Introduction to Programing with Python

Module 01 - Python Basics

# Getting Started

Welcome to this introductory course on programming! In this course, we will explore the fascinating world of programming using the Python programming language. Throughout this course, you will gain a solid understanding of what programming is and develop the skills necessary to become a proficient programmer.

**Python**, our chosen programming language, is an **excellent choice for beginners due to its simplicity and readability**.

Through this course, **we will guide you step by step**, starting **from the basics and gradually building complex applications**.

**Our teaching approach focuses on hands-on learning and practical examples.** We do this through a series of **Demos, Exercises, Labs, and Assignments.**

* **Demo:** An instructor does a walk-through of example code (either live or through a recording.)
* **Exercise:** A minute of reflection to respond to a question in one or two sentences.
* **Lab:** A set of Step-by-step instructions to create a Python program, with an instructor led review (either live or through a recording.)
* **Assignment:** A set of tasks to perform and resources to study

***Important:******Only assignments are submitted and graded.****Grading will happen within a week, but a pre-recorded review of the assignment will be available for self-reflection and learning on the assignment's due date.*

**By the end of this course**, you will have a **solid understanding of programming principles and be able to write Python code** to solve a variety of problems.

# What is Programming?

Programming refers to the process of creating software programs or applications by **writing code using a programming language.** It involves a series of steps that include **designing, coding, testing, and debugging** to develop a functional and reliable software solution.

Here's a breakdown of the typical steps involved in the programming process:

* **Problem Analysis**: Understanding and analyzing the problem or task that the software program needs to solve.
* **Design**: Planning and designing the software solution by breaking down the problem into smaller components, determining the program's structure, and designing algorithms and data structures to implement the desired functionality.
* **Coding**: Writing the actual code using a programming language based on the design.
* **Testing**: Executing and evaluating the program to ensure that it produces the expected results and behaves as intended.
* **Debugging**: The process of locating and fixing errors or bugs in the program. This involves analyzing the code, examining error messages or unexpected behaviors, and making necessary corrections to resolve the issues.
* **Documentation**: Creating documentation that describes the program's purpose, functionality, and usage. This documentation helps other developers understand and work with the code, and it also serves as a reference for future maintenance and updates.
* **Maintenance**: Ongoing support and updates to the program after it has been deployed. This may involve fixing bugs, adding new features, optimizing performance, or adapting the program to changes in the operating environment or requirements.

The programming process **requires logical thinking, problem-solving skills, attention to detail, and a solid understanding of programming concepts and techniques.** It is an iterative and creative process that involves translating ideas and requirements into working software solutions.

# What is a Programming Language?

A programming language is a formal language used to communicate instructions to a computer or a computing device. It provides a **set of rules and syntax for writing software programs**, which are sequences of instructions that specify how the computer should perform a particular task or solve a problem.

**Each programming language has its own syntax**, features, and areas of specialization. Common examples include Python, Java, C++, JavaScript, Ruby, and many others.

# What is Python?

Python is a programming language renowned for its blend of simplicity and power. Here are some important facts:

* It is one of the **easiest** programming languages to learn.
* It has a **vast collection** of pre-built code modules with **ready-to-use code snippets**.
* Python is **cost free**, even for commercial use.
* Python is **compatibility with Windows, Linux/Unix, and Mac OS X**.
* Python code is **easy to read** and understand.
* Python can be **used for web development, data analysis, scientific computing, artificial intelligence, and more.**

***Exercise****: Take a minute or less to write down one or two sentences answering the following question (imagine answering a coworker or interviewer):****What is Python?***

## Python Versions

There are two main versions of Python: 2.x and 3.x. **For this course**, please ensure you install and **use Python 3.x**.

It's important to note **a few facts** about these versions:

1. Both versions, **2.x and 3.x, can coexist** on the same computer.
2. **Older Macs already have Python 2.x** pre-installed.
3. **Version 3.x** of Python offers **improved and advanced features**.

To learn more about the differences between Python 2.x and 3.x, you can visit the Python website or refer to the following link: (<http://wiki.python.org/moin/Python2orPython3>) (external site).

***Important:*** If you choose to install both versions, it's crucial to remember to use the correct version when running your scripts. You can **check the version you are running by using the "-V" switch** in the command terminal of your computer (Figure 1).

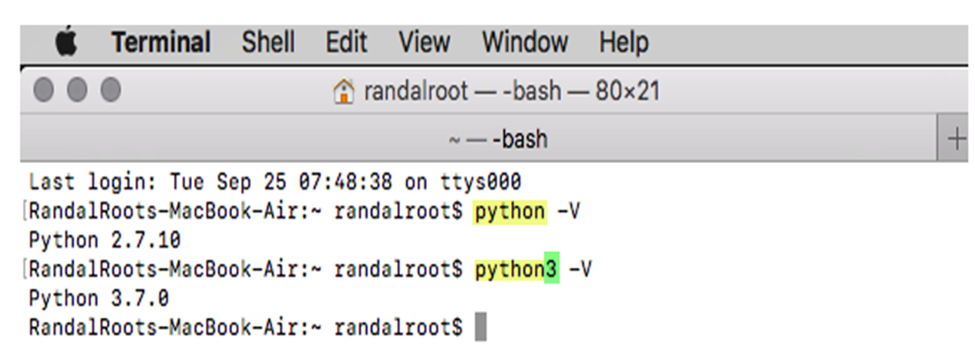


Figure01-Running multiple versions of Python on macOS.

## Installing Python

Installing Python is a straightforward process, but **let's cover the simplest installation option which is using the official installer.**

**For Windows Installation:**

1. **Download** the Python installation program (.exe) from the official Python website's downloads page: <https://www.python.org/downloads/> (external link). The button page should automatically display the correct button for the Windows version of Python.



Figure02-Downloading Python's installation program.

1. **Run** the .exe file on Windows or .dmg file on macOS to start the installation.
2. During the installation process, **ensure** that the "Add Python 3.x to PATH" checkbox is selected. This will include the Python executable in the operating system's PATH ( Figure 3).

