Introduction to Data Science Using Python

Afrand Agah, Ph.D.

A Member of The Pennsylvania Alliance for Design of Open Textbooks

The contents of this eTextbook were developed under a grant from the Fund for the Improvement of Postsecondary Education, (FIPSE), U.S. Department of Education. However, those contents do not necessarily represent the policy of the Department of Education, and you should not assume endorsement by the Federal Government.

About PA-ADOPT

The Pennsylvania Alliance for Design of Open Textbooks (PA-ADOPT) is made up of four participating institutions from the Pennsylvania State System of Higher Education (PASSHE) that are all regional and primarily undergraduate institutions, situated in Southeastern Pennsylvania. The PA-ADOPT project addresses gaps in the open textbook marketplace, improves student learning, and mitigates rising student costs. PA-ADOPT was made possible by the US Department of Education Open Textbook Pilot Program.

Preface

Data science is the process of representing models that fit data. Its goal is to predict future output based on past observations of inputs. In data science, one collects information and interprets it to make decisions.

A data scientist must have programming skills and an understanding of mathematics and statistical concepts. The first part of this book is an introduction to Python programming, which is a highly used language by data scientists, and the second part is an introduction to machine learning and statistical knowledge required for data science. Python is a popular programming language because of its scalability, readability, and strong community support. But perhaps the most important aspect is its extensive libraries and frameworks.

About OER

Open Educational Resources (OER) are instructional, learning, and research materials, digital or non, that are open-source and in the public domain or that are licensed so that users have free and perpetual permission to engage in the following activities:

- Retain: the right to make, own, and control copies of the content
- Reuse: the right to use the content in a wide range of ways
- Revise: the right to adapt, adjust, modify, or alter the content itself
- Remix: the right to combine the original or revised content with other open content to create something new
- Redistribute: the right to share copies of the original content, revisions, and remixes with others.

Table of Contents

About PA-ADOPT **3**

Preface **4**

About OER **5**

Table of Contents **6**

1. Installing Python **8**

2. Introduction to Programming **10**

2.1. Variables 11

2.1.1. Boolean Variables *13*

2.1.2. Random Variables *15*

2.2. Strings 15

2.3. ASCII Code 16

2.4. Practice Questions *21*

3. Decision Structures **22**

3.1. Nested Decision Structures 23

3.2. Practice Questions *25*

4. Repetitions **27**

4.1. While Loops 27

4.2. For Loops 29

4.3. Practice Questions *32*

4.4. Nested Loops 33

4.5. Practice Questions *34*

5. Functions **37**

5.1. Void and Value Returning Functions 38

5.2. Passing Data to and From Functions 39

5.3. Mathematical Built-in Functions 40

5.4. Practice Questions *42*

6. Recursion **43**

7. File Access **45**

7.1. Read From a File 46

7.2. Write to a File 46

7.2.1. New File *47*

7.2.2. An Existing File *47*

7.3. Notable Built-in Functions 47

7.4. Practice Questions *49*

8. Lists **50**
8.1. Practice Questions *55*
9. Arrays **56**
9.1 Practice Questions *57*
10. Plotting Graphs **58**
11. Object Oriented Programming **65**
11.1 Constructor 66
11.2 Inheritance 66
11.3 Polymorphism 67
12. Using Python Packages **69**
13. Python and Graph Theory **71**
13.1 Networkx 72
13.2 Matplotlib 75
14. Python and Machine Learning **77**
14.1 Supervised Learning 78
14.1.2 Regression *78*
14.1.3 Linear Functions *78*
14.1.4 Polynomial Functions *79*
14.2. Unsupervised Learning 81
15. Python and Statistics **84**
15.1 Standardizing Data by Scaling 86
15.2 T-Test 87
References **89**
Appendix **90**
Solutions for Practice Questions (2.4) 90
Solutions for Practice Questions (3.2) 93
Solutions for Practice Questions (4.3) 99
Solutions for Practice Questions (4.5) 103
Solutions for Practice Questions (5.4) 106
Solutions for Practice Questions (7.4) 110
Solutions for Practice Questions (8.1) 114
Solutions for Practice Questions (9.1) 115

1. Installing Python

A computer program is a list of step-by-step instructions to solve a given problem. Imagine you need to have a typewritten essay. The first issue is in what language the essay should be written: English, French, German, etc. Then, you need to use software to type in your essay. Programming languages recognize only plain text files that use a character encoding American Standard Code for Information Interchange (ASCII) to represent numbers, letters, and symbols.

Here, we will use Python as our programming language, so you will need to install Python 3.12 (the latest version of this writing) or a later version, as depicted in Figure 1-1.

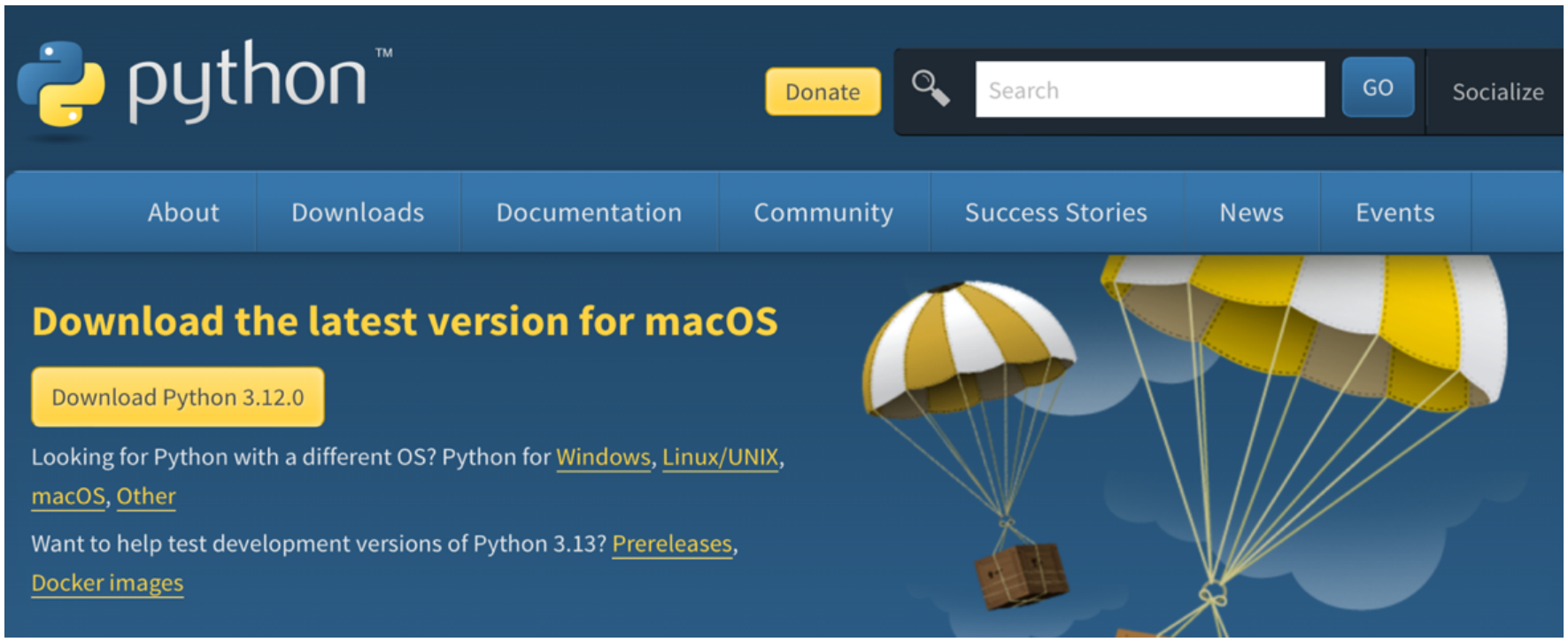

Figure 1-1 Python Interpreter

Python is available for Windows, Mac, and other platforms. When you install the Python interpreter, IDLE will also automatically be installed. IDLE is an integrated development environment for Python—a Python shell is depicted in Figure 1-2.

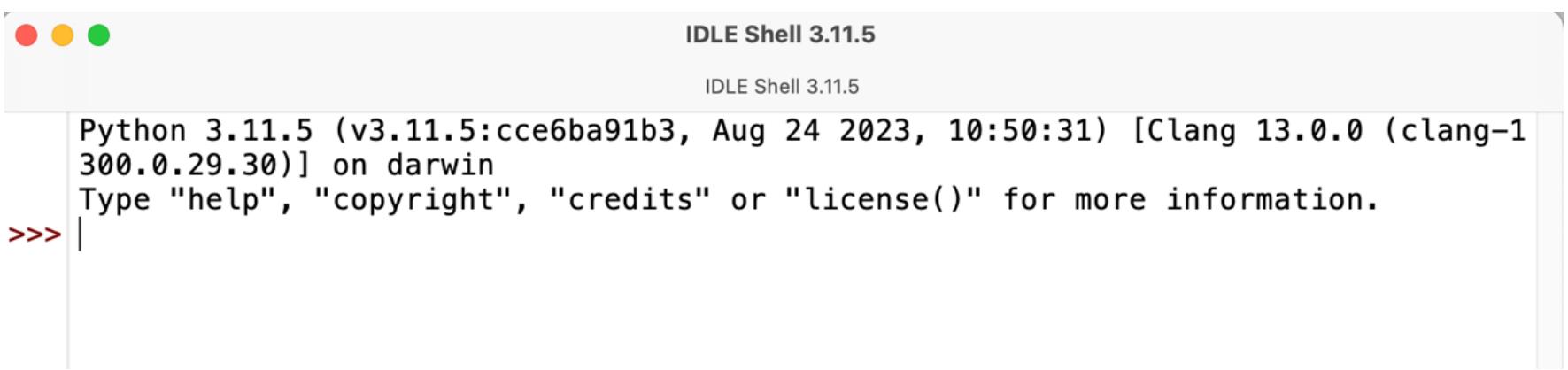

Figure 1-2 Screenshot of Python shell

Using a Python shell, you can test your Python program in an interactive mode. Or you can use the text editor to type in your Python program; this is called script mode. Python keywords are displayed in orange, comments are shown in red, string literals are indicated in green, and built-in functions are displayed in purple. The syntax of Python is easy so beginners can learn Python with no difficulty. (Python is named after a comedy show, *Monty Python*.) Python has several applications, including web development, machine learning, and data analysis.

A high-level language lets you create programs without knowing how the internal units of a computer work. A compiler translates a high-level language into a machine language program. Machine language programs can then be executed at any time. Python uses an interpreter, which translates and runs in a high-level language.

Download Python from their website. Installation steps are simple: accept the agreement and finish the installation. There are many helpful Python resources on the web, such as:

- Python Software Foundation
- Python Documentation

Running Python programs: Mistakes in programs are called errors. IDLE catches syntax errors; run time errors only occur while a program runs. When an error occurs, Python stops executing and instead displays several lines of text containing helpful information about the error.

2. Introduction to Programming

Programming is the process of writing computer programs. Python is a straightforward but powerful object-oriented programming language. A built-in function in Python is called a function, which is a prewritten program. Let us begin with print(argument), which is a widely used function. Consider the Python statement of *print("west")*; the argument is the message you write inside parentheses. The word "west" is a word or a phrase but in Python, it is considered a string literal and is the program's output, which will be displayed on the screen.

By default, the operating system connects standard output to the terminal window. String literal is any text that is enclosed in quotation marks. The quotes surrounding a string are called delimiters because they tell Python where a string begins and ends.

To display a string literal, you can use single-quote marks (') or double-quote marks ("). If you have multiple arguments in a print () function, they are separated by a space when they are displayed.

