Intro to Python
Lab 1

1 Online References

1.1 Basic references

https://www.w3schools.com/python/

1.2 Intermediate references

https://www.pythontutorial.net/

https://www.programiz.com/python-programming/first-program

https://pynative.com/python/

1.3 Advanced references

https://realpython.com

Official tutorial
https://docs.python.org/3.11/tutorial/

1.4 Community forums

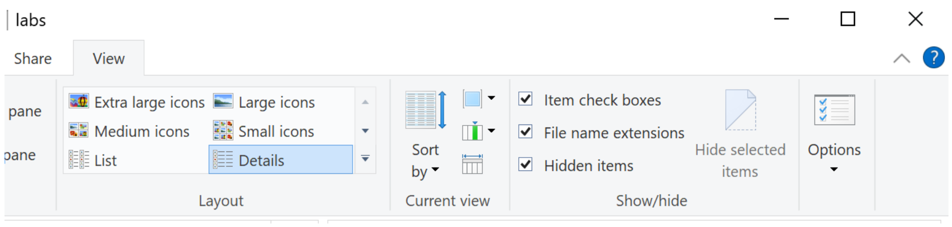
https://forums.feedspot.com/python_forums/

https://www.devglan.com/programming/python-forums

2 Lab Setup

You should have received installation instructions on setting up Anaconda, which includes the Spyder IDE.

We will need to be able to view file name extensions directly in order to be able to directly manipulate Python source code files. If you are using Windows, make sure that you have checked the File name extensions in your File Explorer in order for us to be able to directly manipulate the file extensions when creating or modifying files.



For MacOs:
Show or hide file name extensions - Article 1
Show or hide file name extensions - Article 2

3 Viewing Python source code files in Spyder

Python programs are text files containing Python source code and are known as either scripts or modules. By convention, those files will use the `.py` extension. On Windows systems the extension can also be `.pyw`.

You can either open the source code files individually (File -> New File), or create a new project from the directory holding all your source code files (Project -> New Project) -> Choose existing directory and select the folder containing the sample source code files (`labcode`)

https://docs.spyder-ide.org/current/panes/projects.html

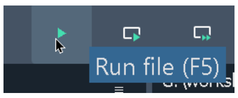
4 Basic Python program

File to use: `basic-program.py`

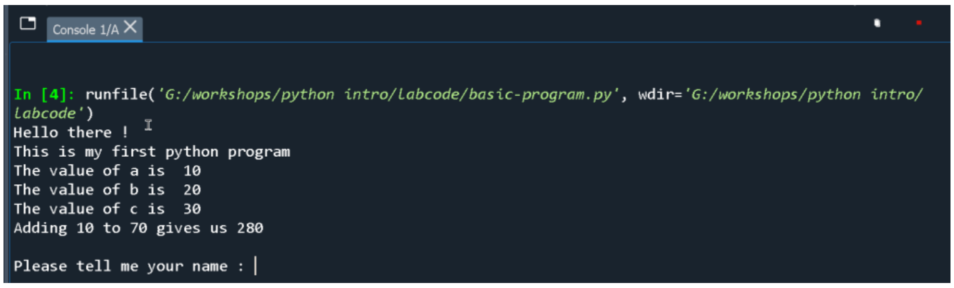
Python code is executed by the interpreter. There are 3 main ways to use the interpreter to execute Python code:

1. Execute code contained in a script / module from the from inside an IDE.
2. Execute code contained in a script / module from the CLI using the interpreter directly
3. Typing Python statements directly into the interactive / standard shell. For the case of the Spyder IDE, the interactive shell is provided by the IPython console

To run a Python script (`*.py`) from inside the Spyder IDE, ensure the tab for the editor holding the Python script you want to execute is highlighted and click the Run button (or F5 as shortcut).



The output from the `print` statements as well as any input required from the user will be shown in the IPython console that is by default located at the lower right hand corner of the IDE.

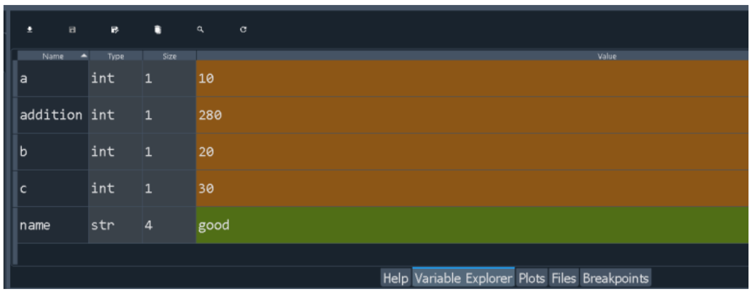


```
In [4]: runfile('G:/workshops/python intro/labcode/basic-program.py', wdir='G:/workshops/python intro/
labcode')
Hello there !
This is my first python program
The value of a is  10
The value of b is  20
The value of c is  30
Adding 10 to 70 gives us 280

Please tell me your name :
```

Variables are used to store different types of values (such as text and numbers) in a program.
You assign values to a variable in an assignment statement.

