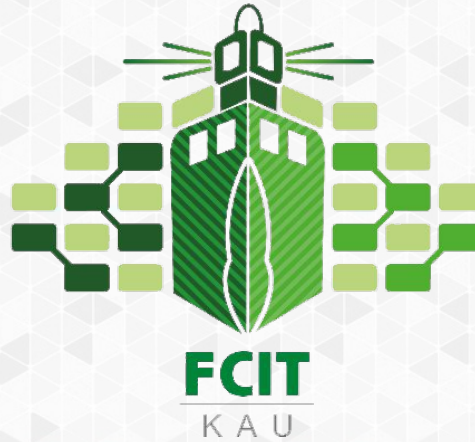FACULTY OF COMPUTING
& INFORMATION TECHNOLOGY

KING ABDULAZIZ UNIVERSITY

FCIT
K A U

كلية الحاسبات
وتقنية المعلومات

جامعة الملك عبدالعزيز

# Chapter 1
# Introduction to Computers, Programs, and Python

## CPIT 110 (Problem-Solving and Programming)

Introduction to Programming Using Python, By: Y. Daniel Liang

# Sections

# Programs

- Program 1: Welcome with Two Messages

- Program 2: Welcome With Three Messages

- Program 3: Compute an Expression

# Check Points

- Simple Examples
  - #1
  - #2

- Section 1.8
  - #3

# Objectives

- To understand computer basics, programs, and operating systems (1.2-1.4).

- To write and run a simple Python program (1.5).

- To explain the basic syntax of a Python program (1.5).

- To describe the history of Python (1.6).

- To explain the importance of, and provide examples of, proper programming style and documentation (1.7).

- To explain the differences between syntax errors, runtime errors, and logic errors (1.8).

# Textbook

Introduction to Programming Using Python, By: Y. Daniel Liang
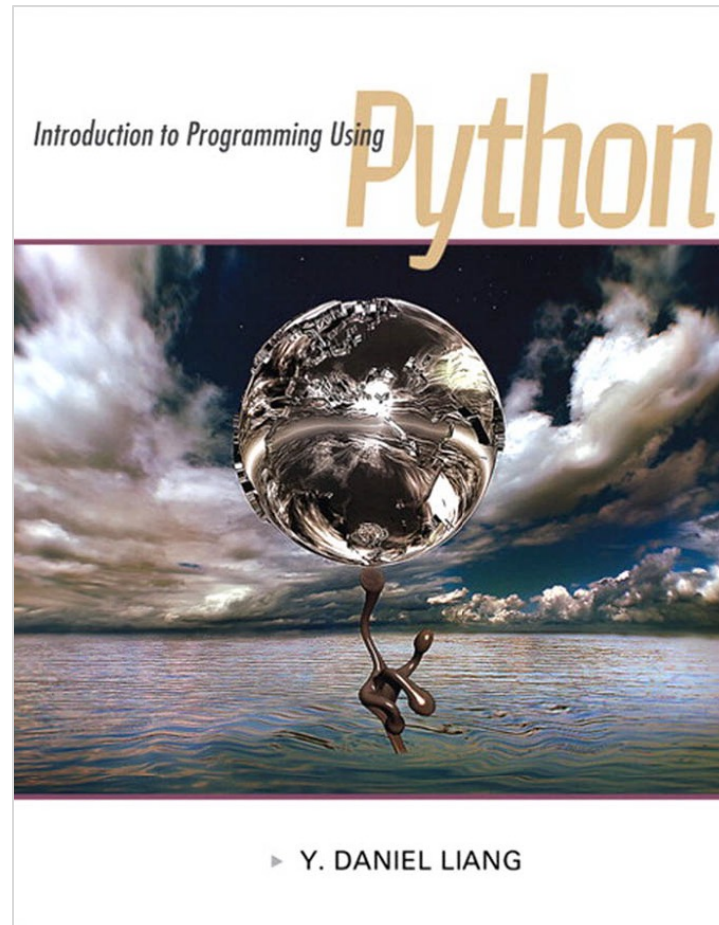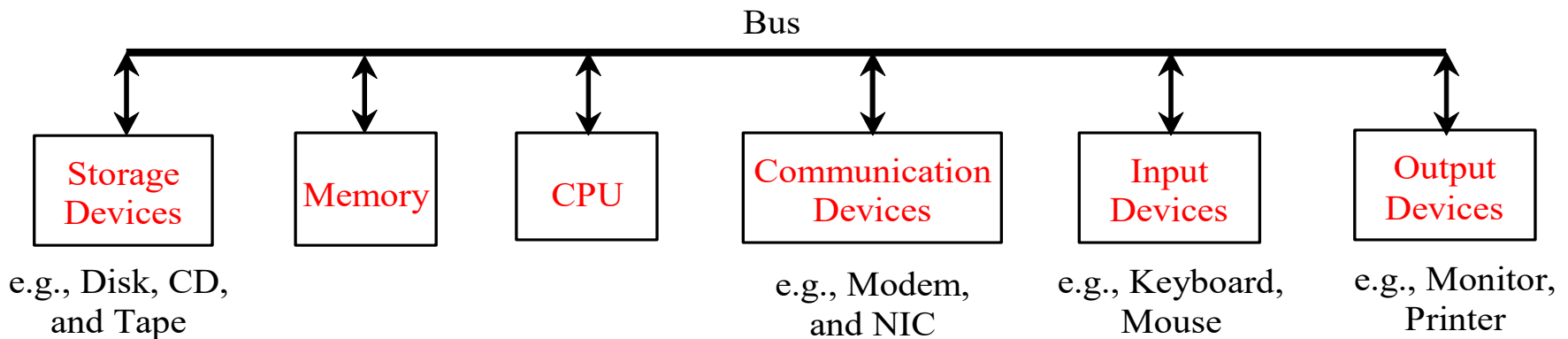
# 1.2. What is a Computer?

- CPU

- Memory

- Storage Devices
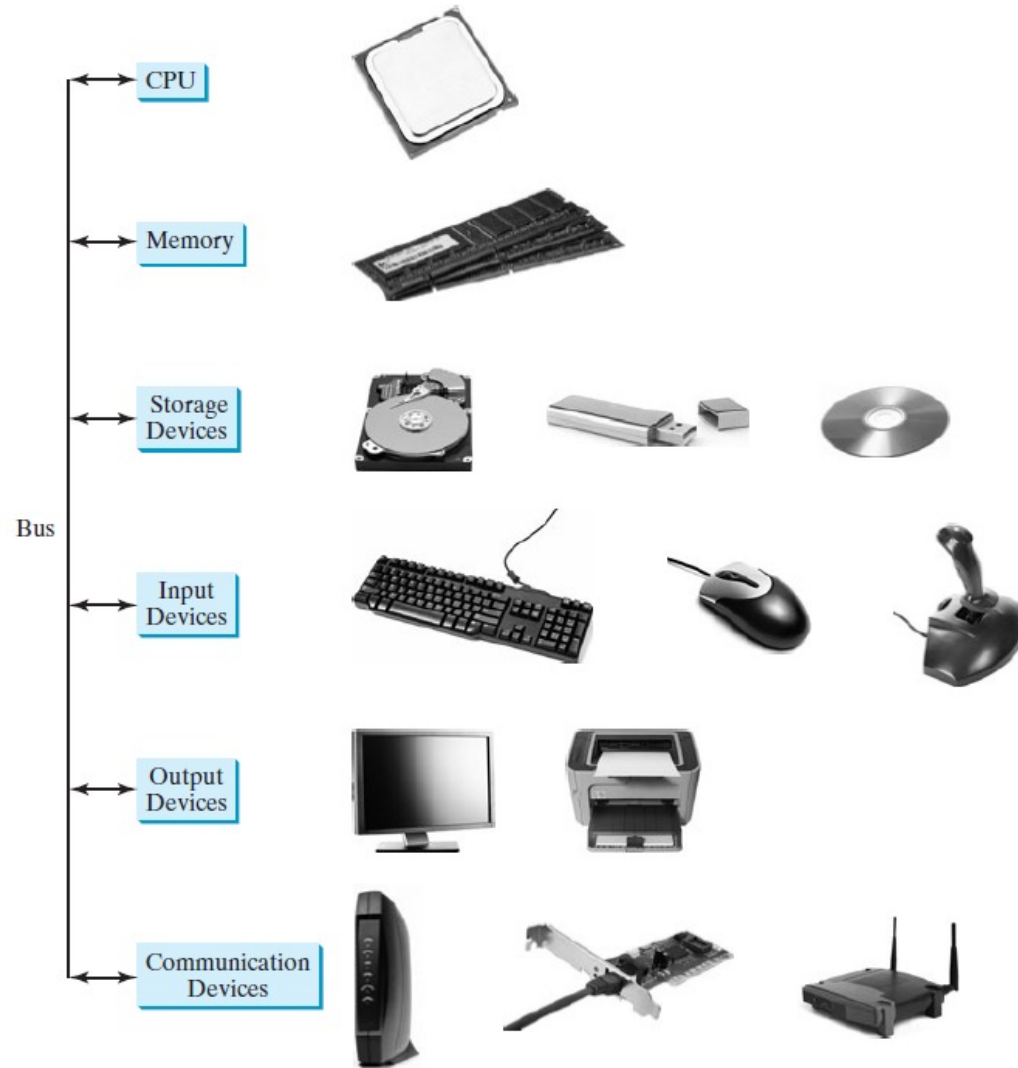
- Output Devices

- Input Devices

# What is a Computer?

- A computer is an electronic device that stores and processes data.

- A computer includes both hardware and software.

- In general, hardware comprises the visible, physical elements of the computer, and software provides the invisible instructions that control the hardware and make it perform specific tasks.

- A computer consists of: CPU, memory, storage devices (such as disks and CDs), input devices (such as the mouse and keyboard), output devices (such as monitors and printers), and communication devices (such as modems and network interface cards).

# What is a Computer?

- A computer's components are interconnected by a subsystem called a bus.

- You can think of a bus as a sort of system of roads running among the computer's components; data and power travel along the bus from one part of the computer to another.

- In personal computers, the bus is built into the computer's motherboard, which is a circuit case that connects all of the parts of a computer together.

Bus
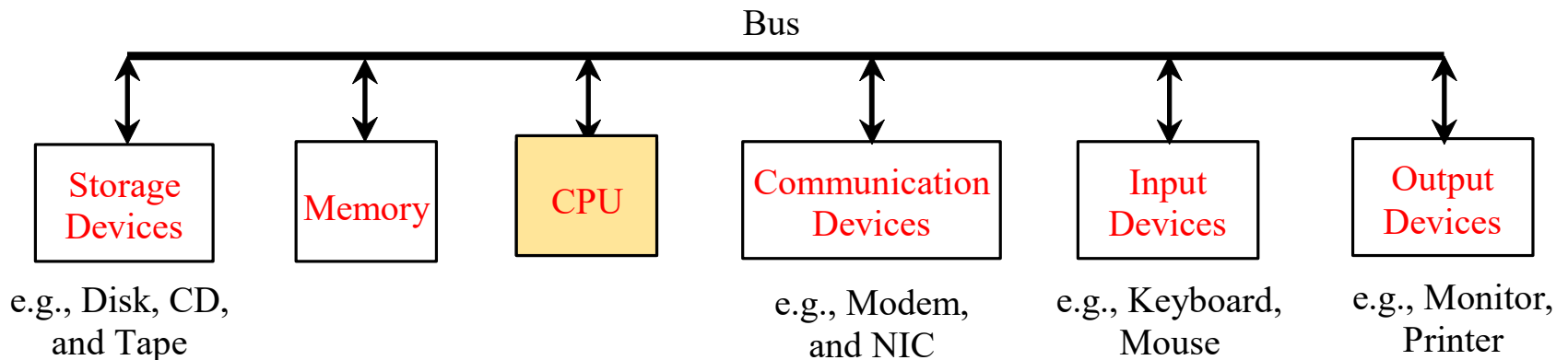
| Storage Devices | Memory | CPU | Communication Devices | Input Devices | Output Devices |

e.g., Disk, CD, and Tape                    e.g., Modem, and NIC        e.g., Keyboard, Mouse        e.g., Monitor, Printer
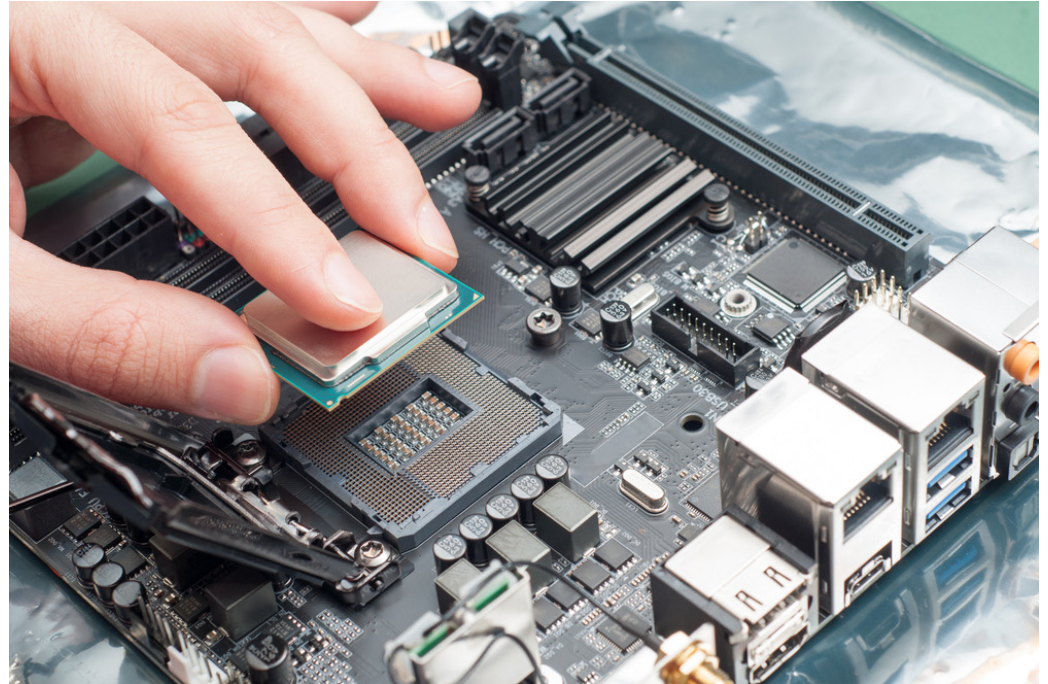
# A computer consists of

# CPU

- The central processing unit (CPU) is the brain of a computer.