- Use *sep* to specify how to separate multiple items.
- Python inserts a space between each of the arguments of the print function.
- Use \n to display the output on the following line.
- Use \' or \" to display single-quote or double-quote marks.
- The *print( )* function automatically advances to the following line.
- Use *end* to keep the print function from advancing to the following line.
- To deal with long strings, break them into multiple lines and put a backslash at the end of every line except for the last line.

Comments improve the readability of a program, but they do not change the outcome of a program. Use the # symbol to mark the start of a line as a comment line. For comments that span several lines, use triple quotes.

2.1. Variables

The variable is a symbolic name representing a value whose associated value may change. A variable must first be created to be utilized. Variables keep values accessible. Values are assigned to variable names using the assignment operator =. The assignment operator takes the value to the right side of the operation and gives it to the name on the left side of the operator.

A variable can be overwritten for any number of times, but only the latest one will be used in the program. Consider the following statements:

```
name= 'West'

name= 'Chester'

name= 'University'
```

The variable name is being overwritten, the latest value assigned to the variable name is university, and the program cannot access the old values of the variable name. To name a variable, you must follow specific rules:

- Names are case-sensitive, so a variable named temp differs from one named Temp.
- The name of a variable cannot contain space.
- The first symbol must be a letter between a to z or A to Z.
- After the first symbol, you may use any letter, digit, or underscore symbol.
- The first symbol can be the underscore symbol.
- Descriptive names are better than short names.
- Limiting variable names to a maximum of three or four is a good rule of thumb.
- Do not use Python keywords; Here is a list of some Python keywords: and, def, elif, else, FALSE, for, if, import, in, not, or, pass, return, TRUE, and while.

There are different numerical types in Python, such as integers and float. Consider 7 and 7.1234, both are numerical values, the first one is an integer and the second one is a float. *int()* and *float()* can produce numbers of a specific

type. *Int()* returns an integer constructed from a number or string. *Float()* returns a floating-point number built from a number or string.

Floating-point numbers always have at least one decimal place of precision. Python supports mixed arithmetic: when a binary arithmetic operator has operands of different numeric types, the operand with the narrower type is widened, where the integer is more limited than the floating point.

To read data from the keyboard as input, use the *input(prompt)* function, which reads a line from input and converts it to a string (stripping a trailing new line). If the prompt argument is present, it is written to standard output. Consider the following two statements:

```
name= input ('Please enter a name')

print (name)
```

This is a Python program to get input from the user. The string literal is the prompt displayed on the screen, and the print statement will show the input on the screen.

One primary reason to use a variable is to remember a value from one part of a program to be used in another part. Let's assume that we need to have three variables, and we call them x, y, and z. If

```
x = 10

y = 100

z = -2

y = 10

x = x + y + z
```

Then, variable x is initialized with the value of 10; variable y is initially 100, then overwritten by 10, and the final value of x is 18.

All numeric types support the four primary mathematical operations: sum (+), difference (-), product (*), and quotient (/). Additionally, you can use the following operations: exponent (**), floored quotient (//), and the remainder (%).

2.1.1. Boolean Variables

Boolean variables: Any object can be tested for truth value in conditional statements. A Boolean variable is either true or false. Boolean returns zero for false and one for true. You can use and, or and not to combine Boolean variables. Table 2-1. depicts the logical meaning of using and, or operators.

P	Q	P AND Q	P OR Q
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE

Table 2-1 Logical And, Or

AND operator takes two expressions and generates one new expression that is true if both expressions are true. *OR* operator takes two expressions and generates one new expression that is true if at least one of the expressions is true.

Precedence of operation: To determine which operation needs to be evaluated first, an operator with the highest precedence will be considered first if an expression has more than one operator.

The highest priority is given to the following:

- exponentiation
- remainder, multiplication, division
- sum and difference

Python provides several ways to format a text. F-string embeds expressions inside string literals. F-string is an expression that will be evaluated at run time and is not a constant value. A string must end with the same character; if it begins with a single quote, it must end with a single quote.

F-strings are string literals that have an *f* before the opening quotation mark. Include Python expressions enclosed in curly braces. Python will replace those expressions with their resulting values.

Consider the following statements in Python:

```
number = 1234.56

print (f'{number}')

print (f'{number:10.4f}')

print (f'{number:.1f}')

print (f'{number:,.5f}')
```

This example illustrates formatting a given number in four different ways. The number after the ":" represents the width of the displayed output, the number after the "." means the total number of digits after the decimal point and will be rounded up or down accordingly, "," is used to separate every three digits from the integer part of the number.

A Python program to convert a degree from Fahrenheit to Celsius can be written as follows:

```
F = 212

C = (F - 32) * 5/9

print (C)
```

Another example is to write a Python program to convert seconds into hours, minutes, and seconds. We may have the following program:

```
Num = 86400

H = Num / 3600

M = (Num % 3600) // 60

S = Num % 60

print (H, ":", M, ":", S)
```

2.1.2. Random Variables

There are several ways to generate random numbers in Python using its functions. The built-in *random()* function generates a random float number between 0.0 and 1.0, with 53-bit precision. Each time this function is called, it returns values uniformly distributed in the given range.

A Python program can gain access to another Python program by importing. By importing, you bind two or more programs together. The interpreter has built-in functions, such as the *print()* function. If a function is not part of the standard library of Python, then you need to use import to bind it to your Python program; for example, you may write it as the following:

```
import random

number = random.randint (5,10)

print (number)
```

which generates a random number in the given range; the randomly generated number includes five and ten. The built-in randint(a,b) function generates a random number that is an integer and is between a and b, including a and b.

2.2. Strings

The string is a primary data type in Python. Strings are used to represent texts. A string contains individual symbols or elements. Another built-in function in Python is *len()*, which returns the total number of elements in each string.

One can access each string element in Python by referring to its index. Indices start from zero and proceed with increments of one at a time. Therefore, every element of a string has a numbered position. Consider a string such as Name = “ABCD”; here, Name [0] is A, Name [1] is B, and so on.

Sometimes, we may need to select more than one element of a string, technically a span of elements is called a slice. To get a slice of the above Name variable in Python, one can have Name [start: end], which returns elements starting at position start, up to (not including) the element at position end.

Strings are immutable, which means they cannot be changed once they have been created. The “+” operator concatenates two strings; this means “1” + “2” is “12” and not “3”. If the element on either side of “+” is a string, Python performs concatenation; if both sides are numbers, then it is a mathematical addition.

In Python, one can multiply a string by an integer, which will result in displaying the string for an integer number of times. However, you cannot multiply two strings. One can convert strings to numeral values by use of *int(input(prompt))* or *float(input(prompt))*.

2.3. ASCII Code

The American Standard Code for Information Interchange is called ASCII and is used for communication between computers. In this coding system, each character and symbol is translated into a code readable by computers.

ASCII uses one byte of memory to display symbols, therefore it only displays 256 possible characters. This includes numbers, capital and lowercase letters, and a few more symbols. This is not enough and a new standard has been created and is called Unicode, which utilizes more than one byte supports more than 65,000 characters, and includes all letters from all international languages and symbols. Table 2-2 depicts the ASCII table.

DEC.	HEX.	Symbol	Symbol Description
0	00h	NULL	(Null character)
1	01h	SOH	(Start of Header)
2	02h	STX	(Start of Text)
3	03h	ETX	(End of Text)
4	04h	EOT	(End of Transmission)
5	05h	ENQ	(Enquiry)
6	06h	ACK	(Acknowledgement)
7	07h	BEL	(Bell)
8	08h	BS	(Backspace)
9	09h	HT	(Horizontal Tab)
10	0Ah	LF	(Line feed)
11	0Bh	VT	(Vertical Tab)
12	0Ch	FF	(Form feed)
13	0Dh	CR	(Carriage return)
14	0Eh	SO	(Shift Out)
15	0Fh	SI	(Shift In)
16	10h	DLE	(Data link escape)
17	11h	DC1	(Device control 1)
18	12h	DC2	(Device control 2)

DEC.	HEX.	Symbol	Symbol Description
19	13h	DC3	(Device control 3)
20	14h	DC4	(Device control 4)
21	15h	NAK	(Negative acknowledgement)
22	16h	SYN	(Synchronous idle)
23	17h	ETB	(End of transmission block)
24	18h	CAN	(Cancel)
25	19h	EM	(End of medium)
26	1Ah	SUB	(Substitute)
27	1Bh	ESC	(Escape)
28	1Ch	FS	(File separator)
29	1Dh	GS	(Group separator)
30	1Eh	RS	(Record separator)
31	1Fh	US	(Unit separator)
32	20h		(Space)
33	21h	!	(Exclamation mark)
34	22h	"	(Quotation mark; quotes)
35	23h	#	(Number sign)
36	24h	$	(Dollar sign)
37	25h	%	(Percent sign)
38	26h	&	(Ampersand)
39	27h	'	(Apostrophe)
40	28h	(	(round brackets or parentheses)
41	29h	)	(round brackets or parentheses)
42	2Ah	*	(Asterisk)
43	2Bh	+	(Plus sign)
44	2Ch	,	(Comma)
45	2Dh	-	(Hyphen)
46	2Eh	.	(Dot, full stop)
47	2Fh	/	(Slash)
48	30h	0	(number zero)
49	31h	1	(number one)
50	32h	2	(number two)
51	33h	3	(number three)
52	34h	4	(number four)

DEC.	HEX.	Symbol	Symbol Description
53	35h	5	(number five)
54	36h	6	(number six)
55	37h	7	(number seven)
56	38h	8	(number eight)
57	39h	9	(number nine)
58	3Ah	:	(Colon)
59	3Bh	;	(Semicolon)
60	3Ch	<	(Less-than sign)
61	3Dh	=	(Equals sign)
62	3Eh	>	(Greater-than sign; Inequality)
63	3Fh	?	(Question mark)
64	40H	@	(At sign)
65	41h	A	(Capital A)
66	42h	B	(Capital B)
67	43h	C	(Capital C)
68	44h	D	(Capital D)
69	45h	E	(Capital E)
70	46h	F	(Capital F)
71	47h	G	(Capital G)
72	48h	H	(Capital H)
73	49h	I	(Capital I)
74	4Ah	J	(Capital J)
75	4Bh	K	(Capital K)
76	4Ch	L	(Capital L)
77	4Dh	M	(Capital M)
78	4Eh	N	(Capital N)
79	4Fh	O	(Capital O)
80	50h	P	(Capital P)
81	51h	Q	(Capital Q)
82	52h	R	(Capital R)
83	53h	S	(Capital S)
84	54h	T	(Capital T)
85	55h	U	(Capital U)
86	56h	V	(Capital V)

DEC.	HEX.	Symbol	Symbol Description
87	57h	W	(Capital W)
88	58h	X	(Capital X)
89	59h	Y	(Capital Y)
90	5Ah	Z	(Capital Z)
91	5Bh	[	(square brackets or box brackets)
92	5Ch	\	(Backslash)
93	5Dh	]	(square brackets or box brackets)
94	5Eh	^	(Caret or circumflex accent)
95	5Fh	_	(underscore, understrike, underbar or low line)
96	60h	`	(Grave accent)
97	61h	a	(Lowercase a)
98	62h	b	(Lowercase b)
99	63h	c	(Lowercase c)
100	64h	d	(Lowercase d)
101	65h	e	(Lowercase e)
102	66h	f	(Lowercase f)
103	67h	g	(Lowercase g)
104	68h	h	(Lowercase h)
105	69h	i	(Lowercase i)
106	6Ah	j	(Lowercase j)
107	6Bh	k	(Lowercase k)
108	6Ch	l	(Lowercase l)
109	6Dh	m	(Lowercase m)
110	6Eh	n	(Lowercase n)
111	6Fh	o	(Lowercase o)
112	70h	p	(Lowercase p)
113	71h	q	(Lowercase q)
114	72h	r	(Lowercase r)
115	73h	s	(Lowercase s)
116	74h	t	(Lowercase t)
117	75h	u	(Lowercase u)
118	76h	v	(Lowercase v)
119	77h	w	(Lowercase w)
120	78h	x	(Lowercase x)

DEC.	HEX.	Symbol	Symbol Description
121	79h	y	(Lowercase y)
122	7Ah	z	(Lowercase z)
123	7Bh	{	(curly brackets or braces)
124	7Ch	|	(vertical-bar, vbar, vertical line or vertical slash)
125	7Dh	}	(curly brackets or braces)
126	7Eh	~	(Tilde; swung dash)
127	20h	DEL	(Delete)

Table 2-2 ASCII Characters

Every character in the Unicode System has a numerical equivalent. Python utilizes built-in functions such as *chr()* and *ord()* for these coding systems. Use *chr()* to get the numerical value of a character and use *ord()* to get the character equivalent to a numerical value.

