PYTHON FOR AI

Table of Contents

Getting Acquainted with Python

Python is an open source, high-level programming language developed by Guido van Rossum in the late 1980s and presently administered by Python Software Foundation. It came from the ABC language that he helped create early on in his career.

Python is a powerful language that you can use to create games, write GUIs, and develop web applications.

It is a high-level language. Reading and writing codes in Python is much like reading and writing regular English statements. Because they are not written in machine-readable language, Python programs need to be processed before machines can run them.

Python is an interpreted language. This means that every time a program is run, its interpreter runs through the code and translates it into machine-readable byte code.

Python is an object-oriented language that allows users to manage and control data structures or objects to create and run programs. Everything in Python is, in fact, first class. All objects, data types, functions, methods, and classes take equal position in Python.

Programming languages are created to satisfy the needs of programmers and users for an effective tool to develop applications that impact lives, lifestyles, economy, and society. They help make lives better by increasing productivity, enhancing communication, and improving efficiency. Languages die and become obsolete when they fail to live up to expectations and are replaced and superseded by languages that are more powerful. Python is a programming language that has stood the test of time and has remained relevant across industries and businesses and among programmers, and individual users. It is a living, thriving, and highly useful language that is highly recommended as a first programming language for those who want to dive into and experience programming.

❖ Advantages of Using Python

Here are reasons why you would prefer to learn and use Python over other high level languages:

- Readability

  Python programs use clear, simple, and concise instructions that are easy to read even by those who have no substantial programming background. Programs Written in Python are, therefore, easier to maintain, debug, or enhance.

- Higher productivity

  Codes used in Python are considerably shorter, simpler, and less verbose than other high- level programming languages such as Java and C++. In addition, it has well-designed built-in features and standard library as well as access to third party modules and source libraries. These features make programming in Python more efficient.

- Less learning time

Python is relatively easy to learn. Many find Python a good first language for learning programming because it uses simple syntax and shorter codes.

- Runs across different platforms

Python works on Windows, Linux/UNIX, Mac OS X, other operating systems and small-form devices. It also runs on microcontrollers used in appliances, toys, remote controls, embedded devices, and other similar devices.

Interacting with Python

Python is a flexible and dynamic language that you can use in different ways. You can use it interactively when you simply want to test a code or a statement on a line-by-line basis or when you're exploring its features. You can use it in script mode when you want to interpret an entire file of statements or application program.

To use Python interactively, you can use either the Command Line window or the IDLE Development Environment.

Command Line Interaction

The command line is the most straightforward way to work with Python. You can easily visualize how Python works as it responds to every completed command entered on the >>> prompt. It may not be the most preferred interaction with Python, but it is the simplest way to explore how Python works.

Starting Python

There are different ways to access Python's command line depending on the operating system installed on your machine:

* If you are using Windows; you can start the Python command line by clicking on its icon or menu item on the Start menu.

* You may also go to the folder containing the shortcut or the installed files and click on the Python command line.

* If you're using GNU/Linux, UNIX, and Mac OS systems, you have to run the Terminal Tool and enter the Python command to start your session.
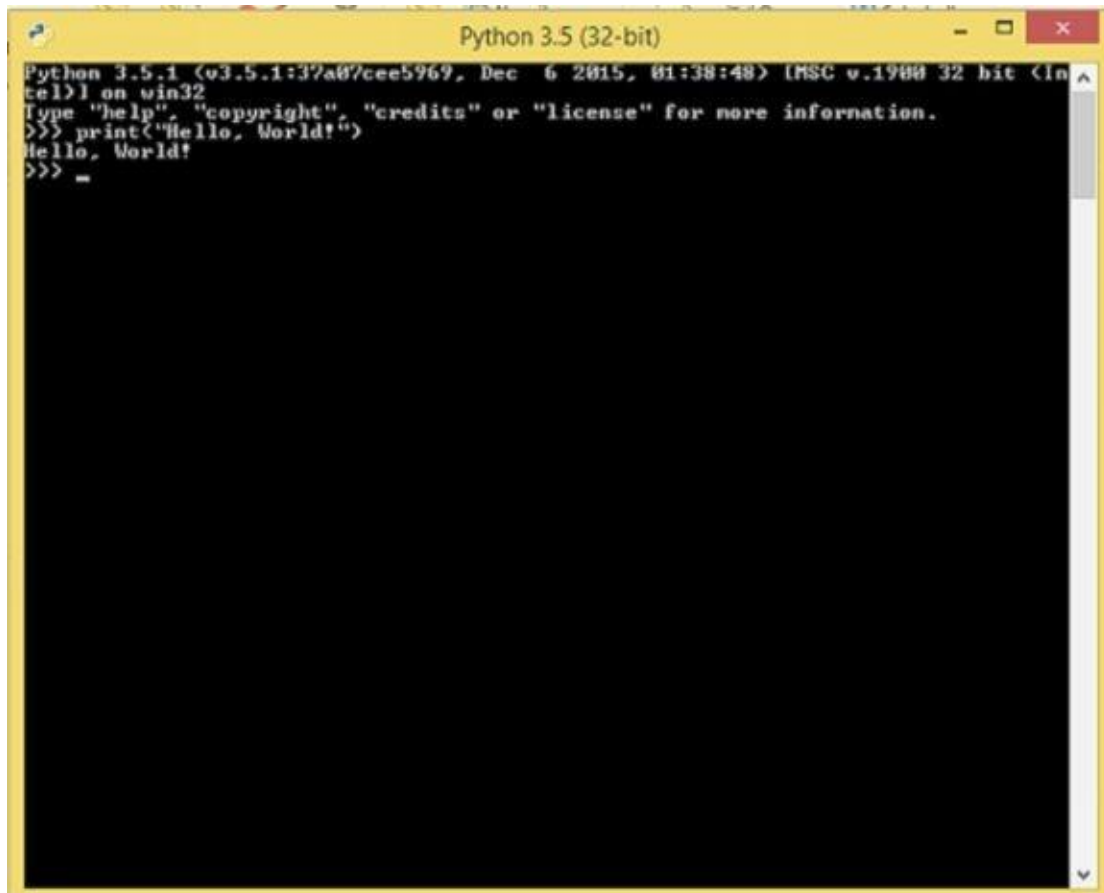
We use commands to tell the computer what to do. When you want Python to do something for you, you have to instruct it by entering commands that it is familiar with. Python will then translate these commands to instructions that your computer or device can understand and execute.