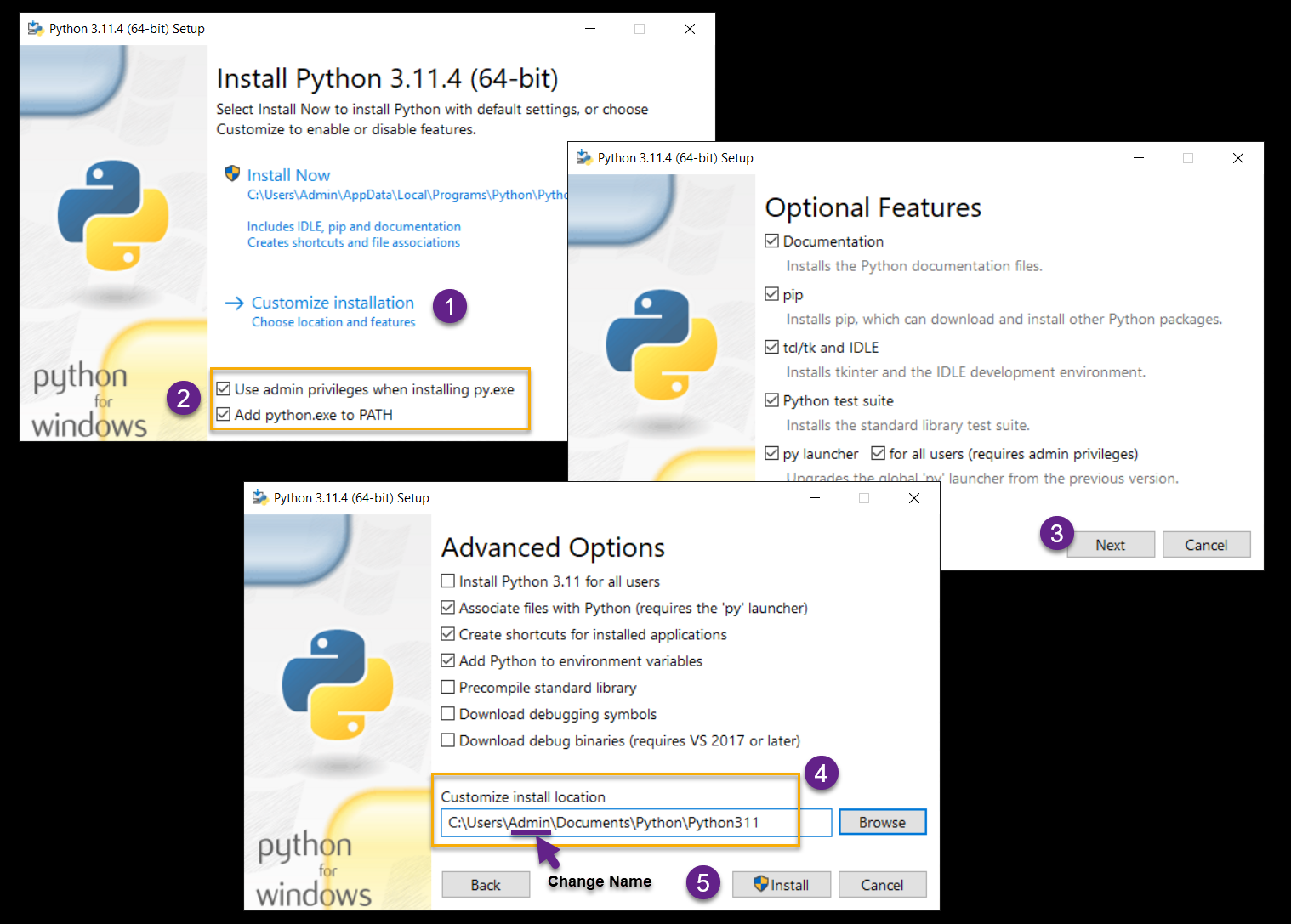


Figure 3-Customizing the Python installation.

1. **Choose** the custom option and change the location **Document/Python**/Python3.x

a) **Change** the path to use your Windows login name (instead of Admin).

b) **Use** the documents folder with a subfolder called Python (See above image).

c) Please **use** a local administrator account or check the "use admin privileges when installing py.exe" box if you have problems installing.

d) ***Do not*** use a work computer. This almost always causes problems due to increased security settings.

**For macOS Installation:**

The simplest way to install Python 3 on a Mac is as follows:

1. **Download** the latest Python version from the Python.org downloads page: <https://www.python.org/downloads/>. The button on the page should automatically display the correct button for the Mac version of Python.
2. **Run** the Python Installer to install Python 3 on your Mac using the default settings.

**Note:** More advanced courses may use "Package Installers" like Visual Studio on Windows or HomeBrew on macOS. However, in this course, we will keep things as basic as possible.

***Exercise****: Take a minute or less to write down one or two sentences answering the following question (imagine answering a coworker or interviewer):****How do you install Python?***

## Testing the Python Installation

Once you have completed the installation of Python, it is **crucial to test your installation** to ensure that everything is functioning correctly.

Here are the steps to test your installation on different operating systems:

1. **Open** a console or terminal window.

a. On the **Windows** OS first **use** the **Windows key + R** keyboard combination to open the \"Run\" dialog window, then type \"**CMD**\" and click the OK button in the \"Run\" dialog box (Figure 4). This will open a command prompt window (Figure 5).

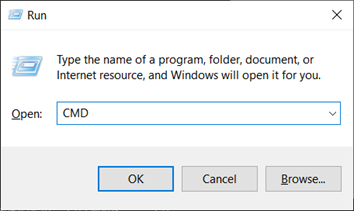


Figure04-The Run dialog window

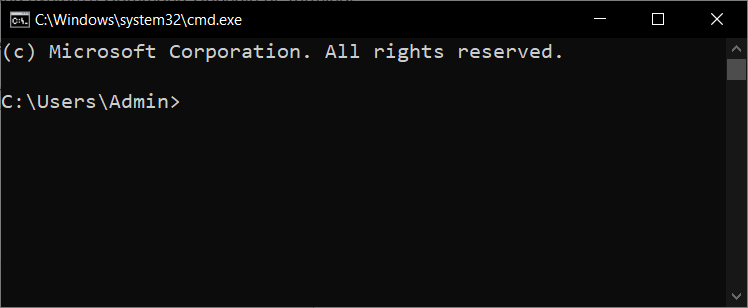


Figure05-The Windows Command Prompt window

b. For **Mac OS** users, **open** Finder, navigate to Applications, then Utilities, and finally click on Terminal.app (Figure 6).

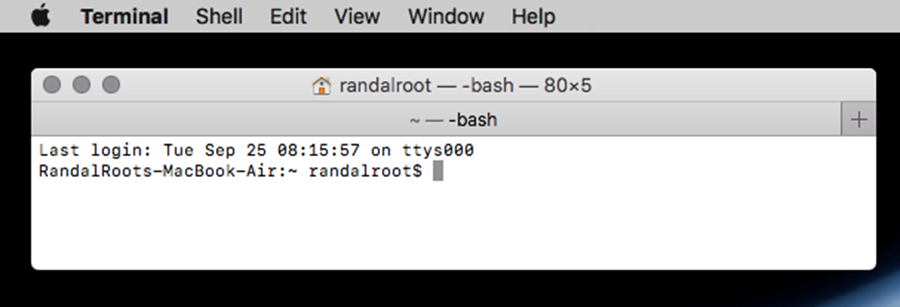


Figure06-A Mac terminal window

1. In both the Command Console or Terminal window, **type** "**python -V**" to ensure that python has been correctly installed.

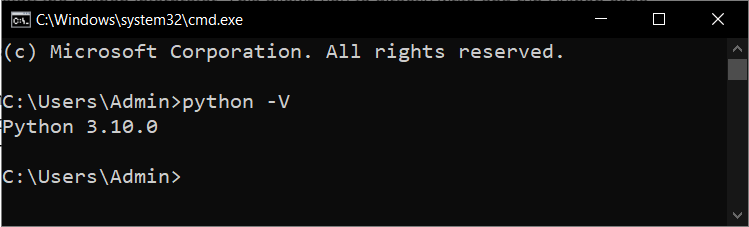


Figure07-Verifying Python has correctly installed

**Important:** Here are some important facts about this command:

* The letter after the dash (-) is **case-sensitive**, so it must be typed as an uppercase "V".
* The **number** you get back in is **based on the version** number of python.
* Any number that is **higher than 3.10 will work** fine in this course.
* If you receive a number that is **lower or starts with a 2**, then you have two choices:

1. **Try** typing in the command as "**python3 -V**". This will work on an Apple computer with an older version of the Mac OS, since they come with version 2.x pre-installed.
2. The installation is not working correctly, so **try uninstalling, rebooting your computer, reinstalling** once more, and try the command again.

**Important:** If on Mac OS using \"python3\" solves the problem, then you must remember to always use \"python3\" instead of \"python\" throughout this course. This will ensure that you are using the correct version of Python to run your programs.

## Demo/Video: Demo01-Installing Python

***Exercise****: Take a minute or less to write down one or two sentences answering the following question (imagine answering a coworker or interviewer):****How do you test that python is installed?***

# The Command Shell

A command shell, also known as a command-line interface (CLI), is a **text-based interface used to interact with an operating system** or execute commands. It provides a way to control and manage a computer system by typing commands into a terminal or command prompt. For example, the CD command changes the shell\'s focus in a different directory and the DIR command shows a list of the files and folders in a directory (another word for folder).

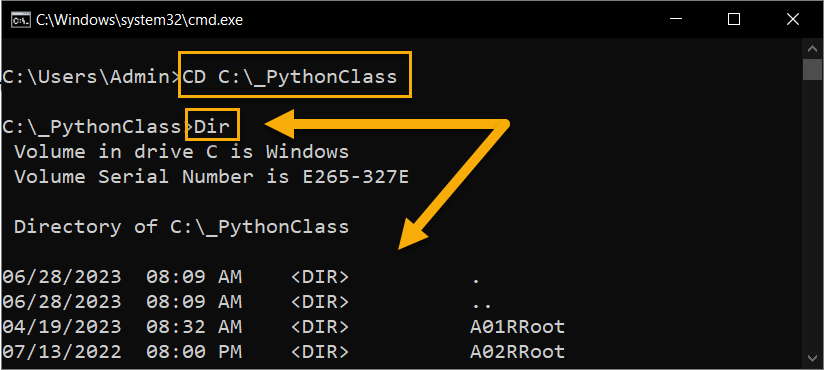


Figure08- Using the Command Shell

**TIP:** On macOS and Linux the "dir" is replaced with "ls" for listing

The command shell allows users to **navigate the file system, run programs, perform administrative tasks, and access various system utilities and tools**. It provides a powerful and efficient way to interact with the underlying operating system.