Table 2-3 depicts Python *print()* commands, which use some of the above built-in functions.

Code	Output
`print(ord('A'))`	An Integer representing character A is 65
`print(ord('a'))`	An Integer representing character a is 97
`print(chr(65))`	A character representing unicode 65 is A
`print("\u221a")`	√ (The symbol of square root)
`print("\u270E")`	✎ (A symbol of a pen pointing at right)
`print("\u2709")`	✉ (A symbol of an envelope)

Table 2-3 Some special symbols using ASCII extended characters

2.4. Practice Questions

1. Write a Python program to get a number, then calculate its squared and cubed value.

2. Write a Python program to get a number such as *n* as input and then calculate n^n.

3. Write a Python program to get a number that contains three digits and then display each digit on a single line.

4. Write a Python program to get a circle's radius and calculate its area.

5. Write a Python program to find the roots of a quadratic equation.

6. Write a Python program to get a name as input and display it five times.

7. Write a Python program to get today’s date as three integers and display it as m/day/year.

8. Write a Python program to simulate the roll of a die.

9. Write a Python program to simulate the roll of a pair of dice.

10. Write a Python program to generate an even random number between one and one hundred.

Go to 2.4 Practice Question Solutions.

3. Decision Structures

Decision is the process of deciding something. This decision is always based on a logical question or condition in programming languages. Decision structures, also known as if-then-else or conditional statements, change the path of executing a program. A program is a sequence of several lines of code. Anytime we have a decision structure in a program, the program flow could go to different paths, and it is all based on the given logical condition. Many times, a computer problem can not be solved by executing sequential lines of codes, and alternative paths of executions are needed, where based on a condition, the program could take alternative paths. We use decision structures to handle these alternative paths of programs.

For example, whether an integer is even or odd, we would like to display a message; accordingly, we must have two separate printing messages in this case, only one will execute based on the given input. Therefore, the control of the program goes to different paths. Let us consider the following program:

```
Num=12

if (Num %2 ==0):
    print (“Even”)
else:
    print (“Odd”)
```

If the condition in front of the if statement is true, the program displays “Even”; otherwise, it shows “Odd.” It would be impossible to show both messages. In Python, the structure of a conditional statement is as follows:

```
If (question/condition):
    Action 1
else:
    Action 2
```

The indentation is required, and control of the program goes either to Action 1 or Action 2. After both if and else statements, you need to include a colon. The if and else statements must be aligned together. Spaces matter at the beginning of lines but not elsewhere.

Comparison operations: A logical condition that uses numerical values can incorporate operations such as less, less or equal, more, more and equal, equal, and not equal to compare several operands, as depicted in Table 3-1.

Operation	Meaning
<	Less than
<=	Less or Equal
>	Greater
>=	Greater or Equal
!=	Not Equal
= =	Equal

Table 3-1 Comparison Operators

Conditional statements can be combined by use of relational operators, as discussed in Table 2-1.

3.1. Nested Decision Structures

Conditional statements can be nested, as it is possible to have a program with a series of conditions that need testing. In the example given in section 3, Action 1 and Action 2 each could be another conditional statement such as the following:

```
if (Q1):
    if (Q2):
        Action 1
    else:
        Action 2
else:
    if (Q3):
        Action 3
    else:
        Action 4
```

Since the logic of nested conditional statements can be complex, a more straightforward approach is available, which is called an if-elif-else statement.

```
if(Q1):
    Action 1
elif(Q2):
    Action2
elif(Q3):
    Action 3
else:
    Action 4
```

In each conditional statement, if the condition is true, the statements following if are executed, and if the condition is false, the statements following else are executed. The following program finds the largest value among three numbers.

```
num1=0
num2=-1
num3=1
if(num1>=num2) and (num1>=num3):
    maximum=num1
elif(num2>=num1) and (num2>=num3):
    maximum=num2
else:
    maximum=num3
print("The maximum value is",maximum)
```

3.2. Practice Questions

1. Write a Python program to get a number and verify if it is divisible by five.

2. Write a Python program to get a number and verify if it is divisible by five or by three.

3. Write a Python program to get a year and verify whether it is a leap or a common year. A year is a leap year if it is divisible by four, except those years divisible by a hundred are not leap years unless they are also divisible by four hundred.

4. Write a Python program to get a date and check if it is a magic date. A date is magical when you multiply month by day and get the year. For example, February 12th, 2024 is a magical date: $2 \times 12 = 24$.

5. Write a Python program to get a number between one and five and convert it to a Roman numeral.

6. Write a Python program to check if a given input is odd or even without using remainder (%).

7. Write a Python program to get a word that contains three symbols and display it in reverse order.

8. Write a Python program to get a word with a length of four as input and then display each string element on one line.

9. Write a Python program to get a word with a length of four as input, then switch the first and last elements and redisplay the new word.

10. Write a Python program to get a word with a length of four as input and verify if it is a palindrome.

11. Write a Python program to get a word with a length of four as input and then reverse it.

12. Write a Python program to get a word with a length of four as input, and if it starts with ‘a’, replace ‘a’ with ‘A’ and redisplay the name; otherwise, display the last two elements of the name.

13. Write a Python program to get a word, and if the length of the word is even, extract the first element and display it; otherwise, display the last element of the string.

14. Write a Python program to get a word with a length of four as input and count the total number of times it contains ‘e’ or ‘E.’

15. Write a Python program to get a word with a length of four as input and then replace every ‘e’ with ‘X.’

Go to 3.2 Practice Question Solutions.

4. Repetitions

Repetition is the recurrence of an action. In programming, it means repeating something that has already been written. Imagine you need to write a program to get a student's name, last name, and ID and then display these on the screen. A Python program to do so would be like the following example:

```
Name=input("Please enter a name")

Last=input("Please enter a last name")

Number=input("Please enter an ID#")

print(Name,Last,Number)
```

4.1. While Loops

Now imagine that you must write the above program for five hundred students. One way is to copy the above program five hundred times, which is impractical. And what if we had an even more significant number of students? A feasible way is incorporating repetitions, and that means utilizing loops.

There are several ways that you can use loops. The first approach is called a while loop. Let us have a closer look at a while loop. Generally, the structure of the while loop is as follows:

```
while (condition):

    Here is the body of the loop
```

The loop's body is the part of the program that will be repeated; this part should be indented. The total time of the repetition depends on the given condition at the beginning of the while loop. This condition is either true or false.

- If the condition is true, the body of the while loop will be repeated.
- As soon as the condition is not valid anymore, the program's control goes out of the loop and directly to the first line right after the while loop, and repetition stops.
- If the condition is not valid, the while loop never begins.

- If the condition is always true, the program goes into an infinite loop, and the programmer must halt the program by pressing external keys on the keyboard to stop the loop.
- The condition must be updated inside the loop's body to ensure that the while loop would not stick in an infinite loop.

To write a Python program to display numbers between one and five, we have:

```
n=1
while(n<6):
    print(n)
    n=n+1
```

Here, *n* begins at one, and if *n* is less than six, it prints the current value of *n* and increases the value of *n* at each iteration of the while loop by one. To write a Python program to display numbers between five and one, we have the following:

```
n=5
while(n>0):
    print(n)
    n=n-1
```

Here, *n* begins at five, and if *n* is greater than zero, it prints the current value of *n* and decreases the value of *n* at each iteration of the while loop by one. A Python program to display the sum of numbers from one to ten would be:

```
n=1
total=0
while(n<=10):
    total=total+ n
    n=n+1
print(total)
```

Here, we need to have two variables: total, which represents the total summation of numbers, and *n*, which represents the total ten numbers that we

are adding together. At each iteration inside the while loop, we need to add the current value of *n* to the total summation.

In our last example, we write a Python program to display the following sum $0+10+20+30+\ldots+100$. As you can see, the only difference is the increments of *n*, which go by increments of ten at each iteration.

```
n=0

total=0

while (n<=100):

    total=total+ n

    n=n+10

print (total)
```

4.2. For Loops

There are several ways that you can utilize loops. The second approach is called a for loop. Generally, the structure of the for loop is as follows:

```
for (condition):

    Here is the body of the loop
```

The condition can be written as a variable name in range (number of times to repeat). As we saw in the while loop, the body of the loop is the part of the program that will be repeated; this part should be indented. The total time of the repetition depends on the given condition at the beginning of the for loop. This condition is always a Boolean variable.

- The condition should be given as a range.
- Elements of the range should be separated by commas and listed inside a set of brackets.

Python has a built-in function, which is called range. The *range ()* function returns a sequence of numbers. By default, it begins at zero, with increments of one.

- range (m)
  - ✦ Generates numbers from 0 up to and including m-1.

- range (m, n)
  - ✦ Generates numbers from m up to and including n-1.
- range (m, n, k)
  - ✦ Generates numbers from m up to and including n-1 with the increments of k.

Now, let us rewrite the programs that we discussed in section 4-1, by using for loops. To write a Python program to display numbers between one and five, we have the following:

```
for n in range (1,6)

    print(n)
```

Here, *n* begins at one, and increases by one at each iteration of the loop, the last value of *n* is five.

To write a Python program to display numbers between five and one, we have the following:

```
for n in range (5,0,-1)

    print(n)
```

Here, *n* begins at five and decreases by one at each iteration, the last value of *n* is 1.

A Python program to display the sum of numbers from one to ten would be as follows:

```
total=0

for n in range (1,11)

    total=total+ n

print(total)
```

Here, we need to have two variables: total, which represents the total summation of numbers, and *n*, which represents the total ten numbers that we are adding together. At each iteration inside the for loop, we need to add the current value of n to the total summation.

In our last example, we write a Python program to display the following sum $0 + 10 + 20 + 30 + \ldots + 100$. As you can see, the only difference is the increments of *n*, which go by increments of ten at each iteration.

```
total=0

for n in range (0,101,10)

    total=total+ n

print(total)
```

4.3. Practice Questions

1. Write a Python program to display numbers between one and five.

2. Write a Python program to display numbers between five and one.

3. Write a Python program to show at which temperature Fahrenheit and Centigrade have the same reading.

4. Write a Python program to calculate the following sum: $1 + 2 + 3 + ... + 10$.

5. Write a program to calculate the following sum: $\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{10}$.

6. Write a program to display numbers between one and five using a for loop.

7. Write a program to display numbers between five and one using a for loop.

8. Write a program to simulate rolling a pair of dice and show how many tries it takes to get a pair.

9. Write a program to get five numbers as input and calculate their average.

10. Write a program to get inputs from the keyboard, where “-1” indicates the end of inputs, and then calculate their average.

11. Write a program to check whether a given word is a palindrome.

12. Write a program to get a word as input and replace every string character with its first character; for example,“abcd” would change to “aaaa”.

Go to 4.3 Practice Question Solutions.