To see how Python works, you can use the print command to print the universal program "Hello, World!"

1.   Open Python's command line.

2. At the >>>prompt, type the following

        print("Hello, World")

3. Press enter to tell Python that you're done with your command. Very quickly, the command line window will display Hello, World! on the following line:

Python responded correctly because you gave it a command in a format that it requires. To see how it responds when you ask it to print the same string using a wrong syntax for the print command, type and enter the following command on the Python command prompt:
        Print("Hello, World!")
        This is how Python will respond:
        Syntax error: invalid syntax
You will get syntax error messages whenever you enter invalid or incomplete statements. In this case, you typed print with a capital letter that is a big no to a case-sensitive language like Python.

 If you're just using Python interactively, you can do away with the print command entirely by just typing your statement within quotes such as "Hello, World!"

Exiting Python

To exit from Python, you can type any of these commands:

quit()

exit()

Control-Z then press enter

IDLE: Python's Integrated Development Environment (IDE)

The IDLE (Integrated Development and Learning Environment) tool is included in Python's installation package but you can choose to download more sophisticated third party IDEs.

The IDLE tool offers a more efficient platform to write your code and work interactively with Python. You can access IDLE on the same folder where you found the command line icon or on the start menu. As soon as you click on the IDLE icon, it will take you to the Python Shell window.

The Python Shell Window

The Python Shell Window has dropdown menus and a >>>prompt that you have seen earlier in the command line window. Here you can type and enter statements or expressions for evaluation in the same way that you used the command line earlier. This time however, IDLE's editing menu allows you to scroll back to your previous commands, cut, copy, and paste previous statements and make modifications. IDLE is quite a leap from the command line interaction.

The Python Shell window has the following menu items: File, Edit, Shell, Debug, Options, Windows, and Help.

The Shell and Debug menus provide capabilities you would find useful when creating larger programs.

The Shell menu allows you to restart the shell or search the shell's log to find the most recent reset.

The Debug Menu has useful menu items for tracing the source file of an exception and highlighting the erring line. The Debugger option will usher in an interactive debugger window that will allow you to step through the running program. The Stack Viewer option displays the current Python stack through a new window.

The Options window allows you to configure IDLE to suit your Python working preferences.

The Help option opens Python Help and documentation.

The File Window

The items on the File menu allows you to create a new file, open an old file, open a module, and/or save your session. When you click on the 'New File' option, you will be taken to a new window, a simple and standard text editor where you can type or edit your code. Initially, this file window is named 'untitled' but its name will soon change as you save your code.

The File window's menu bar varies only slightly with the Shell Window. It doesn't have the 'Shell' and 'Debug' menu found in the Shell Window but it introduces two new menus: the Run and the Format menu. When you choose to Run your code on the file window, you can see the output on the Shell Window.

The Script Mode

When working in script mode, you won't automatically see results the way you would in interactive mood. To see an output from a script, you'll have to run the script and/or invoke the print() function within your code.

Python Syntax

Python syntax refers to the set of rules that defines how human users and the system should write and interpret a Python program. If you want to write and run your program in Python, you must familiarize yourself with its syntax.

Keywords

Python keywords are reserved words in Python that should not be used as variable, constant, function name, or identifier in your code. Take note of these keywords if you don't want to run into errors when you execute your program:

and	assert
break	class
continue	def
del	elif
else	except
exec	finally
for	from
global	if
import	in
is	lambda
not	or
pass	print
raise	return
try	while
with	yield

Python Identifiers

A Python Identifier is a name given to a function, class, variable, module, or other objects that you will be using in your Python program. Any entity you will be using in Python should be appropriately named or identified, as they will form part of your program.

Here are Python naming conventions that you should be aware of:

- An identifier can be a combination of uppercase letters, lowercase letters, underscores, and digits (0-9). Hence, the following are valid identifiers: my Class, my variable, var_1, and print_hello_world.

- Special characters such as %, @, and $ are not allowed within identifiers.
- An identifier should not begin with a number. Hence, 2variable is not valid, but variable2 is acceptable.

- Python is a case-sensitive language and this behavior extends to identifiers. Thus, Labor and labor are two distinct identifiers in Python.

- You cannot use Python keywords as identifiers.

- Class identifiers begin with an uppercase letter, but the rest of the identifiers begin in lowercase.

- You can use underscores to separate multiple words in your identifier.

You should always choose identifiers that will make sense to you even after a long gap. Hence, while it is easy to set your variable to $c = 2$, you might find it more helpful for future reference if you use a longer but more relevant variable name such as count $= 2$.

Using Quotations

Python allows the use of quotation marks to indicate string literals. You can use single, double, or triple quotes but you must start and end the string with the same type. You would use the triple quotes when your string runs across several lines.

Python Statements

Statements are instructions that a Python interpreter can execute. When you assign a value to a variable, say my_variable = "dog", you're making an assignment statement. An assignment statement may also be as short as c = 3. There are other kinds of statements in Python, like *if* statements, *while* statements, *for* statements, etc.

Multi-line statements

A statement may span over several lines. To break a long statement over multiple lines, you can wrap the expression inside parentheses, braces, and brackets. This is the preferred style for handling multi-line expressions. Another way to wrap multiple lines is by using a backslash (\) at the end of every line to indicate line continuation.

Indentation

While most programming languages such as Java, C, and C++ use braces to denote blocks of code, Python programs are structured through indentation. In Python, blocks of codes are defined by indentation not as a matter of style or preference but as a rigid language requirement. This principle makes Python codes more readable and understandable.

