\_new\_string* that replaces the word “OOP” in the string with “Object Oriented Programming”.
  - d. Create a new variable *my\_string\_length* that stores the length of the string (Note: use a search engine or the lecture slides to help you how to do this).

F-Strings (also called [formatted string literals](#)) allow you to evaluate expressions within a string. When you create a new string, you will need to preface it with the letter *f* and can then evaluate your expressions within the quotes of a string if you limit them by curly brackets **{expression}**.

This can allow us to write less code if we want to combine multiple variables into a string for example. Consider the following:

```
class_name = "OOP"
number_of_students = 20
my_string = "Welcome to the " + class_name + " module. There are " + str(number_of_students) + " students in this class."
```

You will need to write parts of the string, and then concatenate them with other values to create your output “my\_string”.

F-Strings make this much easier, the code above can be converted simply to:

```
my_string = f"Welcome to the {class_name} module. There are {number_of_students} students in this class."
```

and will yield the same result. Note the *f* at the beginning before the opening quotation and the variable names wrapped in {}.

### Task – f-Strings

1. Create variables called *my\_name*, *number\_of\_classes*, *campus*. Store your name, the number of classes you are taking this term and then the campus you are studying at.
2. Create an f-string using the instructions above called *my\_text* that combines the text to the following scheme. (Example: *my\_name* = Peter, number of classes = 10, campus = Paisley):  
“My name is Peter and I am studying 10 classes in Paisley”.
3. Print the string, run the program, and ensure that you have done everything correctly.

## 3. Your First Python Program

In this exercise, you will write a Python program that allows the user to convert temperatures from Celsius to Fahrenheit and Kelvin.

Your **Task** is to look at the following requirements and implement the program in the file called *temperature\_converter.py*.

Requirements:

- The user stores the Celsius value as a number in a variable called *celsius\_input*
- The Celsius value then gets converted to Fahrenheit and stored in a variable called *degree\_f*

- The Celsius value then gets converted to Kelvin and stored in a variable called *degree\_k* (Note: Use the internet to find out the formulas on how to convert C to K and F and implement that using Python operators)
- Generate and print an output that follows the following format:

*Welcome to the Temperature Converter!*

*The temperature you have entered is **{USER ENTERED C}** degree Celsius.*

*Converted Temperatures:*

***{USER ENTERED C}** degree Celsius is equal to **{COMPUTED degree\_f}** Fahrenheit.*

***{USER ENTERED C}** degree Celsius is equal to **{COMPUTED degree\_k}** Kelvin.*

*Thank you for using the Temperature Converter!*

Once you are finished with this task, please show the output of this and the code to your lab tutor for formative feedback.