4.4. Nested Loops

A nested loop means a loop inside another loop, and these are typically used for working with two dimensions. When a loop is nested inside another loop, the inner loop runs many times inside the outer loop. The inner loop will be re-started in each iteration of the outer loop. The inner loop must finish all its iterations before the outer loop can continue to its next iteration. Consider the following program:

```
for i in range (1,4):
    for j in range (1,7):
        print ("*", end=" ")
    print ("\n")
```

Here, for every iteration of the outer loop, the program must finish all iterations of the inner loop. The outer loop iterates three times, and the inner loop iterates six times. The program generates three lines; on each line, it prints six asterisks, (as depicted in Figure 4-1).

```
* * * * * *

* * * * * *

* * * * * *
```

Figure 4-1

4.5. Practice Questions

1. Write a Python program to display the pattern in Figure 4-2.

```
*
* *
* * *
* * * *
```

Figure 4-2

2. Write a Python program to display the pattern in Figure 4-3.

```
1
2 2
3 3 3
4 4 4 4
```

Figure 4-3

3. Write a Python program to display the pattern in Figure 4-4.

```
1
1 2
1 2 3
1 2 3 4
```

Figure 4-4

4. Write a Python program to display the pattern in Figure 4-5.

```
1 2 3 4
1 2 3
1 2
1
```

Figure 4-5

5. Write a Python program to display the pattern in Figure 4-6.

```
1 2 3 4    1 2 3    1 2    1
1 2 3 4    1 2 3    1 2
1 2 3 4    1 2 3
1 2 3 4
```

Figure 4-6

6. Write a Python program to display the pattern in Figure 4-7.

```
    *
   * *
  * * *
 * * * *
* * * * *
```

Figure 4-7

7. Write a Python program to display the pattern as depicted in Figure 4-8.

```
    0
   11
  222
 3333
44444
```

Figure 4-8

8. Write a Python program to display the pattern as depicted in Figure 4-9.

```
    0
   1 1
  2 2 2
 3 3 3 3
4 4 4 4 4
```

Figure 4-9

9. Write a Python program to display the pattern as depicted in Figure 4-10.

```
    0
   0 1
  0 1 2
 0 1 2 3
0 1 2 3 4
```

Figure 4-10

10. Write a Python program to display the pattern as depicted in Figure 4-11.

```
    1
   2 1
  3 2 1
 4 3 2 1
5 4 3 2 1
```

Figure 4-11

Go to 4.5 Practice Question Solutions.

5. Functions

A function is a block of code that only runs when called. You can pass data into a function. Therefore, you can write a code block, call it a function, and then use it many times. A large Python program can be broken into several functions by isolating each task into one function.

To call a function, write the function's name followed by two parentheses (). The same set of restrictions for a variable's name is applied while naming a function. The structure of a function in Python is as follows:

```
def function's name():

    Body of the function
```

The first line of a function is its definition and is called the function header; it begins with the keyword def. Every line of code that is part of a block of codes is indented. To run a function, it must be called. To call a function, write its name followed by a set of parentheses.

Let us consider the following program:

```
def func1():

    print("you just called function1")

def func2 ():

    print("you just called function2")

for m in range(0,3):

    func1()

    func2()
```

Here *func1* and *func2* are two functions we defined; each prints a literal string. In the for loop, each function is called three times. When you define a function, it is created, but you still need to call it. When a function is called, the program's control is transferred to that function.

Now let's consider the following program:

```
m=20
def func1():
    m=100
func1()
print (m)
```

variable *m* is initialized with a value of twenty; in function *func1()*, there is also a variable *m*, two different variables with the same name. Variable m in *func1()* is a local variable, and any changes that happen to m in *func1()* do not affect variable m that is outside of *fucn1()*.

A local variable is any variable that is created inside of a function and is only accessible from inside of the function. Several functions can have variables with the same name, and they do not affect each other, as they cannot access each other's variables.

5.1. Void and Value Returning Functions

One way to categorize functions is based on the value that they return. A void function is a function that does not return any value; it executes its block of code and then terminates. On the other hand, a value-returning function executes its block of code and then returns a value to the program that called it. In the following example, *func1()* is the function that prints a variable and terminates; it is a void function.

```
def func1():
    print(m)
```

A value-returning function must have a return statement as the last line in the function. In the following example, *func1()* is the function that returns a value stored in variable *m*; since it is returning a variable, *func1()* is a value-returning function. A function may return more than one value; in Python, if you have more than one value to return, the result is displayed as a list, such as (a,b,c), members in a set of parentheses separated by commas.

```
def func1():
    return(m)
```

5.2. Passing Data to and From Functions

Data passed to a function when called is referred to as an argument, and the variable that receives an argument is called a parameter. Consider the following example:

```
def func1(word):
    m=word[0]
    return (m)
func1(name)
```

In this example, the name is data passed to *func1()* and is an argument, and the word receives this argument and is called a parameter. In this example, func1() extracts and returns the first symbol of the argument word.

In the following example, func1() has two parameters as indicated in its header, it will append them together and display the result. When we call this function we need to list two parameters, otherwise it would be a compile error.

```
def func1(fname,lname):
    print(fname+" "+lname)
func1("Steven","Jobs")
```

In the following example, func1(), we use a loop to print all elements of a given list.

```
dir=["West","East","North","South"]
def func1(dir):
    for d in dir:
        print(d)
func1(dir)
```

5.3. Mathematical Built-in Functions

In addition to writing your defined functions, Python has built-in mathematical functions that you can use. And, like using random numbers, you need to import a standard library into your Python program. Some built-in functions and variables in Python that you can incorporate into your program are as follows:

- sqrt(x): to display the square root of a number.

```
import math

print(math.sqrt(64))
```

- floor(x): to round down the number to the nearest integer.

```
import math

x=123.45

print(math.floor(x))
```

- ceil(x): to round up the number to the nearest integer.

```
import math

x=123.45

print(math.ceil(x))
```

- exp(x): to return E raised to the power of x.E is the base of the natural system of logarithms or the Euler's number (2.71).

```
import math

x=2

print(math.exp(x))
```

- math.e: to return E, the base of the natural system of logarithms or the Euler's number.

```
import math

print(math.e)
```

- math.pi: to return the value of PI (3.14149).

```
import math

print(math.pi)
```

- sin(x): to return the sine of the x, which should be in radians.

```
import math

x=90

print(math.sin(x))
```

- cos(x): to return the cosine of the x, which should be in radians.

```
import math

x=0

print(math.cos(x))
```

- tan(x): to return the tangent of the x, which should be in radians.

```
import math

x=90

print(math.tan(x))
```

- log(x): to return the logarithm base 2 of x.

```
import math

x=2

print(math.log(x))
```

5.4. Practice Questions

1. Write a Python program to generate a random number between zero and five, then simulate a magic ball. You may select your message from these options: without a doubt, better not tell you now, my sources say no, ask again later, outlook hazy.

2. Write a Python program to get a word from a user through the keyboard and then display the first character of that word.

3. Write a Python program to get a word from a user through the keyboard and then display the first character of that word. Extracting the character should be done in a function, but displaying the character should be done in the main program.

4. Write a Python program to get two numbers as input and calculate their average.

5. Write a Python program to get a degree in Fahrenheit and convert it to Celsius.

6. Write a Python program to get an input (integer) and calculate its factorial, where the factorial of a number is denoted by n! and is the product of all positive integers less than or equal to n. For example: $5! = 5 \times 4 \times 3 \times 2 \times 1$.

7. Write a Python program to generate the first ten Fibonacci numbers, where each is the sum of the two preceding ones. These are numbers in the Fibonacci series: $1, 1, 2, 3, 5, 8, \ldots$.

8. Write a Python program to get a word from a user through the keyboard and then reverse the word through the use of a function.

Go to 5.4 Practice Question Solutions.

6. Recursion

When a function calls itself, it is called a recursive function. Making a function call itself is a way to break down complicated problems into more straightforward problems that are identical in structure to the original problem. In recursive programming, there is always one base case, which does not require recursion and stops the chain of recursive calls.

Any problem that can be solved recursively can be solved by using loops. Repetitive problems are more easily solved using recursion.

Example 1: a recursive program to calculate the summation of the first ten positive integers.

```
n=10

def Sum (n):

    if (n==0):

        return 0

    else:

        return Sum(n-1)+n

print (Sum(n))
```

To calculate the summation of the first ten positive integers, we need summation of the first nine positive integers and then add ten to that summation; however, to calculate the summation of the first nine positive integers, we need summation of the first eight positive integers and then add nine to that, and inductively for each new summation you need the previous summation. The base case here is the summation of numbers from zero to zero, which is zero.

Example 2: a recursive program to calculate the factorial of five, where the factorial of a non-negative integer *n* is a product of all positive integers less than or equal to *n*.

```
n=5

def Fact(n):

    if (n==1):

        return 1

    else:

        return n*Fact(n-1)

print (Fact(n))
```

The *n* factorial also equals the product of *n* with the next smaller factorial.

Example 3: a recursive program to generate the first ten elements of the Fibonacci sequence. The Fibonacci sequence is a sequence in which each number is the sum of the two preceding ones. The sequence starts with two ones.

```
def Fib (n):

    if ((n==1) or (n==0)):

        return 1

    else:

        return Fib(n-1) +Fib(n-2)

for i in range (0,10):

    print (Fib(i), " ", end= "").
```

7. File Access

A file is a series of bytes used to store data; this data is organized in a specific format; some are easy to read by a human, like text files. However, all files are then translated into binary for processing by the computer. Every time that you use Word processing programs or image editors, you work with files.

For every program that we wrote, that required the user to enter data, as the input data was not retained at a permanent memory location, we needed to re-enter the data for each run of the program. To save the data between the runs, one way is to save the data on a file, which means saving the data on the computer's disk, one form of permanent memory.

You can read and write all types of files; in this chapter, we only work with text files. In a text file, data is encoded as a text using ASCII or Unicode formatting, therefore even numerical values are stored as a series of characters. A text file can be opened and viewed by any text editor.

There are two different ways to access data in a file, sequential access and random access.

In sequential access, you must access data from the beginning of the file to the end of the file; you can not skip any piece of data; you have to read all the data that comes before the desired data. On the other hand, in random access, direct access to any piece of data in the file is possible. This is like accessing different tracks on a CD or DVD.

In a text file, each line of the text is ended with the new line character ('\n'). You can open a file, read from it, or write on it; if the file does not exist, it will be created; you can also append data to a file if it already exists.

Remember that at the end of a Python program, you should permanently close every open file, freeing up the memory space used by the opened files. When a file is closed, the connection between the file and the program is removed, and should you need to access the same file, you need to reopen the file.

7.1. Read From a File

Retrieving data from a file is known as reading from a file. Suppose you need to access a text file to read it; the following statement reads the first line of the file and copies it into a string line.

```
line=open('filefortest.txt',‘r’)
```

If the text file is not in the current subdirectory, then you must specify the entire path to your file.

```
line=open('c:/users/Agah/desktop/filefortest.txt',‘r’)
```

- You can use *readline()*, which reads a line from a file and returns it as a string.
- You can use *read()*, which reads the entire file.
- You can use *read(n)*, which reads n bytes of a file.
- ‘\n’ is considered two bytes. It marks the location where a new line begins in a file.
- When strings are printed, ‘\n’ causes an extra blank line to appear.

7.2. Write to a File

Every file has a name and a type; the type of a file is a series of three characters, which appear after the filename followed by a dot. The .*txt* indicates that the file is a text file.

To write on a file in Python, first, you need to open the file for writing. To write on a file, you can use the same *print ()* function we used previously. When you are done accessing a file, you must close the file.

To convert a numerical value to a string, use the *str ()* function.

7.2.1. New File

To create a new file, you can use the following statements:

```
file=open('filefortest.txt','w' )

file.write( "WCU \n" )

file.write(f'{"abc"}\n')

file.close()
```

Here 'w' indicates that the file is being accessed for writing, and a new file will be created. If the file already exists, it will be overwritten, and the data on the original file is not retrievable anymore. The statement *open()*, opens the file for writing, and *write()*, writes data on the file; every time that you write on a file, it is in the form of a string; you can either use string literals or use F-formatting to write the data.