A block of code can be easily identified when you look at a Python program as they start on the same distance to the right. If it has to be more deeply nestled, you can simply indent another block further to the right. For example, here is a segment of a program defining car_rental_cost:

```python
def car_rental_cost(days):
    cost = 35 * days
    if days >= 8:
            cost -= 70
        elif days >= 3:
            cost -= 20
        return cost
```

You have to make sure that the indent space is consistent within a block. When you use IDLE and other IDEs to input your codes, Python intuitively provides indentation on the subsequent line when you enter a statement that requires indentation. Indentation, by convention, is equivalent to 4 spaces to the right.

Comments

When writing a program, you'll find it helpful to put some notes within your code to describe what it does. A comment is very handy when you have to review or revisit your program. It will also help another programmer who might need to go over the source code. You can write comments within your program by starting the line with a hash (#) symbol. A hash symbol tells the Python interpreter to ignore the comment when running your code.

For multi-line comments, you can use a hash symbol at the beginning of each line. Alternatively, you can also wrap multi-line comment with triple quotes.

Variables and Data Types

Variables

A variable is like a container that stores values that you can access or change. It is a way of pointing to a memory location used by a program. You can use variables to instruct the computer to save or retrieve data to and from this memory location.

Python is a lot more flexible when it comes to handling variables. If you need a variable, you'll just think of a name and declare it by assigning a value. If you need to, you can change the value and data type that the variable stores during program execution.

To illustrate these features:
In Python, you declare a variable by giving it a value:

my_variable = 10

Take note that when you are declaring a variable, you are not stating that the variable my_variable is equal to 10. What the statement actually means is "my_variable is set to 10".

To increase the value of the variable, you can enter this statement on the command line:

>>>my_variable = my_variable + 3

To see how Python responded to your statement, invoke the print command with this statement:

>>>print(my_variable)

You'll see this result on the next line:

13

To use my_variable to store a literal string "yellow", you'll simply set the variable to "yellow":

>>>my_variable = "yellow"

To see what's currently store in my_variable, use the print command:

>>>print(my_variable)

On the next line, you'll see:

yellow

Data Types

Python handles several data types to facilitate the needs of programmers and application developers for workable data. These include strings, numbers, Booleans, lists, date, and time.

Strings

A string is a sequence of Unicode characters that may be a combination of letters, numbers, and special symbols. To define a string in Python, you can enclose the string in matching single or double quotes:

>>>string1 = "I am enclosed in single quotes."
>>>string2 = "I am enclosed in double quotes."

If a literal string enclosed in single quotes contains a single quote, you'll have to place a backslash (\) before the single quote within the string to escape the character. For example:

>>> string3 = 'It doesn't look good at all.'

To print string3:

>>> print(string3)

It doesn't look good at all.

Of course, you wouldn't have to do this if you used double quotes to enclose the string:

>>>string3 = "It doesn't seem nice"

Similarly, you'll have to place a backslash before a double quote if your string is enclosed in double quotes:

>>>txt = "He said: \"You should get the same results no matter how you choose to enclose a string.\""

>>> print(txt)

He said: "You should get the same results no matter how you choose to enclose a string."

Strings may be indexed or subscripted. In Python, indexing starts from 0 (zero) instead of 1. Hence, a string's first character has a zero index.

To illustrate how string indexing works in Python, define the string "Hello Python" on the command line:

>>>s = "Hello Python"
This is how Python would index the string:

-12	-11	-10	-9	-8	-6	-6	-5	-4	-3	-2	-1
H	e	l	l	o		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

To access the first character on the string you just created, type and enter the variable name s and the index 0 within square brackets like this:

>>>s[0]

You'll get this output:

'H'

Accessing the first character is easy because you know that its index number is zero. You do not have this advantage when you want to access the last character on the string.

To access the last character, you can use this expression:

>>>s[len(s)-1]
You'll get the output:
'n'

The expression introduces you to the *len* function. There is actually an easier way to access the last item on the string:

>>>s[-1]

'n'

To access the penultimate character:

>>>s[-2]

'o'

Besides indexing, you can use other functions and mathematical operators on a string.

Concatenating Strings

Strings can be added together with plus (+) operator. To concatenate the string "Hello Python":

>>> "Hello" + "Python"

'HelloPython'

You can easily repeat strings or its concatenation with the * operator. For example:

Entering "**^**" * 5 will yield:

'**^****^****^****^****^**'

You'll get the same result with this:

>>>s = "**^**"

>>>s * 5

'**^****^****^****^****^**'

You can get the size of a string with the len() function. For example, to get the size of the string "World":

>>>len("World")

5

You can create substrings with the slicing notation. You can do this by placing two indices (separated by a colon) within square brackets. The first index marks the start of the substring while the second index indicates the index number of the first character you don't want to include in the substring.

For example:

>>>"Program"[3:5]

will result in:

'gr'

>>>"Program"[3:6]

will yield:

'gra'

Another way of doing this is by storing "Program" to a variable and manipulating the variable to produce the desired result:

>>>p = "Program"

>>>p [3:6]

'gra'

If you want the substring to start from a character to the end of the original string, you can just omit the second index. For example:

>>>p = "Program"

>>>p [4:]

'ram'

Conversely, if you want your substring to start from the first character of the original string, you can omit the first index and write the last index to be included on the substring. For example:

>>>p = "Program"

>>>p [:4]

'Prog'

If you have a string like "Grand River" and you have decided that you need your data to be all in lower case, you can use the lower() function to print the string in lower case.

Example:

The *str()* function makes strings out of non-strings character. This allows programmers to print non-string characters as if they are string characters. This is very handy when you want, for instance, to print an integer along with strings.

Example:

>>>pi =3.1416

>>>str(pi)

'3.1416'

>>>print("This my favorite number: " + str(pi))

This my favorite number: 3.1416

Numbers

Numeric Data Types

One of the many conveniences of using Python is that you don't really have to declare a numeric value to distinguish its type. Python can readily tell one data type from another when you write and run your statement. It has four built-in numeric data types. Python 3 supports three types: integer, floating-point numbers, and complex numbers. Long integers ('long') no longer form a separate group of integers but are included in the 'int' or integer category.

1. *Integer (int)*

Integers are whole numbers without decimal point. They can be positive or negative as long as they don't contain a decimal point that would make a number a floating number, a distinct numeric type. Integers have unlimited size in Python 3.