- It retrieves instructions from memory and executes them.

- The CPU speed is measured in megahertz (MHz).

- 1 megahertz equaling 1 million pulses per second.

Bus

| Storage Devices | Memory | CPU | Communication Devices | Input Devices | Output Devices |

e.g., Disk, CD, and Tape

e.g., Modem, and NIC
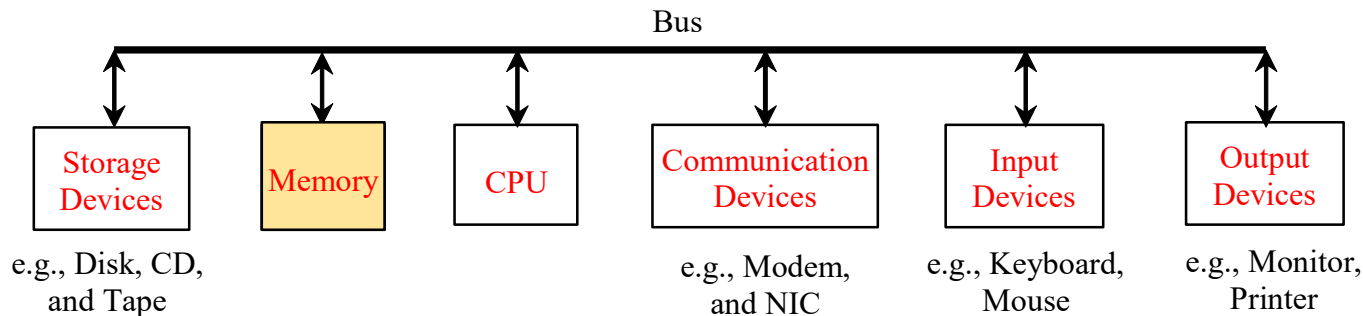
e.g., Keyboard, Mouse
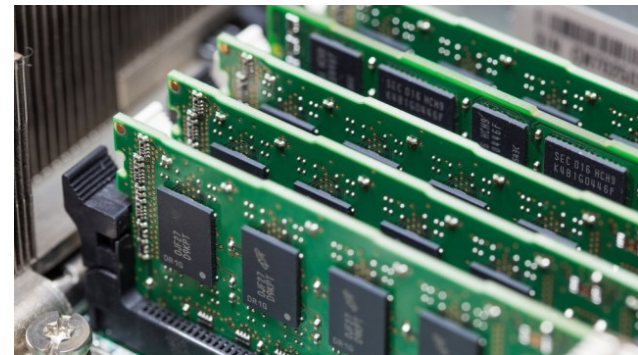
e.g., Monitor, Printer

# CPU

# Memory

- Memory is to store data and program instructions for CPU to execute.

- A memory unit is an ordered sequence of bytes, each holds eight bits. (a bit = 0 or 1)

- A program and its data must be brought to memory before they can be executed.

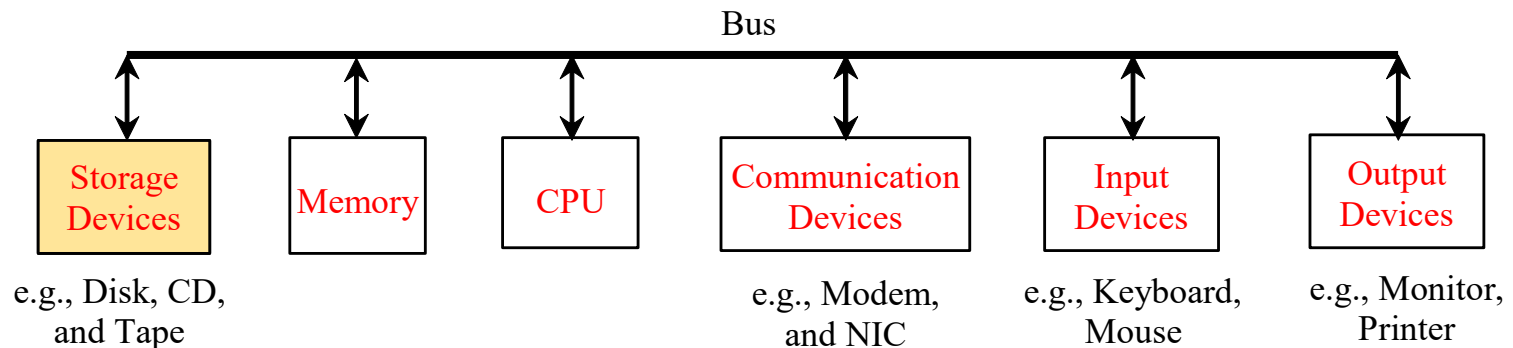- The current content of a memory byte is lost whenever new information is placed in it.

Bus

| Storage Devices | Memory | CPU | Communication Devices | Input Devices | Output Devices |

e.g., Disk, CD, and Tape

e.g., Modem, and NIC

e.g., Keyboard, Mouse

e.g., Monitor, Printer

# Memory

# Storage Devices

- Memory is volatile, because information is lost when the power is off.

- Programs and data are permanently stored on storage devices and are moved to memory when the computer actually uses them.

- There are three main types of storage devices: Disk drives (hard disks and floppy disks), CD drives (CD-R and CD-RW), and Tape drives (magnetic tape).
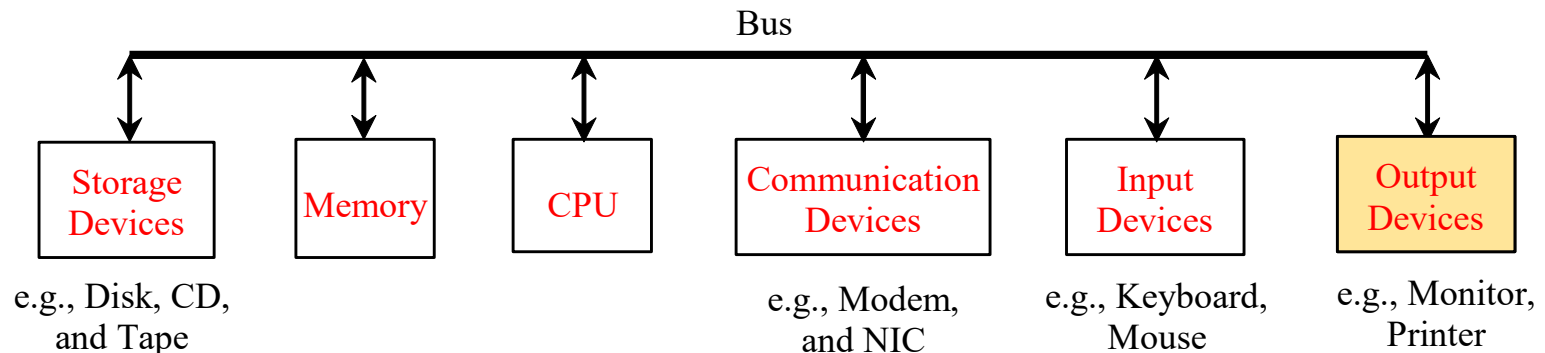
Bus

| Storage Devices | Memory | CPU | Communication Devices | Input Devices | Output Devices |

e.g., Disk, CD, and Tape

e.g., Modem, and NIC

e.g., Keyboard, Mouse

e.g., Monitor, Printer

# Storage Devices

# Output Devices

- An output device is any device used to send data from a computer to another device or user.

- Most computer data output that is meant for humans is in the form of audio or video such as monitors.

- The monitor displays information (text and graphics).

- The resolution and dot pitch determine the quality of the display.

Bus

| Storage Devices | Memory | CPU | Communication Devices | Input Devices | Output Devices |

e.g., Disk, CD, and Tape

e.g., Modem, and NIC
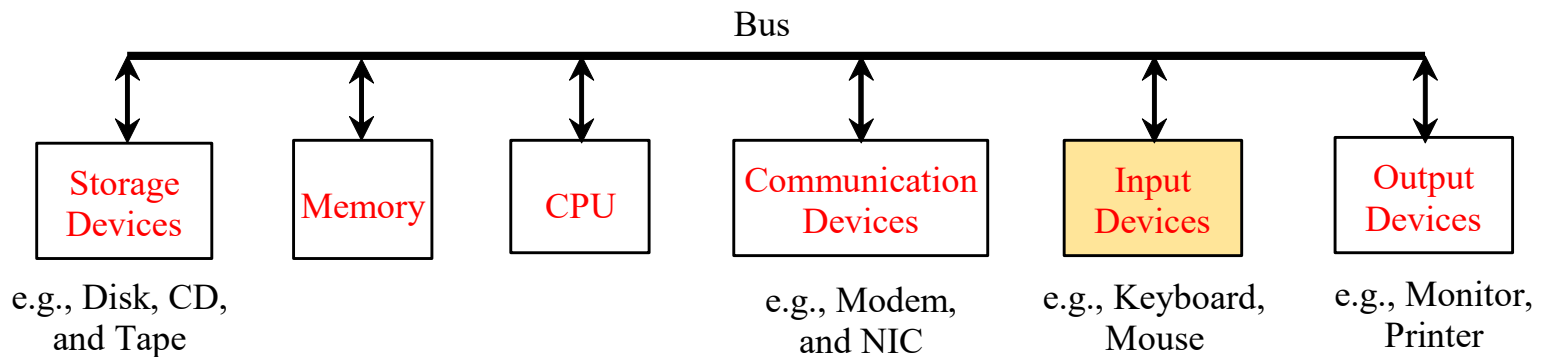
e.g., Keyboard, Mouse

e.g., Monitor, Printer

# Output Devices

# Input Devices

- An input device is any hardware device that sends data to a computer.

- Input and output devices let the user communicate with the computer.

- The most common input devices are keyboards and mice.



| | Bus | | | | |
|---|---|---|---|---|---|
| Storage Devices | Memory | CPU | Communication Devices | Input Devices | Output Devices |
| e.g., Disk, CD, and Tape | | | e.g., Modem, and NIC | e.g., Keyboard, Mouse | e.g., Monitor, Printer |

# Input Devices

# 1.3. Programming Languages

- Programs
- Machine Language
- Assembly Language
- High-Level Language
- Popular High-Level Languages
- Interpreting/Compiling Source Code
- Interpreting Source Code
- Compiling Source Code

# Programs

- Computer programs, known as software, are instructions to the computer.

- You tell a computer what to do through programs.

- Without programs, a computer is an empty machine.

- Computers do not understand human languages, so you need to use computer languages to communicate with them.

- Programs are written using programming languages.

# Programming Languages

**Machine Language**     Assembly Language     High-Level Language

- Machine language is a set of primitive instructions built into every computer.

- The instructions are in the form of binary code, so you have to enter binary codes for various instructions.

- Program with native machine language is a tedious process.

- Moreover the programs are highly difficult to read and modify.

- For example, to add two numbers, you might write an instruction in binary like this:

## 1101101010011010

# Programming Languages

## Machine Language    **Assembly Language**    High-Level Language

- Assembly languages were developed to make programming easy.
- Since the computer cannot understand assembly language, however, a program called assembler is used to convert assembly language programs into machine code.
- Writing code in assembly language is easier than in machine language. However, it is still tedious to write code in assembly language.
- For example, to add two numbers, you might write an instruction in assembly code like this:

## ADD 2, 3, result
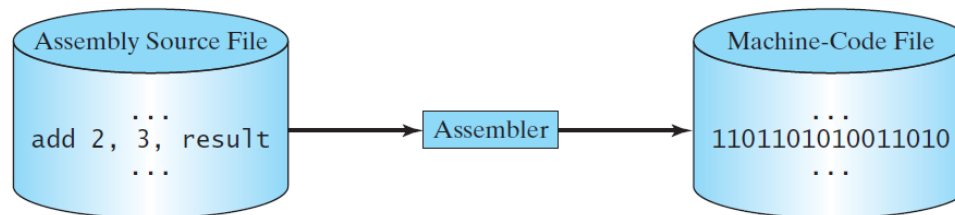
# Programming Languages

- The high-level languages are English-like and easy to learn and program.