7.2.2. An Existing File

If the file already exists and you wish to append data to the end of the existing file, then use the following statements:

```
file=open('filefortest.txt','a' )

file.write( "WCU \n" )

file.write(f'{"abc"}\n')

file.close()
```

Here, 'a' indicates that the data will be appended to the end of the existing file.

7.3. Notable Built-in Functions

When a file is opened for reading for the first time, a read pointer is positioned at the beginning of the file, after the first read, it is maintained to mark the location of the next item that will be accessed from the file.

As discussed in section 7.1, '\n' separates items in a file, and it causes an extra blank line to appear in a file. If you need to remove the extra line, you can utilize a built-in function called *rstrip()*. This function strips a specific character from the end of a given string. For example, *name.rstrip('\n')*, removes the trailing '\n' from the end of the string name.

Each time that you open a file for writing, data is written on the file as strings. Python has a built-in function *str()*, which converts a value into a string, which you can use to directly write numerical values as strings on a file.

7.4. Practice Questions

1. Write a Python program to read a file and print every line of the file on the screen by using a for loop.

2. Write a Python program to read a file and print every line of the file on the screen by using a while loop.

3. Write a Python program to read a file and print the total number of lines in the file.

4. Write a Python program to read a file and print every word in the file that begins with ‘A’.

5. Write a Python program to get an input from the keyboard and then add it to an existing file.

6. Write a program in Python to generate five random numbers and write them in a file.

7. Write a program in Python to read a file and check if it contains a specific word.

8. Write a program in Python to read a file and count the total number of e’s in the file.

9. Write a program in Python to write five numbers on a file and then display their summation.

Go to 7.4 Practice Question Solutions.

8. Lists

A list is an entity that contains multiple data items. Use square brackets to indicate the start and end of the list and separate items in a list with commas. [] is the empty list. You can change items that are in a list, which means a list is mutable, a programmer may add or remove items from a list. We use lists to store multiple items in a single variable. A list of items enables a programmer to keep related data values together.

The Indexing and slicing that we used as built-in functions over strings can be utilized on lists too. Each item of a list is called an element of the list. Here is a list of four integers:

num=[1,2,3,4]

A list may contain elements that have different types. For example:

num=[1,`east',`west',2.5]

is a list that contains four different elements of three different types. You can use the print() function to print entire elements of a list.

Similar to strings, num[0] is the first element of the list num, and also num[:3] returns the first three elements of the list, which is one slice of a list, the same concept that we had with strings. Negative indexing means to start from the end, therefore -1 refers to the last element of the list.

You can use the "+" operator to add one list to the end of another list. And use the '*' operator to repeat a list. For example ['*']*13, displays thirteen '*'.

In Python you can use the following built-in functions on a list:

- index(a): to return the index of the first occurrence of a in the list; consequently the output of the following program is 1.

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

print(num.index(2))
```

- len: to return the total number of elements in a list; consequently the output of the following program is 5.

```
num=[1,2,3,4,15]

print(len(num))
```

- max: to return the largest elements in a list; consequently the output of the following program is 5.

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

print(max(num))
```

- min: to return the smallest element in a list; consequently the output of the following program is 1.

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

print(min(num))
```

- remove (a): to remove the first occurrence of the element a from the list; consequently the output of the following program is [1, 2, 4, 5].

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

num.remove(num[2])

print(num)
```

- sort: to sort elements in a list in ascending order; consequently the output of the following program is [-5, -2, 1, 4, 31].

```
num=[1,-2,31,4,-5]

num.sort()

print(num)
```

- sum: to return the sum of the elements in a list; consequently the output of the following program is 1.

```
num=[1,-2,3,4,-5]

print(sum(num))
```

- append: to add an element to the end of an existing list.

- clear: to remove all elements from a list.
- copy: to return a copy of the list.
- count: total number of elements.
- reverse: to reverse the order of items in a list.

Here is a Python program that utilizes the above five functions:

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

num.append(7)

num.append(num[0])

num.append(num[-2])

print(num)

num=[1,2,3,4,5]

num.clear()

print(num)

num=[1,2,3,4,5,4,4]

print(num.count(4))

num=[1,2,3,4,5]

num.reverse()

print(num)
```

You can copy a list into a file. The following example creates a file for writing and by use of a for loop, it copies all elements of a list onto a file.

```
num=[1,-2,3,4,-5]

myfile=open("data.txt","w")

for n in num:

    myfile.write(str(n))

    myfile.write("\n")

myfile.close()
```

You can copy a file into a list. The following example reads a file and copies all lines of the file into a list.

```
myfile=open("data.txt","r")

newlist=myfile.readlines()

myfile.close()

print(newlist)
```

The following example checks to see if an item is an element of a given list.

```
mylist=["West","East","North","South"]

if "West" in  mylist:

    print("Yes, West is in the list")

else:

    print("No it is not in the list”)
```

The following example inserts a new element into an existing list.

```
mylist=[“West","East","North","South"]

mylist.insert(2,”Ocean”)
```

As a result, the new list contains five elements and the third element is *Ocean*.

The following example appends an item as the last element to a given list.

```
mylist=[“West","East","North","South"]

mylist.append(”Ocean”)
```

As a result, the new list contains five elements and the fifth element is *Ocean*.

The following example appends an entire list to the end of another list.

```
mylist=["West","East","North","South"]

newlist=["Ocean","Sea","Creek"]

mylist.extend(newlist)
```

8.1. Practice Questions

1. Write a Python program to replace every element of a list with its square.

2. Write a Python program to find the smallest and the second smallest elements in a list.

3. Write a Python program to remove the first and last elements of a list and redisplay the new list.

4. Write a Python program to calculate the average of the elements in a list.

5. Write a Python program to replace every even element of a list with “*”.

Go to 8.1 Practice Question Solutions.

9. Arrays

As mentioned in chapter eight, Numpy is a library in Python that can generate lists, matrices, linear algebra, and so on. Using Numpy, you can generate arrays. Arrays are used to keep the related data at the same location in the memory. And since arrays are stored at one continuous location in memory, it is quicker to access elements of an array than elements of a list. Therefore by using arrays, programmers store multiple values as one variable. And by using each element's index we can quickly retrieve a particular element of an array. Consider the following Python program:

```
import numpy

MyArray=numpy.array([-1,0,1])

print(MyArray)
```

In the above Python program, MyArray is the name of the array that contains three elements: -1,0,1. The following Python program generates a two-dimensional array. An array can have any number of dimensions.

```
import numpy

B=numpy.array([[-1,0,1],[-2,0,2]])

print(B)
```

To access each element of an array, you need to refer to its index, and as with lists, you begin from index zero. In the above program, B[0][0]=-1, is the element on the first row and the first column.

You can select more than one element of an array at a time. A[m:n] means several elements from array A, from index m to (not including) index n.

9.1 Practice Questions

1. Write a Python program to generate an array and reverse it, without changing the order of elements in the original array.

2. Write a Python program to generate an array as input and reverse it. The original array will be reversed.

3. Write a Python program to find the common elements in two arrays.

4. Write a Python program to find repeated items in an array.

5. Write a Python program to find the second largest value in an array.

6. Write a Python program to check if an array contains a value.

Go to 9.1 Practice Question Solutions.

10. Plotting Graphs

Matplotlib is a library in Python that you can utilize for the visualization of a graph. However, it is not part of the standard Python library, and you need to install it. To do so you need to open a terminal and on a Mac type:

```
sudo pip3 install matplotlib
```

and if you are using a Windows system, then enter the following command:

```
pip install matplotlib
```

Sudo is a command in Unix-based operating systems that grants administrator privileges to a user.

PIP is a package manager for all the libraries that you may need to import into your Python programs.

Once matplotlib is installed, you can import it into your Python programs.

Plot(x,y) is a built-in function to draw lines in a graph, where *x* is the horizontal axis and y is the vertical axis. *The plot ()* function creates a line that connects a series of data points with straight lines.

Let us consider the following Python program:

```
import matplotlib.pyplot as plt

x=(1,2,3,4,5,6,7,8)

y=(0,50,100,50,-50,0,100,-50)

plt.plot(x,y,marker='s',linestyle=":",color="g",linewidth=2.0)

plt.show()
```

Here, import *matplotlib.pyplot* as *plt* means that you can refer to that library as plt. When you use *plot()*, markers are used to emphasize each point on a graph with a specified marker, line style indicates the style of the line, color indicates the color of the line, and line width indicates the width of the line, all of which are line properties that you can use as built-in functions. Finally, to display a graph on the screen, you need to use another built-in function *show()*, as plot() builds the graph in memory but does not display it by default. Figure 10-1 is the output of the above program.

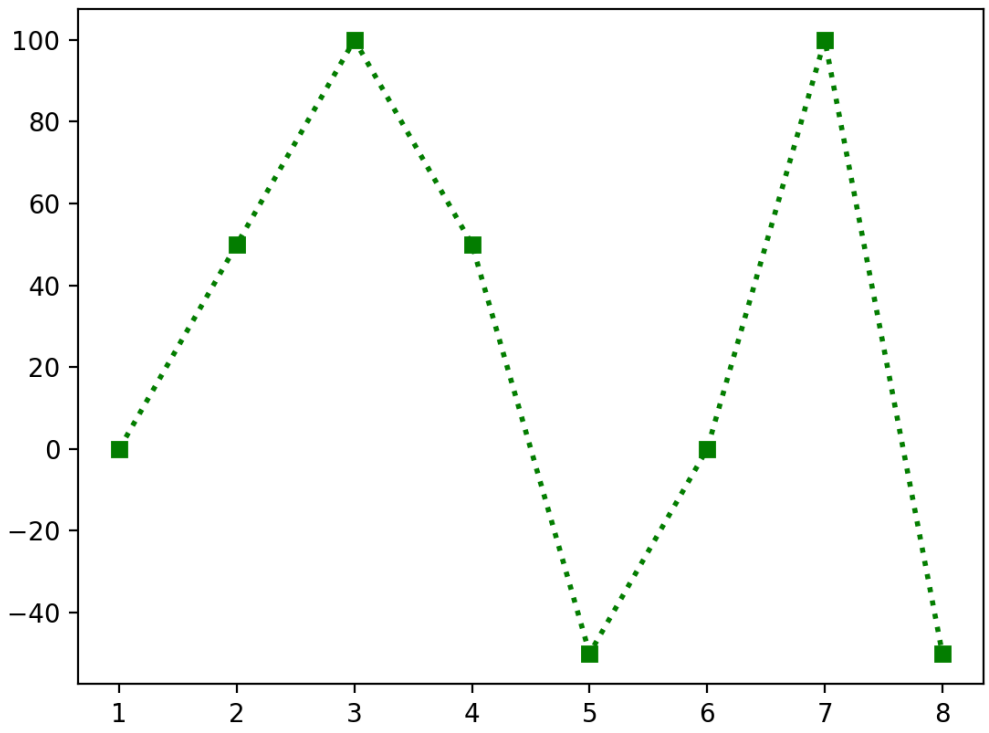

Figure 10-1

Table 10-1 indicates color choices for colors in any graph.

Color	Meaning
B	Blue
G	Green
R	Red
C	Cyan
M	Magenta
Y	Yellow
K	Black
W	White

Table 10-1

You can use the following built-in functions to draw graphs:

- bar(): to draw bar graphs.
- grid(): to add grid lines to the plot.
- pie(): to draw pie charts.
- subplot(m,n,p): to show multiple graphs, in *m* rows, *n* columns and the current plot is the plot number *p*.

- title(): to add a title to a graph.
- xlabel() and ylable(): to add a title to the x and y axes.
- xlim() and ylim(): to change the lower and upper limits of numbers on the x and y axes.

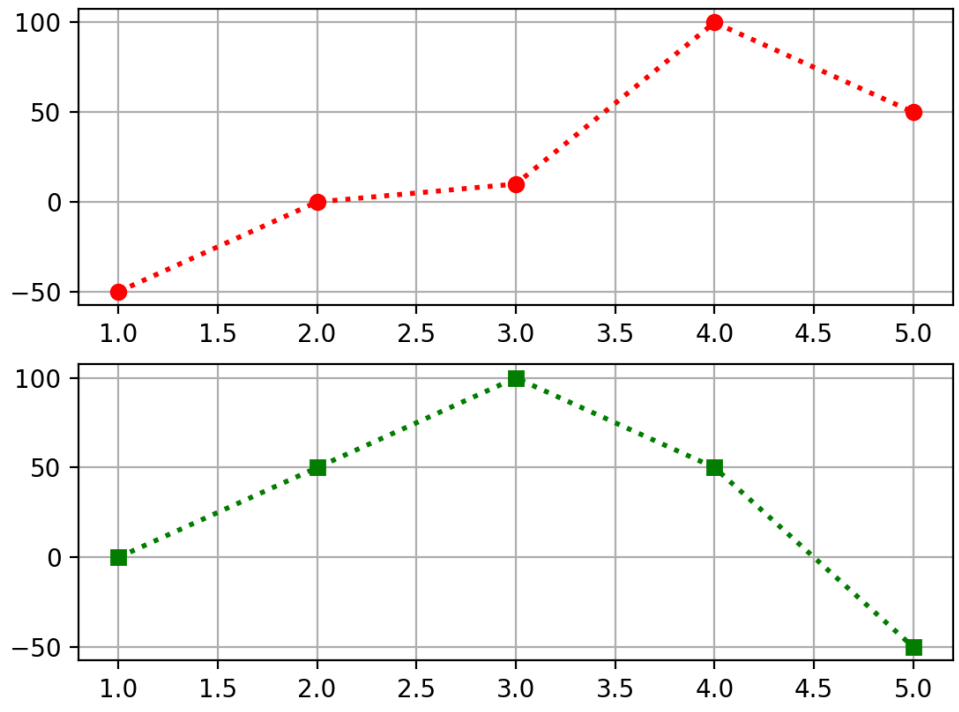

Figure 10-2

The output of the following Python program is depicted in Figure 10-2, here we have used a subplot.