Here are a few key aspects of a command shell:

* **Prompt**: The command shell displays a prompt, which is typically a symbol or text indicating that it is ready to accept a command. This prompt may include information like the current working directory or username.
* **Commands**: Users can enter commands directly into the command shell. Commands can perform a wide range of tasks, such as creating files or directories, manipulating files, launching programs, configuring system settings, and more. Each command has a specific syntax and set of parameters.
* **File System Navigation**: The command shell allows users to navigate the file system by using commands to change directories, list directory contents, move or copy files, and perform other file-related operations.
* **Command History**: Most command shells maintain a command history, which allows users to access previously executed commands. This feature is helpful for recalling and reusing commands without retyping them.

Command shells **vary depending on the operating system**. For example:

* On **Windows**, the command shell is called **Command Prompt** (cmd.exe) or Windows PowerShell. Command Prompt uses a syntax similar to MS-DOS, while PowerShell offers more advanced scripting capabilities.
* On Unix-like systems (including **Linux and macOS**), the default command shell is the \*\* Terminal\*\* or **Bash** (Bourne Again SHell).

## Common Console Commands

In recent years, there has been **a resurgence of use of console commands** due to several factors:

* Growing popularity of Linux and open-source software
* Cloud computing and containers
* Increased focus on automation and scripting

Here are some common console commands for Windows and macOS you should be aware of:

### Windows Command Prompt (CMD):

**dir:** Lists the files and folders in the current directory.

**cd:** Changes the current directory.

**md:** Creates a new directory.

**cls:** Clears the console screen.

### macOS Terminal:

**ls:** Lists the files and folders in the current directory.

**cd:** Changes the current directory.

**mkdir:** Creates a new directory.

**clear:** Clears the console screen.

**Tip:** These are just a few examples of commonly used commands. **Try asking an AI to "List the common console command for Windows or macOS."**

## Demo/Video: Demo02-Using the Command Shell

[TODO: Make video]

***Exercise****: Take a minute or less to write down one or two sentences answering the following question (imagine answering a coworker or interviewer):****What is the Command Shell?***

# Console Applications

A **console application** is any program that **operates a terminal or command window**. The user interacts with the program by entering commands or providing input through **text-based interfaces.**

These applications are not graphically pleasing, but they do allow you to **accomplish useful tasks on a computer with minimal effort**.

The network interface configuration programs **(IPConfig and IFConfig) are an excellent examples** of a Console application.

To see what it does, type in the command "IPConfig.exe" on the Windows OS or "ifconfig" on the macOS and hit the Enter key to run the application. Here are examples of these programs.

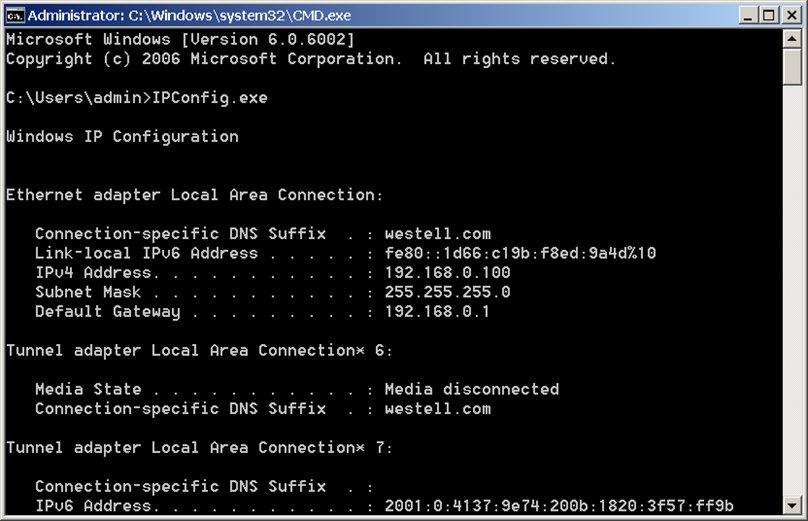


Figure09: Running the IPConfig application on Windows

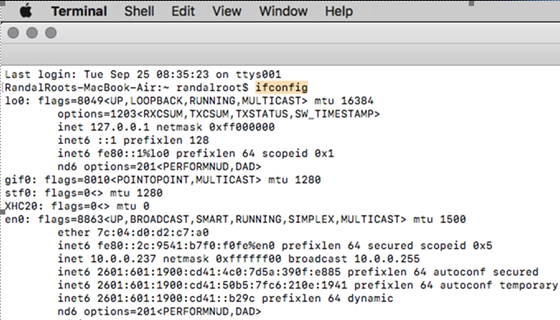


Figure 10: Running the IFConfig application on macOS

**Note:** In this course you will learn to create your own Python console application to perform useful tasks and automations.

# Running Python Programs

**Another example of a console application is python.exe**, which was added to your computer when you installed Python. **This program is responsible for executing Python code.** It is this program that reads Python source code, interprets it, and executes the instructions.

**The Python interpreter can be used** in different ways depending on the operating system:

* **On Windows**, you can open the command prompt and **type python.exe** to start the Python interpreter.
* **On Mac OS and Linux**, you can open the terminal and **type python or python3** to start the Python interpreter.

**Once** the Python interpreter is **running**, you can directly **interact with it by typing Python code line by line**, and the interpreter will execute each line immediately. This allows you to test and experiment with Python code interactively.

In this example, open the Python interpreter application and tell it to print the message "This is a test" (Figure 11).

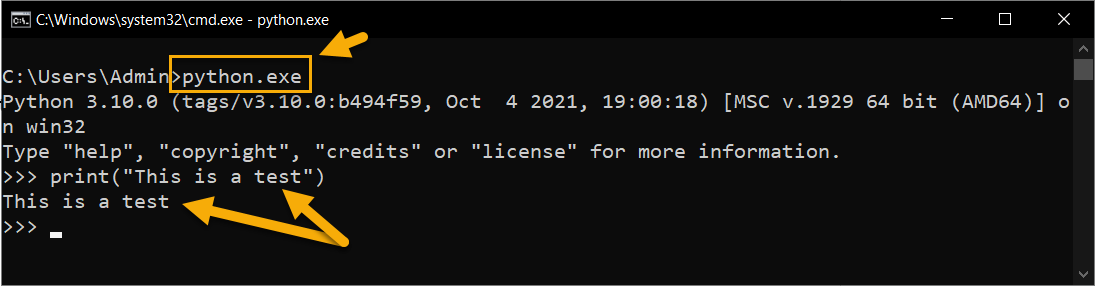


Figure11: Running Python interpreter

The Python interpreter **interprets your code and executes it**. It \*\* also provides access to the standard Python library of modules and any installed third-party libraries\*\*, allowing you to use their code for your programs.

It's worth mentioning that the Python **interpreter can also be used by other applications or integrated into development environments**, allowing for the execution of Python code within those environments.

## Demo/Video: Demo03-The Python Interpreter

***Exercise****: Take a minute or less to write down a one or two sentence answer to the following question as if you were answering a question from a coworker or interviewer:****What is a program language interpreter?***

# Integrated Development Environments

An Integrated Development Environment (or IDE) is **a software application that assists developers in writing, testing, and debugging** software. An IDE combines multiple components into a single integrated interface, making it easier to manage the entire software development process.

## IDLE (Integrated Development and Learning Environment)

IDLE is an Integrated Development Environment for Python programming. It is **included with the standard installation of Python**, starting from version 3.x, and provides a simple and lightweight environment for writing, running, and testing Python code.

Here are some key features of IDLE:

* IDLE includes a **basic code editor with syntax highlighting, indentation, and auto-indentation features**. It provides a clean and straightforward interface for writing Python code.
* IDLE features an **interactive Python shell**, which allows users to **execute Python code and see the results immediately**. It supports executing code line by line or running entire scripts.
* IDLE provides a **debugger to help identify and fix errors** in the code.
* IDLE **integrates Python's built-in documentation system**, making it easy to access documentation for Python modules, functions, classes, and methods.
* IDLE includes features for ***creating, opening, saving, and managing Python script files***.
* IDLE allows users to **run Python scripts directly from the IDE**. It launches a separate Python process to execute the script and displays the output in the shell or console window.
* **IDLE highlights syntax errors in code** and offers basic autocomplete suggestions based on the Python language syntax. It helps catch coding mistakes and improves coding efficiency.