The names of the variables and their contents can be seen in the Variable Explorer tab in the upper right hand corner of Spyder, after the script has completed execution.



Name	Type	Size	Value
a	int	1	10
addition	int	1	280
b	int	1	20
c	int	1	30
name	str	4	good

The other way to execute the Python script is to use the Anaconda prompt, navigate to the directory containing the script and execute it directly from the CLI with:

```
python name-of-script.py
```

A statement is an instruction that a Python interpreter can execute. Python statement ends with the NEWLINE character (no semicolon like other languages such as Java /C#). A Python script consists of a sequence of statements that are executed by the interpreter in the sequential order in which they appear.

There are mainly four types of statements in Python, print statements (which uses the `print` built-in function to output content), Assignment statements, Conditional statements, Looping statements.

Python uses whitespace and indentation to organize the code (indentation will demonstrated later when we study code blocks in the context of `if-else` and `for` constructs). This provides the following advantages compared to other languages:

- Makes it easier to see the beginning or ending of a code block (compared to other languages such as Java or C#)
- Makes the coding style more uniform and provides consistency across a development team

Identifiers are names that identify variables, functions, modules, classes, and other objects in Python. Some common rules for identifiers:

- The name of an identifier needs to begin with a letter or underscore (_). The characters following can be alphanumeric or underscore.
- Identifiers are case-sensitive.
- Identifiers should have meaningful names, for e.g. `salary` instead of `s`
- Multiple words in an identifier can be separated using an underscore, like `salary_for_april`

Python keywords have special meanings and usage. You cannot use Python keywords for naming identifiers.

https://www.programiz.com/python-programming/keyword-list

Python provides three kinds of comments: block (single line) comment, inline comment, and documentation (multi-line) string.

- Single line comment - explains the code that follows it
- inline comment - on the same line as the statement
- Documentation string (docstrings) / multi-line strings - these are used as string literals at the start of functions to document them (to be covered later), however multi line docstrings will work as comments when they appear by themselves. Python doesn't support multiline comments.

Many programming languages have the concept of a constant (a variable whose value cannot change).
https://www.w3schools.com/cs/cs_variables_constants.php

Python **DOES NOT** explicitly support constants, in that ALL variables in Python can have their values changed. To indirectly support constants, developers follow a strong naming convention to distinguish between variables and constants. To tell other programmers that a given value should be treated as a constant, you must write the name in capital letters with underscores separating words

Advantages of using constants
https://realpython.com/python-constants/#why-use-constants
Defining Constants in Python
https://realpython.com/python-constants/#defining-your-own-constants-in-python

5 Variables and data types

File to use: `data-types.py`

Data types specify the type of data that can be stored inside a variable during an assignment statement.

Data type	Description	Example
`int`	To store integer values	`n = 20`
`float`	To store decimal values	`n = 20.75`
`complex`	To store complex numbers (real and imaginary part)	`n = 10+20j`
str	To store textual/string data	`name = 'Jessa'`
bool	To store boolean values	`flag = True`
list	To store a sequence of mutable data	`l = [3, 'a', 2.5]`
tuple	To store sequence immutable data	`t =(2, 'b', 6.4)`
dict	To store key: value pair	`d = {1:'J', 2:'E'}`
set	To store unorder and unindexed values	`s = {1, 3, 5}`
frozenset	To store immutable version of the set	`f_set=frozenset({5,7})`
range	To generate a sequence of number	`numbers = range(10)`

Most of the standard data types found in other languages (int, float, str, bool, etc) are also present in Python.

Boolean data types can only have two possible values: True or False and are typically used in conditional expressions (to be studied later). Related to this data type, we have the concept of truthy and falsy values. This means that non-boolean values can be used in conditional expressions (which normally expect either True or False) based on the idea that a non-boolean value will actually evaluate to True (thereby called a truthy value) or False (thereby called a falsy value).