```
import matplotlib.pyplot as plt

x=(1,2,3,4,5)

y=(0,50,100,50,-50)

plt.subplot(2,1,2)

plt.plot(x,y,marker='s',linestyle=":",color="g",linewidth=2.0)

plt.grid()

x=(1,2,3,4,5)

y=(-50,0,10,100,50)

plt.subplot(2,1,1)

plt.plot(x,y,marker='o',linestyle=":",color="r",linewidth=2.0)
```

```
plt.grid()

plt.show()
```

The output of the following Python program is depicted in Figure 10-3.

```
import matplotlib.pyplot as plt

x=(1,2,3,4,5,6)

y=(100,-10,10,50,20,0)

plt.bar(x,y,width=0.1,color='r')

plt.show()
```

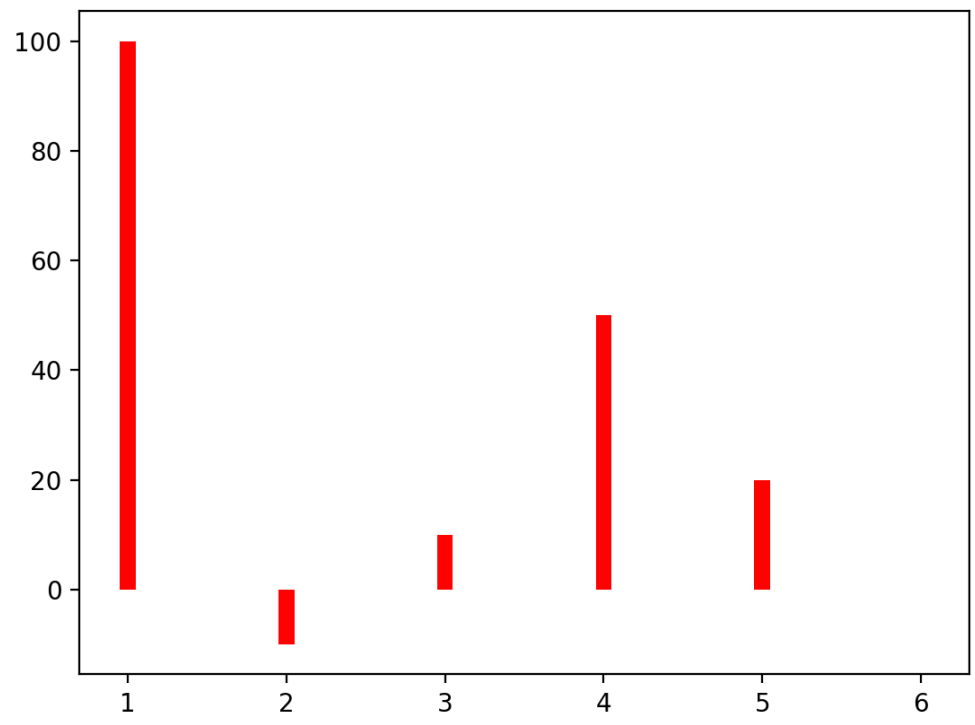

Figure 10-3

NumPy, which stands for Numerical Python is also another Python library, that you can use for working with linear algebra, matrices, and so on.

Assume that you need to draw a graph for Sine(x). Since the Sine function is part of Numerical Python, you need to import it into your program. The output of the following Python program is depicted in Figure 10-4.

```
import matplotlib.pyplot as plt

import numpy as np

x=np.arange(0,10,0.05)

y=np.sin(x)
```

```
plt.plot(x, y)

plt.show()
```

The meaning of import *numpy* as np is that you can refer to that library as np. In this Python program, we used *arange(n,m,p)*, which means *n* is the start of the

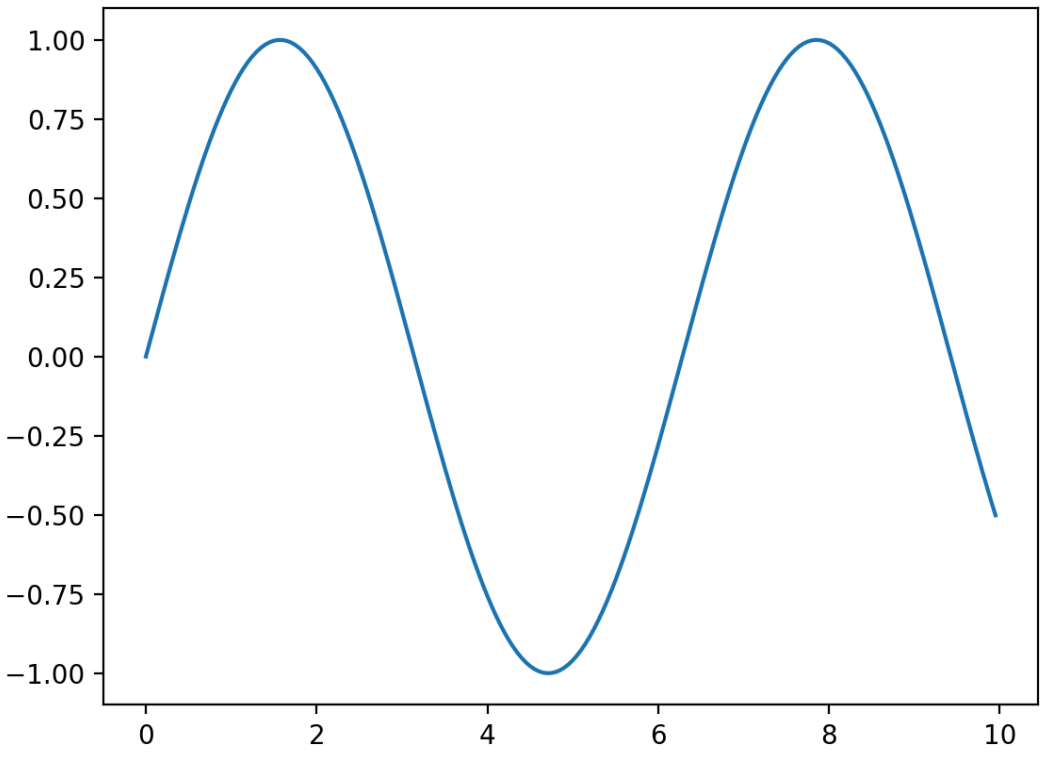

Figure 10-4

interval, *m* is the end of the interval and p is the spacing between the values.

The output of the following Python program is depicted in Figure 10-5. Sine and Cosine functions are part of Numerical Python.

```
import matplotlib.pyplot as plt

import numpy as np

z=np.arange(0,360,10)

x=np.cos(z)

y=np.sin(z)

plt.plot(x,y)

plt.show()
```

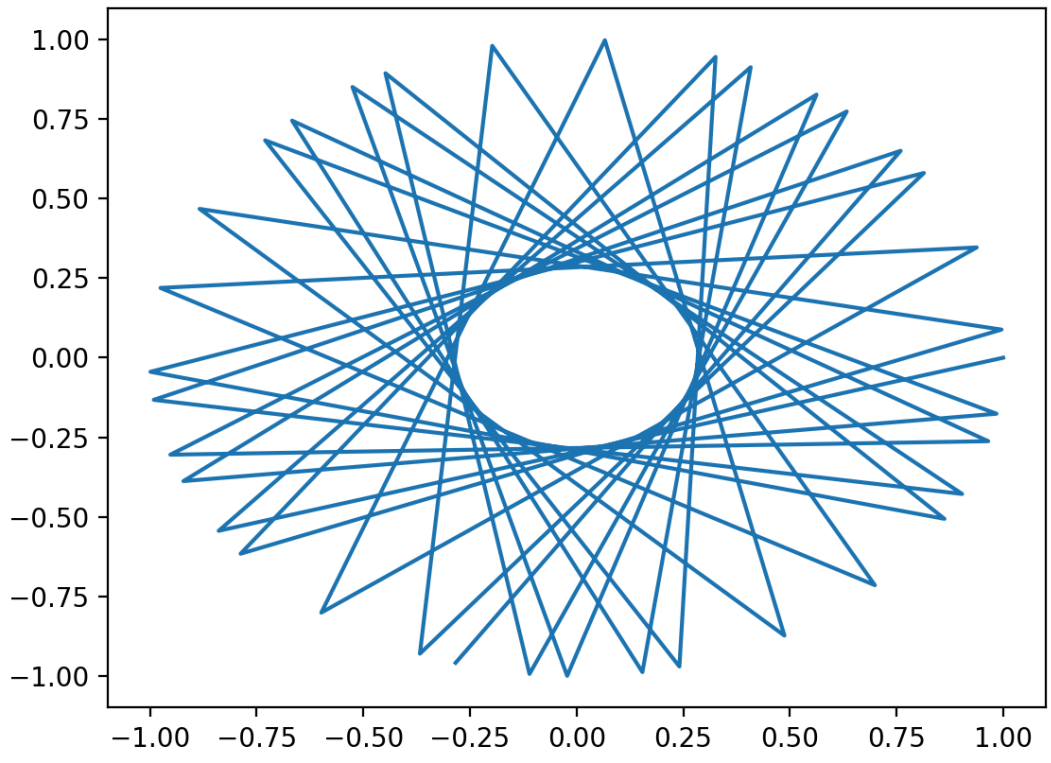

Figure 10-5

The output of the following Python program is a pie chart, which is depicted in Figure 10-6.

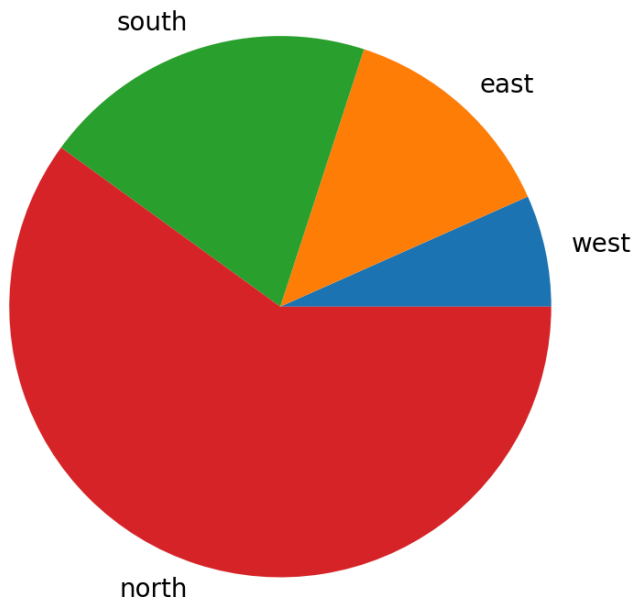

Figure 10-6