- Since the computer cannot understand high-level languages, however, a program called interpreter or compiler is used to convert high-level language programs into machine code.

- For example, the following is a high-level language statement (instruction) that computes the area of a circle with radius 5:

$$\texttt{area = 5 * 5 * 3.1415}$$

# Popular High-Level Languages

**TABLE 1.1   Popular High-Level Programming Languages**

| Language | Description |
|---|---|
| Ada | Named for Ada Lovelace, who worked on mechanical general-purpose computers. The Ada language was developed for the Department of Defense and is used mainly in defense projects. |
| BASIC | Beginner's All-purpose Symbolic Instruction Code. It was designed to be learned and used easily by beginners. |
| C | Developed at Bell Laboratories. C combines the power of an assembly language with the ease of use and portability of a high-level language. |
| C++ | C++ is an object-oriented language, based on C. |
| C# | Pronounced "C Sharp." It is a hybrid of Java and C++ and was developed by Microsoft. |
| COBOL | COmmon Business Oriented Language. Used for business applications. |
| FORTRAN | FORmula TRANslation. Popular for scientific and mathematical applications. |
| Java | Developed by Sun Microsystems, now part of Oracle. It is widely used for developing platform-independent Internet applications. |
| Pascal | Named for Blaise Pascal, who pioneered calculating machines in the seventeenth century. It is a simple, structured, general-purpose language primarily for teaching programming. |
| Python | A simple general-purpose scripting language good for writing short programs. |
| Visual Basic | Visual Basic was developed by Microsoft and it enables the programmers to rapidly develop Windows-based applications. |

# Interpreting/Compiling Source Code

- The instructions in a high-level programming language are called statements.

- A program written in a high-level language is called a source program or source code.

- Because a computer cannot understand a source program, a source program must be translated into machine code for execution.

- The translation can be done using another programming tool called an interpreter or a compiler.

# Interpreting Source Code

- An interpreter reads one statement from the source code, translates it to the machine code or virtual machine code, and then executes it right away.

- Note that a statement from the source code may be translated into several machine instructions.



High-Level Source File

```
. . .
area = 5 * 5 * 3.1415;
. . .
```

Interpreter

Output
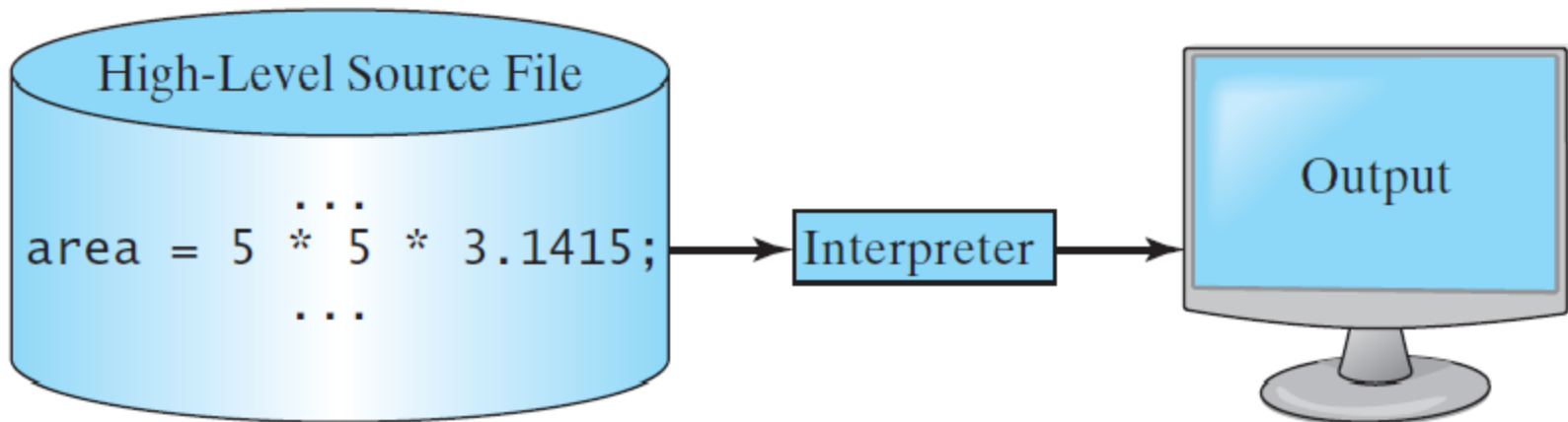
# Compiling Source Code

- A compiler translates the entire source code into a machine-code file, and the machine-code file is then executed.
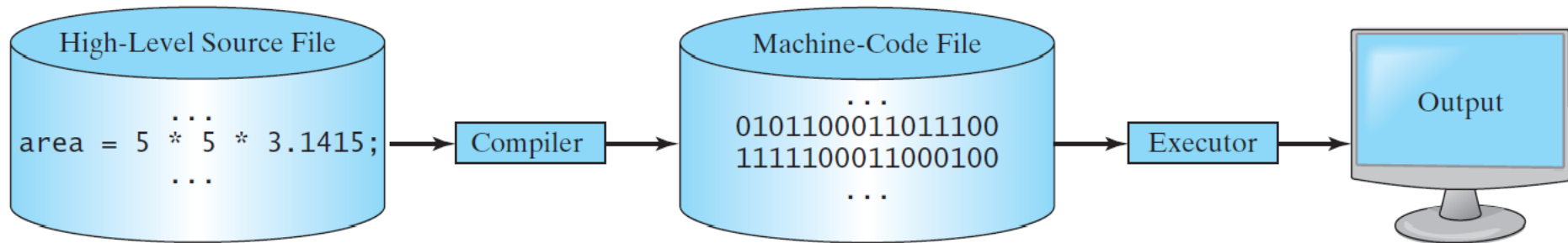
# 1.4. Operating Systems

# Operating Systems

- The operating system (OS) is a program that manages and controls a computer's activities.

- You are probably using Windows 10, Linux, or macOS.

- Windows is currently the most popular PC operating system.

- Application programs - such as an Internet browser and a word processor - cannot run without an operating system.



User

Application Programs

Operating System

Hardware
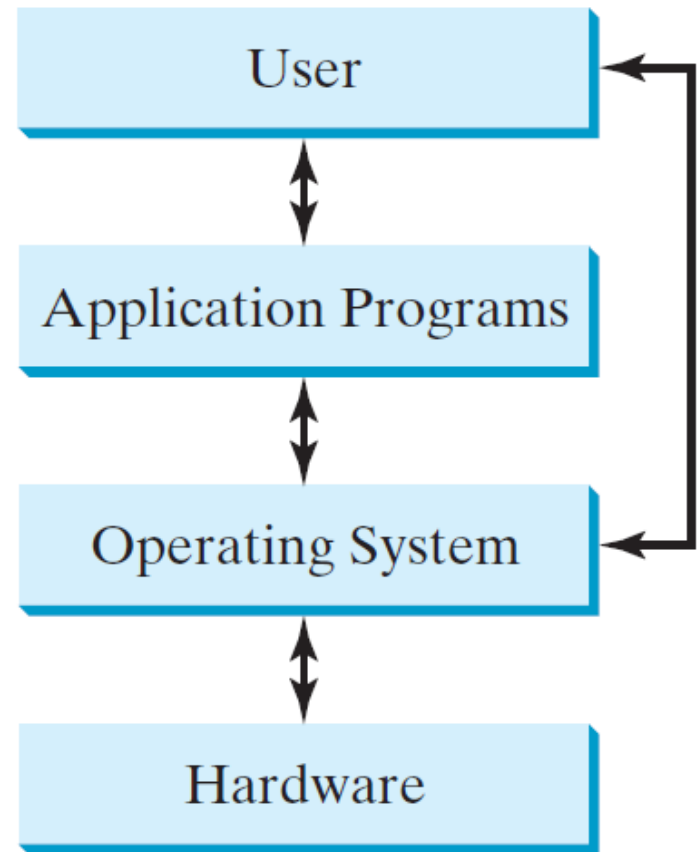
# 1.5. The History of Python

- [What is Python?](#)

- [Python's History](#)

- [Python 2 vs Python 3](#)

# What is Python?

**General Purpose**     Interpreted     Object-Oriented

- Python is a general-purpose programming language.
- That means you can use Python to write code for any programming tasks.
- Python are now used in Google search engine, in mission critical projects in NASA, in processing financial transactions at New York Stock Exchange.

# What is Python?

General Purpose    **Interpreted**    Object-Oriented

- Python is interpreted.
- Which means that python code is translated and executed one statement at a time by an interpreter.
- In a compiled language, the entire source code is compiled and then executed altogether.

# What is Python?

General Purpose      Interpreted      **Object-Oriented**

- Python is an object-oriented programming language.
- Data in Python are objects created from classes.
- A class is essentially a type that defines the objects of the same kind with properties and methods for manipulating objects.
- Object-oriented programming is a powerful tool for developing reusable software.

# Python's History

- Python is created by *Guido van Rossum* in Netherlands in 1990.

- Python is open source.

- Open-source software is a type of computer software in which source code is released under a license in which the copyright holder grants users the rights to study, change, and distribute the software to anyone and for any purpose.

# Python 2 vs Python 3

- Python 3 is a newer version, but it is not backward compatible with Python 2.

- That means if you write a program using Python 2, it may not work on Python 3.

- For example, the following command works on Python 2, but it doesn't work on Python 3: print "Hello World".
  - To get the previous command working on Python 3, you can write it as the following: print("Hello World").

- We will learn and use Python 3 in this book.

# 1.6. Getting Started with Python

- [Install Python](#)

- [PyCharm IDE](#)

- [Install PyCharm](#)

- [Modes of Python Interpreter](#)

- [Interactive vs Script Mode](#)

# Install Python

- Go to [www.python.org/downloads](www.python.org/downloads) and then download and install the last version of **Python 3.x.**x for your operating system.

- See Lab 1 for more details.
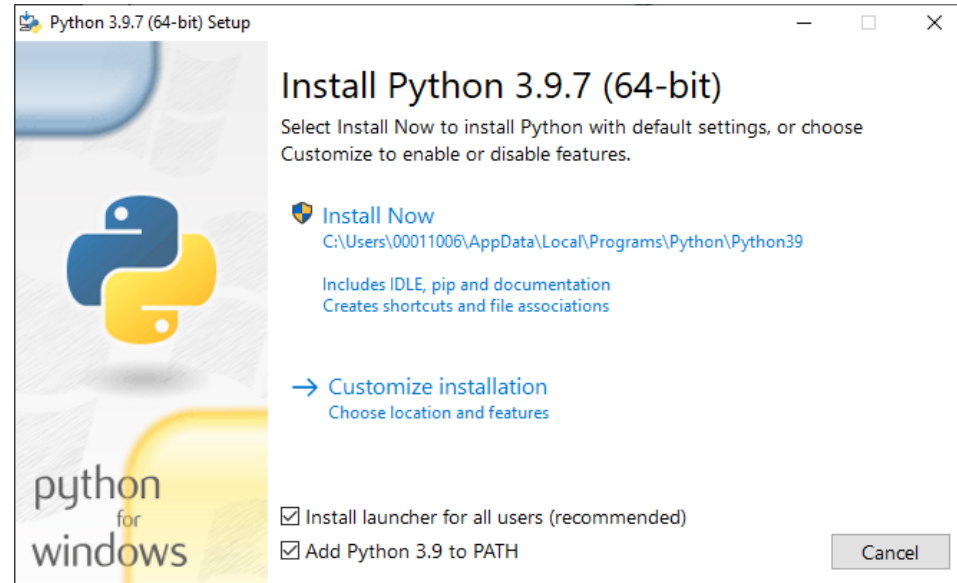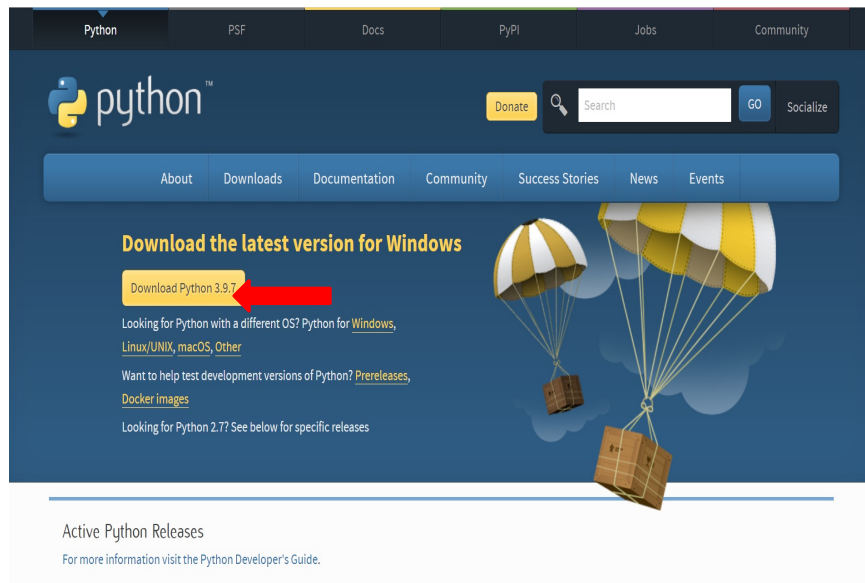
# PyCharm IDE

- An integrated development environment (IDE) is an application that provides comprehensive facilities to programmers for software development.