While IDLE does **not offer all the advanced features of other full-featured IDEs like PyCharm or Visual Studio Code.** However, IDLE\'s **simplicity and ease of use make it a good choice for beginners** or for quick prototyping and testing of Python code.

**To launch IDLE, you find it in your system's Start menu or applications list after installing Python**. Simply open IDLE, and you will be presented with a new Python shell window and a code editor window where you can start writing and running Python code.

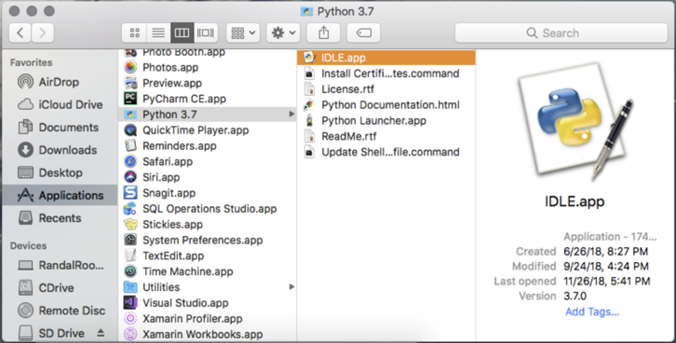


Figure12-Finding IDLE on Mac

**Note**. On Mac, there may be two versions of IDLE installed and make sure to open version 3.x.

Overall, IDLE serves as a beginning and user-friendly environment for learning Python and developing simple Python programs, but **we will learn to use a more advanced IDE later in the course.**

## Demo/Video: Demo04-The IDLE IDE

***Exercise****: Take a minute or less to write down a one or two sentence answer to the following question as if you were answering a question from a coworker or interviewer:****What Python's IDLE application used for?***

**Learn More Here:** [Guide to Setup Python Environment & Understanding Python IDLE (external site)](https://towardsdatascience.com/guide-to-setup-python-environment-understanding-python-idle-7ff3e00941d6)

# Python Script Files

Python is a scripting language designed for executing **scripts**, which are typically **short, simple programs that automate tasks or perform specific functions**.

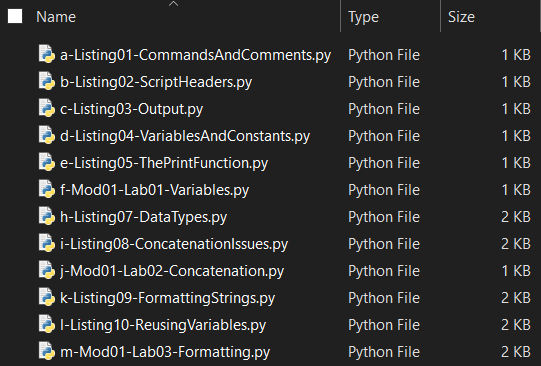


Figure13-A set of script files

Script files are **commonly used to automate repetitive tasks, perform system administration tasks, or execute a series of commands in a specific order.** They provide a **convenient way to bundle a set of instructions** into a single file, which can be executed as a script rather than manually entering each command.

Here are a few key points **about script files**:

* **Scripting Languages**: Script files are written in scripting languages such as Python, Perl, JavaScript, Ruby, PowerShell, and Bash. Each scripting language has its syntax and set of commands or functions.
* **Text-Based**: Script files are plain text files that can be created and edited with any text editor. They typically have a file extension associated with the scripting language used, such as .py for Python, .pl for Perl, .js for JavaScript, or .sh for Bash.
* **Commands and Instructions**: Script files contain a series of commands or instructions that are executed sequentially. These commands can include file operations, variable assignments, conditionals, loops, function calls, and more, depending on the capabilities of the scripting language.
* **Execution**: Script files are executed by an interpreter specific to the scripting language used. The interpreter reads the script file, interprets the instructions, and executes them one by one. To execute a script file, you typically need to provide the name of the script file to the interpreter or run it from a command prompt or terminal.
* **Automation and Task Automation**: Script files are often used for automation purposes. They can automate repetitive tasks, perform system configuration, automate backups, process data, generate reports, and more. By running a script file, you can execute a series of commands without manually entering them each time.
* **Portability**: Script files are portable across different systems and platforms as long as the scripting language and interpreter are available. This allows scripts to be easily shared and run on different computers or operating systems.

# Creating Python Scripts

**Any file that contains Python code is a Python script.** By **executing** a Python script, you \*\* instruct the Python interpreter to read and interpret the code\*\* within the file, executing each command **in sequence**.

These script files **typically have a .py extension**, although technically any extension can be used as long as the file contains valid Python commands.

The .**py extension convention helps users and the operating system easily identify that a file contains Python code.** It enables developers to organize and distinguish Python scripts from other types of files and allows the operating system to associate the file with the correct interpreter for execution.

Script files are often **organized into three main sections:**

* **Header** comments
* **Setup** code
* The **main body**.

Using this **structure helps improve code readability and maintainability**.

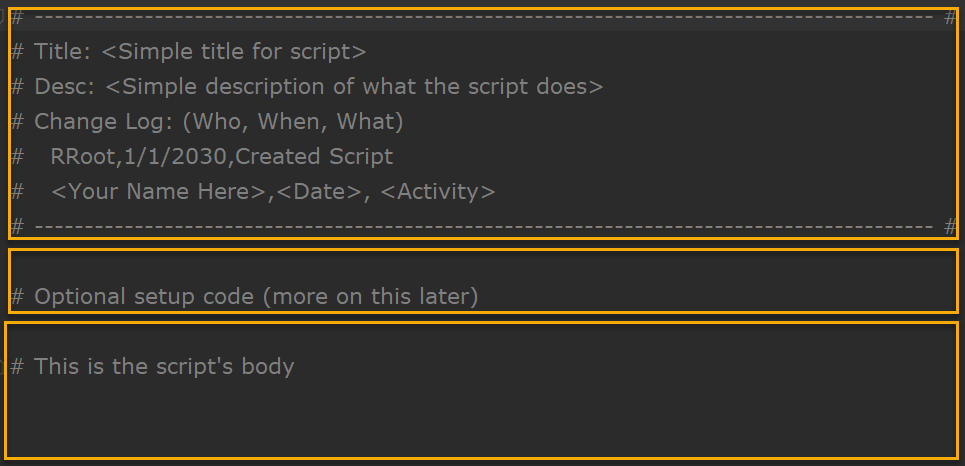


Figure14: A script organized by header setup code and a main body

## Script Headers

**Script files should start with a script header.** The header comments section, also known as the script header or script comment block, is typically placed at the beginning of the script file. It consists of **comments that provide important information about the script, such as its title, description, author, creation date, and change history.** The header comments serve as simple documentation for the script, providing context and details for team members who read or maintain the script.

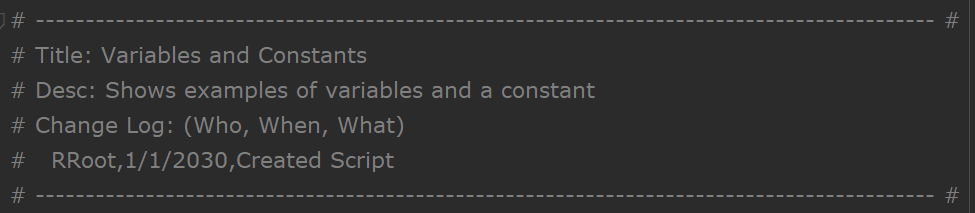


Figure15: An example of a script header

## Setup Code

Following the header comments, the **setup code section contains any initial setup or configuration code required by the script.** This includes importing necessary modules or libraries, defining global variables, setting up connections or resources, and performing any other **preparatory tasks before the main functionality of the script is executed.** The setup code ensures that the script has the necessary environment and resources in place before proceeding to the main body.