```
import matplotlib.pyplot as plt

x=(10,20,30,90)

lab=("west","east","south","north")
```

```
plt.pie(x,labels=lab)

plt.show()
```

11. Object Oriented Programming

Python is an object-oriented programming language. This model of programming is based on objects. An object is an entity that has predefined attributes and behavior. The Python program makes objects interact with other objects. Class is the template for an object, and an object is an instance of a class.

A smartphone is an object. Examples of attributes are traits such as screen size, camera, weight, or number of external ports. Examples of behavior are taking a picture, recording a movie, or connecting to Wi-Fi.

Class is a blueprint of the object. Every cell phone that is released by a particular manufacturer will come with certain predefined attributes and behavior, such as the examples above. The class provides all the common elements.

The statement

```
class Vehicle:
```

creates one class, which is named Vehicle. The statement

```
v1=Vehicle()
```

creates one instance of the class vehicle, which is called v1.

In programming, we refer to behavior as methods or functions. The following Python program consists of a class car. Every car has a model and a year associated with it. C1 and c2 are two objects and are instances of the class car.

```
Class Car:

    def __init__(self,model,year):

        self.model=model

        self.year=year

c1=Car("TESLA",2024)

c2=Car("FERRARI",2023)

print(c1.model,"***",c1.year)

print(c2.model,"****",c2.year)
```

11.1 Constructor

A constructor is a method for initializing objects of a class. All classes have a constructor, which is executed when the class is being initiated. In Python, a constructor's name is always init and it must have a prefix and suffix of double underscores. Since it is a method, you must include def before its name. As with any other method, a constructor may or may not accept any arguments. A default constructor does not accept any arguments. A parametrized constructor accepts arguments, and in this case, you can pass data during object creation, which is used to initialize the instance members.

Objects can also have methods. The following Python program depicts one class called vehicle, v1 is one instance of the class, and Driver is one method that belongs to the object. Every object has three attributes, model, year, and number of drivers.

```
class Vehicle:

    def __init__(self,Model,year,NoD):

        self.model=model

        self.year=year

        self.NoD=NoD

    def Driver(self):

        print("Number of Driver" + self.NoD)

v1=Vehicle("TESLA",2024,0)

v1.driver()
```

Self represents the instance of the class; it is a reference to the current instance of the class. Use self to create an attribute that belongs to an object that self is referencing. This is called an instance of an object.

An object cannot be created without a constructor, if you do not declare a constructor in your program, Python generates one by default.

11.2 Inheritance

Inheritance allows one class to inherit attributes and methods from another. Parent class or base class is the class being inherited from, and child class is the

class that is inherited from a parent class. The child class is a derived class from the base class; the child class creates an instance of the class. Child class may override the methods of its parent class. Overriding means that a child class has an implementation of a method that is already defined in the parent class.

In the flowing Python program shown below, the class vehicle is the parent class, and the car is the child class. Therefore, the class car has the same properties and methods as the vehicle class.

```
class vehicle:

    def __init__(self,model,year):

        self.model=model

        self.year=year

    def printcar(self):

        print(self.model,self.year)

class car(vehicle):

    def __init__(self,model,year):

v1=vehicle("Toyota",2000)

v1.printcar()

v1=car("Honda",2020)

v1.printcar()
```

When you include the following in the child's class:

```
def __init__(self,model,year):
```

then it no longer inherits this method and instead uses its own, which is called overriding.

11.3 Polymorphism

Class polymorphism is when multiple classes have the same method name. For example, a class shape() by itself does not have any definition of area. However, if you consider class triangle, circle, or hexagon, they all have the concept of area in common, and you could have a separate method named area() for each

one of them, which calculates the area of that particular shape. This is depicted in the following Python program.

```
class Shape:

    def area(self):

        print("Needs more specifications")

class Circle(Shape):

    def area(self):

        print("3.14*r*r")

class Triangle(Shape):

    def area(self):

        print("h*b/2")

s1=Shape()

c1=Circle()

t1=Triangle()

s1.area()

c1.area()

t1.area()
```

12. Using Python Packages

Python has several libraries that make analyzing data very easy. It also contains highly powerful machine-learning libraries. The libraries that we will utilize are for data analysis, visualization, machine learning, and data mining.

To get ready to work in a Python environment, you need to install some packages. A package is an archived file that we download from the internet and install on your computer. To be able to run the programs that are provided in this book, you need to be sure that you are using the latest version of each released package.

At the time of writing this book, the latest edition of pip is 23.3.1, and the latest edition of numpy is 1.26.3. Follow these directions to install these packages:

To install a given package, you need to run this command:

```
pip install name of the package.
```

In a Python shell type-in the following commands:

```
Pip install -U NumPy

Pip install --upgrade pip

Pip install network

Pip install matplotlib

Pip install fpgrowth_py
```

- Pip is a management system written in Python, which you can use to install other packages. Pip connects to an online repository of packages.
- Networkx is used for the creation and studying the structure of networks and graphs. It is free software, which means users can run the software, change it, and distribute it. The term “free” means users have the freedom to use the software in any way that they choose, but the source of the software is not openly available.
- Matplotlib is a plotting library used for static and interactive visualizations in Python. It is designed to be like MATLAB, can use Python, is available for free, and is an open-source software. Open-source software is freely

available and can be modified and redistributed, by encouraging open collaborations.

- Numpy is a library for Python programming, where the type and size of data, arrays, and mathematical functions are supported. Numpy is the fundamental package for scientific computing with Python and is an open-source software.

- fpgrowth_py is a Python package used for frequent pattern mining and associations in data sets.

- "-U" or "—upgrade" means upgrade the current edition to the latest version.

Now that you have installed the above packages, you can import them into your Python programs. If the Python interpreter generates errors, it means one or more packages have not been installed correctly. All Python packages are available at the Python Package index.

13. Python and Graph Theory

A graph is a structure, which contains vertices (nodes) and edges. Graphs can be used to represent acquaintances between people such as friendship relationships on Facebook, followers on X (Twitter), and so on. In each graph, each edge connects two vertices, and each of the vertices is called an endpoint. If there is an edge between two vertices, those two vertices are called adjacent nodes.

A simple graph is a graph where no two edges connect the same pair of vertices. A railway system between two cities represents a simple graph. A graph with multiple edges connecting the same vertices is called a multigraph. For example, imagine a graph where vertices are cities in a country and edges are direct flights between two cities.

A graph may be directed or undirected; if it is needed to assign a direction to an edge between two vertices, then the graph is directed. For example, being a follower of a person on an online social network makes a graph a directed graph. In some online social networks, certain people can influence others. Therefore, a directed graph can be used to represent when one node influences another node; there is a direction assigned to the edge between the two vertices. A directed multigraph can be utilized to represent text messages sent and received between two phone numbers.

The degree of a vertex in an undirected graph is the total number of edges that it touches. In a directed graph the total number of edges that are incoming to a node is called its in-degree, and the number of edges that are outgoing from a node is called its out-degree.

A path is a sequence of edges that begin at one node of a graph and traverses from one node to another node along the edges of the graph. For example, two people on an online social network are linked where there is a path between the two. This path could represent influence, such as being a follower of a person or retweeting messages in a network.

In a graph, the shortest path is a path between two nodes such that it contains a minimized number of edges. Also, the closeness centrality of a node is the average length of the shortest path between the node and all other nodes in that graph. Therefore, a node’s centrality indicates the number of shortest paths that it is part of. The higher the number of centralities, the more important the node is.

13.1 Networkx

Networkx is a Python package for the creation and analysis of networks. A network could be an online social network, or a graph that represents friendships, sending the receiving messages such as tweets, and so on. You can use Networkx over large data sets; you can use it to design a new algorithm or build network models. To create a graph with no edges and no nodes, you can execute the following Python program:

```
import networkx as nx

G = nx.Graph()
```

Using the kite_graph data set (a data set is any collection of data), which is part of the Networkx package, the output of the following Python program is depicted in Figure 13-1, where we have 10 vertices and 18 edges. In Networkx, you can use graph objects to generate graphs. A directed graph is specified by the class DiGraph() and a multigraph is specified by the class MultiGraph().

```
import networkx as nx

import matplotlib.pyplot as plt

G = nx.krackhardt_kite_graph()

nx.draw_networkx(G)

plt.show()
```

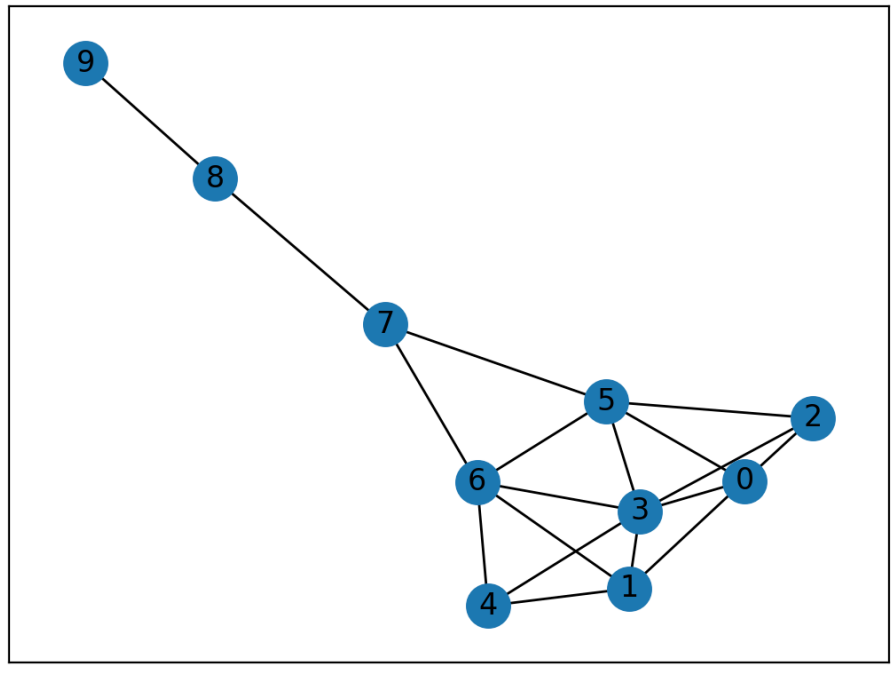

Figure 13-1

In the above program, we can examine the degree of each node; for example, print(G.degree(3)) will display five as it is the degree of node number three. Or print(G.adj[3]) will generate the following output, which is the list of all of the adjacent nodes to node number three.