- IDEs are large size programs, and many of them are not free.

- For Python programmers, PyCharm is one of the best IDE for Python.

- Also, it has a free version called "Community Edition".

- In general, using IDEs are the best way to develop programs especially mid-large programs.

# Install PyCharm

- Go to [https://www.jetbrains.com/pycharm/download/](https://www.jetbrains.com/pycharm/download/) and then download and install "Community" version.

- See Lab 1 for more details.
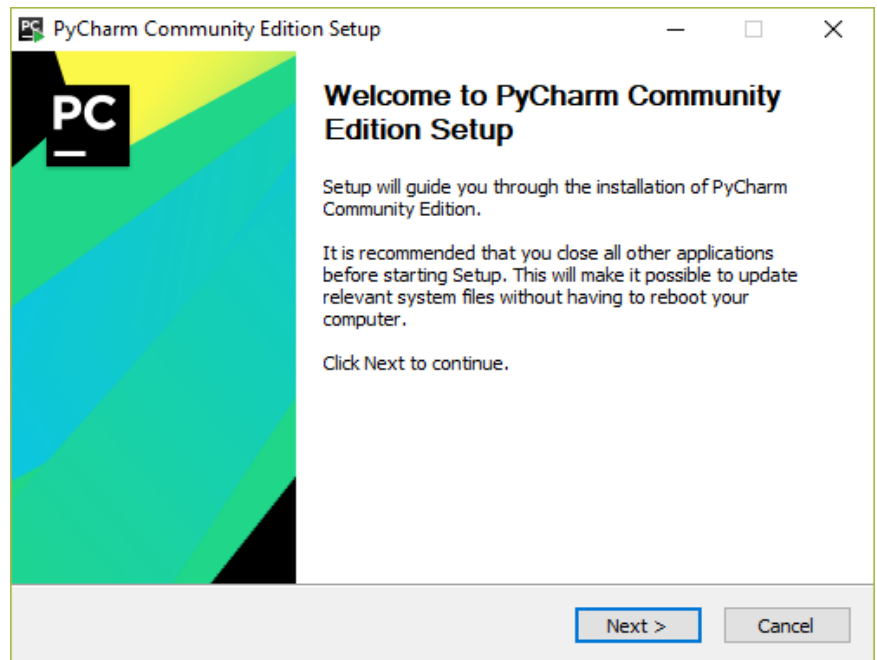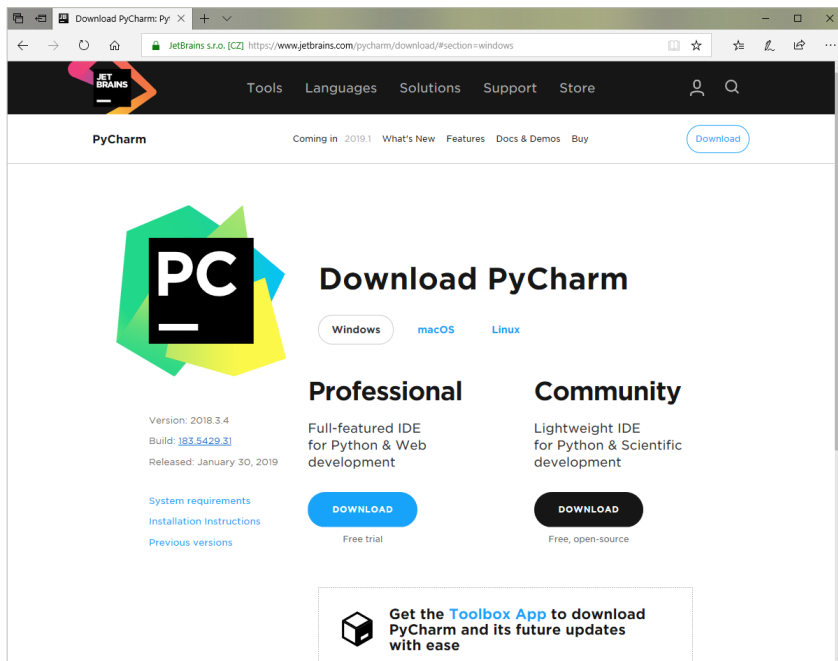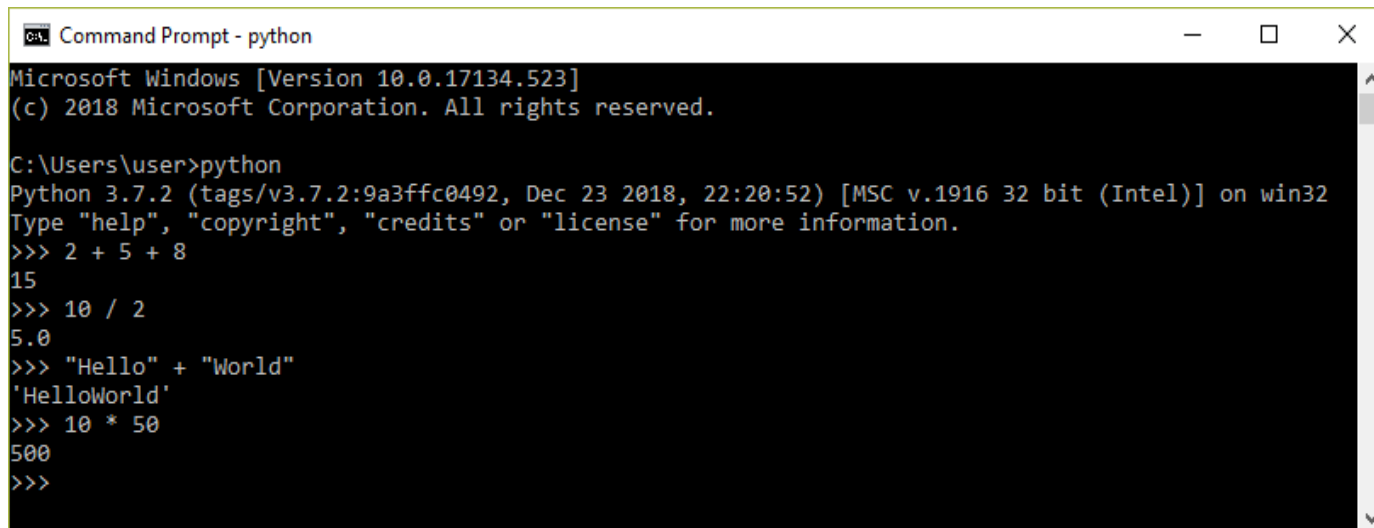
# Modes of Python Interpreter

**Interactive Mode**          Script Mode

- Interactive mode provides us with a quick way of running blocks or a single line of Python code.

- The code executes via the Python Shell (also known as Python Interactive Shell), which comes with Python installation.



Python Shell on CMD

# Modes of Python Interpreter

**Interactive Mode**          Script Mode

- The **>>>** indicates that the Python shell is ready to execute and send your commands to the Python interpreter.

- The result is immediately displayed on the Python shell as soon as the Python interpreter interprets the command.
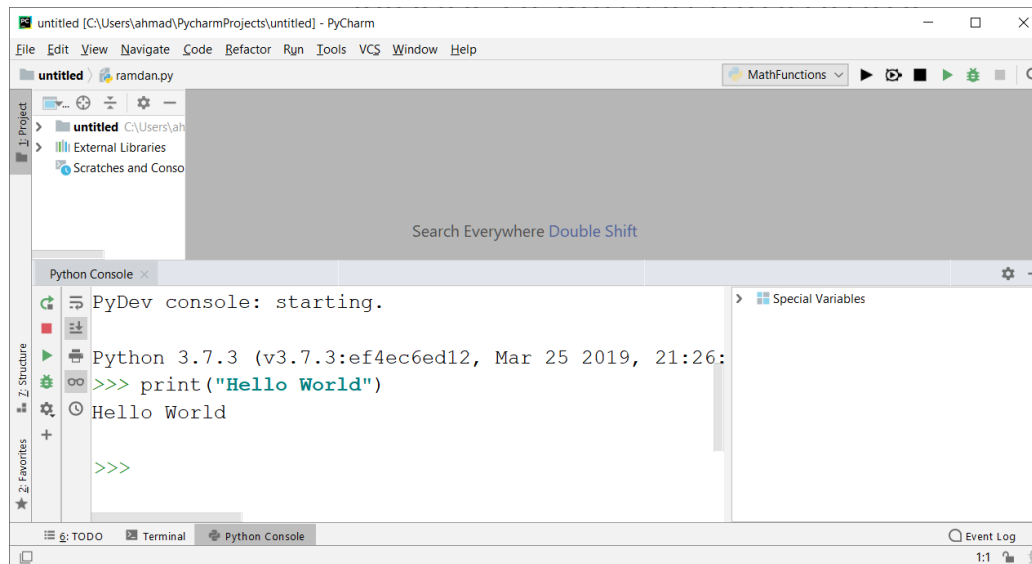


Python Shell on PyCharm IDE

# Modes of Python Interpreter

Interactive Mode          **Script Mode**

- This is the normal mode where a python code is written in a text file with a '.py' extension, and Python interpreter executes the file.

- The result of the code will be displayed after the Python interpreter runs the file.
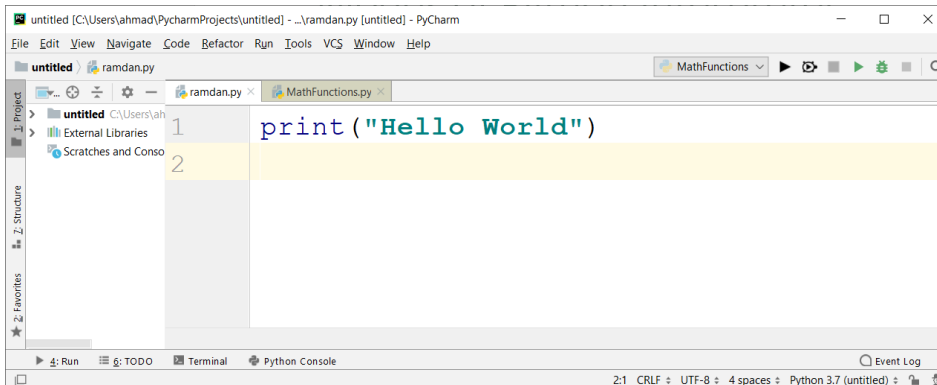


PyCharm IDE
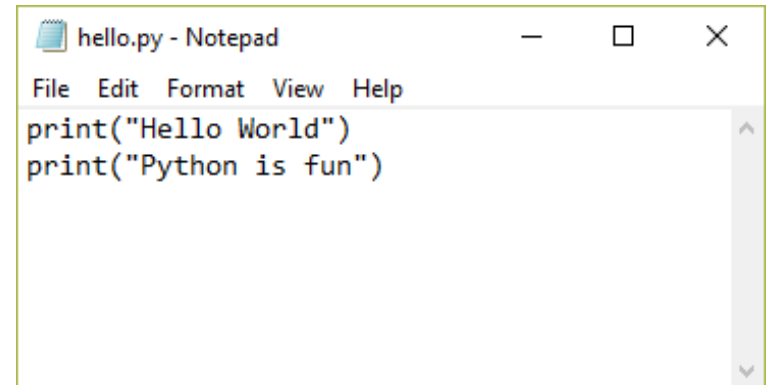


Notepad

# Interactive vs Script Mode

The key differences between programming in interactive mode and programming in script mode:

1.  In script mode, a file must be created and saved before executing the code to get results. In interactive mode, the result is returned immediately after pressing the **<enter>** key from the keyboard.