The main body is the core section of the script **where the primary logic and functionality are implemented.** It contains the sequence of commands, functions, loops, conditional statements, and any other code necessary to achieve the script\'s intended purpose. The main body executes the primary actions or **computations, processes data, performs calculations, and interacts with users or external systems** as required.

## The Main Body of a Script

The main body of a script refers to the **central portion of the script file where the primary code and instructions are written**. It contains the sequence of commands, functions, statements, and any other code necessary to accomplish the intended purpose of the script.

By **organizing script files** into these three sections, it becomes **easier to understand** the script's purpose, review its documentation, and navigate the codebase. It **promotes modularity, code reuse, and separates the script's specific functionality from the initial setup and metadata.**

It's important to note that the **specific sections and their order may vary depending on coding style or project conventions used in each organization**. However, the general principle of dividing a script into header comments, setup code, and the main body remains a common practice in software development.

# Input and Output

One of the most common tasks in a script is to perform Input or Output. \*\* Input/Output, or I/O for short, refers to the process of transferring data between a program and a destination\*\*.

In Python, **the input() and print() functions are commonly used** for receiving input from the user and displaying output, respectively.

## Input:

The input() function is used to **obtain user input from the console**. It displays a prompt (specified as an argument) and waits for the user to enter a value. The entered value is treated as a string and can be assigned to a variable for further use in the program. Here's an example:

name = input("Please enter your name: ")

print("Hello, " + name + "!") # Prints a personalized greeting

In this example, **the input() function displays the prompt** "Please enter your name: " and waits for the user to type their name. Once the user enters their name and presses "Enter", **the input is stored in the variable name**. The following print() statement then outputs a greeting message using the entered name.

## Output

The print() function is used to **display output in the console**. It takes one or more arguments ( separated by commas) and prints them to the console. Here's an example:

name = "John"

age = 25

print("Name:", name, "Age:", age) # Prints the name and age

These functions are fundamental to the creation of interactive Console applications programs, and you will use them a lot in this course.

# Running Python Scripts

**Once you have created the script you need to test** that it runs as expected. To run a Python script from a command console, you can follow these steps:

1. **Open the command console or terminal** on your operating system. The method to open the console varies depending on your operating system:

* On Windows: Press the Windows key, type "Command Prompt" or "CMD", and press Enter.
* On macOS: Press Command + Spacebar, type "Terminal", and press Enter. On Linux: Press Ctrl + Alt + T, or search for "Terminal" in your "Applications" menu.

1. **Navigate to the directory** where your Python script is located. Use the cd command followed by the directory path to change the current directory. For example, if your script is in the "Documents/PythonClass/Mod01" folder, you can navigate to it using:

cd Documents/PythonClass/Mod01

1. Once you are in the correct directory, you can **run the Python script by typing python** followed by the script's file name. For example, if your script file is named myscript.py, you would run:

python myscript.py

Press Enter to execute the command. The Python interpreter will run your script, and any output or results will be displayed in the command console.

**Note:** **If you have multiple versions of Python installed**, you may need to specify the version you want to use. Instead of python, **you might have to use python3** or python2 followed by the script's file name.

You will get a chance to practice this process in the next lab!

## Demo/Video: Demo05-CreatingScript

***Exercise****: Take a minute or less to write down a one or two sentence answer to the following question as if you were answering a question from a coworker or interviewer:****How do you run a Python program from the command line?***

# Mod01-Lab01-Creating Python Scripts

|  |
| --- |
|  |

In this lab, you use the IDLE integrated development environment to create and execute a simple Python script that displays a test message.

**Important:** This course utilizes standard replacement tokens noted by <some tex>. Whenever you encounter these tokens, please replace them with the appropriate text. For example, the tokens <your name here>,<Date>,Created file would be replaced with Bob Smith,1/1/2000,Created file.

Lab steps:

1. **Create** a folder named **"documents/Python/PythonCourse"** folder

**Important:** This is the folder we will be using for the rest of the course!

1. **Launch** the IDLE Python IDE on your computer.
2. In the IDLE editor, **create** a new Python script file (**File > New File**).
3. **Add** the following code the new file.

# ------------------------------------------------------------------------------------------ #

# Title: Getting Started with IDLE and Python Scripting

# Desc: This script serves as an introduction to IDLE and Python scripting.

# Change Log: (Who, When, What)

# < Your Name Here>, <Date>, Created File

# ------------------------------------------------------------------------------------------ #

# This is the script's body

print("This is a test")

1. **Save** the file by clicking on "File" in the menu bar and selecting "Save" or "Save As" using the name "Mod01-Lab01-CreatingAScript.py" in the **"documents/Python/PythonCourse"** folder.
2. **Go** to the "Run" menu in the IDLE editor and **select** "Run Module" or use the shortcut key F5.
3. **Verify** that the output in the IDLE Python shell displays the message: "This is a test".

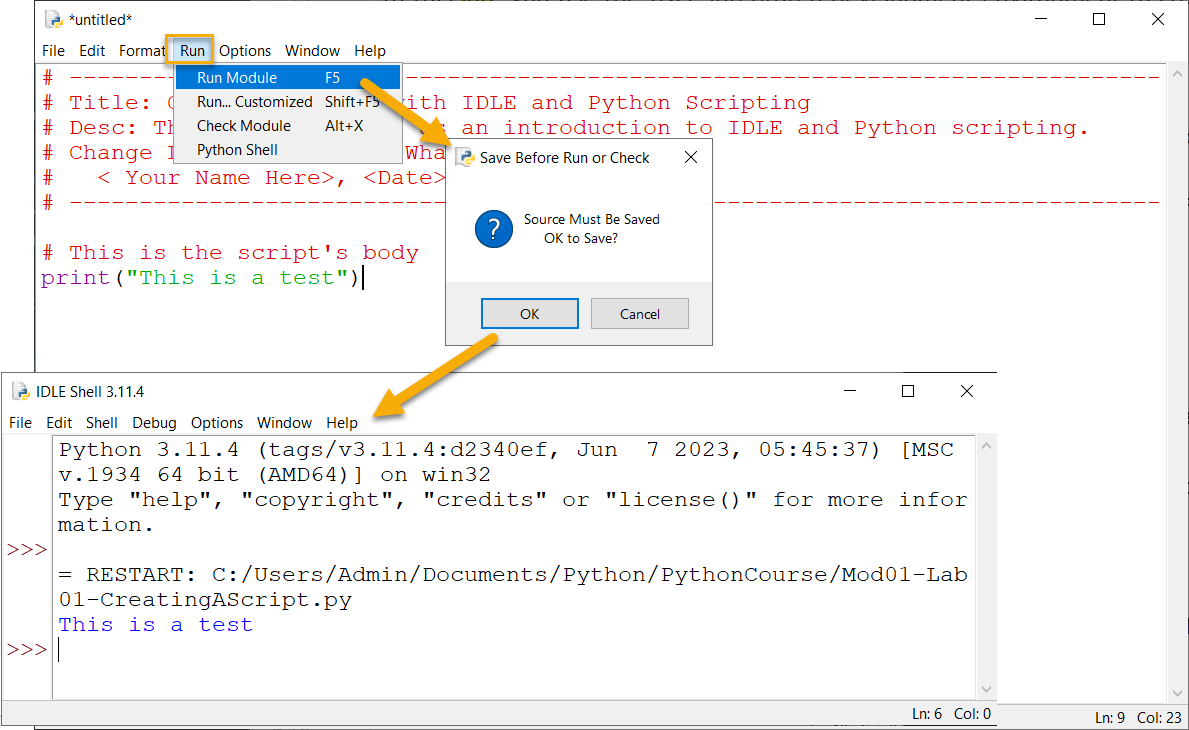


Figure16-The results of Mod01-Lab01 in IDLE.

1. **Open** you command shell (CMD on Windows and Terminal on Mac).
2. **Execute** the script file by typing the command **python (or python3 if needed)** followed by the path and name of the script file. Here are two examples:

Windows

cd C:\Users\admin\Documents\Python\PythonCourse\

python "Mod01-Lab01-CreatingAScript.py"

Mac OS

cd documents/Python/PythonCourse/

python "Mod01-Lab01-CreatingAScript.py"

1. **Verify** that the output in the command shell displays the message: "This is a test".

