

INTRODUCTION TO PROGRAMMING WITH PYTHON

Chapter 1

Introduction to programming with Python

- Computer Programs
- Programming Languages
- Interpreting Source Code
- Compiling Source Code
- What is Python
- Python's History
- Python Version
- Install Python
- PyCharm IDE

Computer 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** 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

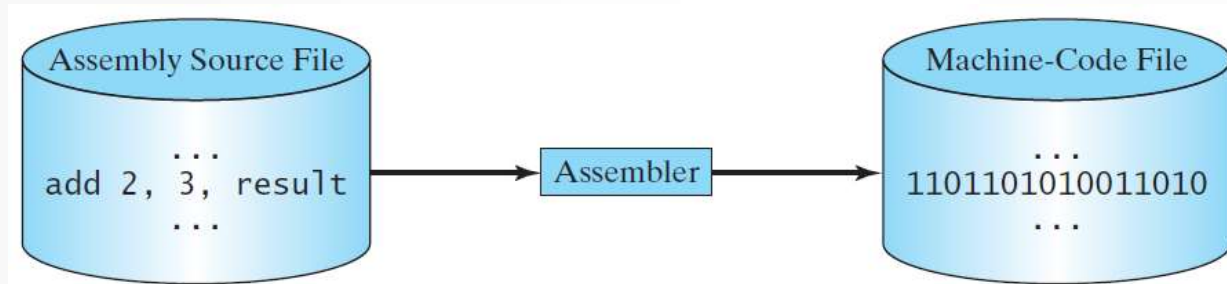
Programming Languages

- **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.

Programming Languages

- 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:

area = 5 * 5 * 3.1415

Programming Languages

TABLE 1.1 Popular High-Level Programming Languages

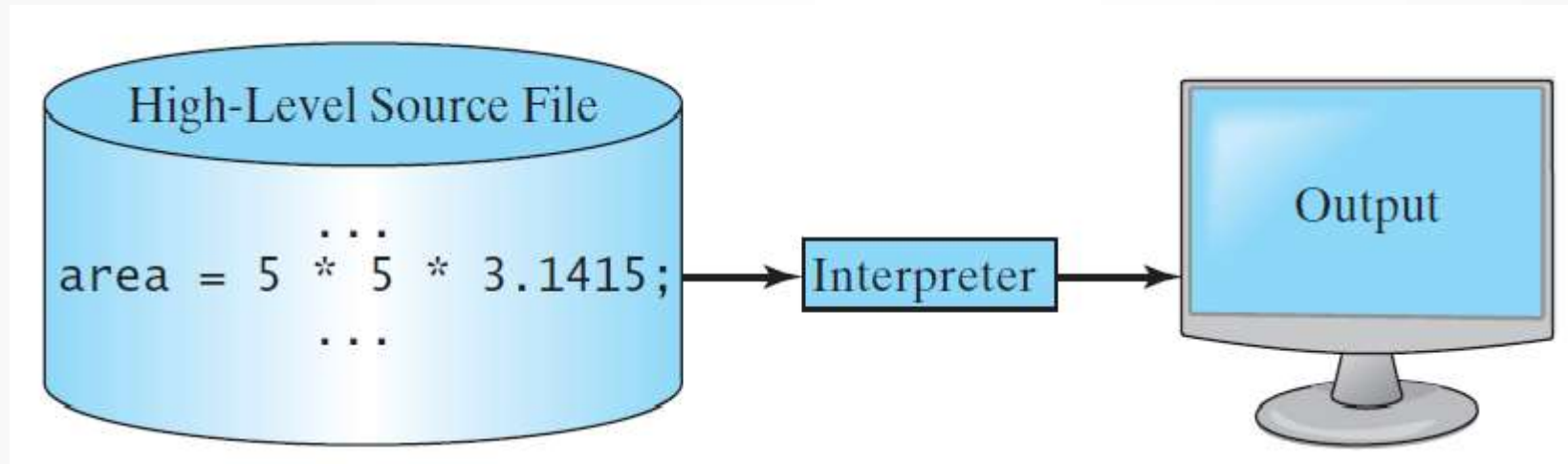
<i>Language</i>	<i>Description</i>
Ada	Named for Ada Lovelace, who worked on mechanical general-purpose computers. The Ada language was developed for the Department of Defense and is used mainly in defense projects.
BASIC	Beginner's All-purpose Symbolic Instruction Code. It was designed to be learned 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.

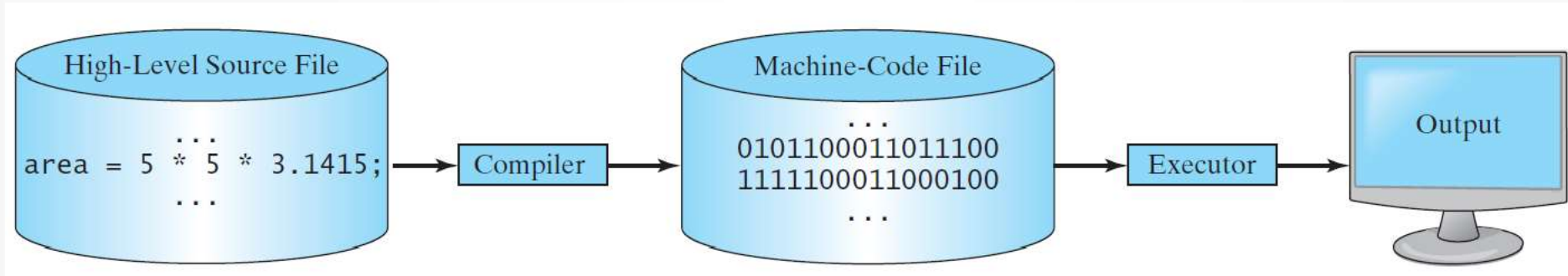
Interpreting Source Code

- 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.



Compiling Source Code

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



What is Python?

- 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.
- Python is interpreted.
- Python is an object-oriented programming language.
- 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 Version

- 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 .

Install Python

- Go to www.python.org/downloads and then download and install the last version of **Python 3.7.x** for your operating system.

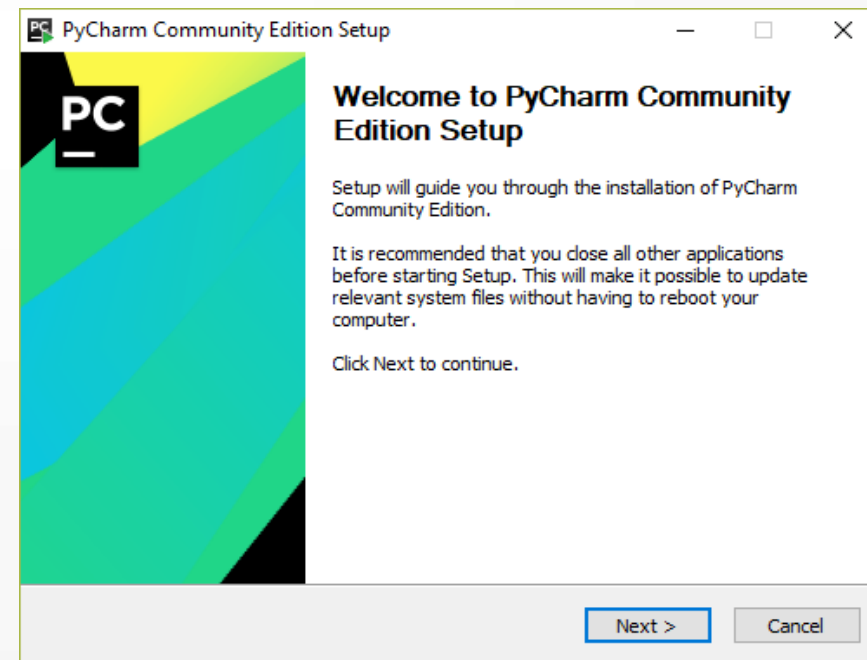


PyCharm IDE

- 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/> and then download and install “Community” version.



END



A SIMPLE PYTHON PROGRAM

Chapter 2

Introduction to programming with Python

- Welcome Message Program
- Run the program
- Statement
- Indentation
- Comment
- Syntax Errors
- Python Version
- Runtime Errors
- Logic Errors

