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

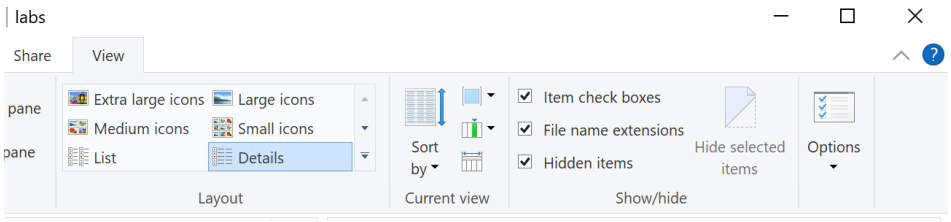
https://forums.feedspot.com/python_forums/

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

2 Lab Setup

You should have received installation instructions on setting up the Python interpreter and 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 Starting the Spyder IDE

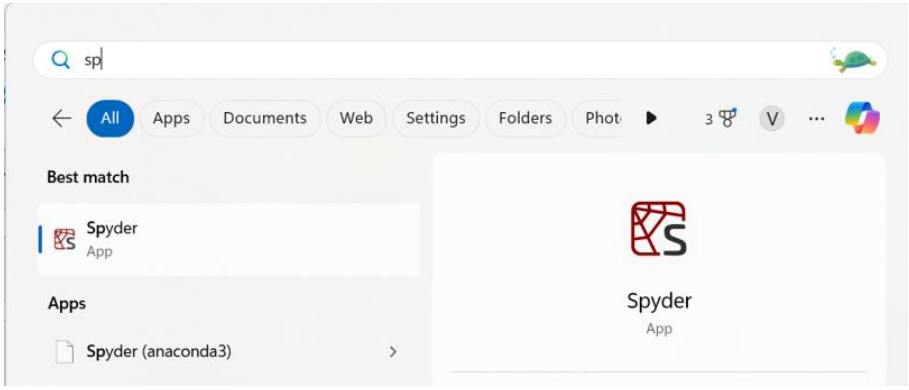
Spyder is a free open source IDE written in Python for the sole purpose of developing Python applications. It was designed by and for scientists, engineers and data analysts and is key component of the Anaconda data science platform. It includes all the features of a standard IDE (such as code editing, analysis, debugging, and profiling) as well as the functionalities related to data science, machine learning and AI (such as data exploration, interactive execution, and visualization capabilities).

There are several ways to start up Spyder, depending on how it was installed.

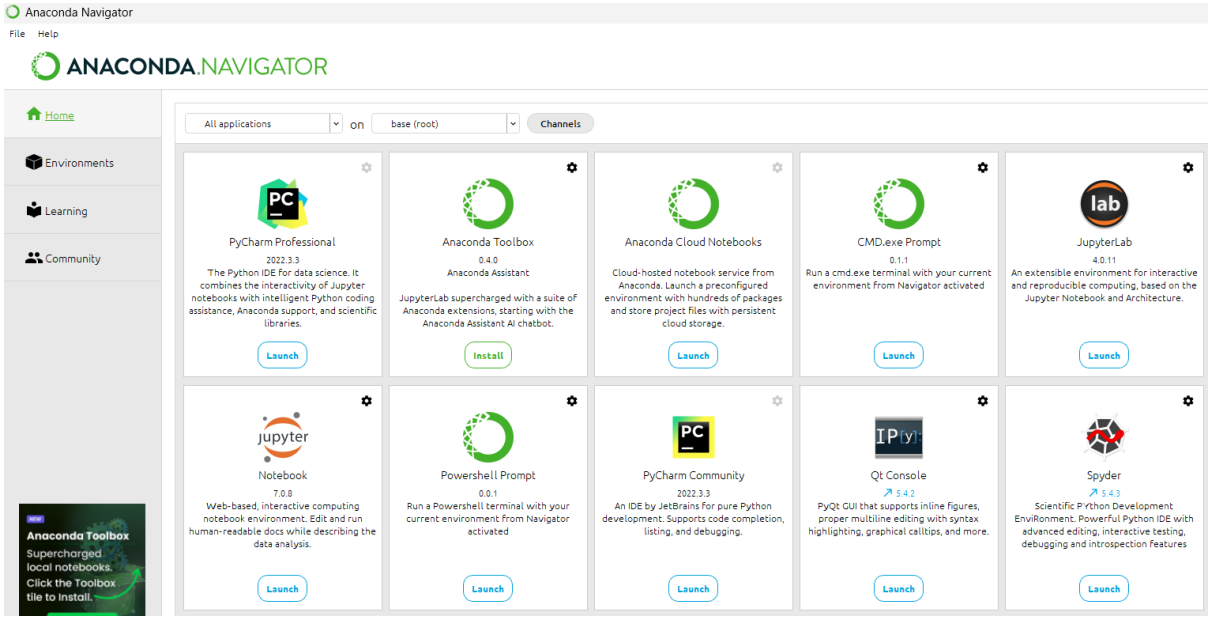
If you installed Spyder using original Python, then you will need to start a DOS command prompt, activate the appropriate environment and then start Spyder:
https://gohyk.bitbucket.io/python_setup/official_python/#starting-spyder-and-jupyter