2.  In script mode, you are provided with a direct way of editing your code. This is not possible in interactive mode.

# A Simple Python Program

- [Program 1: Welcome with Two Messages](#)

- [Creating and Editing Using PyCharm](#)

- [Tracing The Program Execution](#)

- [Code Tracing](#)

# Welcome with Two Messages
## Program 1

Write a program that displays Welcome to Python and Programming is fun. The output should be as the following:

```
Welcome to Python
Python is fun
```

➤ The Solution:

**LISTING 1.1 Welcome.py**

```python
1  # Display two messages
2  print("Welcome to Python")
3  print("Python is fun")
```

▶ Run

# Welcome with Two Messages
## Creating and Editing Using PyCharm

- Open PyCharm.
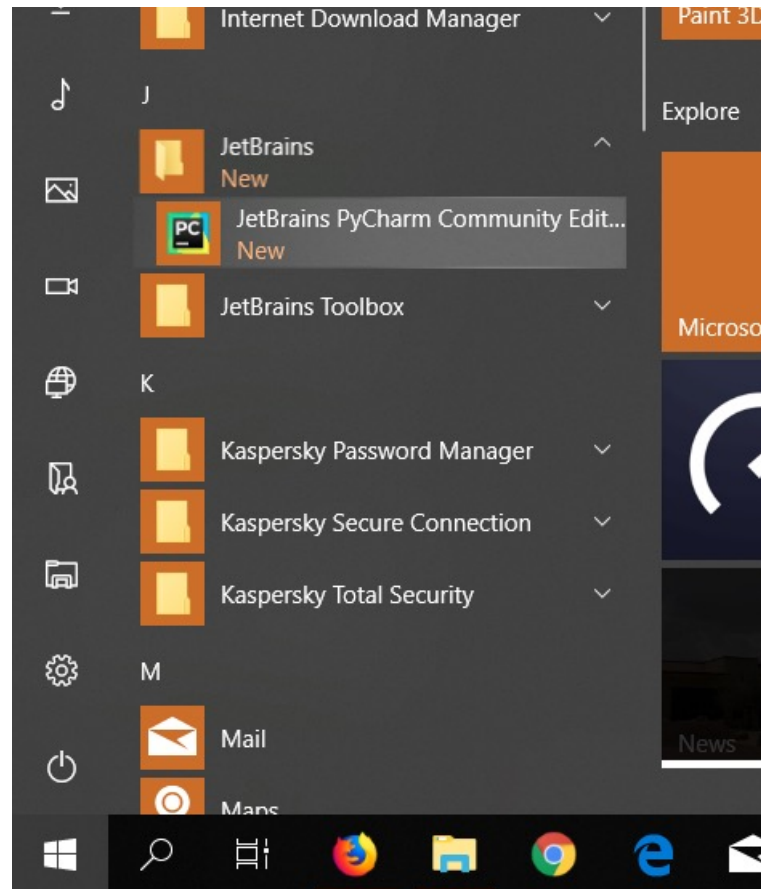
# Welcome with Two Messages
## Creating and Editing Using PyCharm

- Click on "New Project".
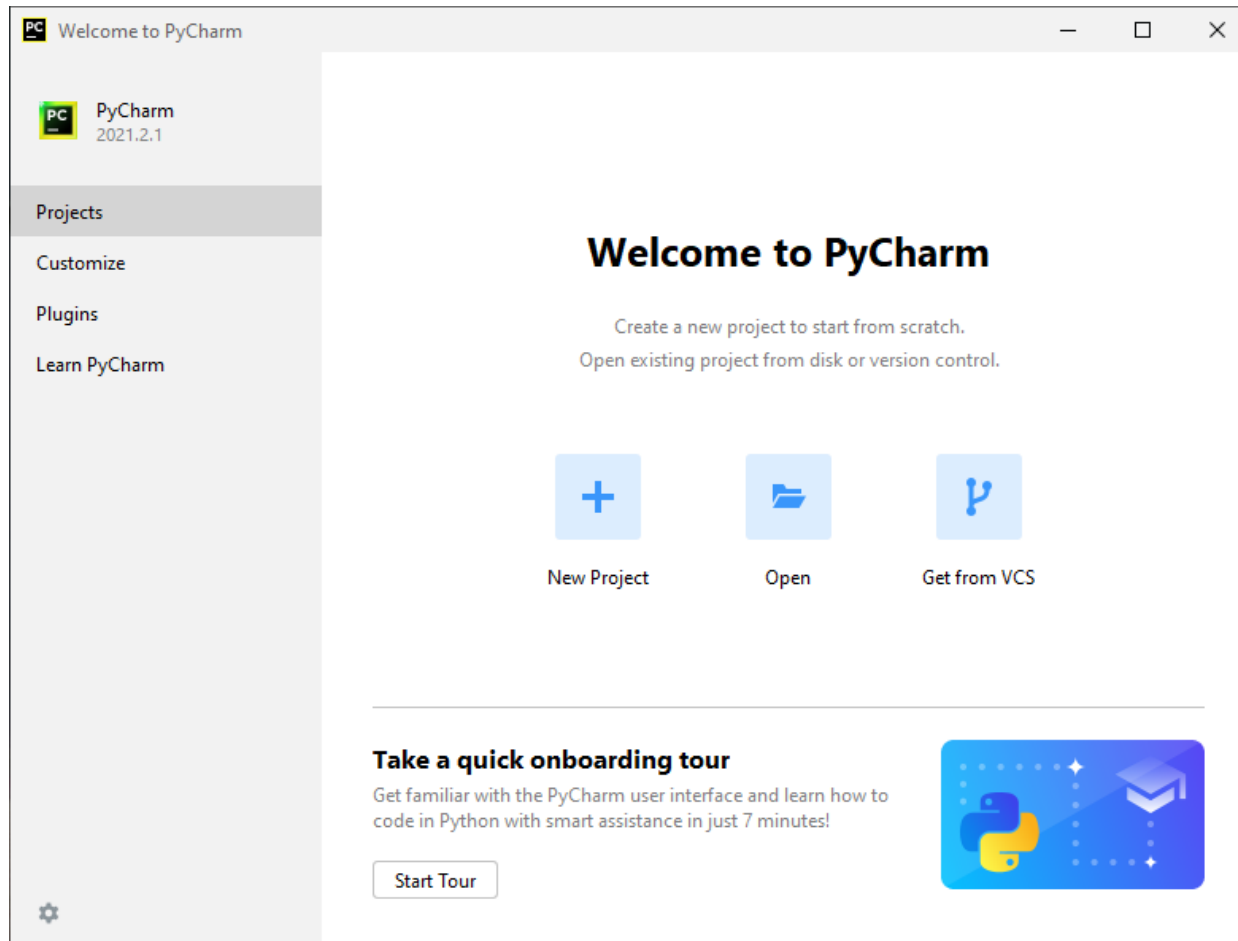
# Welcome with Two Messages
## Creating and Editing Using PyCharm

- Then, change the default name of the project "pythonProject". For example, name it as "My First Project", and then click on "Create".
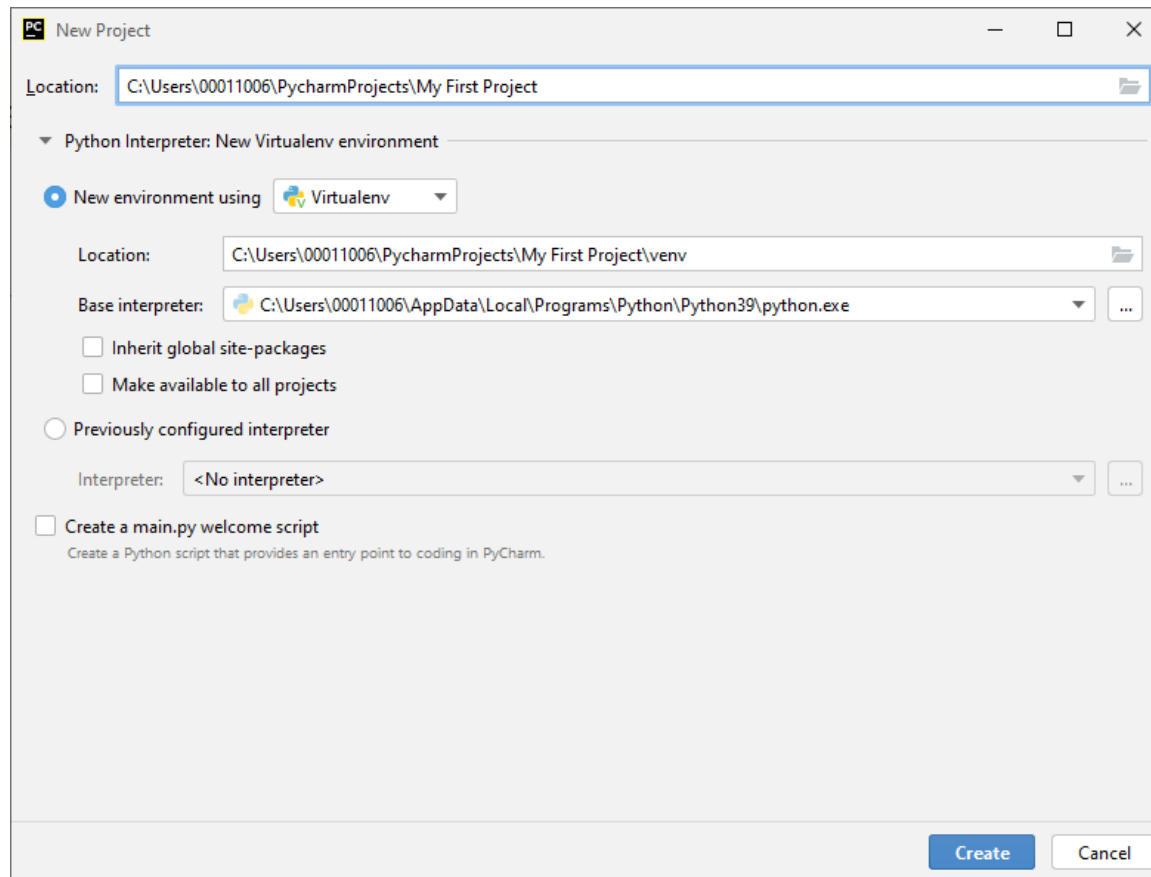
# Welcome with Two Messages
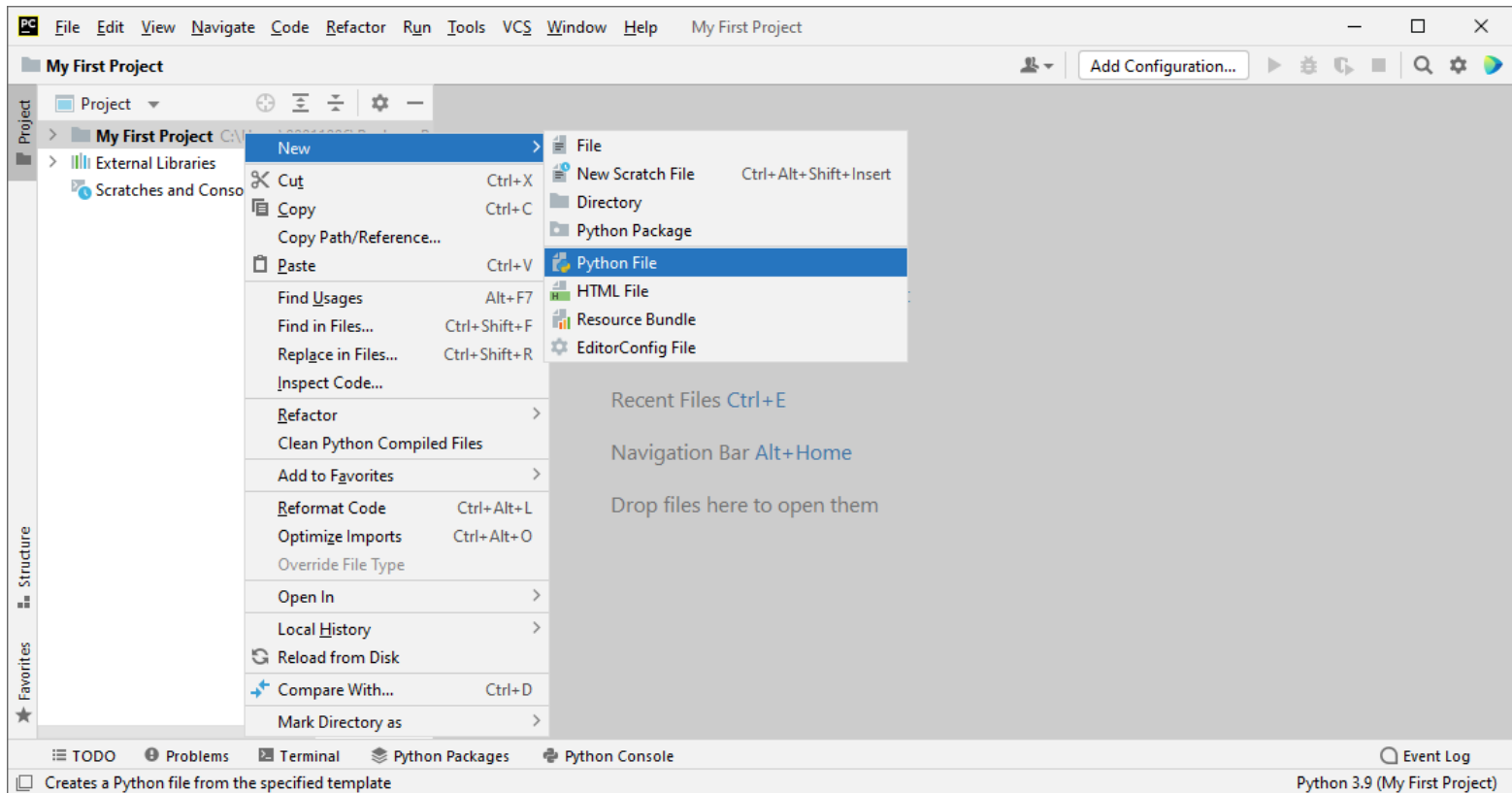## Creating and Editing Using PyCharm