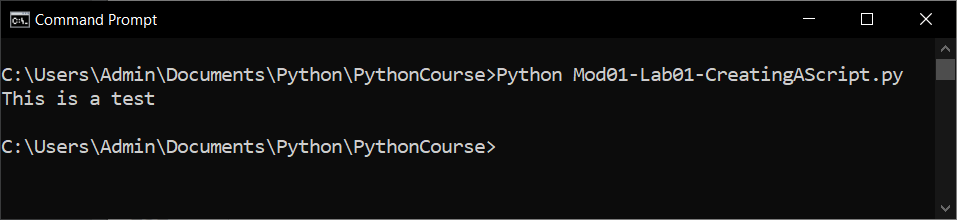


Figure 17: he results of Mod01-Lab01 in the Console

*After completing this lab, please watch the following video review:*

## Demo/Video: Mod01-Lab01-Review

[TODO: Make Video]

In this lab, you used the IDLE integrated development environment to execute a simple Python script. You learned how to create a new script file, code to the file, and run the script within IDLE. The script displayed a test message in the Python shell to verify that the script executed successfully.

|  |
| --- |
|  |

# Programming Basics

**Programming involves instructing a computer on what actions to perform**. These instructions are given through commands known as statements, which carry out operations typically involving data. Data and operations form the **fundamental components** of any program:

* **Data** is the stored information that you wish to manipulate or work with, such as a person's name or phone number.
* **Operations** perform actions on the data, such as printing it or performing calculations.

While data and operations are central to a program, the code can also incorporate additional elements such as comments, directives, and statements.

## Comments

Comments in programming serve as explanatory notes or annotations within the code. They are essential for providing additional information to programmers and anyone who reads the code. This helps in understanding the purpose, logic, and functionality of the code.

In Python, there are **two common types of comments**: line comments and block comments.

* **Line Comments:** Line comments are single-line comments that begin with the hash symbol (#) and continue until the end of the line. They are used for short comments or explanations about a specific line of code.
* **Block Comments:** Block comments, also known as multiline comments, are used to provide more extensive explanations, documentation, or descriptions of larger code blocks. While Python does not have a specific syntax for block comments, it is common practice to use multiline strings enclosed in triple quotes (''' ''') as a workaround.

# Programs are made of code files

# Code files contain comments, statements, and data

# Comments can be line or block comments

# Line comment use a # sign

""" Block comments use triple quotes """

# print() is a statement that displays data

print(1.23)

**Tip:** Many other languages use the symbols **//** to indicate a like comment and **/\* \*/** to indicate a block comment!

Using **meaningful comments** throughout your code helps **improve code readability, maintainability, and collaboration among developers**. It allows others (including yourself) to understand the code\'s intent, making it easier to debug, modify, and maintain the codebase.

## Directives

In Python, there is a **special type of comments or commands called a directive that provides instructions or information to the Python interpreter or other tools**. One common type of directive is the coding declaration directive, which specifies the encoding used in the source code file. It is typically placed at the top of the script file.

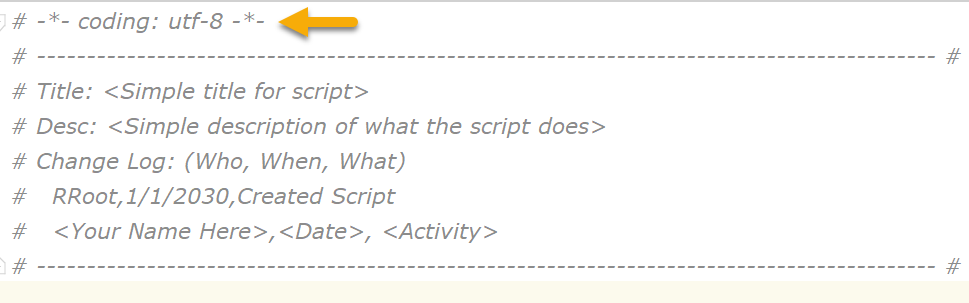


Figure18: The unicode 8 declaration directive

In the example above, **-\*- coding: utf-8 -\*-** is a coding declaration directive. It indicates that the source code file is written using the UTF-8 character encoding. This directive helps ensure that the interpreter correctly interprets characters from different languages or character sets.

**Tip:** We will see more of these later in the course.

## Statements

**A programming statement, also known as a code statement or simply a statement**, is a unit of code that expresses an action or command to be executed by a computer program. Statements are the **fundamental building blocks** of a program and are composed of specific syntax and keywords that conform to the rules of a programming language.

In programming, statements are used to **perform various tasks**, such as assigning values to variables, manipulating data, controlling program flow, making decisions, looping, and calling functions. **Each statement represents a single action or instruction** that the computer should execute.

Here are a few examples of common programming statements in Python:

# Assignment Statements:

x = 5 # This statement assigns the value 5 to the variable x.

# Function Call Statements:

print("Hello, world!") # This statement calls the print() function.

Programming statements **can be simple or complex**, depending on the task at hand. They allow programmers to define the sequence of actions and logic within a program, enabling the computer to perform specific operations and achieve the desired results.

## Demo/Video: Demo06- Commands and Comments

***Exercise****: Take a minute or less to write down a one or two sentence answer to the following question as if you were answering a question from a coworker or interviewer:****What is the difference between a command and a comment?***

## Expressions

In programming, **expressions can be as simple as a single value or variable, or they can be complex combinations involving multiple operators and operands**. Expressions are evaluated by the programming language\'s interpreter or compiler, which follows predefined rules for interpreting the expressions and generating the resulting value.

Here are a few examples of programming expressions in Python:

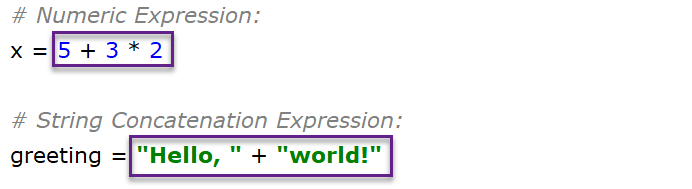


Figure19: An example of two Expressions

It's important to note that expressions differ from statements in that **expressions produce a value, while statements perform an action.**

## Demo/Video: Demo07 - Expressions

***Exercise****: Take a minute or less to write down one or two sentences answering the following question (imagine answering a coworker or interviewer):****What are the basic components of programming?***

## Case-Sensitivity

**Python is a case-sensitive programming language**. This means that Python distinguishes between uppercase and lowercase letters when interpreting code.

In Python, **variables x and X are treated as separate entities due to the case sensitivity** of the language. As a result, x and X are two distinct variables.

So, the in the following code:

x = 4 # This places the value of four into a variable called x.

X = 13 # But, these places the value of four into a variable called X!

print(X) # This displays the value 13 to the user.

PRINT(X) # But, this command is \*\*not\*\* understood by Python.

It's important to note that since Python is case-sensitive, the **PRINT(X) command in this code will result in an error** because the print() function must be written in lowercase letters.

Make sure to use consistent capitalization and adhere to the case sensitivity of Python, so you can ensure that variables and functions are referenced correctly throughout your code.

# Python Coding Standards

Python coding standards, also known as **Python style guides or PEP 8 (Python Enhancement Proposal 8)**, are a set of guidelines and recommendations for writing Python code that promotes consistency, readability, and maintainability. These **standards help improve code quality, enhance collaboration among developers, and make code easier to understand and maintain over time.**

Here are some key principles and guidelines of Python coding standards:

* **Indentation and Line Length**: Python code should be indented using 4 spaces per indentation level. This helps improve code readability. Each line should preferably be **no longer than 79 characters**, and if necessary, line continuation should be used.
* **Naming Conventions**: **Variable, function, and module** names should be written in lowercase, with words separated by underscores (**snake\_case**). Class names should follow the CapWords convention (TitleCase). **Constants** should be written in **uppercase**. **Descriptive and meaningful names** are encouraged.
* **Whitespace and Blank Lines**: Use **whitespace around operators and after commas** to improve readability. Leave **blank lines between functions and classes**, as well as within functions to separate logical blocks of code.
* **Comments and Documentation**: Include comments to explain complex or non-obvious parts of the code. Use docstrings to provide documentation for modules, classes, and functions. **Comments and docstrings should be clear, concise, and meaningful.**

These are **just a few key aspects of Python coding standards**. Adhering to these standards helps create consistent and readable code that is easier to understand, maintain, and collaborate on.