If you installed Spyder using Conda Forge in Miniforge, then you will need to start the MiniForge prompt, activate the appropriate environment and then start Spyder:
https://gohyk.bitbucket.io/python_setup/conda_forge_python/#starting-spyder-and-jupyter

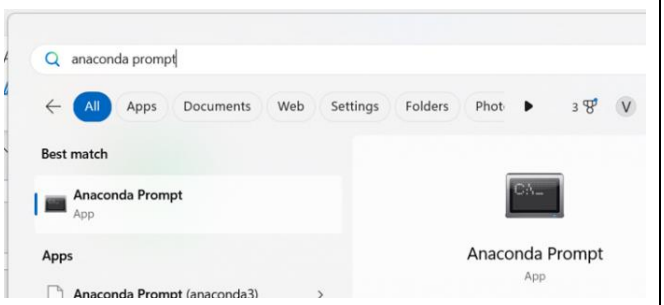
If you installed Spyder via Anaconda, then the easiest way to start it is to search for it directly from the Start Menu.

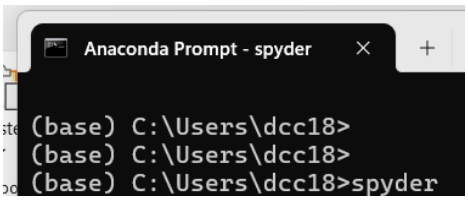


Alternatively, you can search for and start Anaconda Navigator from the Start Menu. From here, you can select Spyder.



Finally, if you are already working directly from the Anaconda Prompt (which is also accessible from the Start Menu), you can start it Spyder by just typing its name into the prompt.