- Then, the new project is created and opened. After that, you have to create a new Python file inside the project to write the code on it.
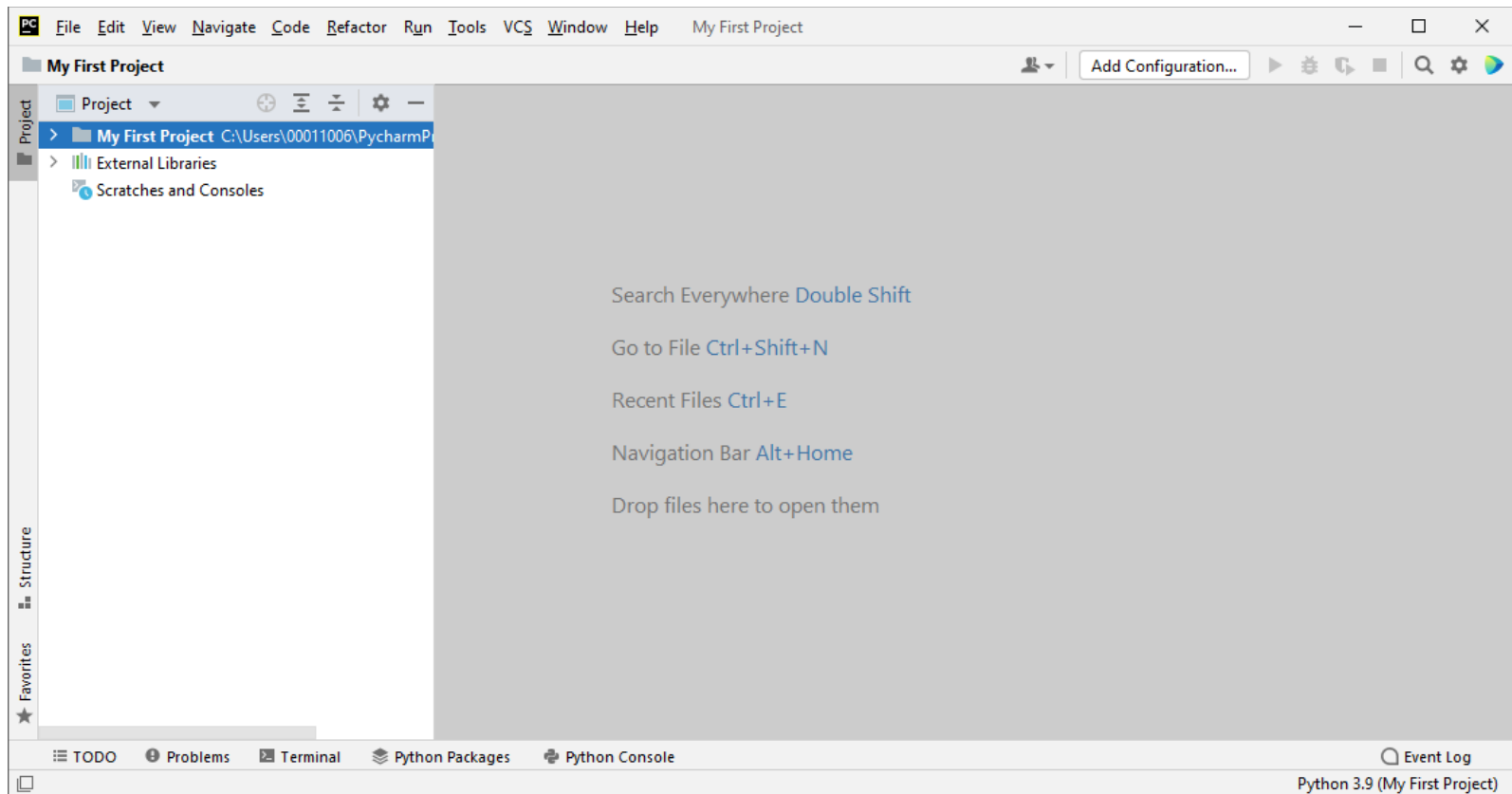
# Welcome with Two Messages
## Creating and Editing Using PyCharm

- Select the project name on the left menu, right click on it and select "New" → "Python File".

# Welcome with Two Messages
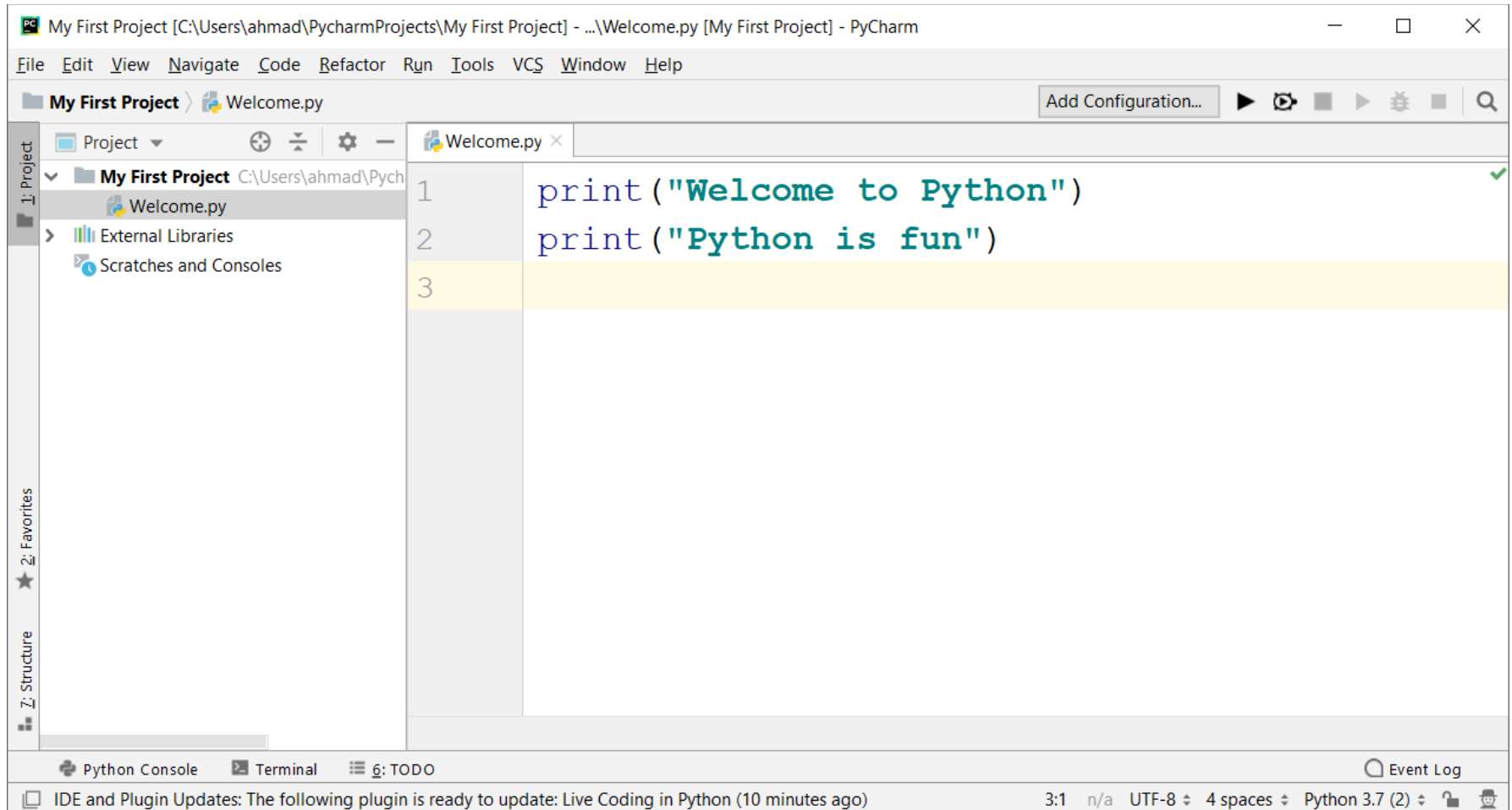## Creating and Editing Using PyCharm

- Then, name the new file "Welcome".
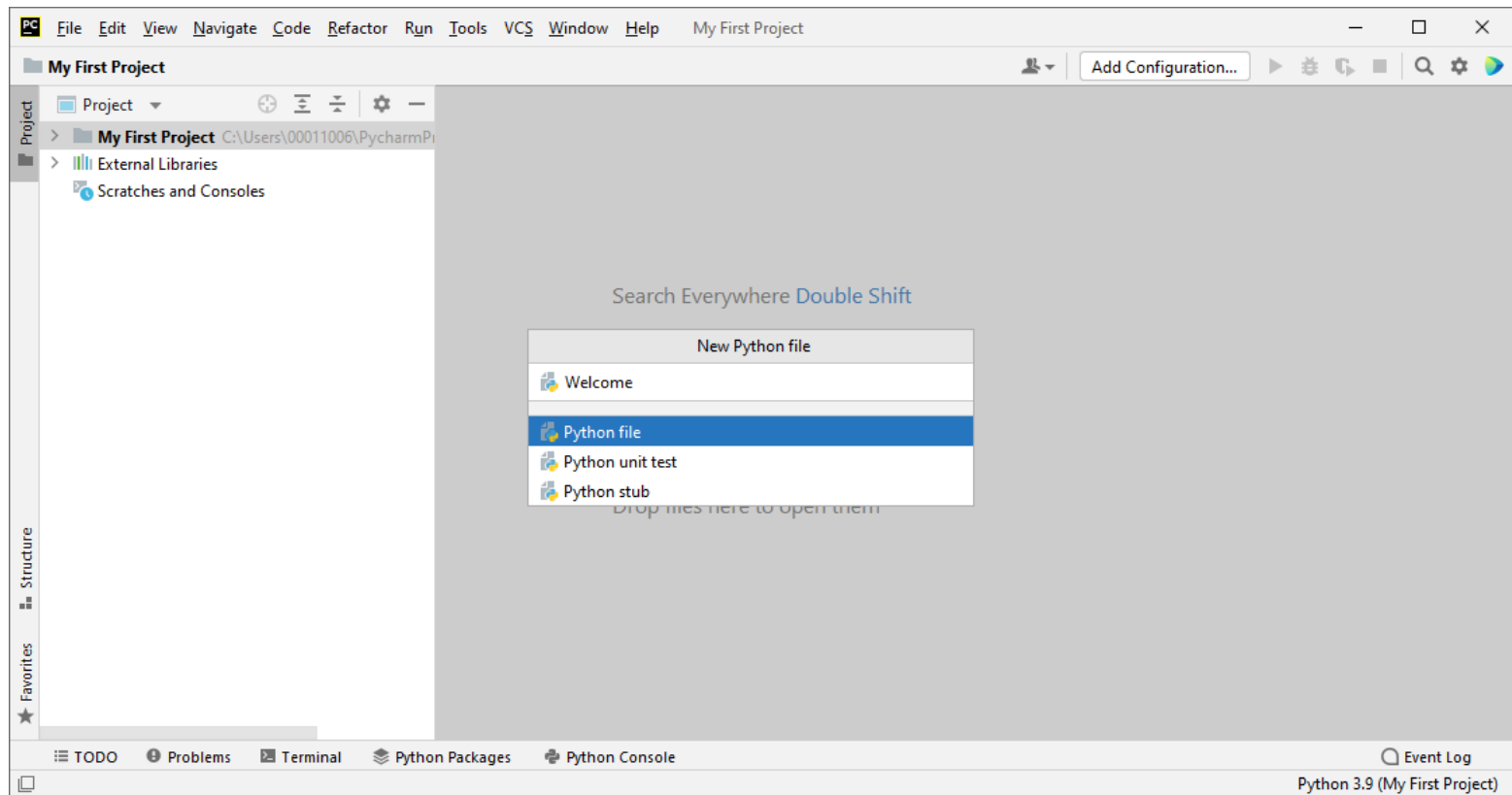
# Welcome with Two Messages
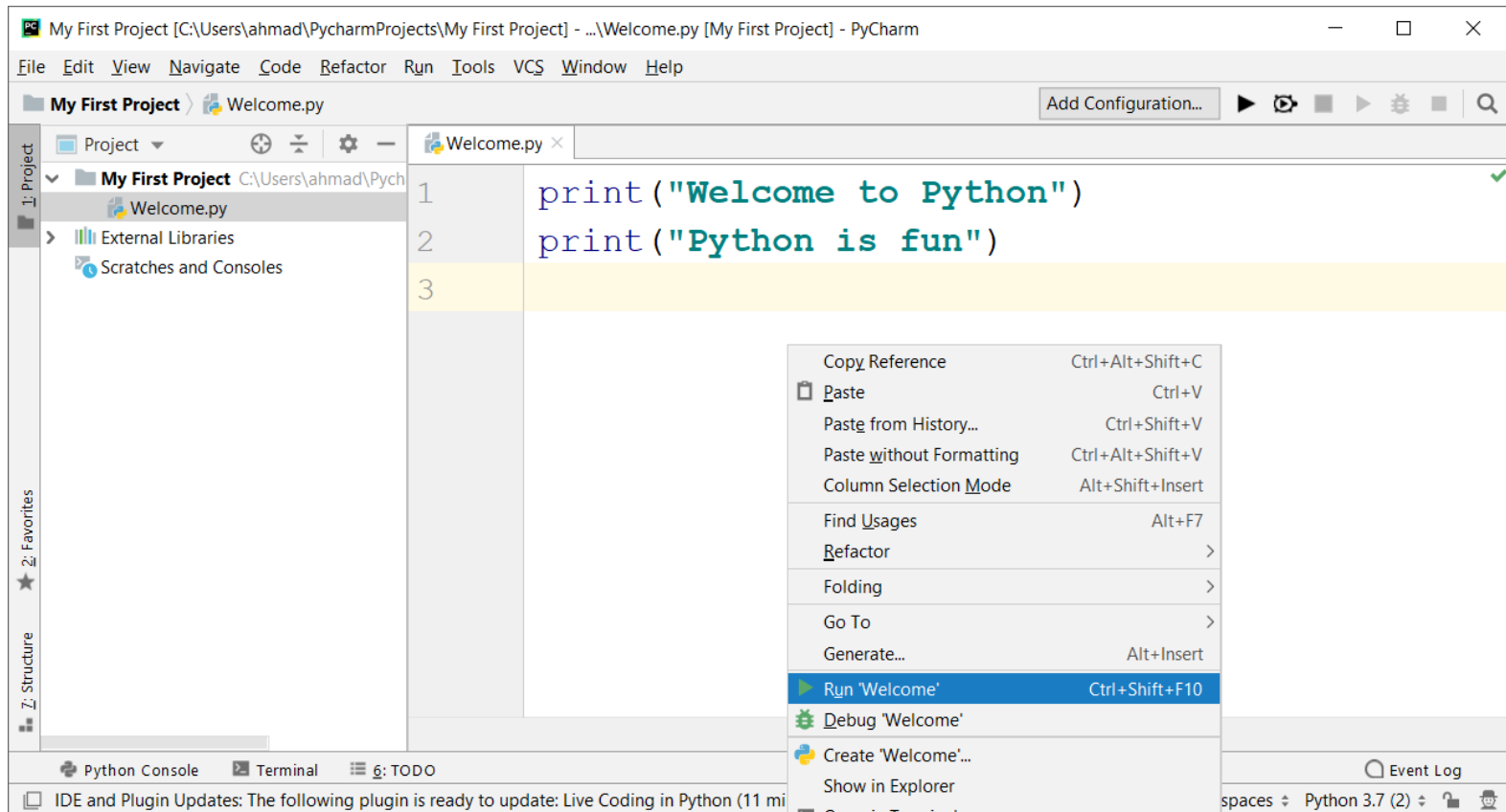## Creating and Editing Using PyCharm