The following numbers and literals are recognized by Python:

Regular integers

Examples: 793, -254, 4

Octal literals (base 8)

To indicate an octal number, you will use the prefix 0o or 0O (zero followed by either a lowercase or uppercase letter 'o').

Example:

>>>a = 0O7
>>>print(a)
7

Hexadecimal literals (base 16)

To indicate hexadecimal literals, you will use the prefix '0X' or '0x" (zero and uppercase or lowercase letter 'x').

Example:

```
>>>hex_lit = 0xA0C
>>>print(hex_lit)
2572
```

Binary literals (base 2)

To signify binary literals, you'll use the prefix '0B' or '0b' (zero and uppercase or lowercase 'b').

Example:

```
>>> c = 0b1100
>>> print(c)
12
```

Converting Integers to their String Representation

Earlier, you have seen how the print command converted literals to their equivalent in integers. Python makes it possible for you to work the other way around by converting integers to their literal representation. To convert an integer into its string representation, you can use the functions *hex()*, *bin()*, and *oct()*.

Examples:

To convert the integer 7 to its octal literal, type and enter oct(7) on the command prompt. You'll get the output '0o7':

```
>>>oct(7)
'0o7'
```

Here is what happens when you convert the integer 2572 to a hexadecimal literal:

>>>hex(2572)

'0xa0c'

Finally, see what happens when you use the bin() function to convert the integer 12 to its binary string:

>>>bin(12)

'0b1100'

You can store the result to a variable by defining a variable with the hex(), bin(), and oct() functions:

For example:

>>>x = hex(2572)

>>>x

'0xa0c'

To see the object type created and stored in the variable x, you can use and enter the command type():

>>>type(x)

You should get this result:

<class 'str'>

1. *Floating-point numbers*

Also known as floats, floating-point numbers signify real numbers. Floats are written with

a decimal point that segregates the integer from the fractional numbers. They may also be written in scientific notation where the uppercase or lowercase letter 'e' signifies the 10th power:

```
>>>6.2e3
6200.0
```

```
>>>6.2e2
620
```

2. Complex numbers

Complex numbers are pairs of real and imaginary numbers. They take the form 'a + bJ' where 'a' is a float and the real part of the complex number. On the other side is bJ where 'b' is a float and J or its lowercase indicates the square root of an imaginary number, -1. This makes 'b' the imaginary part of the complex number.

Here are examples of complex numbers at work:

```
>>>a = 2 + 5j
>>>b = 4 – 2j
>>>c = a + b
>>>print(c)
(6 + 3j)
```

Complex numbers are not extensively used in Python programming.

Conversion of Number Type

You can expect Python to convert expressions with mixed types of numbers to a common type to facilitate evaluation. In some situations, however, you may have to convert one number type to another explicitly, like when the conversion is required by a function parameter. You can type the following expressions to convert a number to another type:

To convert x to a float: >>>float(x)

Example:

>>>float(12)
12.0

To convert x to a plain integer: int(x)

>>>int(12)
12

To convert x to a complex number: type complex(x)

>>>complex(12)
(12+0j)

Boolean Data Type

Comparisons in Python can only generate one of two possible responses: True or False. These data types are called booleans.

To illustrate, you can create several variables to store Boolean values and print the result:

bool_1 = 4 == 2*3

bool_2 = 10 < 2 * 2**3

bool_3 = 8 > 2 * 4 + 1

print(bool_1)

print(bool_2)

print(bool_3)

The Python Shell will display these results:

False
True
False

Lists

A list is a data type that can be used to store any type and number of variables and information.

You can define and assign items to a list with the expression:

my_list = [item_1, item_2, item_3]

Python also allows creation of an empty list:

my_list = []

To illustrate, let's create a list of colors:
colors = ["red", "orange", "yellow", "green", "indigo", "white"]

Since this is an indexed list, the first item on colors has zero as its index.

To access the first item on the list, you can print the color with the command:
>>> print(colors[0])
red

To print the color name of the third color on the list, you can enter:

>>> print(colors[4])
indigo

To see how many colors are on the list, you can use the len() function:

>>> len(colors)
6

There are only six colors on your list but you want to have all seven colors of the rainbow in your list. To see what colors are on the list, you can use the print to see what color might be missing:

>>> print(colors)
['red', 'orange', 'yellow', 'green', 'indigo', 'white']

It appears that the colors list doesn't just lack one color name. It also has one member that should not have been included – 'white'. To remove 'white' from the list, you can use the remove() method:

>>> colors.remove("white")

You can view the updated list with the print command:
>>> print(colors)
['red', 'orange', 'yellow', 'green', 'indigo']

The list is still short of 2 colors – violet and blue.

To add violet to your colors list, you can use the append command:

>>> colors.append("violet")

Let's check out the updated list with the print command:

```
>>> print(colors)
['red', 'orange', 'yellow', 'green', 'indigo', 'violet']
```

The color 'violet' was added to the end of the list. Now, you only need to add one more color - blue. Let's say you want to have 'blue' inserted between 'green' and 'indigo'.

You can use Python's insert()method with the syntax:

```
list.insert(index, obj)
```

The parameters are index and object. Index refers to the position where you want the new item to be located. The object is the item you want to insert.

Applying the syntax to the above example, you'll have the command:

```
>>> colors.insert(4, "blue")
```

To see the new list:

```
>>> print(colors)
['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']
```
Slicing lists

You can also slice lists in the same way that you slice strings.

For example, if you only want to display the colors 'green', 'blue', and 'indigo', with index of 3, 4, 5 respectively, you can use this command:

```
>>> colors[3:6]
['green', 'blue', 'indigo']
```

Dictionary

A dictionary is like a list but instead of looking up an index to access values, you'll be using a unique key, which can be a number, string, or tuple. Dictionary values can be anything but the keys must be an immutable data type. A colon separates a key from its value and all are enclosed in curly braces. Here is the dictionary structure:

d = {key_1 : a, key_2 : 2, key_3 : ab}

An empty dictionary will have this format:

d = {}

A dictionary can be a very useful tool for storing and manipulating key-value pairs such as those used in phone books, directory, menu, or log-in data. You can add, modify, or delete existing entries within the dictionary.