**Notes:**

* **We will talk more about each of these aspects later in the course.**
* You can find more detailed information about Python coding standards in PEP 8 (<https://pep8.org/>) and various style guide resources available online. Additionally, there are tools like linters and formatters that can automatically check and enforce coding standards for Python code.

# Program Data

Your program's data exists in one of **two states:** **stored on a drive or loaded into memory.**

**Data stored on a drive still must load into memory before you can use it**. All data must be read or modified while in memory, even simple applications like Notepad or TextEdit do this.

**When the program has completed using the data in memory, it must be saved back as stored data**, or the data is lost when the program ends, or the computer shuts down.

**Note that not all program data must be saved.** For example, often you ask a user to supply temporary values as they are using your program. **A Calculator is an example** of this kind of program.

Whenever you are loading data into memory, either from stored data or from the collected user input, **you need to tell the computer what kind of data is being loaded**. This does two things:

1. It **allows the computer to reserve enough memory space** to hold the data you are going to load
2. It allows the computer to **restrict certain types of data from that space**.

For example, **if you create a space in memory for an integer**, the program tells the computer to set aside typically 4 bytes of memory and **only allow whole numbers within that memory space, and no characters**. If you later input characters or non-whole numbers to that memory space, the computer reports an error or possibly ignores the values after the decimal point.

## Constants, Variables, and Data Types

Constants and Variables are **both tools for holding data in memory**. In most languages, you need to tell the computer what type of data you wish to store, called **declaring the "data type."** In **Python, "data typing" is done automatically**.

### Variables

In Python, you can create a variable by **assigning a value to a name using the assignment operator (=)**. You can make up just about any name you want, but on the internet, you will see many demos using a few letters.

Here is an example:

x = 5

In the above code, the **variable x is assigned the value 5**. After it has been created you can use variable x the program to perform calculations, make decisions, or display the value.

Python uses the **=** symbol to **assign values to both variables and constants**. One thing every programmer needs to know is that items **on the left of the = symbol** always receive that which is on the right.

**Tip:** It may seem simple if you have some programming experience, but **many beginners are confused by this**. Perhaps, many of use saw our childhood teachers write out equations as follows:

4 + 5 = 9 # This code will \*\*not\*\* work in programming!

It was not until later that a teacher might write the same equation like this:

9 = 4 + 5

By the time we saw the second example, we had seen the first example for years. However, **the second example**, "9 = 4 + 5", **is the correct way** to write the equation in programming. Perhaps this is why some people struggle with the fact that **programming code written like this next example, always writes out the value of x as 10 and never 5.**

x = 5

y = 10

x = y # The variable x is set to the same value as y

print(x)

### Constants

Programmers sometimes **restrict some of their program's data from changes once its value is set**. While these look like variables, these are known as Constants.

You still set the value of a constant at the same time you create the constant, but **once your program starts running, the value remains the same if the program is running, which is different from a Variable, whose value can "vary."**

It is a common programming convention to name your constants using all uppercase letters:

PI = 3.1416

In the above code, PI is used to represent the mathematical constant pi. By convention, it is written in **uppercase letters to indicate that it is intended to be a constant and should not be modified**.

Most **other languages use keywords line "const" to keep constants from changing** values while the program is running. For instance, here is an **example using the C# language**.

SIMPLEPI = 3.1416 // This C# code is in error since SIMPLEPI is a constant

In **Python, there is no specific keyword for defining constants**. **Instead**, it is a convention to **use uppercase letters** and underscores to denote constants. Although **Python does not restrict** the changes to constants, it is **up to the developers to not modify their values** to maintain the intended meaning of constants.

## Demo/Video: Demo08 - Variables and Constants.py

***Exercise****: Take a minute or less to write down a one or two sentence answer to the following question as if you were answering a question from a coworker or interviewer:****What is the difference between a constant and a variable?***

# Mod01-Lab02-variables

|  |
| --- |
|  |

In this lab, You will create a program that displays a student's full name. Your task is to write a program that creates two variables, first\_name and last\_name, to store a student's first name and last name respectively. Then, use the print() function to display the student's full name.

Follow these steps to complete the lab:

1. **Open** IDLE, the Python Integrated Development Environment.
2. **Create** a new file by clicking on "File" in the menu bar and selecting "New File".
3. **Create** a variable named first\_name and assign it a string value representing the student's first name of "Vic".
4. **Create** a variable named last\_name and assign it a string value representing the student's last name of "Vu".
5. **Use** the print() function to display the student's full name by printing the values of the first\_name and last\_name variables separated by a space.
6. **Save** the file by clicking on "File" in the menu bar and selecting "Save" or " Save As" using the name "Mod01-Lab02-Variables.py" Save the file in your Documents/Python/PythonCourse folder.
7. **Run** the code. Click on "Run" in the menu bar and select "Run Module" or press the F5 key.
8. **Verify** the output displays the student's full name.

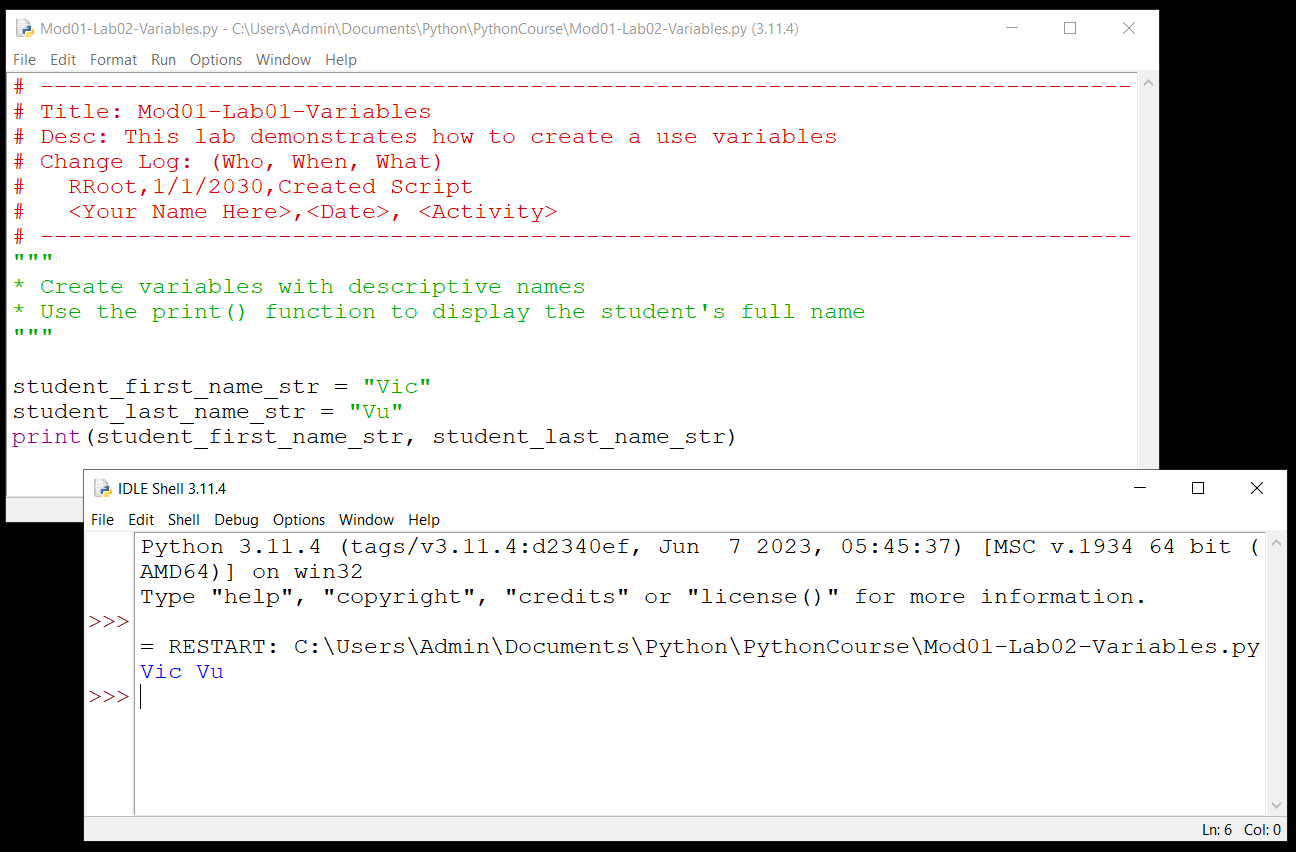


Figure22: The result of Mod01-Lab02

1. **Open** you command shell (CMD on Windows and Terminal on Mac).
2. **Execute** the script file by typing the command **python (or python3 if needed)** followed by the path and name of the script file. Here are two examples:

Windows

cd C:\Users\admin\Documents\Python\PythonCourse\

python "Mod01-Lab02-RRoot.py"

Mac OS

cd documents/Python/PythonCourse/

python "Mod01-Lab02-RRoot.py"

1. **Verify** that the output in the command shell displays the message: "This is a test".