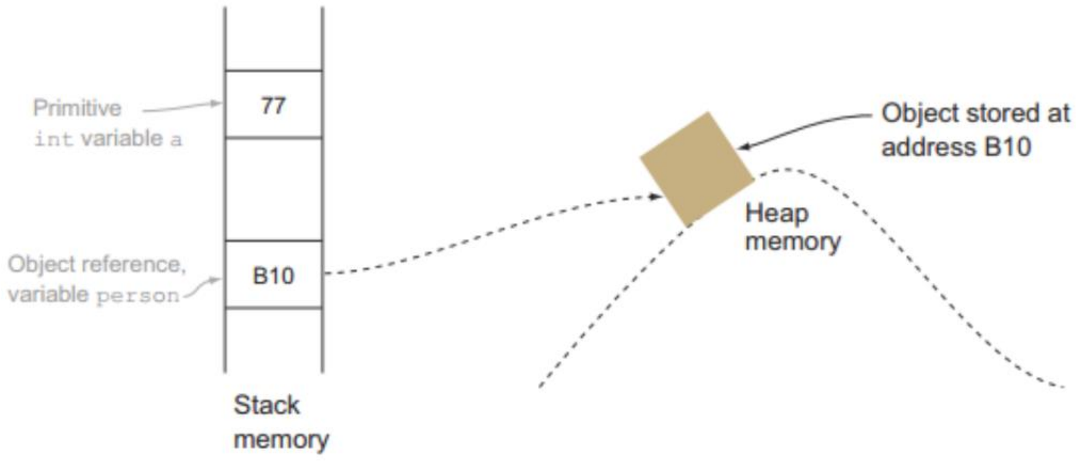
The following are falsy values in Python:

- The number zero (0)
- An empty string ''
- False
- None
- An empty list []
- An empty tuple ()
- An empty dictionary {}

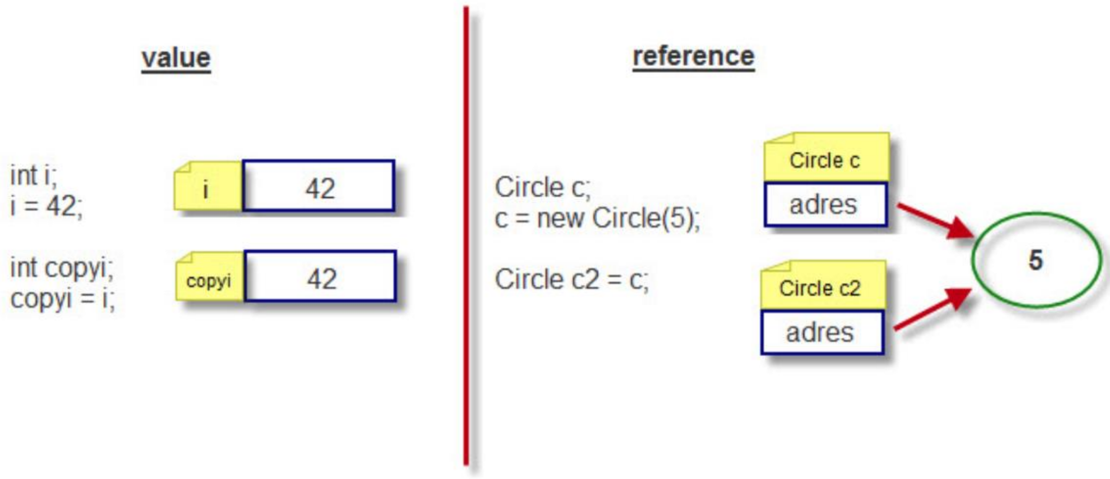
The truthy values are the other values that aren't falsy.

A key distinction between Python and many other languages is that is dynamically typed.

In other languages like C++, C# and Java, there is the concept of **primitive / value data types** (e.g. int, float, double, etc) which are distinct from **reference types** (this is based on the concept of OOP, where you instantiate objects from classes)
https://www.tutorialsteacher.com/csharp/csharp-data-types
https://www.programiz.com/cpp-programming/data-types
https://www.w3schools.com/java/java_data_types.asp


Primitive int variable a
77
Object reference, variable person
B10
Stack memory
Object stored at address B10
Heap memory

Variables of primitive data type hold the actual value, while variables of reference type point to (or reference) another object in memory which holds the actual value



These languages are said to be statically typed and type checking is done at compile time. With static typing, variables generally are not allowed to change types, although mechanisms for casting a variable to a different type may exist.

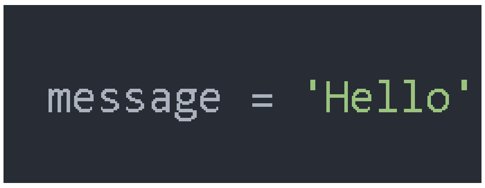
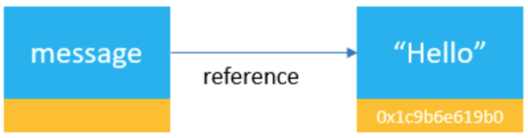
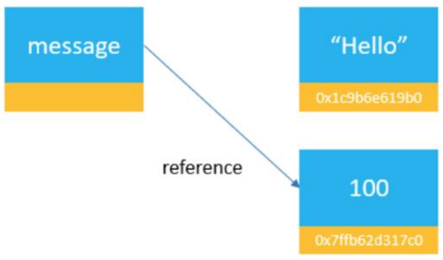
Python is dynamically typed, which means that the Python interpreter does type checking at run time, and that the type of a variable is allowed to change over its lifetime.