To see how dictionaries actually work, you can create a dictionary named menu with dish and prices pairs:

menu = {"spam" : 12.50, "carbonara" : 20, "salad" : 15 }
To see how many key-value pairs are stored in the dictionary, you can use the len() function:

>>>len(menu)
3

To print the current entries in the menu dictionary:

>>> print(menu)
{'salad': 15, 'carbonara': 20, 'spam': 12.5}

To add another entry in the menu dictionary, you can use this format:

d[key_4 : b]

Applying this structure to the menu dictionary, you can add the dish-price entry of cake : 6 with:

menu["cake"] = 6

To see the updated menu, use the print command:

>>> print(menu)
{'spam': 12.5, 'cake': 6, 'carbonara': 20, 'salad': 15}

Assuming you no longer want to include spam in your menu, you can easily do so with the del command:

>>> del menu["spam"]

To see the modified list after deleting spam:
{'cake': 6, 'carbonara': 20, 'salad': 15}

You might want to change the values in any of the keys at one point. For instance, you need to change the price of carbonara from 20 to 22. To do that, you'll just assign a new value to the key with this command:

>>> menu["carbonara"] = 22

You can use the print command once more to see the updated menu:

>>> print(menu)
{'cake': 6, 'carbonara': 22, 'salad': 15}

If you want to remove all entries in the dictionary, you can use the function

dict.clear()

To clear all entries in the menu:

>>>dict.clear(menu)

Use the print command to see what happened to the menu dictionary:

>>> print(menu)
{}

The Python Shell displayed an empty dictionary with the clear command. Now that it contains no data at all, you might decide to delete the dictionary. You can do so with the del command:

del dict
To delete the menu dictionary:

del menu

To see what happened, use the print command.

>>> print(menu)
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in<module>
    print(menu)
NameError: name 'menu' is not defined

You got an error message because menu no longer exists.

Chapter 6   Python Basic Operators

Python operators allow programmers to manipulate data or operands. Here are the types of operators supported by Python:

- Arithmetic Operators
- Assignment Operators
- Relational or Comparison Operators
- Logical Operators
- Identity Operators
- Bitwise Operators
- Membership Operators

Arithmetic Operators

Python does a good job of processing mathematical expressions with its basic arithmetic operators. You can easily make programs to automate tasks such as computing tax, tips, discounts, or rent.

+	Addition	adds the value of the left and right operands
-	Subtraction	subtracts the value of the right operand from the value of the left operand
*	Multiplication	multiplies the value of the left and right operand
/	Division	divides the value of the left operand by the right operand
**	Exponent	performs exponential calculation
%	Modulus	returns the remainder after dividing the left operand with the right operand
//	Floor Division	division of operands where the solution is a quotient left after removing decimal numbers

Addition, subtraction, multiplication, and division are the most basic operators and are invoked by entering the following expressions:

Addition:

```
>>>1 + 3
```

4

Subtraction:

>>>10 – 4

6

Multiplication:

>>>4 * 2

8

Division:

>>>10 / 2

5.o

Exponent

Exponential calculation is invoked by raising the first number to the power defined by the number after the ** operator:

>>>2**3   2 raised to the power of 3

8

Modulus

The modulus operator gives the remainder after performing division:

>>>17 % 5

2

Floor Division

Floor division, on the other hand, returns the quotient after removing fractional numbers:

>>>17 // 5

3

Using Basic Operators to Compute Sales Tax, Tip, and Total Bill

To put your knowledge of variables, data types, and operators to good use, you can design a simple program that will compute the sales tax and tip on a restaurant meal.

Meal cost	$65.50
Sales tax rate	6.6%
Tip	20% of meal + tax

First, set up a variable meal to store the food cost:

meal = 65.50

Next, set up the tax and tip variable. Assign both variables the decimal value of the percentages given. You can do this by using 100 as divisor.

tax = 6.6 / 100

tip = 20 / 100

Your tip is based on meal cost and the added sales tax so you need to get the total amount of the meal and the sales tax. One way to do this is by simply creating a new variable to store the total cost of the meal and tax. Another way is by reassigning the variable meal so that it stores both values:

meal = meal + meal * tax

Now that you have reassigned meal to take care of the meal cost and tax, you're ready to compute for the tip. This time, you can set a new variable to store the value of the tip, meal, and tax. You can use the variable total to hold all values:

total = meal * tip

Here's your code to compute for the total bill amount:

```
meal = 65.50

tax = 6.6 / 100

tip = 20 / 100

meal = meal + meal * tax

total = meal + meal * tip
```

If you're using the file editor in IDLE, you can save the file in a filename of your choice and Python automatically appends the .py extension. As you may have noticed, the file editor will always prompt you to save your file before it does anything about your code. Just like when naming other data files and types, you should use a filename that's descriptive of the file. In this case, a filename like BillCalculator should do the trick.

To get the total amount, go to the Python Shell and type total:

```
>>>total

83.78760000000001
```

Now you have the bill amount: 83.78760000000001

If you're using the line command window, you can simply enter the above code on a per line basis.

This simple program shows how straightforward Python programming is and how useful it could be in automating tasks. Next time you eat out, you can reuse the program by simply changing the figures on your bill calculator. Think forward and visualize how convenient it could be if you could put your code in a bigger program that will simply ask you to input the bill amount instead of accessing the original code. You can do that with Python.

Assignment Operators

These operators are useful when assigning values to variables:

= Operator

You have seen this operator at work in previous chapters when you have assigned different values to variables. Examples:

a = c

a = b + c

a = 8

a = 8 + 6

s = "I love Python."

+= add and

The 'add and' (+=) operator is simply another way to express x = x + a so that you'll end up with the statement x += a.

-= subtract and

The 'subtract and' (-=) operator is equivalent to the expression x = x – a and is expressed with the statement x-=a

*= multiply and

The 'multiply and' (*=) operator is the equivalent of the statement x = x * a and is expressed with x*=a.