- Now, the new file is created and opened. Write the code in it:

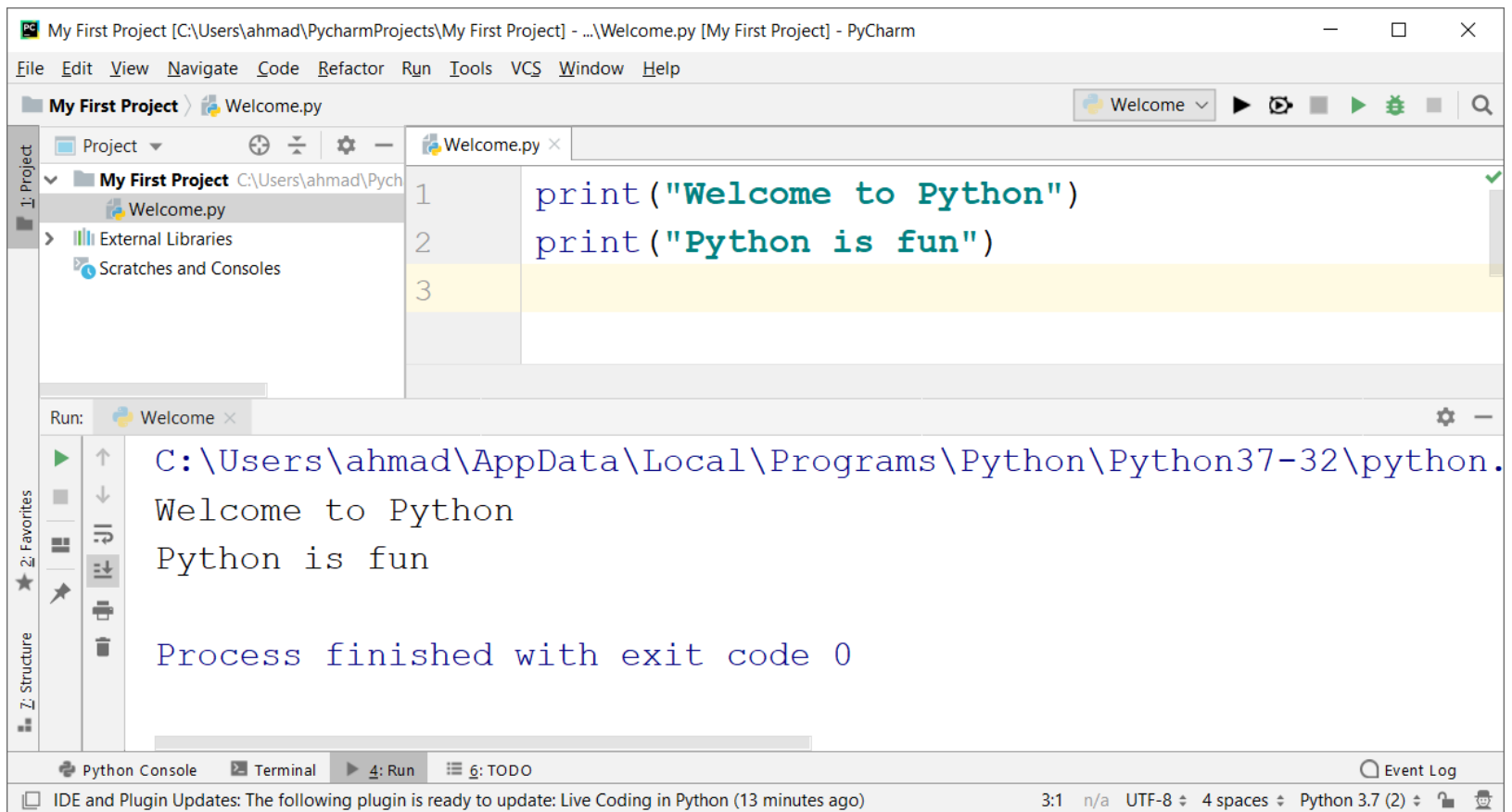# Welcome with Two Messages
## Running The Code Using PyCharm

- To run the file, right click on any area of the editor and click on (Run 'Welcome'), which is the name of the file.

# Welcome with Two Messages
## Running The Code Using PyCharm

- After that, PyCharm is going to run the file using the Python interpreter, and then display the output of the file for you.
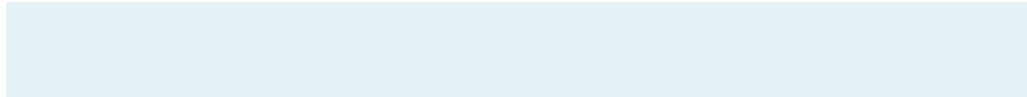
# Welcome with Two Messages
## Tracing The Program Execution

LISTING 1.1 Welcome.py

```
1  # Display two messages
2  print("Welcome to Python")
3  print("Python is fun")
```

It is a comment, so do nothing.

# Welcome with Two Messages
## Tracing The Program Execution

LISTING 1.1 Welcome.py

```
1  # Display two messages
2  print("Welcome to Python")
3  print("Python is fun")
```

Execute a statement

```
Welcome to Python
```

The output of the statement

# Welcome with Two Messages
## Tracing The Program Execution

LISTING 1.1 Welcome.py

```python
1  # Display two messages
2  print("Welcome to Python")
3  print("Python is fun")
```

Execute a statement

```
Welcome to Python
Python is fun
```

The output of the statement

# Code Tracing

- Code tracing is when the programmer interprets the results of each line of code and keeps track of the effect of each statement.

# Simple Examples

- Program 2: Welcome With Three Messages

- Program 3: Compute an Expression

- Check Point #1 - #2

# Welcome With Three Messages
## Program 2

Write a program that displays Welcome to Python , Programming is fun , and Problem Driven . The output should be as the following:

```
Welcome to Python
Python is fun
Problem Driven
```

➢ The Solution:

LISTING 1.2 WelcomeWithThreeMessages.py

```python
1  # Display three messages
2  print("Welcome to Python")
3  print("Python is fun")
4  print("Problem Driven")
```

Run

# Compute an Expression
## Program 3

Write a program that evaluates $\dfrac{10.5 + 2 \times 3}{45 - 3.5}$ and print its result.

➢ The Solution:

```
1  # Compute expression
2  print((10.5 + 2 * 3) / (45 - 3.5))
```

▶ **Run**

➢ The output:

```
0.39759036144578314
```

# Check Point
# #1

Identify and fix the errors in the following code:

```
1   # Display two messages
2     print("Welcome to Python")
3   print("Python is fun").
```

➢ Solution:
The errors are the incorrect indentation in line 2 and the punctuation (.) in line 3.

```
1   # Display two messages
2   print("Welcome to Python")
3   print("Python is fun")
```

# Check Point #2

Show the output of the following code:

```
1  print("3.5 * 4 / 2 - 2.5 is")
2  print(3.5 * 4 / 2 - 2.5)
```

➢ Solution:

```
3.5 * 4 / 2 - 2.5 is
4.5
```

# Anatomy of a Python Program

- [Statement](Statement)

- [Indentation](Indentation)

- [Comment](Comment)

- [Special Symbols](Special-Symbols)

# Statement

- A statement represents an action or a sequence of actions.

- The statement print("Welcome to Python") in the program in Listing 1.1 is a statement to display the greeting "Welcome to Python".

LISTING 1.1 Welcome.py

```
1  # Display two messages
2  print("Welcome to Python")
3  print("Python is fun")
```

It is a statement (action)

It is a statement (action)

# Indentation

- The indentation matters in Python.

- The following figure is a block structure visualizing indentation.

| Block 1 | | | |
|---|---|---|---|
| | Block 2 | | |
| | | Block 3 | |
| | Block 2, Continuation | | |
| Block 1, Continuation | | | |

- Note that the statements are entered from the first column in the new line. It would cause an error if the program is typed as follows:
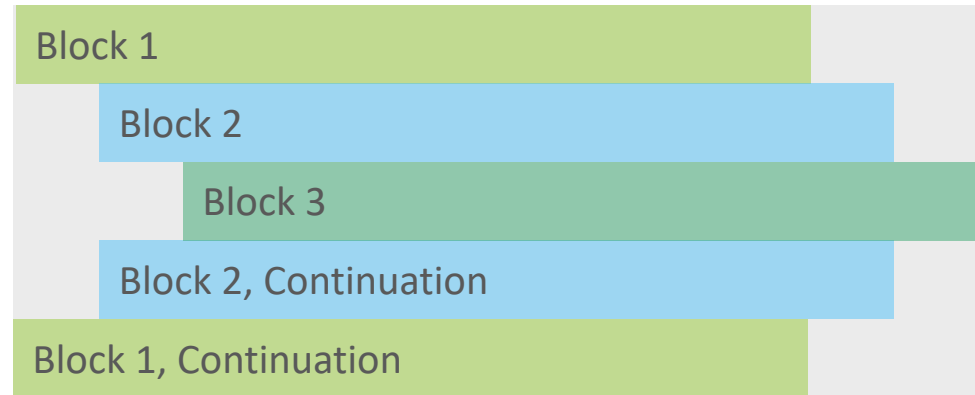
```
1  # Display two messages
2    print("Welcome to Python")
3  print("Python is fun")
```

⚠️

> It would cause an error because this statement has a wrong indentation.

# Caution

- Don't put any punctuation at the end of a statement.

- For example, the Python interpreter will report errors for the following code:

```
1  # Display two messages
2  print("Welcome to Python").
3  print("Python is fun").
```

# Note

- Python programs are case sensitive.

- It would be wrong, for example, to replace print in the program with **P**rint.

```
1  # Display two messages
2  print("Welcome to Python")
3  Print("Python is fun")
```

# Comment

- A comment is a programmer-readable explanation or annotation in the source code of a computer program.

- In Listing 1.1, line 1 is a comment that documents what the program is and how it is constructed.

LISTING 1.1 Welcome.py

```
1  # Display two messages
2  print("Welcome to Python")
3  print("Python is fun")
```

It is a comment, so the Python interpreter will ignore it when executing the program.

# Comment

- Comments help programmers communicate and understand a program.

- They are not programming statements and thus are ignored by the interpreter.