Python at its core is a highly object-oriented language, where **EVERYTHING IS AN OBJECT.** All the different data values: numbers, strings, functions, Boolean as well as even classes, and modules are **all objects**.

Variables in Python are all pointers / references to objects in memory which hold the actual data. When you assign a value to a variable in an assignment statement, you are making the variable reference an object that holds this value internally.

In other words, in Python, **all data types are reference data types**. There is **no concept** of **primitive data types** in Python (unlike in C#, C++, Java, etc)

The content (or internal state) of objects are values which do have types, but the variable itself does not have a type. A single variable can be assigned (or made to reference) different objects of different types during the course of its lifetime (or scope) and it will have the type of the object it references. This is the essence of dynamic typing.

 `message = 'Hello'`	`message = 100`
	 message "Hello" 0x1c9b6e619b0 reference 100 0x7ffb62d317c0

Objects in Python can themselves be either mutable (their internal value can change) or immutable (their internal value cannot change). This will be discussed in a later lab.

For more detailed reference on the concept that **EVERYTHING IS AN OBJECT** in Python:

https://jakevdp.github.io/WhirlwindTourOfPython/03-semantics-variables.html#Everything-Is-an-Object
https://realpython.com/python-mutable-vs-immutable-types/#objects-value-identity-and-type

6 String and print basics

File to use: `string-basics.py`

A string is a series of characters. They can be delimited with either single or double quotes.
Strings also include escape sequences (similar to other programming languages).
An escape sequence is a sequence of characters that, when used inside a character or string, does not represent itself but is converted into another character or series of characters that would be considered illegal or difficult to be included directly in a Python string, for e.g. new line or tab.
The complete list of escape sequence characters are shown here:

https://www.scaler.com/topics/escape-sequence-in-python/

Raw strings are used to simplify the construction of strings where many escape sequence characters need to be used, for e.g. regular expressions or directory paths in Windows. They treat the backslash in a string as a literal character, rather than as a start of an escape character.

There are many ways to format a string in order to display the contents of variables together with a string literal in a specific format.

a) "Old Style" String Formatting (% Operator) - similar to printf style in C, still in use in legacy Python code
b) New Style" String Formatting (str.format) - used in Python 3
c) String Interpolation / f-Strings (Python 3.6+)

Python also provides many shortcuts to concatenate string literals and string variables together.

The `print` function (which is the main approach for output to the console) has several optional parameters that can be specified in order to customized output:

- `sep`: used to define the separation among different objects to be printed. By default an empty string("") is used as a separator.
- `end`: used to set the string that is to be printed at the end. The default value for this is set as line feed("").
- `file`: It is an optional parameter used when writing on or over a file. By default, it is set to produce standard output as part of `sys.stdout`.

7 String methods

File to use: `string-methods.py`

As mentioned earlier, everything in Python (including strings) are objects. Objects have functions (methods) that operate on the internal data or state of the objects. The String object has many useful methods that perform a wide variety of useful string operations that are common in data analytics and automation tasks.

https://www.programiz.com/python-programming/methods/string

8 Type conversion

File to use: `type-conversion.py`

Often we need to convert between different data types, particularly when operations or expressions involve values of 2 different data types. The most common case is converting between strings and numbers (and vice versa).

There are two types of type conversion in Python.

- Implicit Conversion - automatic type conversion by the compiler. This occurs when we try to add variables of two different numeric types (float and int), whereby the int type is converted to the float type to avoid loss of data.

- Explicit Conversion (type casting) - manual type conversion specified by the developer. This is necessary when we are trying to operate on two completely different data types (e.g. str and int).

9 Operators, operator precedence and associativity

Operators are special symbols that perform operations on variables and values. The main classes of operators in Python are:

1. Arithmetic operators
2. Assignment Operators
3. Comparison Operators
4. Logical Operators
5. Bitwise Operators
6. Special Operators

https://www.programiz.com/python-programming/operators

When the Python interpreter encounters any expression containing several operations, all operators get evaluated according to an ordered hierarchy, called operator precedence. All the operators, except exponentiation(**) follow the left to right associativity. It means the evaluation will proceed from left to right, while evaluating the expression.

https://www.scaler.com/topics/operator-precedence-in-python/