/= divide and

The 'divide and' (/=) operator is like saying x = x/a and is expressed with the statement x/=a.

%= modulus and

The 'modulus and' (%=) operator is another way to say x = x % a where you'll end up instead with the expression x%=a.

//= floor division and

Operator	Meaning
==	is equal to
<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to
!=	is not equal to

return the relation as either True or False.

Here are the relational operators in Python:

The 'floor division and' is equivalent to the expression x = x//a and takes the form x//=a.

Relational or Comparison Operators

Relational operators evaluate values on the left and right side of the operator and

Logical Operators

Python supports 3 logical operators:

or

and

not

        x or y   If the first argument, x, is false, then it evaluates the second argument, y. Else, it evaluates x.

        x and y   If x is false, then it evaluates x. Else, if x is true, it evaluates y.

not x   If x is false, then it returns True. If x is true, it returns False.

Examples:

```
>>> (8>9) and (2<9)
False
```

```
>>> (2>1) and (2>9)
False
```

```
>>> (2==2) or (9<20)
True
```

```
>>> (3!=3) or (9>20)
False
```

```
>>> not (8 > 2)
False
```

```
>>> not (2 > 10)
True
```

Precedence of Python Operators

Python operators are evaluated according to a set order or priority:

	Description	Operators
1	Exponentiation	**
2	Ccomplement, unary plus, and minus	~, +, -
3	Multiplication, division, modulo, and floor division	*, /, %, //
4	addition and subtraction	+ -
5	Right and left bitwise shift	>>, <<
6	Bitwise 'AND'	&
7	Regular `OR' and Bitwise exclusive 'OR'	|, ^
8	Comparison operators	<= < > >=
9	Equality operators	== !=
10	Assignment operators	=, +=, -=, *-, /=, %= //= **=
11	Identity operators	is, is not
12	Membership operators	in, not in
13	Logical operators	or, and, not

Chapter 7   Python's Built-in Functions

Functions provide efficiency and structure to a programming language. Python has many useful built-in functions to make programming easier, faster, and more powerful.

The input() Function

Programs usually require input that can come from different sources: keyboard, mouse clicks, database, another computer's storage, or the internet. Since the keyboard is the most common way to gather input, Python provided its users the input () function. This function has an optional parameter called the prompt string.
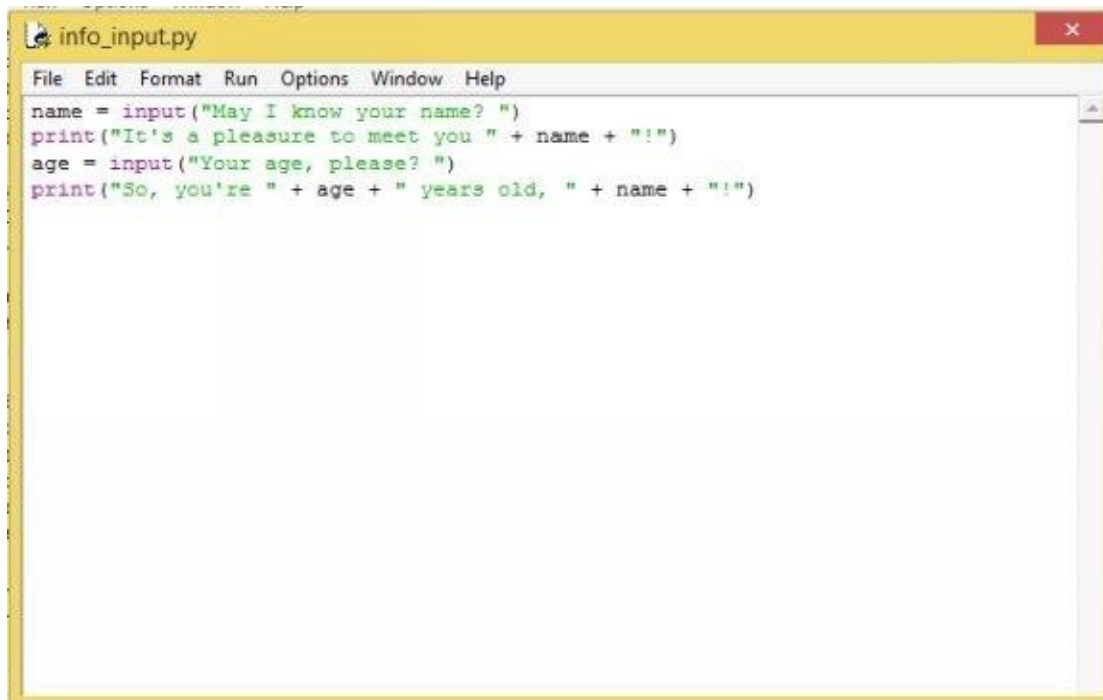
Once the input function is called, the prompt string will be displayed on the screen and the program flow stops until the user has entered an input. The input is then interpreted and the input () function returns the user is input as a string.

To illustrate, here is a sample program that collects keyboard input for name and age:

name = input ("May I know your name? ")
print("It's a pleasure to meet you " + name + "!")
age = input("Your age, please? ")
print("So, you're " + age + " years old, " + name + "!")

Before you save the code, take a close look at the string to be printed on the second line. You'll notice that there is a blank space after 'you' and before the double quote. This space ensures that there will be a space between 'you' and the 'name' input when the print command is executed. The same convention can be seen on the 4th line with the print command where 'you're' is separated by a single space from the 'age' input and 'old' is separated by a space from the 'name' input.

Save the code as info_input.py and run it.

```
info_input.py                                                        ×

File  Edit  Format  Run  Options  Window  Help

name = input("May I know your name? ")
print("It's a pleasure to meet you " + name + "!")
age = input("Your age, please? ")
print("So, you're " + age + " years old, " + name + "!")
```

The Python Shell will display the string on the first line:

 May I know your name?