Welcome Message Program

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
```

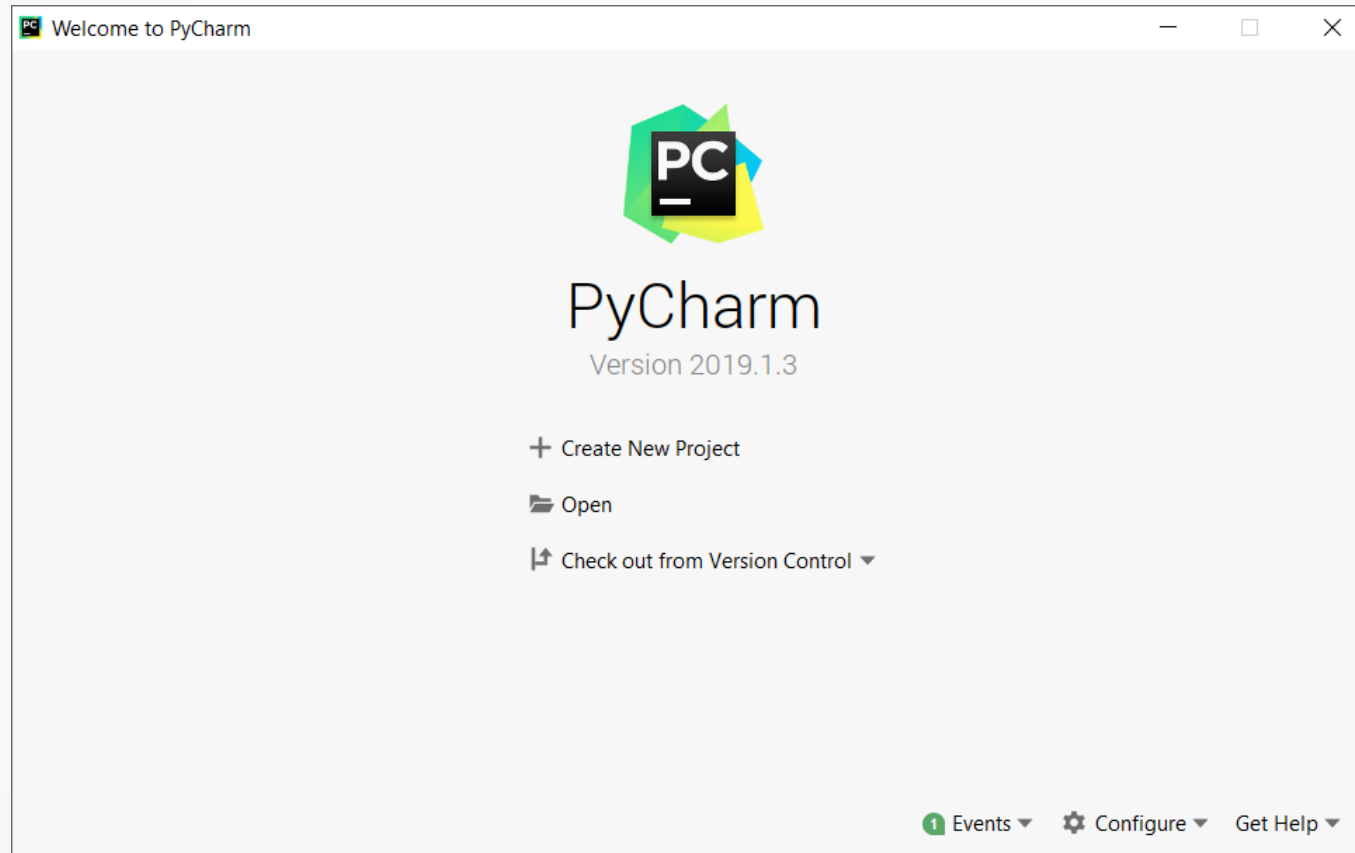
➤ The Solution:

LISTING 1.1 Welcome.py

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

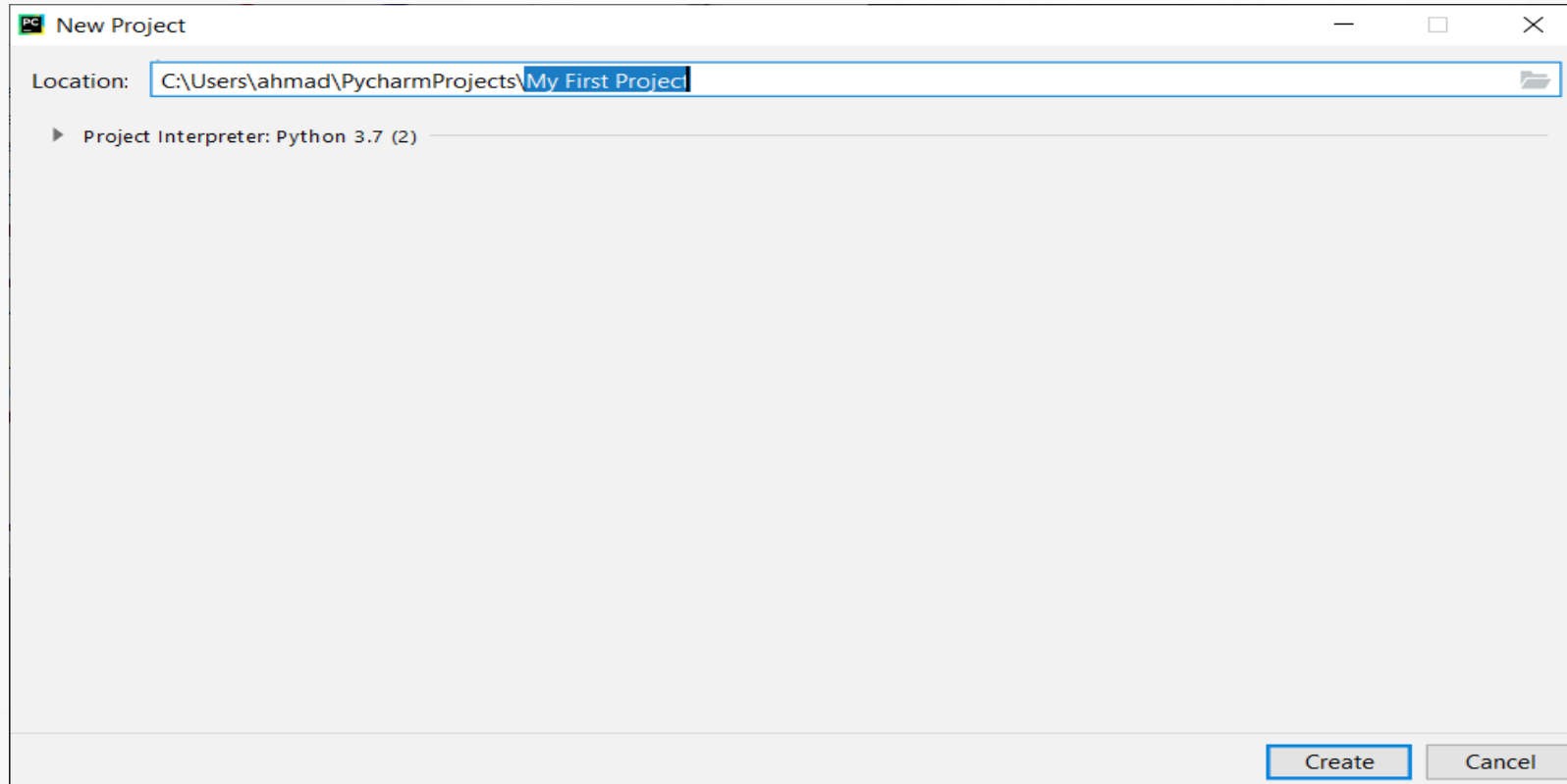
Welcome Message Program

- First step: open PyCharm and click on “Create New Project”.



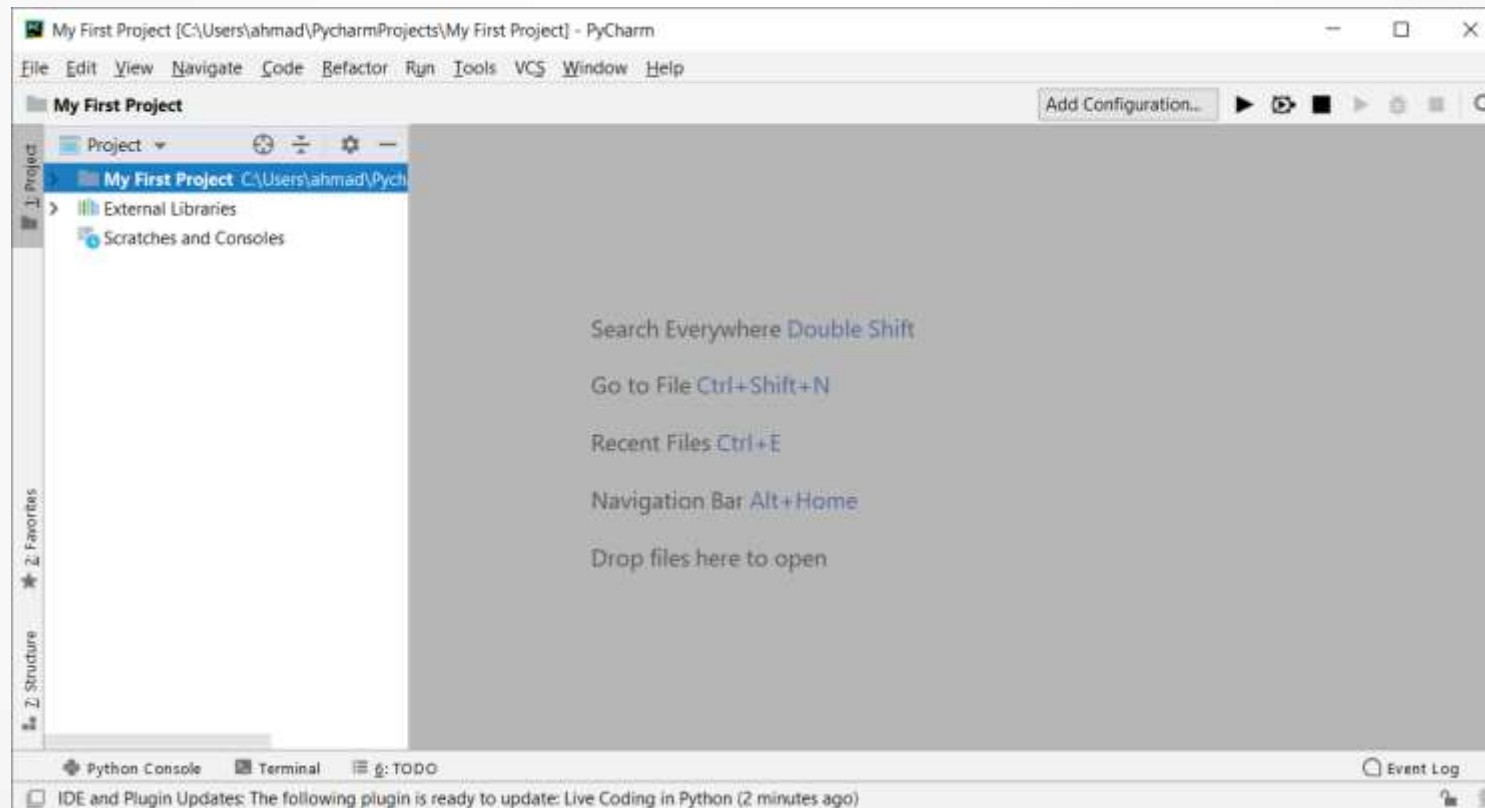
Welcome Message Program

- Then, change the default name of the project “untitled1”. For example, name it as “My First Project”, and then click on “Create”.



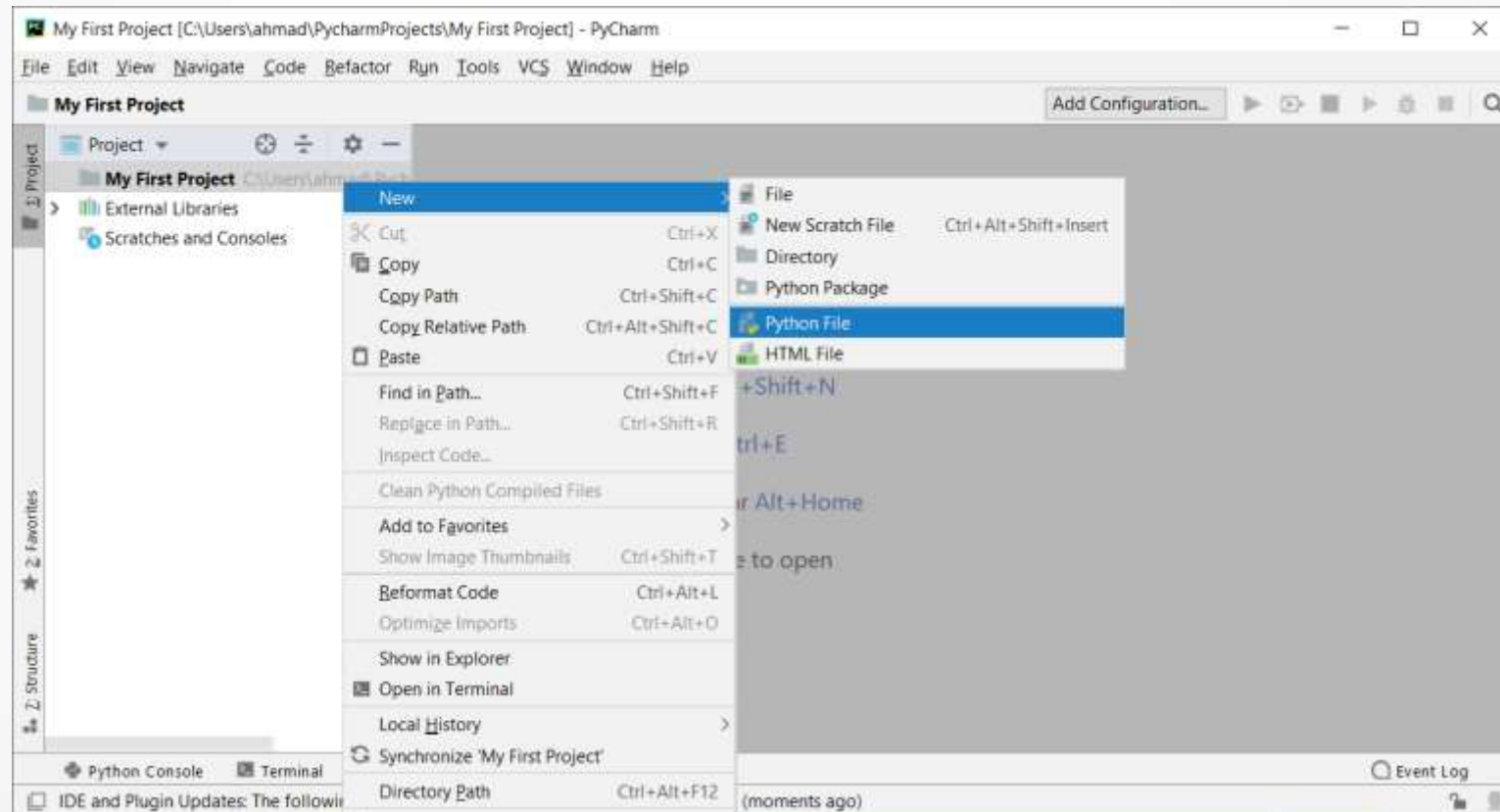
Welcome Message Program

- 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.



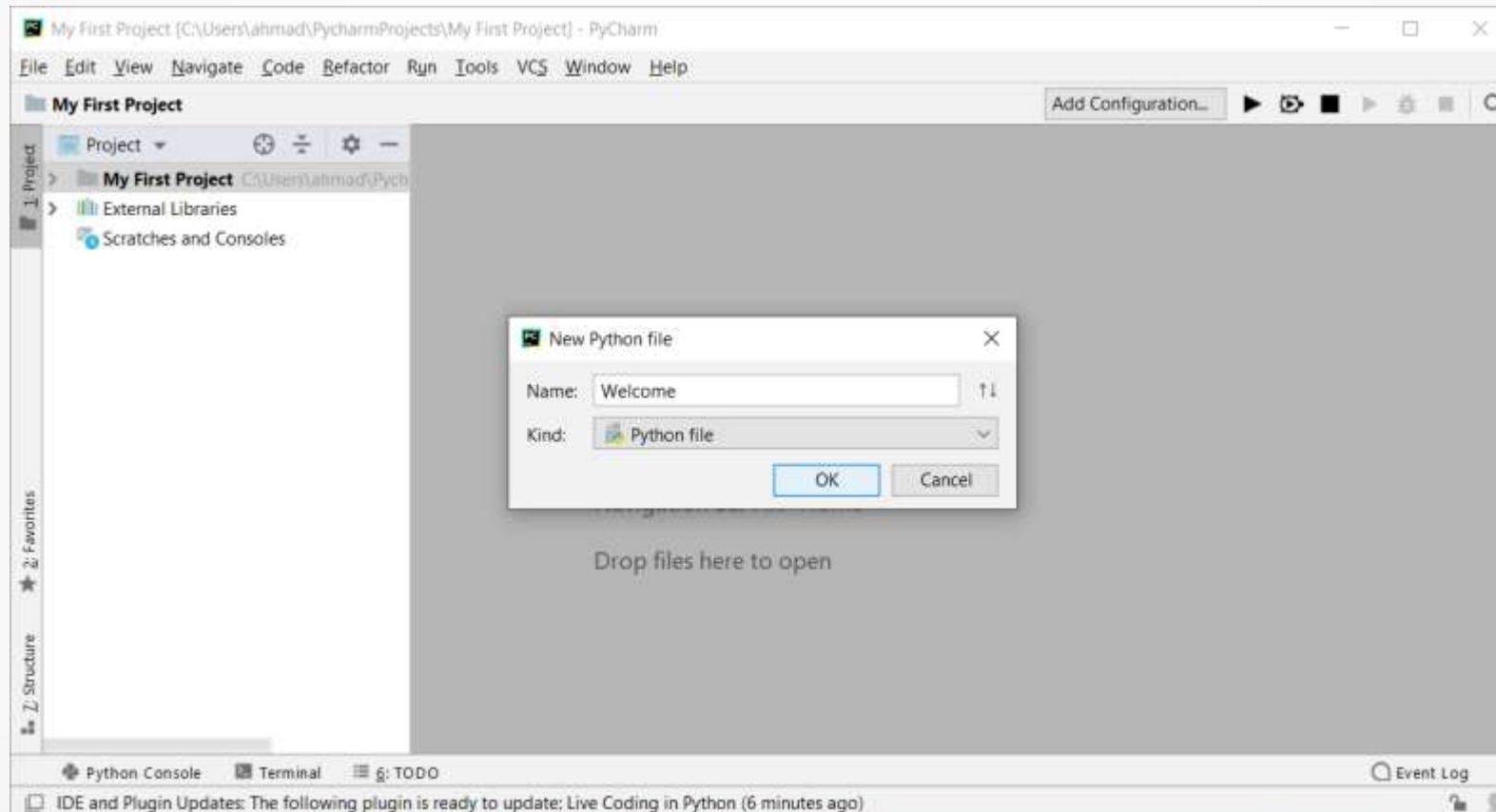
Welcome Message Program

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



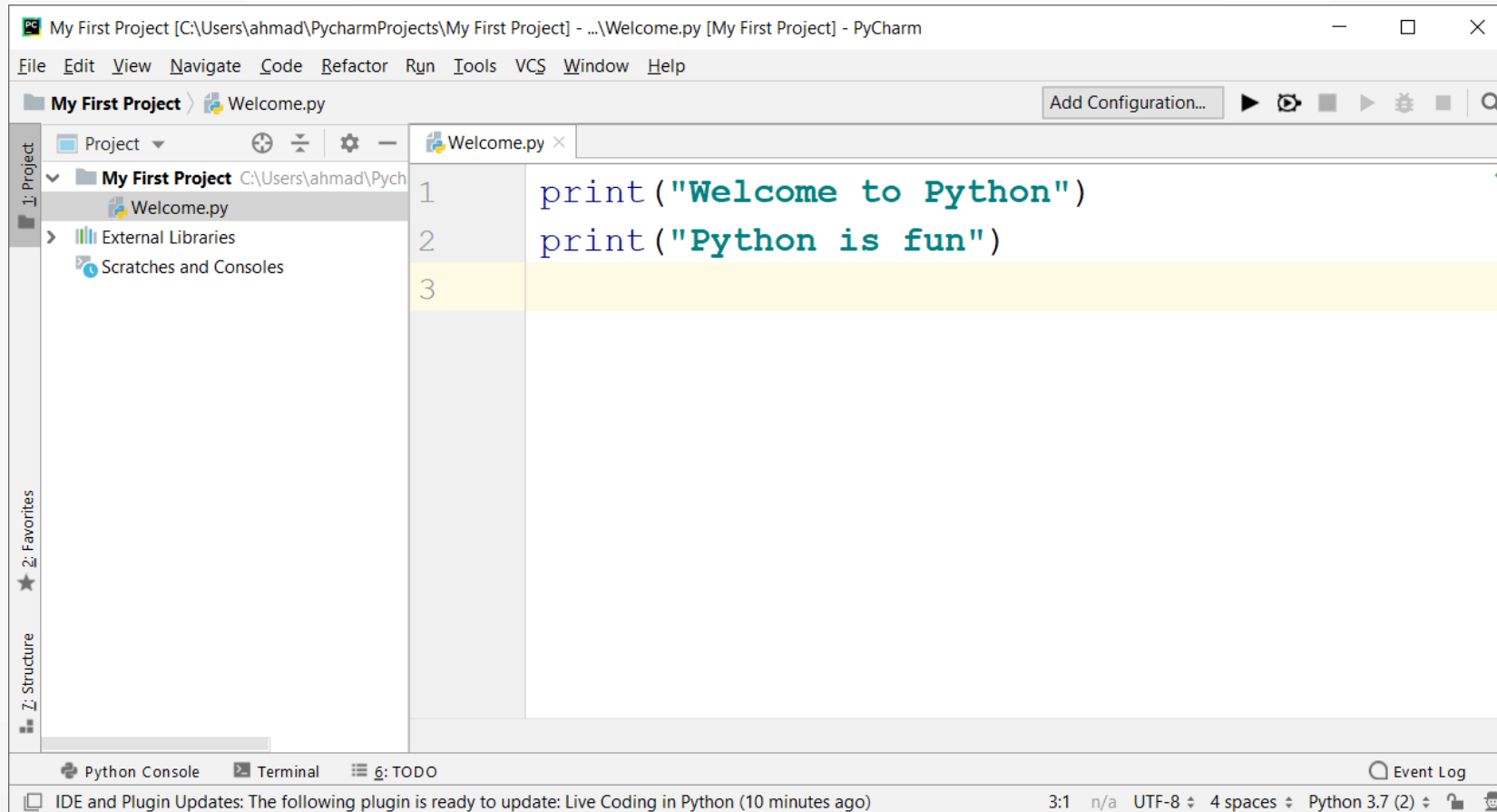
Welcome Message Program

Then, name the new file “**Welcome**”, and click on “**OK**”.



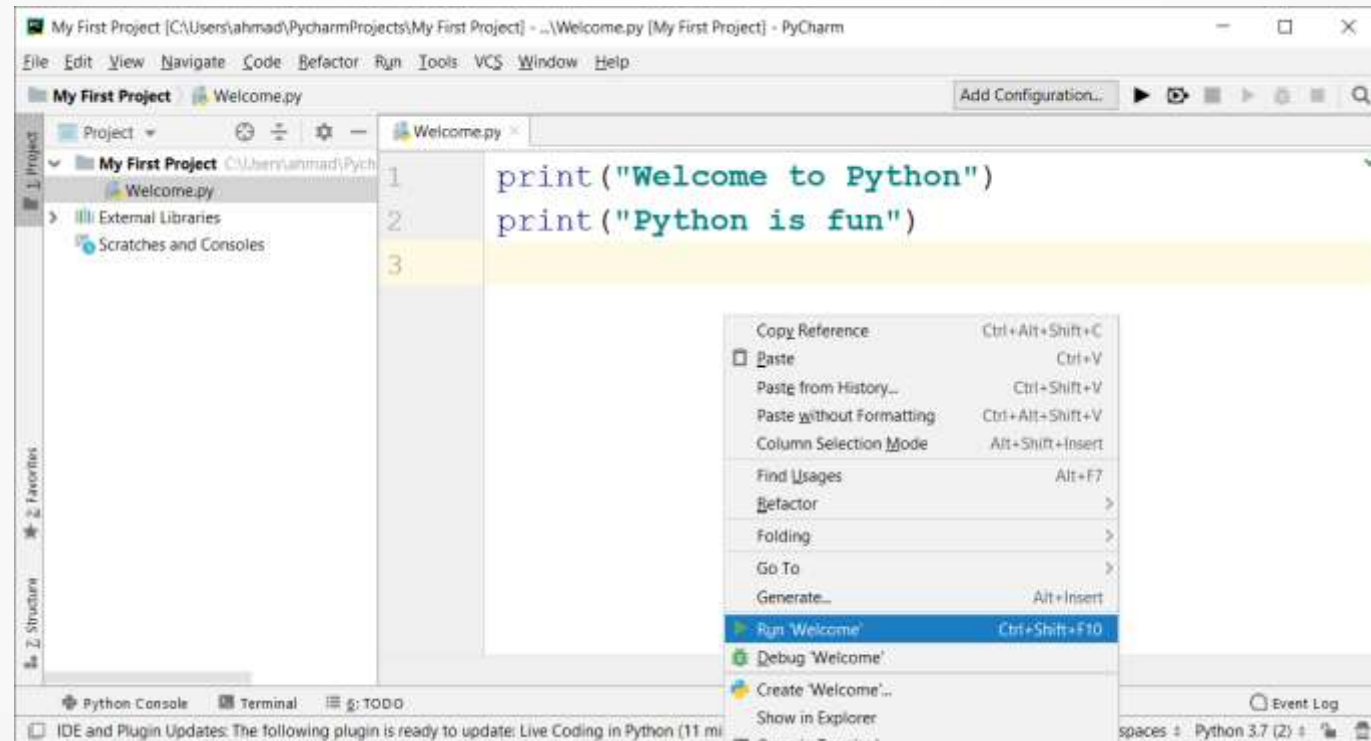
Welcome Message Program

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



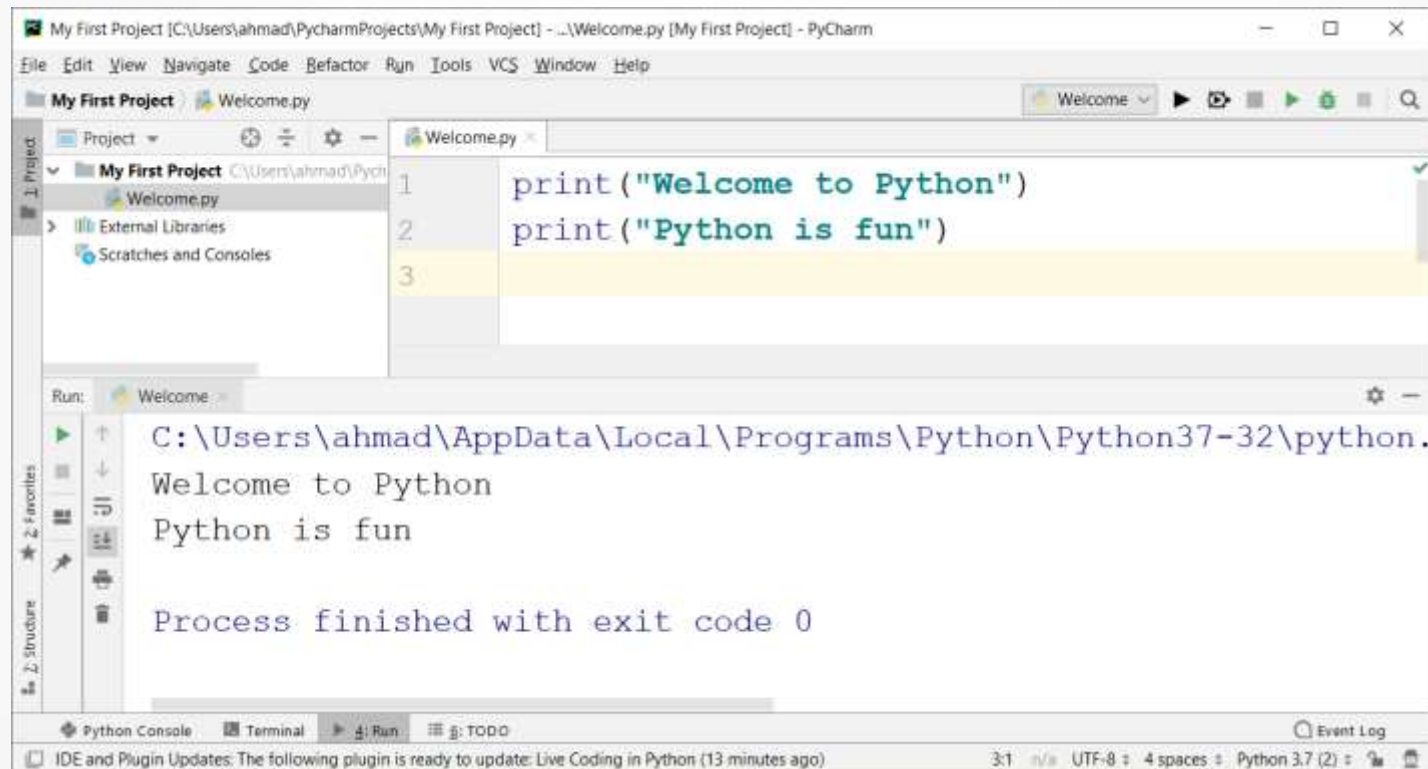
Running The Code

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



Running The Code

- After that, PyCharm is going to run the file using the Python interpreter, and then display the output of the file for you.



Program 2

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

➤ The Solution:

LISTING 1.3

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