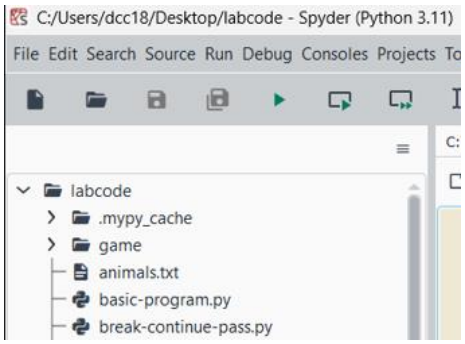




3.1 Viewing Python source code files in Spyder

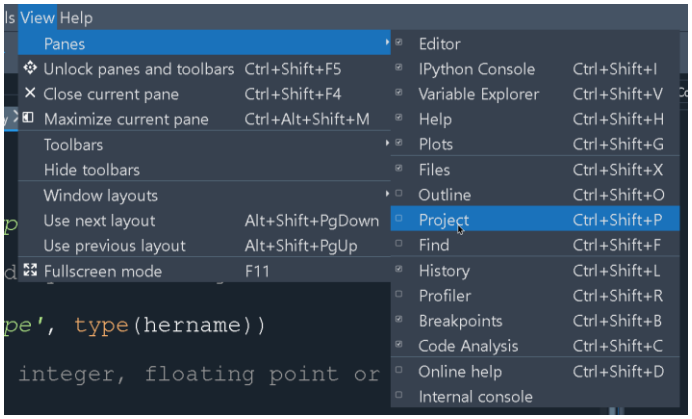
Python programs are text files containing Python source code and are known as either scripts or modules. By convention, those files will use the `.py` extension. On Windows systems the extension can also be `.pyw`.

The sample source code files for this workshop are in the `labcode` folder. You can either open the source code files individually (File -> Open File) from within this folder, or else you can choose to open the `labcode` folder as a single project (Projects -> Open Project), after which you should be able to see all these files in the Project pane of the IDE on the left. You can then simply select and double click on the Python script to open it for editing in the main Editor pane of Spyder.

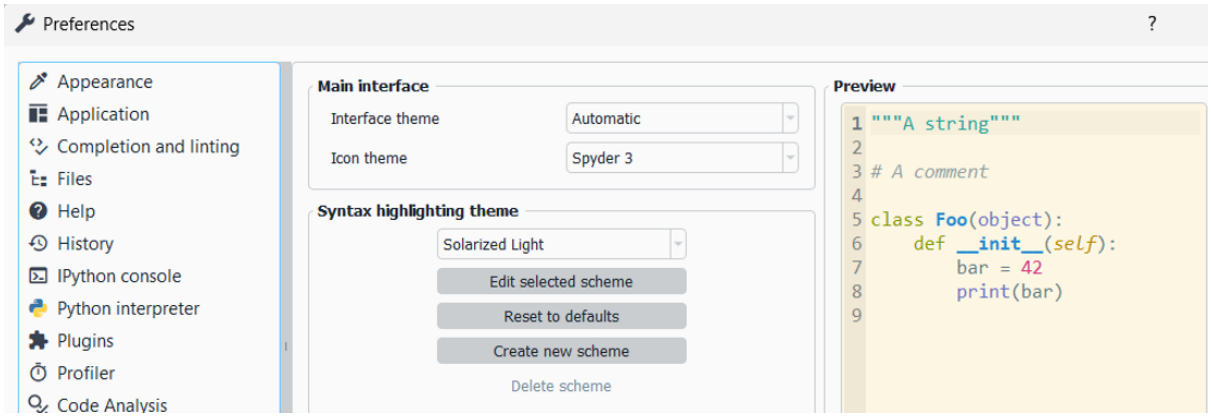


https://docs.spyder-ide.org/current/panes/projects.html

You can toggle the visibility of the Project pane to provide more space for yourself to work with the source code files in the Main Editor Pane.



To facilitate coding in the IDE for the duration of this workshop, you can customize its look-n-feel of by going to Tools -> Preferences -> Appearances, and then choosing a theme for the Main Interface as well Syntax Highlighting. Note that changing the theme will necessitate a restart of Spyder.



4 Basic Python program

File to use: `basic-program.py`

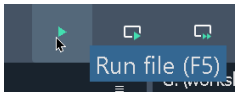
Open this file in the Editor Pane by finding and double clicking on it in the Project Pane.

You can increase / decrease the font size in the source code file using the Ctrl + Scroll wheel on your mouse to facilitate your coding experience.