A response is needed at this point and the program stops executing until a
keyword input is obtained. Let's type and enter the name Jeff to see what
happens:

 It's a pleasure to

meet you Jeff!
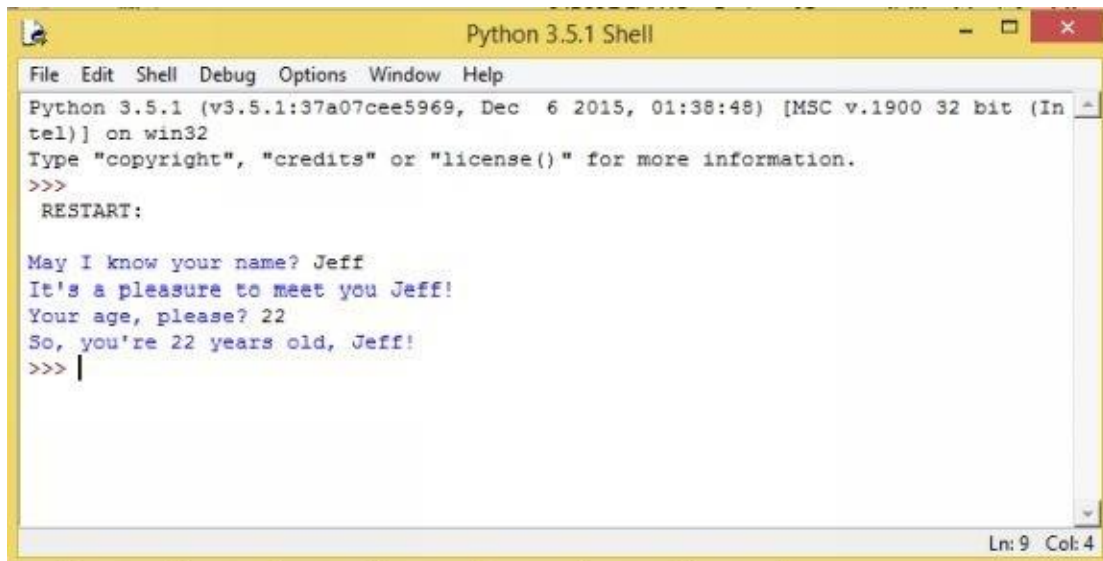
Your age, please?

The program has now proceeded to the next input function and is waiting for
the keyboard input. Let's enter 22 as Jeff's age and see what the program does
next:

So, you're 22 years old, Jeff!

The program printed the last string on the program after a keyboard response was obtained. Here is the entire output on the Python Shell:

May I know your

name? Jeff It's a

pleasure to meet

you Jeff! Your age,

please? 22

So, you're 22 years old, Jeff!

The range() function

Python has a more efficient way to handle a series of numbers and arithmetic progressions and this is by using one its built-in functions: range(). The range function is particularly useful in 'for loops'.

Here is an example of the *range()* function:

```
>>>
range(
5)
range(
0, 5)
```

The expression range(5) above generates an iterator that progresses integers from zero and ends with 4 (5-1). To show the list of numbers, you can use the command list(range(n)):

```
>>>li
st(ran
ge(5))
[0, 1,
2, 3,
4]
```

You can exercise more control over the list output by calling the range() function with two arguments:

```
range (begin, end)
```

Example:

```
>>> range(5, 9)
range(5, 9)
```

To show the list:

```
>>> list
(range(5,
9)) [5, 6,
7, 8]
```

The above examples of *range()* demonstrated an increment of 1. You can change the way Python increments the number by introducing a third argument, the 'step'. It can be a negative or positive number, but never zero.

Here is the

format:

range(begin,

end, step)

Example:

```
>>> range(10, 71, 5)
range(10, 71, 5)
```
Invoking list, we'll see this sequence of numbers:

```
>>> list (range(10, 71, 5))
[10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70]
```

The print() Function

Python 3 turned *print* from a statement into a function. Hence, you must always enclose your print parameters within the round parentheses.

Examples:

print("This is Python 3

print function) print(s)

print(5)

The *print()* function can print any number of values within the parentheses; they must be separated by commas. For example:

a = 3.14

b

=

"

a

g

e

"

c

=

3

2

print("a = ", a, b, c)

The result:

a = 3.14 age 32
The Python shell displayed values with blank spaces between them.

abs()

The abs() function returns the absolute value of a single number. It takes an integer or float number as argument and always returns a positive value.

Examples:

>

>

>

a

b

s

(

-

1

0

)

1

0

```
>>> abs(5)
10
```

When complex numbers are used as argument, the abs() function returns its magnitude:

```
>>> abs(3 + 4j)
5.0
```

max()

The max() function takes two or more arguments and returns the

largest one. Examples:

```
>>> max(9, 12, 6, 15)
15
```

```
>>> max(-2, -7, -35, -4)
-2
```

min()

The min() function takes two or more arguments and returns the

smallest item. Examples:

```
>>> min(23, -109, 5, 2)
-109
```

```
>>> min(7, 26, 0, 4)
0
```

type()

The type() function returns the data type of the

given argument. Examples:

```
>>> type("This is a string")
<class 'str'>
```

```
>>> type(12)
<class 'int'>

>>> type(2 +3j)
<class 'complex'>

>>> type(215.65)
<class 'float'>
```

len()

The len() function returns the length of an object or the number of items in a list given as argument.

Examples:

```
>>>
len("pneumonoultramicroscopicsilicovolcanoconi
osi") 44

>>> s = ("winter", "spring", "summer", "fall")
>
l
e
n
```

Chapter 9   Loops

A loop is a programming construct that enables repetitive processing of a sequence of statements. Python provides two types of loops to its users: the 'for loop' and the 'while loop'. The 'for' and 'while' loops are iteration statements that allow a block of code (the body of the loop) to be repeated a number of times.

The For Loop

Python implements an iterator-based 'for loop'. It is a type of 'for loop' that iterates over a list of items through an explicit or implicit iterator.

The loop is introduced by the keyword 'for' which is followed by a random variable name which will contain the values supplied by the object.