➤ The output:



0.39759036144578314

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

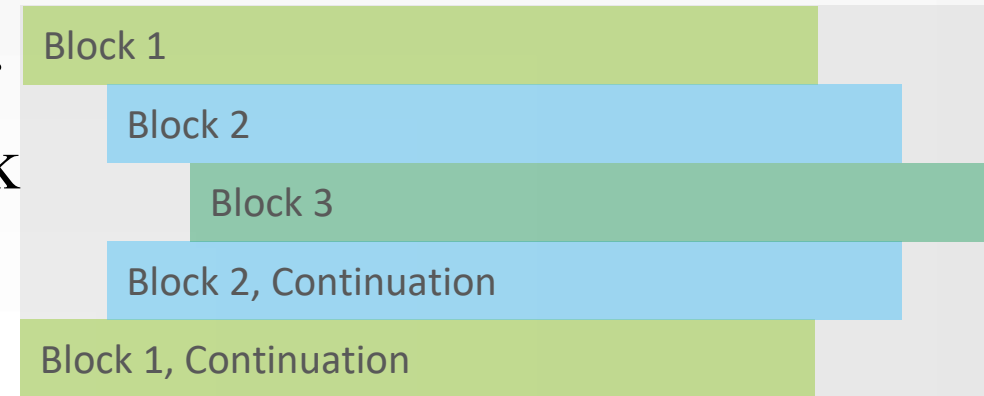
```
1 # Display two messages
2 print("Welcome to Python")
3 print("Python is fun")
```

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.



- 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**.

Comment

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

END



DATA TYPE AND OPERATORS

Chapter 3

Introduction to programming with Python

- Data Types
- Numeric Operators
 - Binary Operator
 - Float Division Operator
 - Integer Division Operator
 - Exponentiation Operator
 - Remainder Operator
- Scientific Notation
- Arithmetic Expressions
- How to Evaluate an Expression

Data Types

- A variable represents a value stored in the computer's memory.
- Every variable has a name and value.
- Every value has a data type, and the data type is to specify what type of the value are being used, such as integers or strings (text characters).
- In many programming languages such as Java, you have to define the type of the variable before you can use it. You don't do this in Python.
- However, Python automatically figures out the data type of a variable according to the value assigned to the variable.

Python Data Types

- Python provides basic (built-in) data types for integers, real numbers, string, and Boolean types.
- Here are some examples of different types of values stored in different variables:

```
var1 = 25          # Integer
var2 = 25.8        # Float
var3 = "Ahmad"     # String
var4 = 'Python'    # String
var5 = True        # Boolean
```


Numeric Data Types

- The information stored in a computer is generally referred to as data.
- There are two types of numeric data: integers and real numbers.
- Integer types (int for short) are for representing whole numbers.
- Real types are for representing numbers with a fractional part.
- Inside the computer, these two types of data are stored differently.
- Real numbers are represented as floating-point (or float) values.

Numeric Operators

TABLE 2.1 Numeric Operators

<i>Name</i>	<i>Meaning</i>	<i>Example</i>	<i>Result</i>
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Float Division	1 / 2	0.5
//	Integer Division	1 // 2	0
**	Exponentiation	4 ** 0.5	2.0
%	Remainder	20 % 3	2

Binary Operator

- The +, -, and * operators are straightforward, but note that the + and - operators can be both unary and binary.
- A unary operator has only one operand; a binary operator has two.
- For example, the - operator in -5 is a unary operator to negate the number 5, whereas the - operator in 4 - 5 is a binary operator for subtracting 5 from 4.

```
e1 = -10 + 50    # 40
e2 = -10 + -50   # -60
e3 = +10 + +20   # 30
e4 = +10++20     # 30
e5 = 10++20      # 30
e6 = -20--30     # 10
```

Float Division (/) Operator

- The / operator performs a float division that results in a floating number. For example:



```
>>> 4 / 2  
2.0  
>>> 2 / 4  
0.5  
>>>
```

Integer Division (//) Operator

- The // operator performs an integer division; the result is an integer, and any fractional part is truncated. For example:



```
>>> 5 // 2
2
>>> 2 // 4
0
>>>
```

Exponentiation (**) Operator

- To compute a^b (a with an exponent of b) for any numbers a and b , you can write `a ** b` in Python. For example:

```
>>> 2.3 ** 3.5
18.45216910555504
>>> (-2.5) ** 2
6.25
>>>
```



Remainder (%) Operator

- The % operator, known as remainder or modulo operator, yields the remainder after division.
- The left-side operand is the dividend and the right-side operand is the divisor.

- Examples:

$$7 \% 3 = 1$$
$$26 \% 8 = 2$$

$$3 \% 7 = 3$$
$$20 \% 13 = 7$$

$$12 \% 4 = 0$$

$\begin{array}{r} 2 \\ 3 \overline{) 7} \\ \underline{6} \\ 1 \end{array}$	$\begin{array}{r} 0 \\ 7 \overline{) 3} \\ \underline{0} \\ 3 \end{array}$	$\begin{array}{r} 3 \\ 4 \overline{) 12} \\ \underline{12} \\ 0 \end{array}$	$\begin{array}{r} 3 \\ 8 \overline{) 26} \\ \underline{24} \\ 2 \end{array}$	Divisor \longrightarrow	$\begin{array}{r} 1 \longleftarrow \text{Quotient} \\ 13 \overline{) 20} \longleftarrow \text{Dividend} \\ \underline{13} \\ 7 \longleftarrow \text{Remainder} \end{array}$
--	--	--	--	---------------------------	---

Remainder (%) Operator

- Remainder is very useful in programming.
- For example, an even number % 2 is always 0
- An odd number % 2 is always 1
- So you can use this property to determine whether a number is even or odd.
- You can also mod by other values to achieve valuable results.



```
>>> 100 % 2
0
>>> 99 % 2
1
>>>
```


Scientific Notation

- Floating-point literals can also be specified in scientific notation.
- Example:
 - $1.23456e+2$, same as $1.23456e2$, is equivalent to 123.456
 - and $1.23456e-2$ is equivalent to 0.0123456
 - E (or e) represents an exponent and it can be either in lowercase or uppercase.



```
>>> 20e2
2000.0
>>> 123.456e3
123456.0
>>> 20e3
20000.0
```



```
>>> 123.456e2
12345.6
>>> 123.456e-2
1.23456
>>>
```

Arithmetic Expressions

- Python expressions are written the same way as normal arithmetic expressions.
- Example:

$$\frac{3 + 4x}{5} - \frac{10(y - 5)(a + b + c)}{x} + 9 \left(\frac{4}{x} + \frac{9 + x}{y} \right)$$

- It is translated into:

```
((3 + (4 * x)) / 5) - ((10 * (y - 5) * (a + b + c)) / x) + (9 * ((4 / x) + ((9 + x) / y)))
```

Zoom In

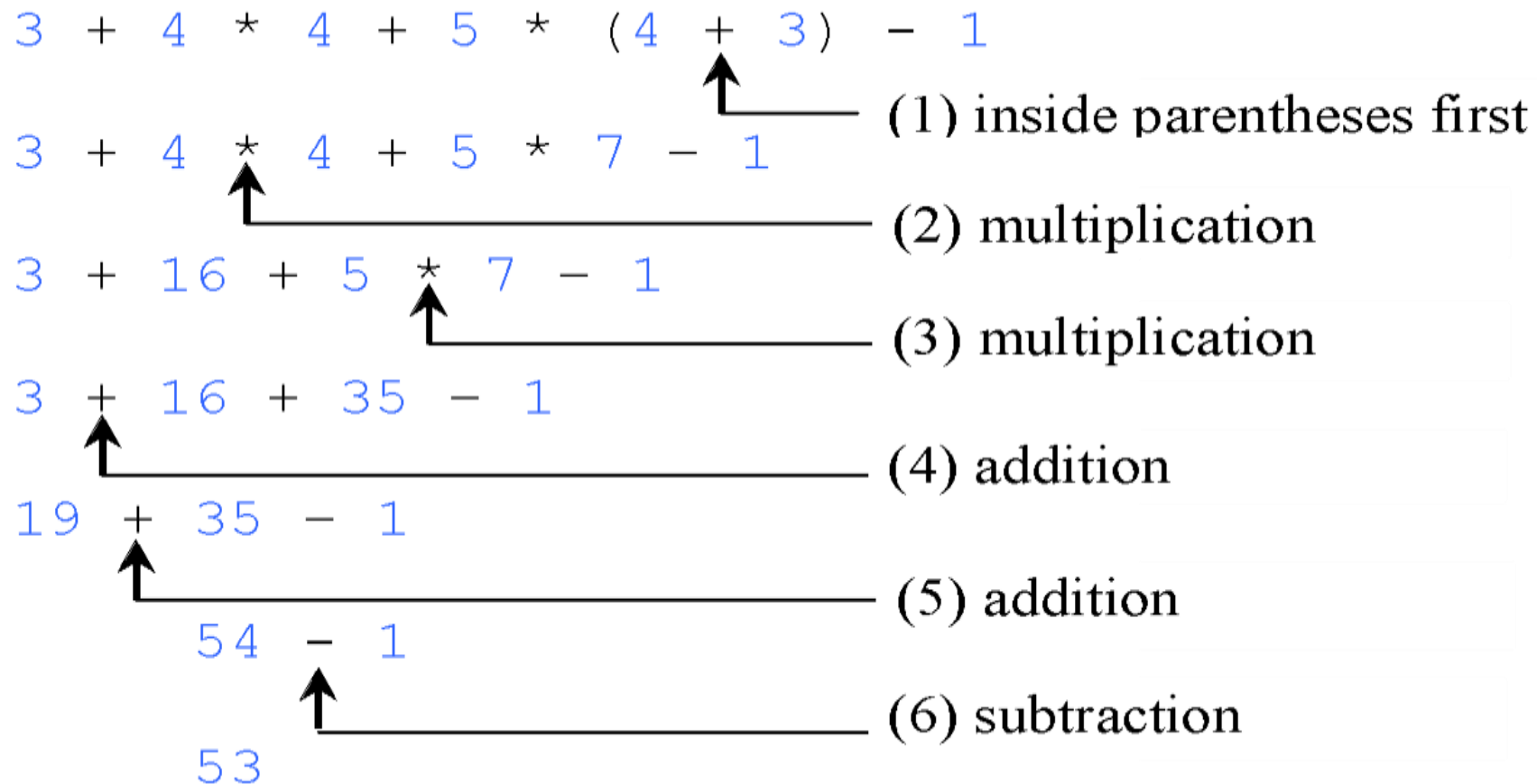
```
((3 + (4 * x)) / 5) - ((10 * (y - 5) * (a + b + c)) /  
x) + (9 * ((4 / x) + ((9 + x) / y)))
```

How to Evaluate an Expression

- You can safely apply the arithmetic rule for evaluating a Python expression.
- Operators inside parenthesis are evaluated first.
 - Parenthesis can be nested
 - Expression in inner parenthesis is evaluated first
- Use operator precedence rule.
 - Exponentiation (**) is applied first.
 - Multiplication (*), float division (/), integer division (//) , and remainder operators (%) are applied next.
 - If an expression contains several multiplication, division, and remainder operators, they are applied from left to right.
 - Addition (+) and subtraction (-) operators are applied last.
 - If an expression contains several addition and subtraction operators, they are applied from left to right.

How to Evaluate an Expression

- Example of how an expression is evaluated:



How to Evaluate an Expression

How would you write the following arithmetic expression in Python?

$$\frac{4}{3(r + 34)} - 9(a + bc) + \frac{3 + d(2 + a)}{a + bd}$$

➤ Solution:

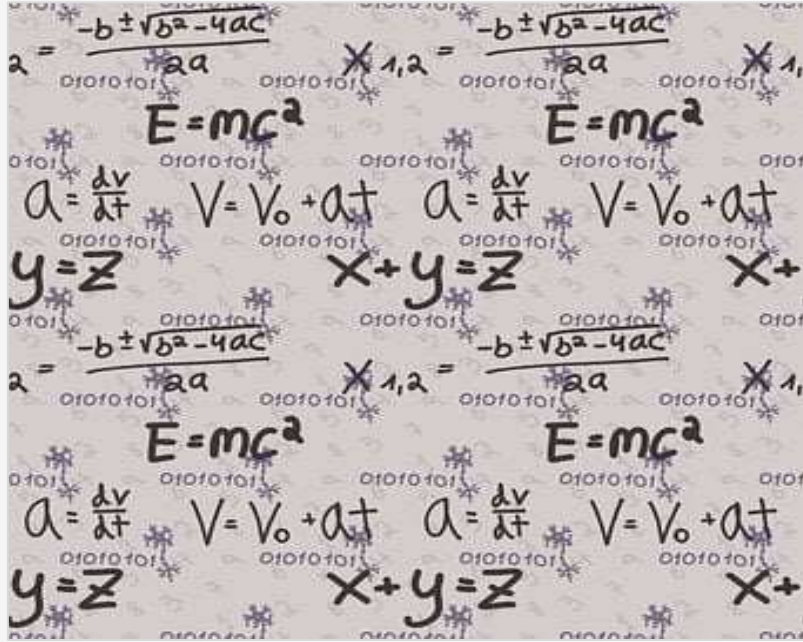
```
(4 / (3 * (r + 34))) - (9 * (a + (b * c))) + ((3 + (d * (2 + a))) / (a + (b * d)))
```

How to Evaluate an Expression

Suppose m and r are integers. Write a Python expression for mr^2 .

➤ Solution:

```
m * (r ** 2)
```



PYTHON VARIABLES

Chapter 4

Introduction to programming with Python

- Variables
- Assignment Statements
- Expression
- Assigning a Value to Multiple Variables
- Scope of Variables
- Augmented Assignment Operators
- Type Conversions

Variables

- Variables are used to reference (represent) values that may be changed in the program.
- They are called variables because their values can be changed!

Variables

- For example, see the following code:

```
# Compute the first area
radius = 1.0
area = radius * radius * 3.14159
print("The area is", area, "for radius", radius)

# Compute the second area
radius = 2.0
area = radius * radius * 3.14159
print("The area is", area, "for radius", radius)
```

Discussion:

`radius` is initially 1.0 (line 2)

then changed to 2.0 (line 7)

`area` is set to 3.14159 (line 3)

then reset to 12.56636 (line 8)

Assignment Statements

- The statement for assigning a value to a variable is called an assignment statement.
- In Python, the equal sign (=) is used as the assignment operator. The syntax for assignment statements is as follows:

```
variable = value
```

- Or

```
variable = expression
```

Expression

- An expression represents a computation involving values, variables, and operators that, taken together, evaluate to a value.
- For example, consider the following code:

```
1 y = 1 # Assign 1 to variable y
2 radius = 1.0 # Assign 1.0 to variable radius
3 x = 5 * (3 / 2) + 3 * 2 # Assign the value of the expression to x
4 x = y + 1 # Assign the addition of y and 1 to x
5 area = radius * radius * 3.14159 # Compute area
```

- You can use a variable in an expression.

Expression

- A variable can also be used in both sides of the = operator.
- For example:

```
x = x + 1
```

- In this assignment statement, the result of $x + 1$ is assigned to x . If x is 1 before the statement is executed, then it becomes 2 after the statement is executed.
- If x is not created before, Python will report an error.

Assigning a Value To Multiple Variables

- If a value is assigned to multiple variables, you can use a syntax like this:

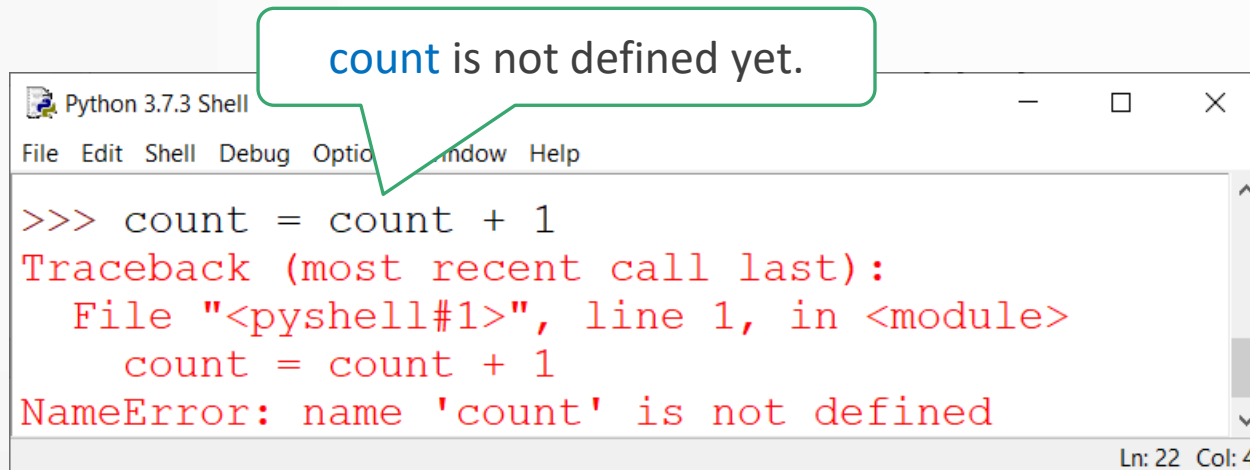
```
i = j = k = 1
```

- which is equivalent to

```
k = 1  
j = k  
i = j
```

Scope of Variables

- Every variable has a scope.
- The scope of a variable is the part of the program where the variable can be referenced (used).
- A variable must be created before it can be used.
- For example, the following code is wrong:



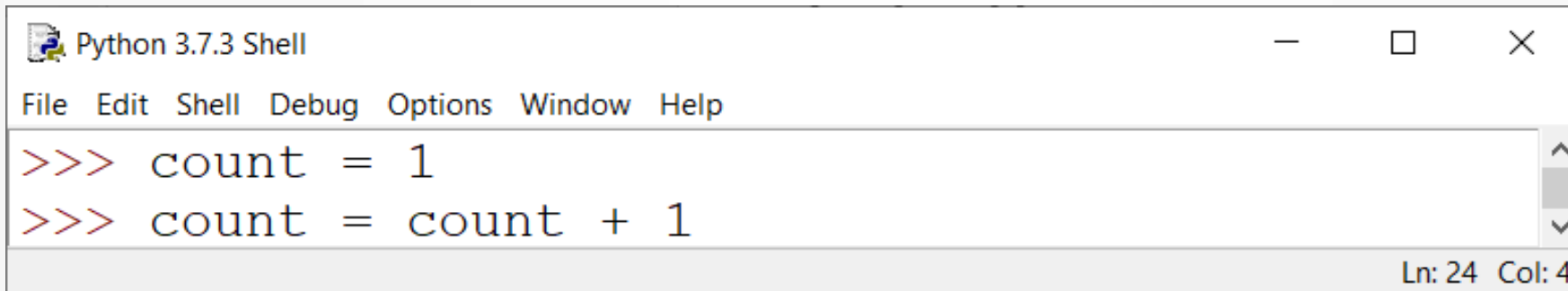
```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help

>>> count = count + 1
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    count = count + 1
NameError: name 'count' is not defined

Ln: 22 Col: 4
```

Scope of Variables

- To fix the previous example, you may write the code like this:



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> count = 1
>>> count = count + 1
Ln: 24 Col: 4
```


Scope of Variables

Assume that $a = 1$ and $b = 2$. What is a and b after the following statement?

```
a, b = b, a
```

Answer:

$a = 2$

$b = 1$

Augmented Assignment Operators

- Very often the current value of a variable is used, modified, and then reassigned back to the same variable.
- For example, the following statement increases the variable count by 1:

```
count = count + 1
```

- Python allows you to combine assignment and addition operators using an augmented (or compound) assignment operator.
- For example:

```
count += 1
```

Augmented Assignment Operators

TABLE 2.2 Augmented Assignment Operators

<i>Operator</i>	<i>Name</i>	<i>Example</i>	<i>Equivalent</i>
<code>+=</code>	Addition assignment	<code>i += 8</code>	<code>i = i + 8</code>
<code>-=</code>	Subtraction assignment	<code>i -= 8</code>	<code>i = i - 8</code>
<code>*=</code>	Multiplication assignment	<code>i *= 8</code>	<code>i = i * 8</code>
<code>/=</code>	Float division assignment	<code>i /= 8</code>	<code>i = i / 8</code>
<code>//=</code>	Integer division assignment	<code>i //= 8</code>	<code>i = i // 8</code>
<code>%=</code>	Remainder assignment	<code>i %= 8</code>	<code>i = i % 8</code>
<code>**=</code>	Exponent assignment	<code>i **= 8</code>	<code>i = i ** 8</code>

Augmented Assignment Operators

- The augmented assignment operator is performed last after all the other operators in the expression are evaluated.
- Example:

```
x /= 4 + 5.5 * 1.5
```

is same as

```
x = x / (4 + 5.5 * 1.5)
```

Augmented Assignment Operators

Assume that $a = 1$, and that each expression is independent. What are the results of the following expressions?

- $a += 4$ \longrightarrow 5
- $a -= 4$ \longrightarrow -3
- $a *= 4$ \longrightarrow 4
- $a /= 4$ \longrightarrow 0.25
- $a //= 4$ \longrightarrow 0
- $a \% = 4$ \longrightarrow 1
- $a = 56 * a + 6$ \longrightarrow 62

Type Conversions

- Can you perform binary operations with operands of different numeric types?
 - Meaning, can we add an integer literal with a floating-point literal?
- **Yes.** If an integer and a float are involved in a binary operation, Python automatically converts the integer to a float value.
- Example: `3 * 4.5` is the same as `3.0 * 4.5`
- This is called type conversion.

Type Conversions

- Sometimes, it is desirable to obtain the integer part of a fractional number.
- You can use the `int(value)` function to return the integer part of a float value. For example:



```
>>> value = 5.6
>>> int(value)
5
>>>
```

- Note that the fractional part of the number is truncated, not rounded up.

Type Conversions

- The `int` function can also be used to convert an integer string into an integer.
 - For example, `int("34")` returns **34**.
- So you can use the `eval` or `int` function to convert a string into an integer. Which one is better?
- The `int` function performs a simple conversion. It does not work for a non-integer string.
 - For example, `int("3.4")` **will cause an error**.
- The `eval` function does more than a simple conversion. It can be used to evaluate an expression.
 - For example, `eval("3 + 4")` returns **7**.

Type Conversions

- You can use the `str(value)` function to convert the numeric value to a string. For example:



```
>>> value = 5.6
>>> str(value)
'5.6'
>>> value
5.6
>>> value = str(value)
>>> value
'5.6'
```

- Note: The functions `str` does not change the variable being converted.