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

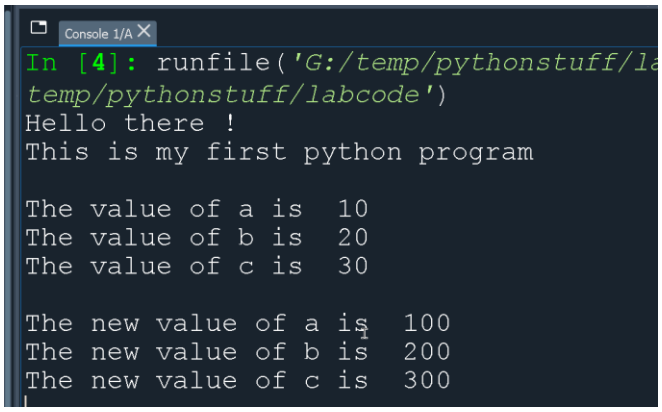
1. Execute code contained in a script / module from the from inside an IDE.
2. Execute code contained in a script / module from the CLI using the interpreter directly
3. Typing Python statements directly into the interactive / standard shell. For the case of the Spyder IDE, the interactive shell is provided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



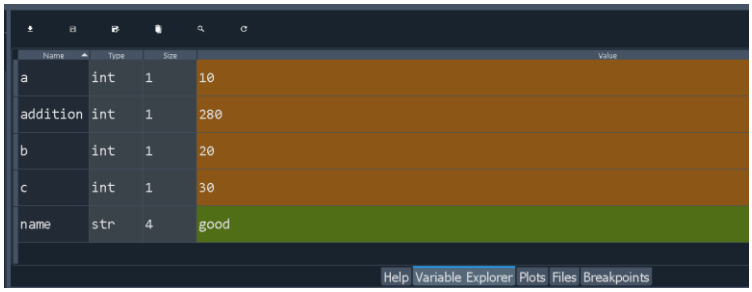
The output from the `print` statements as well as any input required from the user will be shown in the IPython console that is by default located at the lower right hand corner of the IDE.

You can increase / decrease the font size of the text in the IPython console with Ctrl_Shift + or Ctrl_- shortcuts



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.



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:

1. print statements (which uses the `print` built-in function to output content)
2. Assignment statements
3. Conditional statements
4. 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.

You can use operators to evaluate standard mathematical expressions and store their result into a variable or print them out.
https://www.programiz.com/python-programming/operators

Some examples of simple formulas
https://www.idtech.com/blog/important-math-formulas-for-school-students

When you have multiple operators in a mathematical expression, Python will evaluate them in a specific order (termed the operator precedence). To override this default precedence for the case of arithmetic operators, we will frequently use parenthesis.
https://www.upgrad.com/tutorials/software-engineering/python-tutorial/operator-precedence-in-python/

Python provides has an inbuilt library / module (`math`), that has many useful functions to help execute common mathematical expressions. We will be looking at these functions in an upcoming lab.

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

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.

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 is 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.

It is possible to format a string in order to display the contents of variables together with a string literal in a specific format using String Interpolation / f-Strings (which is available from Python 3.6 onwards)

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

7 String methods

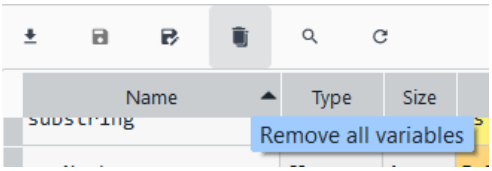
File to use: `string-methods.py`

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

NOTE: The Variable Explorer pane will show all variables registered in the Python environment across successive runs of various Python programs / scripts up to the current point in time. When you run a new Python program / script in Spyder, the variables from the previous script still remain registered in the environment. This can lead to a rather cluttered space in the Variable Explorer pane.

You can choose to remove all the variables registered in the Python environment:



Subsequently, when you run the current script, only the variables used in that script will be registered in the environment and you can then view them in the Variable Explorer.

Note that you can also do save all the registered variables and their current values from the Variable Explorer to a separate file, which can then be later be reloaded to initialize the Python environment.

8 Type conversion and obtaining input from the user

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).

The built-in `input()` function is the most common and straightforward way to get user input from the keyboard. However, this will return the data entered by the user as a string (even if the user enters only numbers). So you will have to ensure that you explicitly cast the input as an numeric type (int or float).

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.upgrad.com/tutorials/software-engineering/python-tutorial/operator-precedence-in-python/

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