This is the syntax of Python's 'for loop':

```
for
    variable
    in list:
    stateme
    nts
else:
    statements
```

Here is an example of a 'for loop' in Python:

```python
pizza = ["New York Style Pizza", "Pan Pizza", "Thin n Crispy Pizza", "Stuffed Crust Pizza"]
for choice in pizza:
    if choice == "Pan Pizza":
    print("Please pay $16.
    Thank you!")
    print("Delicious, cheesy "
    + choice)
else:
    print("Cheesy pan pizza is my all-time
favorite!") print("Finally, I'm full!")
```

Run this and you'll get the following output on Python Shell:

Delicious, cheesy New York

Style Pizza Please pay $16.

Thank you!

Delicious, cheesy Pan

Pizza Delicious, cheesy

Thin n CrispyPizza

Delicious, cheesy Stuffed

Crust Pizza

Cheesy pan pizza is my all-

time favorite! Finally, I'm full!

A Python break statement ends the present loop and instructs the interpreter to starts executing the next statement after the loop. It can be used in both 'for' and 'while' loops. Besides leading the program to the statement after the loop, a break statement also prevents the execution of the 'else' statement.

To illustrate, a break statement may be placed right after the print function of the 'if statement':

pizza = ["New York Style Pizza", "Pan Pizza", "Thin n Crispy Pizza", "Stuffed Crust Pizza"]

for choice in pizza:

   if choice == "Pan Pizza":

   print("Please pay $16.

   Thank you!") break

   print("Delicious, cheezy " +

choice) else:

   print("Cheezy pan pizza is my all-time

favorite!") print("Finally, I'm full!")

The Python Shell will now show:

Delicious, cheezy New York

Style Pizza Please pay $16.

Thank you!

Finally, I'm full!

The continue statement brings back program control to the start of the loop. You can use it for both 'for' and 'while' loops.

To illustrate, the continue statement may be placed right after the print function of the 'for loop' to replace the break statement:

```
pizza = ["New York Style Pizza", "Pan Pizza", "Thin n Crispy Pizza", "Stuffed Crust Pizza"]
for choice in pizza:
    if choice == "Pan Pizza":
    print("Please pay $16.
    Thank you!") continue
    print("Delicious, cheesy "
+ choice) else:
    print("Cheesy pan pizza is my all-time
favorite!") print("Finally, I'm full!")
```

The output will be:
Delicious, cheesy New York

Style Pizza Please pay $16.

Thank you!

Delicious, cheesy Thin n

Crispy Pizza Delicious, cheesy

Stuffed Crust Pizza Cheesy

pan pizza is my all-time

favorite! Finally, I'm full!

The range() function can be combined with the 'for loop' to supply the numbers required by the loop. In the following example, the range(1, x+1) provided the numbers 1 to 50 needed by the 'for loop' to add the sum of 1 until 50:

x = 50

total = 0
for number in
    range(1, x+1):
    total = total +
    number

print("Sum of 1 until %d: %d" % (x, total))

The Python Shell

will display: l.py
Sum of 1 until 50: 1275

The While Loop

A Python 'while loop' repeatedly carries out a target statement while the condition is true.

The loop iterates as long as the defined condition is true. When it ceases to be true and becomes false, control passes to the first line after the loop.

The 'while loop' has the

following syntax: while

condition

   stat

ement

statem

ent

Here is a simple 'while loop':

```
counter = 0
while (counter < 10):
    print('The count is:'
    , counter)
```

```
    counter = counter +
    1

print("Done!")
```

Chapter 10   User-Defined Functions

A function is a set of statements that perform a specific task, a common structuring element that allows you to use a piece of code repeatedly in different parts of a program. The use of functions improve a program's clarity and comprehensibility and makes programming more efficient by reducing code duplication and breaking down complex tasks into more manageable pieces. Functions are also known as routines, subroutines, methods, procedures, or subprograms.

They can be passed as arguments, assigned to variables, or stored in collections.

A user-defined Python function is created or defined by the def statement and follows the syntax:

def function_name(parameter list):
    function body/statements

The indented statements make up the body of the function and are executed when the function is called. Once the function is called, parameters inside round brackets become arguments.

Function bodies can have more than one return statement which may be placed anywhere within the function block. Return statements end the function call and return the value of the expression after the return keyword. A return statement with no expression returns the special value 'None'. In the absence of a return statement within the function body, the end of the function is indicated by the return of the value 'None'.

The docstring is an optional statement after the function title which explains what the function does. While it is not mandatory, documenting your code with a docstring is a good programming practice.

Here is a simple function that prints I

love Pizza! def love_pizza():
    print "I love Pizza!"

Here is a function with a parameter and

return keyword: def

absolute_value(number):

    if
    numb
    er >=
    0:
    retur
    n
    numb
    er
    else:
    return -number

print(absolut
e_value(3))

print(absolut

e_value(-5))

In the above example, number is the parameter of the function absolute_value. It acts as a variable name and holds the value of a passed in argument.

Here is the output when the above code is run:

3
5

Following is a function with an if-then-

else statement. def shutdown(yn):
    if yn.lower() == "y":
    return("Closing files and
    shuttingdown") elif yn.lower()
    == ("n"):
    return("Shutdown
    cancelled") else:
    return("Please check your response.")

print(shut
down("y"
))
print(shut
down("n"

```
))
print(shut
down("x"
))
```

Python Shell will display:

```
Closing files and
shutting down
Shutdown cancelled
Please check your response.
```

Function can take more than one parameter and use them for computations:
```
def
    calculat
    or(x, y):
    return x
    * y + 2

print(calculator(2,6))
print(calculator(3,7))
```

Run the code and you'll get the output:
```
14
23
```
Functions can call other functions

Functions can perform different types of actions such as do simple calculations and print text. They can also call another function.

For example:

```
def
   member
   s_total(
   n):
   return n
   * 3
```

```
def org_total(m):
   return members_total(m) + 5
```

To see what you code does, enter the following print

commands: print(org_total(2))
print(org_total(5))
print(org_total(10))

You'll get these results:
11
20
35

A variable's scope refers to a program's sections where it is recognized. Variables and parameters defined within a function have a local scope and are not visible from outside of the function. On the other hand, a variable's lifetime refers to its period of existence in the memory. Its lifetime coincides with the execution of a function which ends when you return from the function. A variable's value is discarded once the return is reached and a function won't be able to recall a variable's value from its previous value.