READING INPUT FROM THE CONSOLE

Chapter 5

Introduction to programming with Python

- User Input
- Program 1: Compute Area
- Program 2: Compute Average
- Identifiers
- Python Keywords

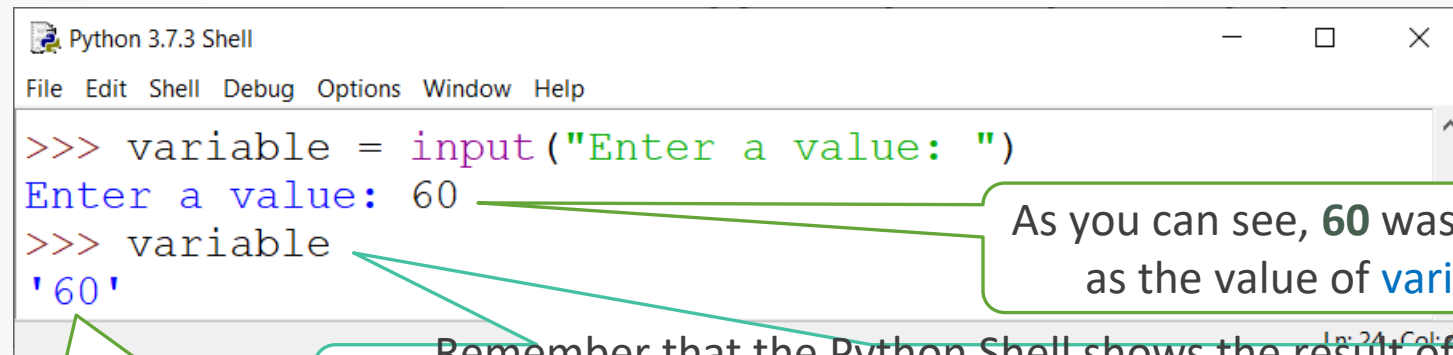
User Input

- Python uses the input function for console input.
- The input function is used to ask the user to input a value, and then return the value entered as a string (string data type).
- The following statement prompts (asks) the user to enter a value, and then it assigns the value to the variable:

```
variable = input("Enter a value: ")
```

User Input

- The example of the output of the previous statement is shown in the following figure (Python Shell – Interactive mode).



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> variable = input("Enter a value: ")
Enter a value: 60
>>> variable
'60'
```

As you can see, **60** was entered as the value of **variable**.

Remember that the Python Shell shows the result of the executed expression automatically even you didn't use the **print** function.

After showing up the value of **variable**, the value (**60**) is enclosed in matching **single quotes** (**'**). This means that the value is stored as a **string** (text) not a **number** into **variable**.

User Input

- Does it matter if a numeric value is being stored as a string, not a number?
- **Yes, it does.** See the following examples:

```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> '20' + '30'
'2030'
>>> 20 + 30
50
>>> '20' + 30
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    '20' + 30
TypeError: can only concatenate str (not "int") to str
```

String + String = Concatenation of the Strings

Number + Number = Summation of the numbers

String + Number = Error (Type Error)

Ln: 11 Col: 54

User Input

- You can use eval function to evaluate and convert the passed value (string) to a numeric value.

```
1 eval("34.5")           # returns 34.5 (float)
2 eval("345")            # returns 345  (integer)
3 eval("3 + 4")          # returns 7    (integer)
4 eval("51 + (54 * (3 + 2))") # returns 321 (integer)
```

- So, to get an input (value) from the user as a number and store it into the variable x, you can write the following code:

```
x = eval(input("Enter the value of x: "))
```

- Also, you can separate the process into two lines if you would like:

```
x = input("Enter the value of x: ") # Read x as string
x = eval(x) # Convert value x to number and save it to x
```

Program 1: Compute Area

Write a program that will calculate the area of a circle.

1. Get the radius of the circle.
2. Compute the area using the following formula:
 $\text{area} = \text{radius} \times \text{radius} \times \pi$
3. Display the result

- Tip:

It's always good practice to outline your program (or its underlying problem) in the form of an algorithm.

Program 1: Compute Area

LISTING 2.2 ComputeAreaWithConsoleInput.py

```
1 # Prompt the user to enter a radius
2 radius = eval(input("Enter a value for radius: "))
3
4 # Compute area
5 area = radius * radius * 3.14159
6
7 # Display results
8 print("The area for the circle of radius " , radius , " is " , area)
```



```
Enter a value for radius: 2.5 <Enter>
The area for the circle of radius 2.5 is 19.6349375
```



```
Enter a value for radius: 23 <Enter>
The area for the circle of radius 23 is 1661.90111
```

Program 1: Compute Area

LISTING 2.2 ComputeAreaWithConsoleInput.py

```
1  # Prompt the user to enter a radius
2  radius = eval(input("Enter a value for radius: "))
3
4  # Compute area
5  area = radius * radius * 3.14159
6
7  # Display results
8  print("The area for the circle of radius " , radius , " is " , area)
```

- Line 2 prompts the user to enter a value (in the form of a string) and converts it to a number, which is equivalent to:

```
# Read input as a string
radius = input("Enter a value for radius: ")
# Convert the string to a number
radius = eval(radius)
```

- After the user enters a number and presses the <Enter> key, the number is read and assigned to radius.

Program 2: Compute Average

Write a program to get three values from the user and compute their average.



```
Enter the first number: 1 <Enter>  
Enter the second number: 2 <Enter>  
Enter the third number: 3 <Enter>  
The average of 1 2 3 is 2.0
```

Program 2: Compute Average

Write a program to get three values from the user and compute their average.

Phase 1: Design your algorithm

1. Get three numbers from the user.
2. Use the input function.
3. Compute the average of the three numbers:
$$\text{average} = (\text{num1} + \text{num2} + \text{num3}) / 3$$
4. Display the result

Program 2: Compute Average

Write a program to get three values from the user and compute their average.

Phase 2: Implementation (code the algorithm)

```
# Prompt the user to enter three numbers  
  
# Compute average  
  
# Display result
```

Program 2: Compute Average

Write a program to get three values from the user and compute their average.

Phase 2: Implementation (code the algorithm)

LISTING 2.3 ComputeAverage.py

```
1  # Prompt the user to enter three numbers
2  number1 = eval(input("Enter the first number: "))
3  number2 = eval(input("Enter the second number: "))
4  number3 = eval(input("Enter the third number: "))
5
6  # Compute average
7  average = (number1 + number2 + number3) / 3
8
9  # Display result
10 print("The average of", number1, number2, number3,
11       "is", average)
```

Program 2: Compute Average

Runs the Program



```
Enter the first number: 1 <Enter>  
Enter the second number: 2 <Enter>  
Enter the third number: 3 <Enter>  
The average of 1 2 3 is 2.0
```



```
Enter the first number: 10.5 <Enter>  
Enter the second number: 11 <Enter>  
Enter the third number: 11.5 <Enter>  
The average of 10.5 11 11.5 is 11.0
```

Program 2: Compute Average

LISTING 2.3 ComputeAverage.py to enter three numbers

```
2  number1 = eval(input("Enter the first number: "))
3  number2 = eval(input("Enter the second number: "))
4  number3 = eval(input("Enter the third number: "))
5
6  # Compute average
7  average = (number1 + number2 + number3) / 3
8
9  # Display result
10 print("The average of", number1, number2, number3,
11       "is", average)
```

- The program prompts the user to enter three integers (lines 2–4), computes their average (line 7), and displays the result (lines 10–11).
- If the user enters something other than a number, the program will terminate with a runtime error.

Program 2: Compute Average

LISTING 2.3 ComputeAverage.py

```
1  # Prompt the user to enter three numbers
2  number1 = eval(input("Enter the first number: "))
3  number2 = eval(input("Enter the second number: "))
4  number3 = eval(input("Enter the third number: "))
5
6  # Compute average
7  average = (number1 + number2 + number3) / 3
8
9  # Display result
10 print("The average of", number1, number2, number3,
11       "is", average)
```

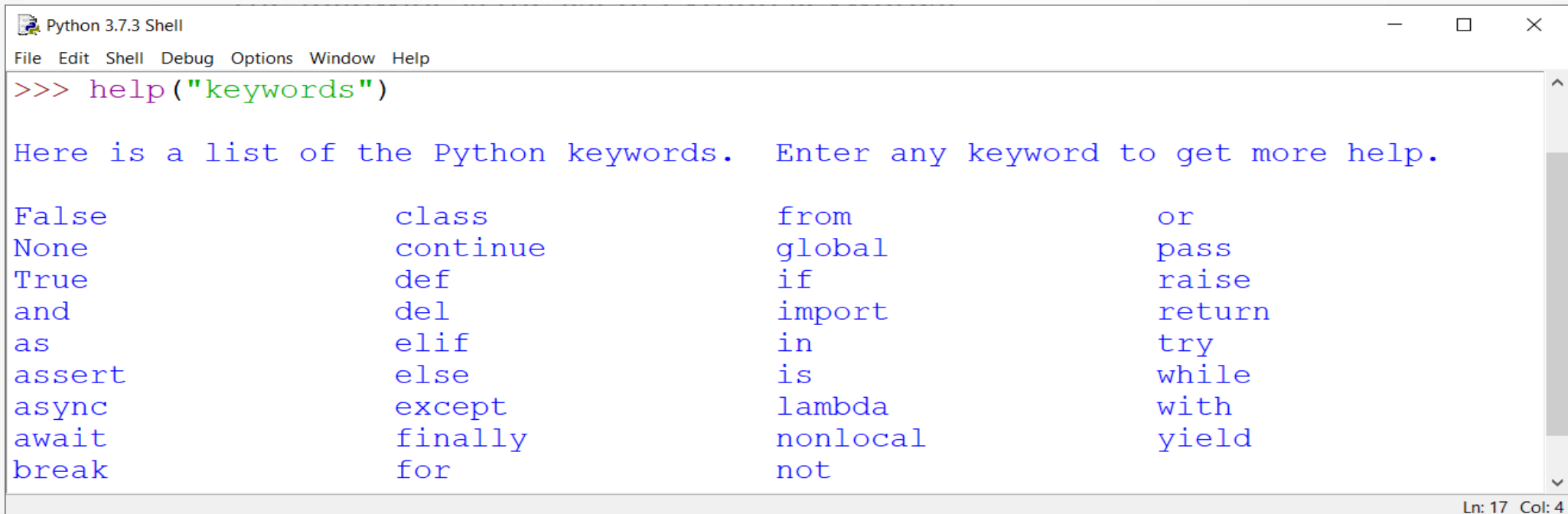
- Normally a statement ends at the end of the line.
- In Line 10, the print statement is split into two lines (lines 10–11).
- This is okay, because Python scans the print statement in line 10 and knows it is not finished until it finds the closing parenthesis in line 11.
- We say that these two lines are joined implicitly.

Identifiers

- Identifiers are the names that identify the elements such as variables and functions in a program.
- All identifiers must obey the following rules:
 - An identifier is a sequence of characters that consists of letters, digits, and underscores (_).
 - An identifier must start with a letter or an underscore. It cannot start with a digit.
 - An identifier cannot be a Keyword.
 - Keywords, also called reserved words, have special meanings in Python.
 - For example, import is a keyword, which tells the Python interpreter to import a module to the program.
 - An identifier can be of any length.

Python Keywords

- Keywords are reserved words by programming language.
- Keywords can not be used as identifiers.
- The following is the list of Python keywords:



```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
>>> help("keywords")

Here is a list of the Python keywords.  Enter any keyword to get more help.

False          class          from          or
None           continue     global        pass
True           def          if            raise
and            del          import        return
as             elif         in            try
assert         else        is            while
async          except      lambda        with
await          finally    nonlocal     yield
break          for        not

Ln: 17 Col: 4
```

Identifiers

Which of the following identifiers are valid? Which are Python keywords?

- | | | |
|----------|---|-------------|
| 1. miles | → | ✓ |
| 2. Test | → | ✓ |
| 3. a+b | → | ✗ |
| 4. b-a | → | ✗ |
| 5. 4#R | → | ✗ |
| 6. \$4 | → | ✗ |
| 7. #44 | → | ✗ |
| 8. apps | → | ✓ |
| 9. elif | → | ✗ (Keyword) |
| 10. if | → | ✗ (Keyword) |
| 11. y | → | ✓ |
| 12. iF | → | ✓ |

Comparison Operators and Conditional

- IF Condition
- Compound if statements
- Comparison Operators
- elif
- IF demo: program 1
- IF demo: program 2
- Nested Condition

IF Condition

In Python code, decisions can be made using a Boolean condition with **if** statement

This allows a programmer to control the flow of code through a program

Statements are either 100% True, or it is False - no maybe

Code either runs, or does not run

IF Condition

- If condition is true do x else do y

```
1 if age > 50:  
2     print("The name you entered is", name + ". You are", age, "years old.")  
3 else:  
4     print("The name you entered is", name + ". You are", age, "years old and still young!.")  
5
```

The colon (:) at the end of the if statement is required. Any lines of code that go with the if statement must be indented.

IF Condition

Boolean string methods can be used in if statements

```
1  name = input("What is your name?")
2  age = int(input("What is your age?"))
3
4  if name.isalnum():
5      print("The name you entered is", name + ". You are", age, "years old.")
6  else:
7      print("Please enter a valid name.")
8
```

Compound if statements

It is possible to use two conditions at the same time using an and

```
1  if name.isalnum() and not(name.isdigit()):
2      print("The name you entered is", name + ". You are", age, "years old.")
3  else:
4      print("Please enter a valid name.")
5
6
```

the **if** statement reads:

"if name is alpha numeric and not a digit"

Compound if statements

It is possible to use two conditions at the same time using an or

```
1  if age > 65 or age < 18:  
2      print("Congrats you meet the age requirements" , name + ".")  
3  else:  
4      print("You are not in the required age ranges", name + ".")  
5  
6
```

or means either condition can be TRUE and the entire statement is TRUE

If age is smaller than 18 **or** greater than 65 the statement is TRUE

Comparison Operators

Comparison operators in Python

> Greater than

< less than

>= Greater than to equal to

<= Less than to equal to

== Equals

!= Does not equal

Note: A single equals sign (=) is the **assignment operator**.
Attempting to check for equality is a syntax error in Python

Comparison Operators

In this example two strings are compared using ==

```
1  name = input("What is your name?")
2  second_name = input("What is another name?")
3
4  if name == second_name:
5      print("You typed the same name!")
6  else:
7      print("You typed different names.")
8
9
```

elif

What is there is more than one condition to test? Using an else if, many conditions can be tested.

In Python to accomplish this use the **elif** keyword.