- In Python, comments are preceded by a pound sign (#) on a line, called a line comment, or enclosed between three consecutive single quotation marks (''') on one or several lines, called a paragraph comment.

# Comment

- When the Python interpreter sees #, it ignores all text after # on the same line.

- When it sees ''', it scans for the next ''' and ignores any text between the triple quotation marks.

- Here are examples of comments:

```
1  # This program displays Welcome to Python (a line comment)
2  ''' This program displays Welcome to Python and
3  Python is fun (a paragraph comment)
4  '''
5  print("Welcome to Python")
6  print("Python is fun")
```

# Special Symbols

**TABLE 1.2**    Special Characters

| Character | Name | Description |
| --- | --- | --- |
| ( ) | Opening and closing parentheses | Used with functions. |
| # | Pound sign | Precedes a comment line. |
| " " | Opening and closing quotation marks | Encloses a string (i.e., sequence of characters). |
| ' ' ' ' ' ' | Paragraph comments | Encloses a paragraph comment. |

# 1.7. Programming Style and Documentation

- Programming Style

- Documentation

- Appropriate Comments and Comment Styles

- Proper Indentation and Spacing

# Programming Style

- Programming style deals with what programs look like.

- When you create programs with a professional programming style, they not only execute properly but are easy for people to read and understand.

- This is very important if other programmers will access or modify your programs.

# Documentation

- Documentation is the body of explanatory remarks and comments pertaining to a program.

- These remarks and comments explain various parts of the program and help others understand its structure and function.

- As you saw earlier in the chapter, remarks and comments are embedded within the program itself; Python's interpreter simply ignores them when the program is executed.

- Good programming style and proper documentation make a program easy to read and prevents errors.

- Programming style and documentation are as important as coding. In the following slides, there are a few guidelines.

# Appropriate Comments and Comment Styles

- Include a summary comment at the beginning of the program to explain what the program does, its key features, and any unique techniques it uses.

- In a long program, you should also include comments that introduce each major step and explain anything that is difficult to read.

- It is important to make comments concise so that they do not crowd the program or make it difficult to read.

- In homework and exams, Include your name, class section, instructor, date, and a brief description at the beginning of the program.

# Proper Indentation and Spacing

- Indentation
  - Indent four spaces.

- Spacing
  - A consistent spacing style makes programs clear and easy to read, debug, and maintain.
  - Use blank line to separate segments of the code.

```
print(20+50)
print(20- 10)
print(60*5+30)
print("A","B")
```

```
print(20 + 50)
print(20 - 10)
print(60 * 5 + 30)
print("A" ,"B")
```

# 1.8. Programming Errors
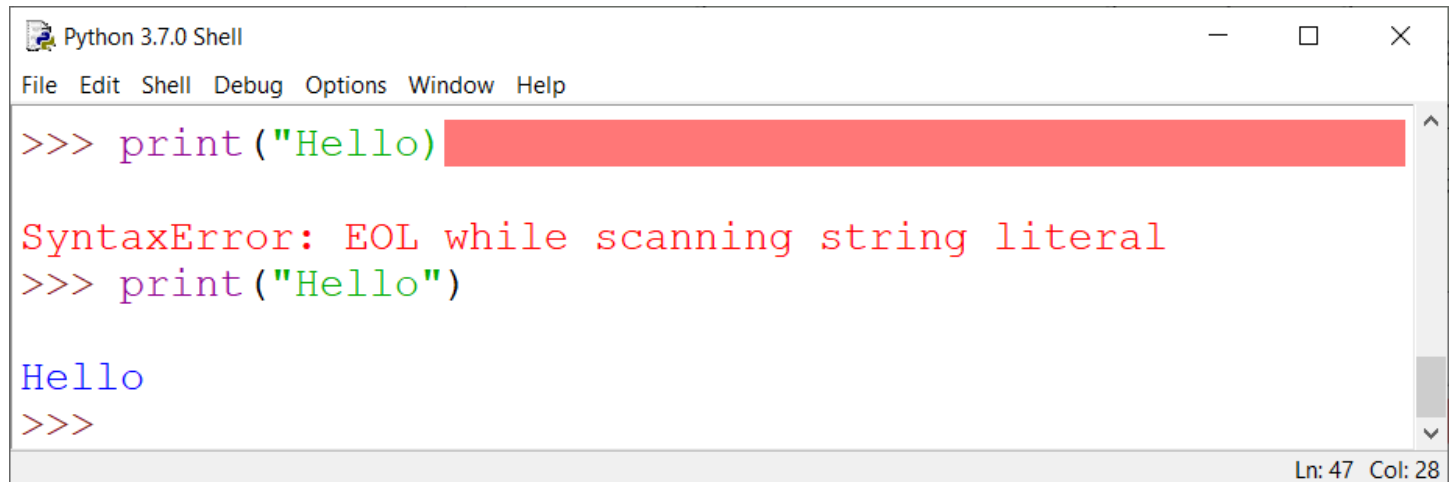
- [Types of Programming Errors](#)

- [Syntax Errors](#)

- [Runtime Errors](#)

- [Logic Errors](#)

- [Notes](#)

- [Check Point #3](#)

# Types of Programming Errors

- Programming errors can be categorized into three types:
  - Syntax Errors
    - Error in code construction.
  - Runtime Errors
    - Causes the program to abort.
  - Logic Errors
    - Produces incorrect result.
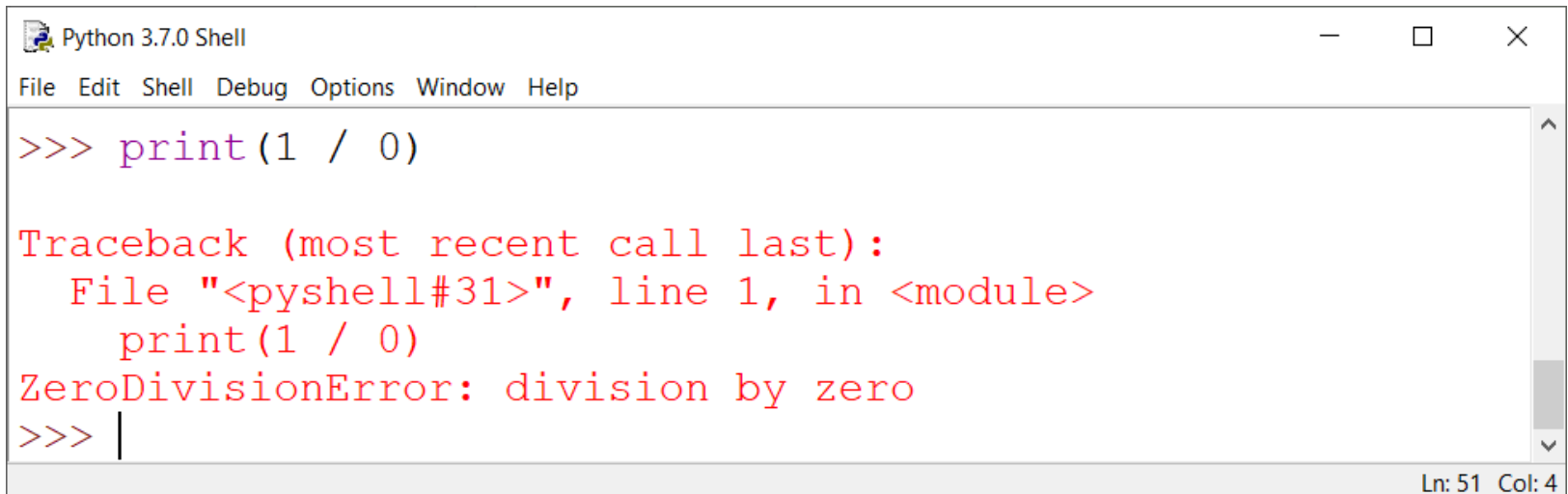
# Syntax Errors

- Syntax errors result from errors in code construction, such as mistyping a statement, incorrect indentation, omitting some necessary punctuation, or using an opening parenthesis without a corresponding closing parenthesis.

- Python has its own syntax, and you need to write code that obeys the syntax rules. If your program violates the rules Python will report syntax errors.

# Runtime Errors

- Runtime errors are errors that cause a program to terminate abnormally.

- They occur while a program is running if the Python interpreter detects an operation that is impossible to carry out.

- Input mistakes typically cause runtime errors.

# Runtime Errors

- An input error occurs when the user enters a value that the program cannot handle.

- For instance, if the program expects to read in a number, but instead the user enters a string of text, this causes data-type errors to occur in the program.
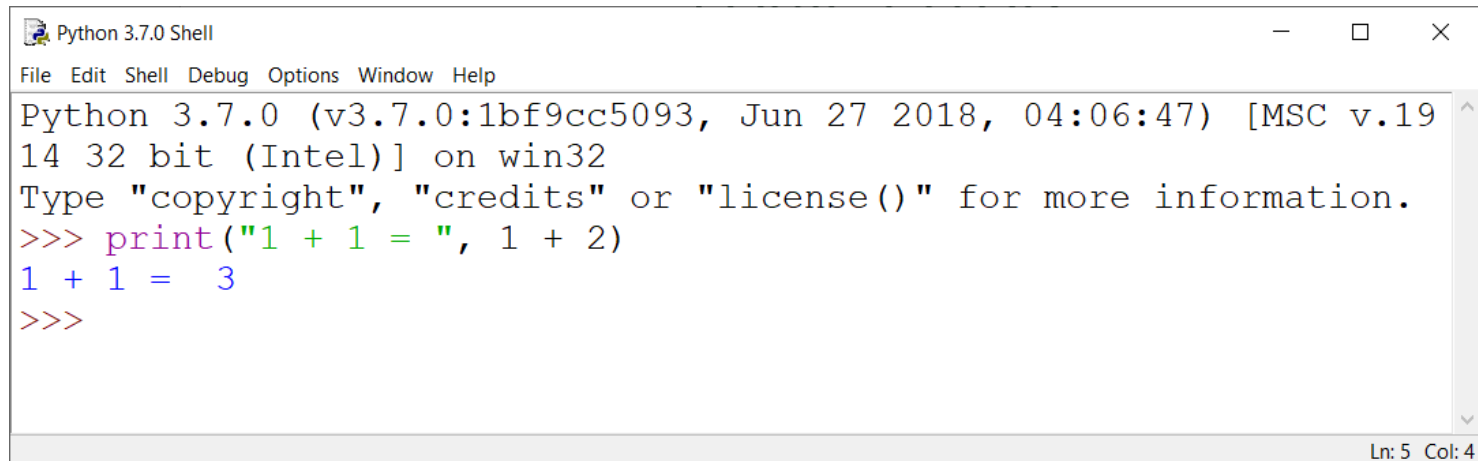
```
Python 3.7.0 Shell                                               —    □    ×
File  Edit  Shell  Debug  Options  Window  Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> x = input("Enter the value of x: ")
Enter the value of x: ahmad
>>> print(x + 1)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print(x + 1)
TypeError: can only concatenate str (not "int") to str
>>>
                                                        Ln: 10  Col: 4
```

# Logic Errors

- Logic errors occur when a program does not perform the way it was intended to.

- Logic errors produce unintended, incorrect or undesired output or other behavior, although it may not immediately be recognized as such.

- In fact, they do not cause the program to terminate abnormally.

# Logic Errors
## Example

The following is a program that converts a temperature (35 degrees) from Fahrenheit to Celsius. $T_{(°C)} = \dfrac{5}{9} \times (T_{(°F)} - 32)$

LISTING 1.4 ShowLogicErrors.py

```
1  # Convert Fahrenheit to Celsius
2  print("Fahrenheit 35 is Celsius degree ")
3  print(5 / 9 * 35 - 32)
```

**Run**

```
Fahrenheit 35 is Celsius degree
-12.55555555555554
```

The produced result (-12.55) is not correct. The correct result is (1.66).

- Replace the expression 5 / 9 * 35 − 32 with 5 / 9 * (35 − 32) to get the correct result.

- That is, you need to add parentheses around (35 − 32) so Python will calculate that expression first before doing the division.

# Notes

- In Python, syntax errors are actually treated like runtime errors because they are detected by the interpreter when the program is executed.

- In general, syntax and runtime errors are easy to find and easy to correct, because Python gives indications as to where the errors came from and why they are wrong.

- Finding logic errors, on the other hand, can be very challenging.

# Check Point
# #3

If you forget to put a closing quotation mark on a string, **what kind of error will be raised?**

➢ Answer: Syntax Error

If your program needs to read data from a file, but the file does not exist, an error would occur when running this program. **What kind of error is this?**

➢ Answer: Runtime Error

Suppose you write a program for computing the perimeter of a rectangle and you mistakenly write your program so that it computes the area of a rectangle. **What kind of error is this?**

➢ Answer: Logic Error

# End

- Test Questions

- Programming Exercises

# Test Questions

- Do the test questions for this chapter online at
  https://liveexample-ppe.pearsoncmg.com/selftest/selftestpy?chapter=1

# Programming Exercises

- Page 27 – 29:
  - 1.1 – 1.11

- [Lab #1](#)

- [Lab #2](#)