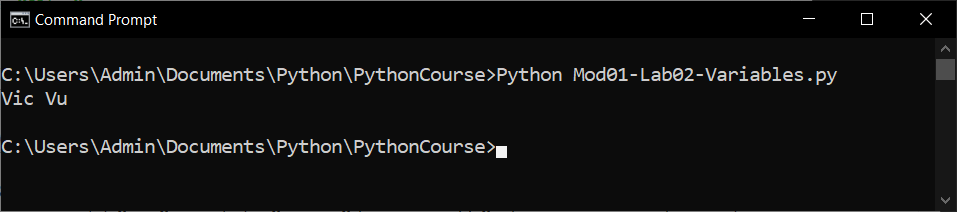


Figure 23: The results in the command shell of Mod01-Lab02

After completing this lab, please watch the following video review:

## Demo/Video: Mod01-Lab02-Review

In this lab, you worked with variables, the print() function, and basic file management in IDLE.

|  |
| --- |
|  |

# Data Types

Computers do not automatically deal with subtle distinctions when it comes to data as humans do. We see a number and think that it is very different from a name. We see a picture and think that it is different from a collection of numbers. However, computers see all of these as ones and zeros.

Still, it is **often essential to force a computer to see these distinctions**. That way, you can receive an error message from the computer when a program tries to set incompatible values to your variables or constants. **To make a computer understand the distinction between numbers, names, dates, etc., you formally tell the computer the difference. This is known as a data type.**

**A data's "type" is a description of the data**. In other words, a type **defines what it can and cannot allow as values**. For example, you can indicate you want to store only numbers in one variable and character data in another.

**Variable and constant data exist at a defined location in a computer's memory.** In almost all languages nowadays, the location is dynamically assigned. **In addition to the value** of the variable and constant, there **is information about its name, type, and size.**

**Python** is a dynamically-typed language, which means that variables can hold values of different data types, and the **data type is determined** **automatically** based on the assigned value.

**Common data types in Python** include:

* **Numeric Types:** Integers (int), floating-point numbers (float), and complex numbers (complex).
* **Strings:** Sequences of individual characters (str).
* **Boolean:** Represents either True or False (bool).
* **none:** Represents a empty variable that will be filled with data later in the program.

**Tip:** Python also supports advanced data types, and we will talk about these much later in the course.

## The Type() Function

The **type() function is a built-in function** in Python that returns the class of the data. It allows you to determine the data type of variable or value.

Here are four examples:

data = 5

print(type(data))

data = 5.8

print(type(data))

data = "5"

print(type(data))

data = True

print(type(data))

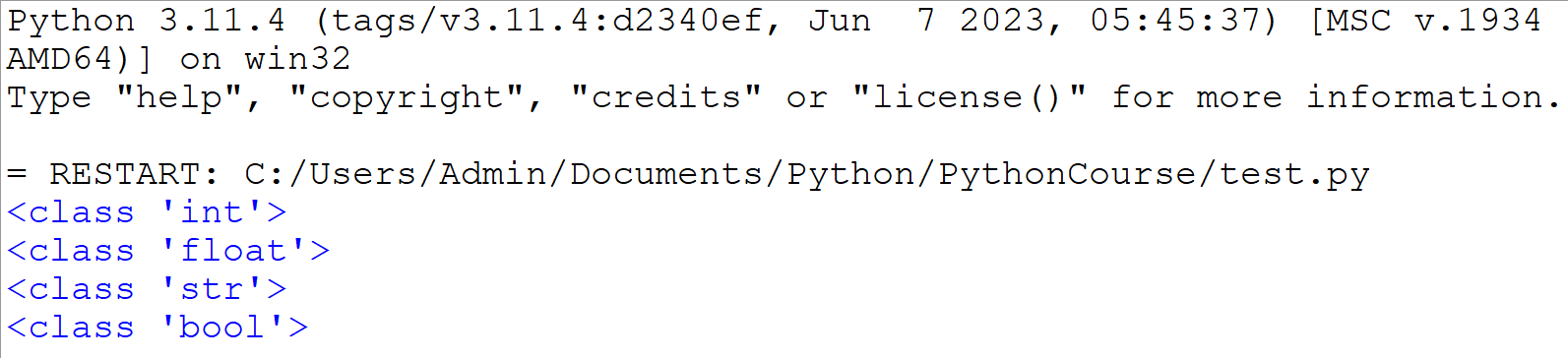


Figure 22: Displaying data types.

In each example, the type() function is used to determine the type of data stored in the variable.

## Data Type Naming Conventions

In programing, it is common simply note what data type is intended to go into a variable. This is usually done by adding a prefix or suffix to the variables name. Here are two examples:

message\_str = "This variable has a \_str suffix"

strMessage = "This variable has a str prefix"

strMessage = "This is Camel Casing (one or more humps in the middle")

All styles are acceptable, but python code usually uses the underscore suffix, while C style languages (C, C++, Java, C#) use the prefix.

Common data type notations include:

* **str:** String
* **int:** Integer
* **flt:** Floating Point number
* **bool:** Boolean
* **obj:** Generic for any type of object.

Here are some examples:

student\_first\_name\_str = "Vic"

student\_last\_name\_str = "Vu"

student\_gpa\_flt = 3.9

student\_passed\_bool = True

**Important:** The **code samples you see on the internet** may not include either suffix or prefix, but these are **not usually intended to be production ready code**. While it is true that modern IDE will show you a variable's data type, using simple type notation makes you code easier to read and less prone to errors. **In this course, please use the underscore suffix option.**

### Type Hints

Python type hints are a way to annotate the types of variables, function parameters, and return values in your code to indicate the expected data types. Like including data type naming conventions, type hints are especially helpful in larger projects, where maintaining and understanding code becomes more complex. Here is an example of a Python function that uses type hints to identify what type of data should be passed to the parameters and what type of data the function will return.

student\_first\_name: str = "Vic"

student\_last\_name: str = "Vu"

student\_gpa: flt = 3.9

student\_passed: bool = True

However, it is important to note that **type hints are not enforced by the Python interpreter** itself but are primarily used by IDEs (Integrated Development Environments) to provide better code understanding, catch potential type-related errors, and improve code readability.

**NOTE:** Developers will often **use prefix or suffix notation for variables and constants**, but will use **type hints for function parameters**. That is the convention we will follow in this course.

## Demo/Video: Demo09 - Data Types

# Summary

In this module, you started you journey in learning programming and Python. You learned what Python is, how it is used, and how it is installed. You learned how to use a command console to verify that Python was installed and working on your computer. You learned how to program and run a console application.

You learned that Python can be used interactively and with a script. You learned how to create and run Python scripts using both an IDE and the command console.

Throughout the module, you have acquired skills such as creating and running Python scripts using both an Integrated Development Environment (IDE) and the command console. Additionally, you were introduced to programming language styles and organizational principles.

You also learned some basics of data types and explored concepts such as concatenation.

At this point, you should try and answer as many of the following questions as you can from memory, then review the subjects that you cannot. Imagine being asked these questions by a co-worker or in an interview to connect this learning with aspects of your life.

* What is python?
* How do you install python?
* How do you test that python is installed?
* What is the command shell?
* Why does Python work the same on all operating systems?
* What are some differences between python and operating system shell commands?
* What is a python script?
* What is the "Main" body of a script?
* What is a Script Header?
* How do you run a python script?
* How can you display information to the user?
* What is a data type?
* What is a type hint?

When you can answer all of these from memory, it is time to complete the module's assignment and move on to the next module.