```
1  name = input("What is your name?")
2  age = int(input("What is your age?"))
3
4
5  if age < 18:
6      print("The name you entered is", name + ". You are", age, "years old my child.")
7  elif age > 18 and age < 30:
8      print("The name you entered is", name + ". You are", age, "years old young adult.")
9  elif age > 30 and age < 50:
10     print("The name you entered is", name + ". You are", age, "years old. This is middle age.")
11 elif age > 50:
12     print("The name you entered is", name + ". You are", age, "years old and very wise.")
13
```

IF Demo: program 1

Write a program that prompts the user to enter an integer. If the number is a multiple of 5, the program displays the result **HiFive**. If the number is divisible by 2, the program displays **HiEven**.



```
Enter an integer: 4 <Enter>  
HiEven
```



```
Enter an integer: 15 <Enter>  
HiFive
```



```
Enter an integer: 30 <Enter>  
HiFive  
HiEven
```

IF Demo: program 1

LISTING 4.1 SimpleIfDemo.py

```
1 number = eval(input("Enter an integer: "))
2
3 if number % 5 == 0:
4     print("HiFive")
5
6 if number % 2 == 0:
7     print("HiEven")
```



```
Enter an integer: 4 <Enter>
HiEven
```



```
Enter an integer: 15 <Enter>
HiFive
```



```
Enter an integer: 30 <Enter>
HiFive
HiEven
```


IF Demo: program 2

Write a program that helps a first-grader practice subtraction. The program should randomly generate two single-digit integers, number1 and number2, with number1 \geq number2 and should then ask the user for the answer. The program will then display a message stating if the answer is correct. If wrong, the program should display the correct answer.



```
What is 6 - 6? 0 <Enter>  
You are correct!
```



```
What is 9 - 2? 5 <Enter>  
Your answer is wrong.  
9 - 2 is 7
```

IF Demo: program 2

LISTING 4.4 SubtractionQuiz.py

```
1  import random
2
3  # 1. Generate two random single-digit integers
4  number1 = random.randint(0, 9)
5  number2 = random.randint(0, 9)
6
7  # 2. If number1 < number2, swap number1 with number2
8  if number1 < number2:
9      number1, number2 = number2, number1 # Simultaneous assignment
10
11 # 4. Prompt the student to answer "what is number1 - number2?"
12 answer = eval(input("What is " + str(number1) + " - " +
13                     str(number2) + "? "))
14
15 # 4. Grade the answer and display the result
16 if number1 - number2 == answer:
17     print("You are correct!")
18 else:
19     print("Your answer is wrong.\n", number1, "-",
20           number2, "is", number1 - number2)
```

Nested IF

- The statement in an if or if-else statement can be any legal Python statement.
 - Including another if or if-else statement.
- The inner if statement is said to be nested inside the outer if statement.
- Example:

```
1  if i > k:
2      if j > k:
3          print("i and j are greater than k")
4  else:
5      print("i is less than or equal to k")
```

Nested IF

More details:

- The inner if statement can contain another if statement.
- In fact, there is no limit to the depth of the nesting.

So what is the purpose?

- The nested if statement can be used to implement multiple alternatives.
- Consider the following example in the next slide, which prints a letter grade according to the final number grade.

Nested IF

```
1  if score >= 90.0:
2      grade = 'A'
3  else:
4      if score >= 80.0:
5          grade = 'B'
6      else:
7          if score >= 70.0:
8              grade = 'C'
9          else:
10             if score >= 60.0:
11                 grade = 'D'
12             else:
13                 grade = 'F'
```

(a)

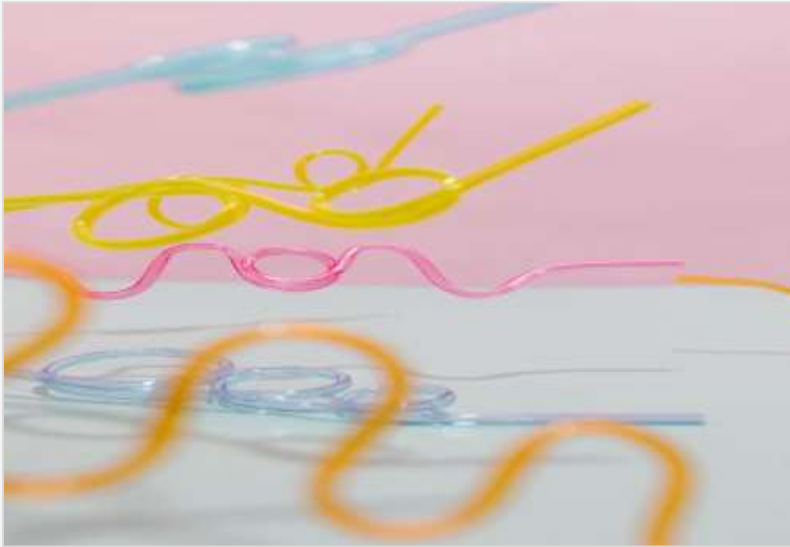
equivalent

This is better

```
1  if score >= 90.0:
2      grade = 'A'
3  elif score >= 80.0:
4      grade = 'B'
5  elif score >= 70.0:
6      grade = 'C'
7  elif score >= 60.0:
8      grade = 'D'
9  else:
10     grade = 'F'
```

(b)

- While (a) works, the preferred format for multiple alternatives is shown in (b) using a multi-way if-elif-else statement.
- This multi-way if-elif-else style avoids deep indentation and makes the program easier to read.



LOOPS

Chapter 7

Loops

- Loops
- The while loop
- The for loop
- Nested loops
- Keywords break and continue

Loops

- A loop is a construct that controls the repeated execution of a block of statements.
- The concept of looping is fundamental to programming.
- Python provides two types of loop statements:
 - while loops
 - The while loops is a condition-controlled loop.
 - it is controlled by a True/False condition.
 - for loops
 - The for loop is a count-controlled loop.
 - It repeats a specified number of times.

The while loop

- A while loop executes statements repeatedly as long as a condition remains true.
- The syntax for the while loop is:

```
while loop-continuation-condition:  
    # Loop body  
    Statement(s)
```

The while loop

- The syntax for the while loop is:

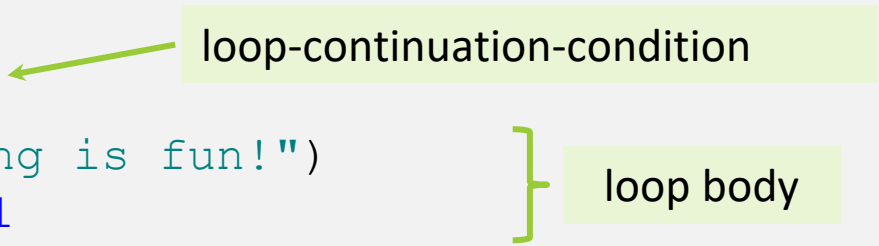
```
while loop-continuation-condition:  
    # Loop body  
    Statement(s)
```

- Each loop contains a loop-continuation-condition.
- This is a Boolean expression that controls the execution of the loop body.
- This expression is evaluated at each iteration.
 - If the result is True, the loop body is executed
 - If it is False, the entire loop will terminate
 - Program control goes to the next statement after the loop.

The while loop

- The loop that displays Programming is fun! 100 times is an example of a while loop.

```
1 count = 0
2 while count < 100:
3     print("Programming is fun!")
4     count = count + 1
```



The diagram illustrates the components of the while loop code. A green arrow points from the text 'loop-continuation-condition' to the condition 'count < 100:' on line 2. A green bracket on the right side of lines 3 and 4 is labeled 'loop body'.

- The continuation condition is `count < 100`
- If True, the loop continues.
- If False, the loop will terminate.
- This type of loop is called a counter-controlled loop.

The while loop


- Here is another example illustrating how a loop works:

```
1 sum = 0
2 i = 1
3 while i < 10:
4     sum = sum + i
5     i = i + 1
6 print("sum is", sum) # sum is 45
```

- Details:
- if $i < 10$ is True, the program adds i to sum.
- The variable i is initially set to 1.
- Then it is incremented to 2, 3, and so on, up to 10.
- When i is 10, $i < 10$ is False, and the loop exits.
- So the sum is $1 + 2 + 3 + \dots + 9 = 45$.

The while loop

- Write a program that randomly generates a number between 0 and 100, inclusive. The program will repeatedly ask the user to guess the number until the user gets the number correct. At each wrong answer, the program tells the user if the guess is too low or too high.



```
Guess a magic number between 0 and 100
Enter your guess: 50 <Enter>
Your guess is too high
Enter your guess: 25 <Enter>
Your guess is too low
Enter your guess: 42 <Enter>
Your guess is too high
Enter your guess: 39 <Enter>
Yes, the number is 39
```

The while loop

LISTING 5.3 GuessNumber.py

```
1  import random
2
3  # Generate a random number to be guessed
4  number = random.randint(0, 100)
5
6  print("Guess a magic number between 0 and 100")
7
8  guess = -1 # Initial value that doesn't meet the loop condition
9  while guess != number:
10     # Prompt the user to guess the number
11     guess = eval(input("Enter your guess: "))
12
13     if guess == number:
14         print("Yes, the number is", number)
15     elif guess > number:
16         print("Your guess is too high")
17     else:
18         print("Your guess is too low")
```

The while loop

- The program generates the random number in line 4.
- Then, it prompts the user to enter a guess continuously in a loop (lines 9–18).
- For each guess, the program determines whether the user's number is correct, too high, or too low (lines 13–18).
- When the guess is correct, the program exits the loop (line 9).
- Note that guess is initialized to -1.
- This is to avoid initializing it to a value between 0 and 100, because that could be the number to be guessed.

The for loop

- Often you will use a while loop to iterate a certain number of times.
- A loop of this type is called a counter-controlled loop.
- In general, the loop can be written as follows:

```
i = initialValue # Initialize loop-control variable
while i < endValue:
    # Loop body
    ...
    i += 1 # Adjust loop-control variable
```

- If you want to iterate a specific number of times, it is better/easier to just use a for loop.

The for loop

```
i = initialValue # Initialize loop-control variable
while i < endValue:
    # Loop body
    ...
    i += 1 # Adjust loop-control variable
```

- A for loop can be used to simplify the preceding loop:

```
for i in range(initialValue, endValue):
    # Loop body
```

- In general, the syntax of a for loop is:

```
for var in sequence:
    # Loop body
```

The for loop

```
for i in range(0, 5):  
    print(i)
```



0
1
2
3
4

The for loop

- k is used as step value in range(a, b, k).
- The first number in the sequence is a.
- Each successive number in the sequence will increase by the step value k.
- b is the limit.
- The last number in the sequence must be less than b.

- Example:

```
1 for v in range(3, 9, 2):  
2     print("v =", v)
```



```
v = 3  
v = 5  
v = 7
```

- The step value in range (3, 9, 2) is 2, and the limit is 9. So, the sequence is 3, 5, and 7

The for loop

- The `range(a, b, k)` function can count backward if `k` is negative.
- In this case, the sequence is still `a`, `a + k`, `a + 2k`, and so on for a negative `k`.
- The last number in the sequence must be greater than `b`.
- Example:

```
1 for v in range(5, 1, -1):  
2     print("v =", v)
```



```
v = 5  
v = 4  
v = 3  
v = 2
```

Nested loops

- A loop can be nested inside another loop.
- Nested loops consist of an outer loop and one or more inner loops.
- Each time the outer loop is repeated, the inner loops are reentered and started anew.
- Example:

```
1 print("Start")
2 print()
3
4 for x in range(1, 4):
5     print("----- x =", x, "-----")
6     for y in range(4, 6):
7         print("y =", y)
8     print()
9
10 print("End")
```

Outer Loop

Inner Loop



Start

```
----- x = 1 -----
y = 4
y = 5
```

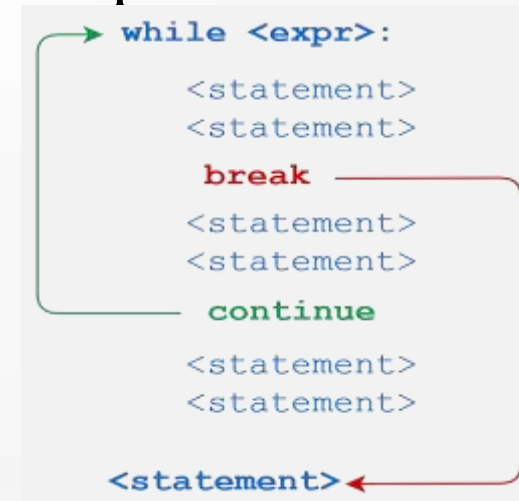
```
----- x = 2 -----
y = 4
y = 5
```

```
----- x = 3 -----
y = 4
y = 5
```

End

Keyword break and continue

- The break and continue keywords provide additional controls to a loop.
- break keyword breaks out of a loop.
- continue keyword breaks out of an iteration.
- Benefits of using these keywords:
- Can simplify programming in some cases.
- Negatives of using these keywords:
- Overuse or improperly using them can cause problems and make programs difficult to read and debug.



Keyword break and continue

- You can use the keyword `break` in a loop to immediately terminate a loop.
- Example:

LISTING 5.11 TestBreak.py

```
1 sum = 0
2 number = 0
3
4 while number < 20:
5     number += 1
6     sum += number
7     if sum >= 100:
8         break
9
10 print("The number is", number)
11 print("The sum is", sum)
```



```
The number is 14
The sum is 105
```

Keyword break and continue

- You can use the continue keyword in a loop to end the **current** iteration, and program control goes to the end of the loop body.
- Example:

LISTING 5.12 TestContinue.py

```
1 sum = 0
2 number = 0
3
4 while number < 20:
5     number += 1
6     if number == 10 or number == 11:
7         continue
8     sum += number
9
10 print("The sum is", sum)
```



The sum is 189



LIST

Chapter 8

List

- List
- List of String
- Altering a list
- Loop and list

List

- List is a data type in Python
- A list can contain multiple data types in the same list: numbers, strings, and even other lists

```
num_list = [1, 2, 3, 4, 5]
```

```
mixed_list = [1, "two", "three", 4, 5]
```

- list use an index to access individual elements of the list :
- list_name[index#]
- List indexes start with zero (0) for the first item in the list

List

```
1  #Create a list
2
3  num_list= [1,2,3,4,5,6,7,8,9]
4
5  print(num_list[4])
6
```

- What number will be printed and why?

List of Strings

```
1 # create a list of cities
2 city_list = ["Seattle", "New York", "Washington", "Houston", "Miami", "San Diego",
3 "Charlotte", "Atlanta", "Boston", "Baltimore"]
4
5 print("The 4th city on the list is:", city_list[3])
6
7
```

- Which city will be printed from index 3?

Altering a List

- Create or update a list with user input
- append add the value to the end of the list
- insert add the value at the given index number
- remove remove the given value from the list

Altering a List

```
1 city_list = ["Seattle", "New York", "Washington",  
2 "Houston", "Miami", "San Diego", "Charlotte", "Atlanta", "Boston", "Baltimore"]  
3  
4 city_list.insert(2, "Greensboro")  
5  
6 print(city_list)  
7
```

This code will add Greensboro into the list at index position 2. Which will be between New York and Washington.

Altering a List

```
1 city_list = ["Seattle", "New York", "Greensboro", "Washington", "Houston",  
2 "Miami", "San Diego", "Charlotte", "Atlanta", "Boston", "Baltimore"]  
3  
4 city_list.remove("Greensboro")  
5  
6 print(city_list)  
7
```

This code will remove Greensboro from the list.

Altering a List

What code is needed after the input?

```
1 city_list = ["Seattle", "New York", "Greensboro", "Washington",  
2 "Houston", "Miami", "San Diego", "Charlotte", "Atlanta", "Boston", "Baltimore"]  
3  
4 option = int(input("1- Add a City, 2- Remove a City, 3- Insert a City. Please enter 1, 2, or 3: "))  
5
```

Altering a List

```
1  if option == 1:
2      print("This option allows you to add a city to the end of the list")
3      print(city_list)
4      city = input("What city would you like to add to the list?")
5      city_list.append(city)
6      print(city_list)
7
```

This option is straightforward- it appends a city to the end of the list.

Altering a List

Always check if item exists in a list before using `.remove()`

```
1  elif option == 2:
2      print("This option allows you to remove a city from the list")
3      print(city_list)
4
5      city_to_remove= input("What city should be removed from the list?")
6
7      if city_to_remove in city_list:
8          city_list.remove(city_to_remove)
9          print(city_list)
10     else:
11         print("That city is not in the list.")
12
```

Altering a List

```
1 elif option == 3:
2     print("This option allows you to insert a city in a specific position on the list")
3     city = input("What city would you like to add to the list?")
4     pos = int(input("Where on the list would you like to add it?"))
5
6     if pos < len(city_list):
7         city_list.insert(pos,city)
8         print(city_list)
9     else:
10        print("Invalid Position")
11
```

The option will insert the city at a specific position in the list. The if statement ensures the index number chosen is valid. This prevents a run time error for index out of range.

Loop and list

```
1  #Create a list of numbers and sum them
2  num_list = [1,3,5,7,9,11,13,15,17,19]
3  sum = 0
4
5  for num in num_list:
6      sum += num
7      print(sum)
8
9  print("The total sum is:", sum)
10
```

What is the total sum of the list?

Loop and list

```
1  # creates list of numbers
2  while count < how_many:
3      # the next line creates a random number between 1 and 9
4      num1 = random.randint(1,9)
5      num_list.append(str(num1))
6      count += 1
7
```

Create the list by generating random integer numbers between 1 and 9.
The loop will run the number of times specified by the user.

[illegible]

FUNCTIONS

Chapter 9

Functions

- Built-in Functions
- User Defined Functions
- Functions and Parameters
- Multiple Parameters
- Python Calculator

Built-in Function

A function is a group of statements that performs a specific task.

Python, as well as other programming languages, provides a library of functions.

You have already used the functions `eval`, `input`, `print`, and `int`.

These are built-in functions and they are always available in the Python interpreter. You don't have to import any modules to use these functions.

Additionally, you can use the built-in functions `abs`, `max`, `min`, `pow`, and `round`, as shown in the following slide.

Built-in Function

TABLE 3.1 Simple Python Built-in Functions

<i>Function</i>	<i>Description</i>	<i>Example</i>
<code>abs(x)</code>	Returns the absolute value for <code>x</code> .	<code>abs(-2)</code> is 2
<code>max(x1, x2, ...)</code>	Returns the largest among <code>x1</code> , <code>x2</code> , ...	<code>max(1, 5, 2)</code> is 5
<code>min(x1, x2, ...)</code>	Returns the smallest among <code>x1</code> , <code>x2</code> , ...	<code>min(1, 5, 2)</code> is 1
<code>pow(a, b)</code>	Returns <code>a^b</code> . Same as <code>a ** b</code> .	<code>pow(2, 3)</code> is 8
<code>round(x)</code>	Returns an integer nearest to <code>x</code> . If <code>x</code> is equally close to two integers, the even one is returned.	<code>round(5.4)</code> is 5 <code>round(5.5)</code> is 6 <code>round(4.5)</code> is 4
<code>round(x, n)</code>	Returns the float value rounded to <code>n</code> digits after the decimal point.	<code>round(5.466, 2)</code> is 5.47 <code>round(5.463, 2)</code> is 5.46

Built-in Function

```
>>> abs(-3) # Returns the absolute value
3
>>> abs(-3.5) # Returns the absolute value
3.5
>>> max(2, 3, 4, 6) # Returns the maximum number
6
>>> min(2, 3, 4) # Returns the minimum number
2
>>> pow(2, 3) # Same as 2 ** 3
8
>>> pow(2.5, 3.5) # Same as 2.5 ** 3.5
24.705294220065465
>>> round(3.51) # Rounds to its nearest integer
4
>>> round(3.4) # Rounds to its nearest integer
3
>>> round(3.1456, 3) # Rounds to 3 digits after the decimal point
3.146
```

User Defined Functions

A power of a programming language is being able to create functions

Functions allows code reuse. We can call a function again and again

Reuse makes programming more efficient and less prone to error

User Defined Functions

To create a function use the def keyword.

Each line of the function under the definition **MUST** be indented. This is how Python knows it is part of the function.

To call a function, type the name of the function like a built-in function.

function call: `get_name()`

```
1 def get_name():  
2     name = input("What is your name?")  
3     print("The name you entered is", name + ".")  
4  
5     print("This is a sample funciton")  
6     get_name()  
7
```

User Defined Functions

This simple function adds numbers, but returns a value to the function call

```
1  def add_numbers():
2      num1= input("Enter a number:")
3      num2 = input("Enter a second number:")
4      num3 = int(num1) + int(num2)
5      return num3
6
7  print("Welcome to the Magic!")
8  num4= add_numbers()
9  print(num4)
10
```

User Defined Functions

In the `add_numbers()` function the user is asked for 2 numbers which are added together. The `return` keyword is used to send the result back to the `num4` variable

```
1  def add_numbers():
2      num1= input("Enter a number:")
3      num2 = input("Enter a second number:")
4      num3 = int(num1) + int(num2)
5      return num3
6
7  print("Welcome to the Magic!")
8  num4= add_numbers()
9  print(num4)
10
```

Why is type casting needed here?

User Defined Functions

We call the function more than once. This is the real power of a function

```
1 print("Welcome to the Magic!")
2 num4= add_numbers()
3 num5 = add_numbers()
4 print(num4 + num5)
5
```

So now the function is called twice and all 4 numbers are added together.

But the addition code was only written once.

Functions and Parameters

Built-in functions like `print()` and `type()` take arguments.

User-defined functions can define parameters it will accept

```
1  # define the function
2  def format_name(name_input):
3      print("the name you entered is", name_input + ".")
4
5  name1 = input("What is your name?: ")
6
7  #Call the function
8  format_name(name1)
9
```

Multiple Parameters

Separate each parameter with a comma.

```
1  # define the function
2  def format_name(name_input,age_input):
3      print("the name you entered is", name_input + ". You are", age_input, "years old.")
4
5      name1 = input("What is your name?: ")
6      age1 = input("What is your age?: ")
7
8  #Call the function
9  format_name(name1, age1)
10
```

The number of parameters passed must exactly match the number in the def statement or Python will throw an error. Unless a default value is defined.

Python Calculator

Let's write a simple Python program that will add, subtract, multiply, or divide 2 numbers! First, the functions.

```
1
2 def add(num1, num2):
3     sum = num1+ num2
4     return sum
5
6 def subtract(num1, num2):
7     difference = num2 - num1
8     return difference
9
10 def multiply(num1, num2):
11     product = nun1 * num2
12     return product
13
14 def divide(num1, num2):
15     if num1 != 0:
16         quotient = num2 / num1
17         return quotient
18     else:
19         return 0
20
```

Python Calculator

Calling the calculator function

```
1  num1 = int(input("Please enter a number:"))
2  num2 = int(input("Please enter a second number:"))
3  operator = input("What is your operation? Enter + - * or / only.")
4
5  if operator == "+":
6      answer = add(num1,num2)
7
8  elif operator == "-":
9      answer= subtract(num1, num2)
10
11 elif operator == "*":
12     answer = multiply(num1,num2)
13
14 elif operator == "/":
15     answer = divide(num1,num2)
16
17 else:
18     print("This is not a valid operation!")
19     print(str(answer))
20
```