```
{0: {}, 1: {}, 2: {}, 4: {}, 5: {}, 6: {}}
```

The output of the following Python program is depicted in Figure 13-2. Since G is a directed graph, add.edge(m,n) adds an edge from node m to node n.

```
import networkx as nx

import matplotlib.pyplot as plt

G = nx.DiGraph()

G.add_edge(1,2)

G.add_edge(1,3)

G.add_edge(2,1)

G.add_edge(2,4)

nx.draw_networkx(G)

plt.show()
```

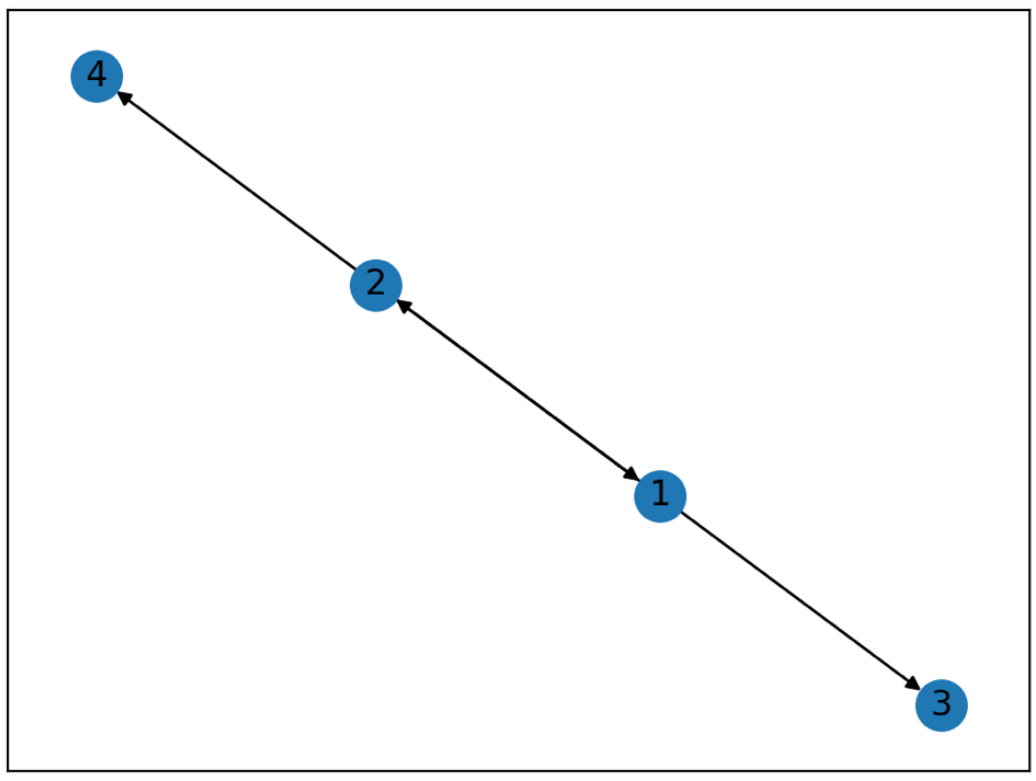

Figure 13-2

The following Python program prints the node centrality of each node in a kite graph, as given in Figure 13-1.

```
import networkx as nx

import matplotlib.pyplot as plt

G = nx.krackhardt_kite_graph()

print(nx.betweenness_centrality(G))
```

And as you can see the node with the highest centrality is node number seven.

```
{0: 0.02314814814814143,
1: 0.02314814814814143,
2: 0.0,
3: 0.10185185185185183,
4: 0.0,
5: 0.2314814814814148,
6: 0.2314814814814148,
7: 0.38888888888888884,
8: 0.222222222222222,
9: 0.0}
```

The following Python program displays the degree centrality of each node in a kite graph, given in Figure 13-1.

```
import networkx as nx

import matplotlib.pyplot as plt

G = nx.krackhardt_kite_graph()

print(nx.degree_centrality(G))
```

And as you can see the node with the highest node centrality is node number three.

```
{0: 0.4444444444444444,
1: 0.4444444444444444,
2: 0.3333333333333333,
3: 0.6666666666666666,
4: 0.3333333333333333,
5: 0.5555555555555556,
6: 0.5555555555555556,
7: 0.3333333333333333,
8: 0.222222222222222,
9: 0.1111111111111111}
```

```
G.number_of_nodes()
```

and

```
G.number_of_edges()
```

respectively, display the total number of nodes and the total number of edges in a graph.

```
G.remove_edge(m,n)
```

removes the edge between node m and node n.

13.2 Matplotlib

Many utilities that you can use in Matplotlib are imported under the plt alias. Use the following command to use the package as plt in your Python programs:

```
import matplotlib.pyplot as plt.
```

- Plot() is used to draw a line between two points.
- Use marker() to mark each point on a line with a specified marker and use ms to set the size of the marker.

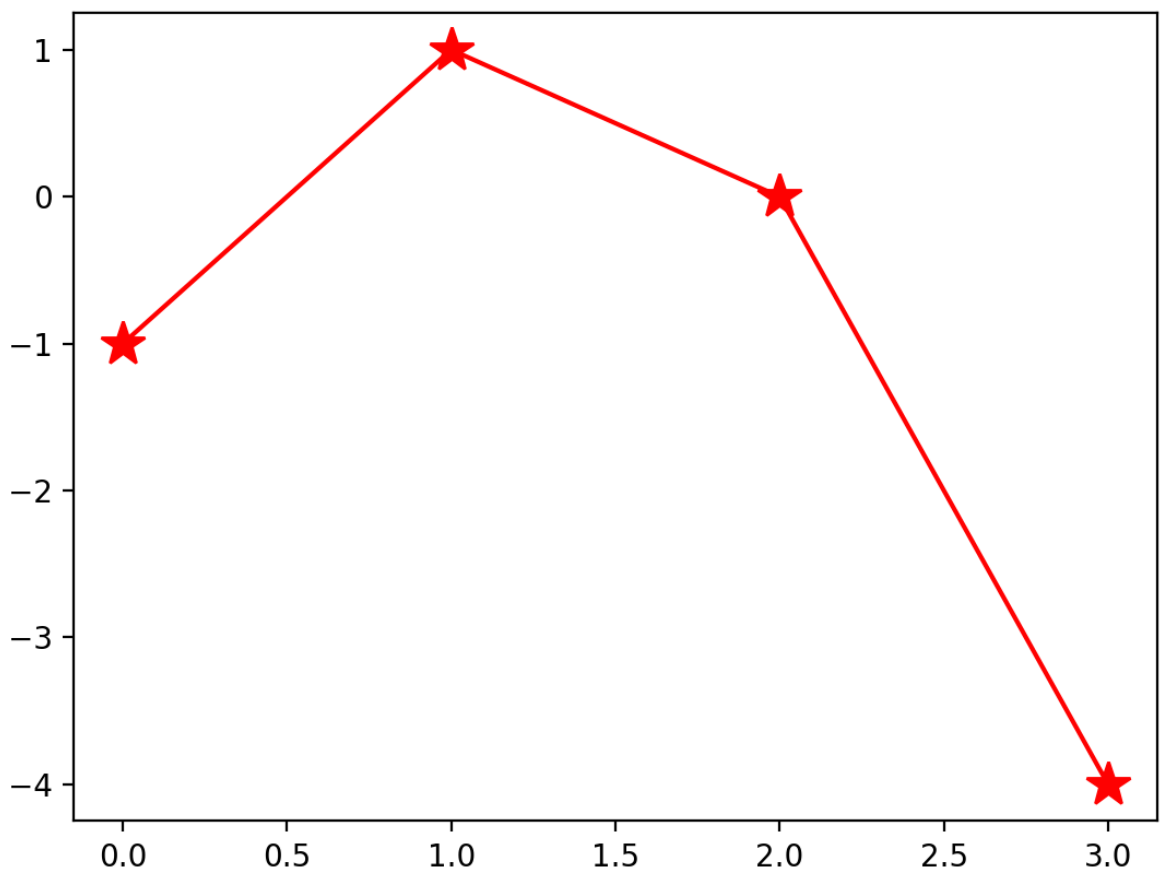

Figure 13-3

The following program draws a line graph, as depicted in Figure 13-3:

```
import matplotlib.pyplot as plt

import numpy as np

y=np.array([-1,1,-0,-4])

plt.plot(y,marker='*',c="red" ,ms=15)

plt.show()
```

14. Python and Machine Learning

When you improve your performance based on your observations of past events, learning occurs. Learning is needed because you cannot anticipate all possible situations that can occur in the future.

Machine Learning is the study of statistical algorithms that can learn from data; It is the process of analyzing data and predicting the possible outcomes. Perhaps classification is the most basic form of analyzing data. An example of this is an incoming email being marked as spam or not spam, and this marking classifies emails into two classes.

There are three different types of data: numerical, categorical, and ordinal. Numerical data can be counted, such as the total number of emails that one receives, or it can be measured data, such as the size of an attachment to an email. Categorical data are not measured in numbers; for example, a person's gender cannot be measured numerically. Ordinal numbers are non-numerical but with an implied order, such as a rating that you provide for an item listed on Amazon. There are different techniques that you may apply to data based on its type.

There are three types of learning: supervised learning, unsupervised learning, and reinforcement learning.

- In supervised learning through a computer program, a computer learns from the inputs to generate the desired output. A computer observes pairs of input-outputs and the correlation between them. The input could be an image, and the output be a traffic light. Therefore, when someone acts as a teacher, it is thus called supervised learning.

- In unsupervised learning, a computer must find patterns within the data. There is no learning process; a spam-detecting system may use unsupervised learning to classify different emails into several possible spam categories, each with a different probability of being spam. Then, each category uses supervised learning to learn how many of those emails were a spam message.

- In reinforcement learning, an agent must interact with the environment dynamically; interactions come in the form of rewards and punishments. The goal is to maximize the rewards.

14.1 Supervised Learning

Classification is one type of classification, where a program is trained on a dataset to predict the category of new data. Classification is either binary or multi-class. For example, consider if an incoming message is either a text or a picture. An incoming email may be spam or not. We can train the email system that if an email contains certain words, then with a high probability, it is a spam message. We can use linear classifiers to create a linear decision; this classification is based on a linear combination of characteristics of the data.

14.1.2 Regression

Prediction is like classification; you predict a numerical value of a variable. Data mining is examining data where the classification will occur in the future. One approach for supervised learning is regression. Regression is predicting the behavior of one variable based on the behavior of another variable. The variable that is being predicted is the response variable. For example, if an influencer in an online social network purchases an item or gives a high approval to an object, several of their followers will purchase the same object. What makes a person an influencer depends on many factors, such as fame, degree centrality, betweenness, and so on. If you have a business, then you would like to recognize these influencers as they can influence others to buy your product, which in return could boost your business. Therefore, you need to be able to predict the future behavior of people many times and make business decisions accordingly, which means you need to apply regression.

14.1.3 Linear Functions

In the equation, $y = a*x + b$, you can calculate the value of y if the value of x is given. The relationship between x and y is linear. Regardless of the values of a and b, the graph representing this equation is always a straight line. The variable a is called slope as it defines the slope of the line and variable b is called the intercept. The intercept is the value where the plotted line intersects the y-axis. Slope and intercept are key values of linear regression.

The output of the following program as depicted in Figure 14-1 is a linear regression between two variables, and the straight line is used to predict the future values of the two variables.

```
import matplotlib.pyplot as plt

from scipy import stats

x=[1,2,3,-2,-5,20,-10,100,2]

y=[100,-2,20,-10,200,5,20,100,-11]

slope,intercept,r,p,std_err=stats.linregress(x, y)

def fun(x):

return (x*slope-0.5)

pred=list(map(fun,x))

plt.scatter(x,y,c="red")

plt.plot(x,pred)

plt.show()
```

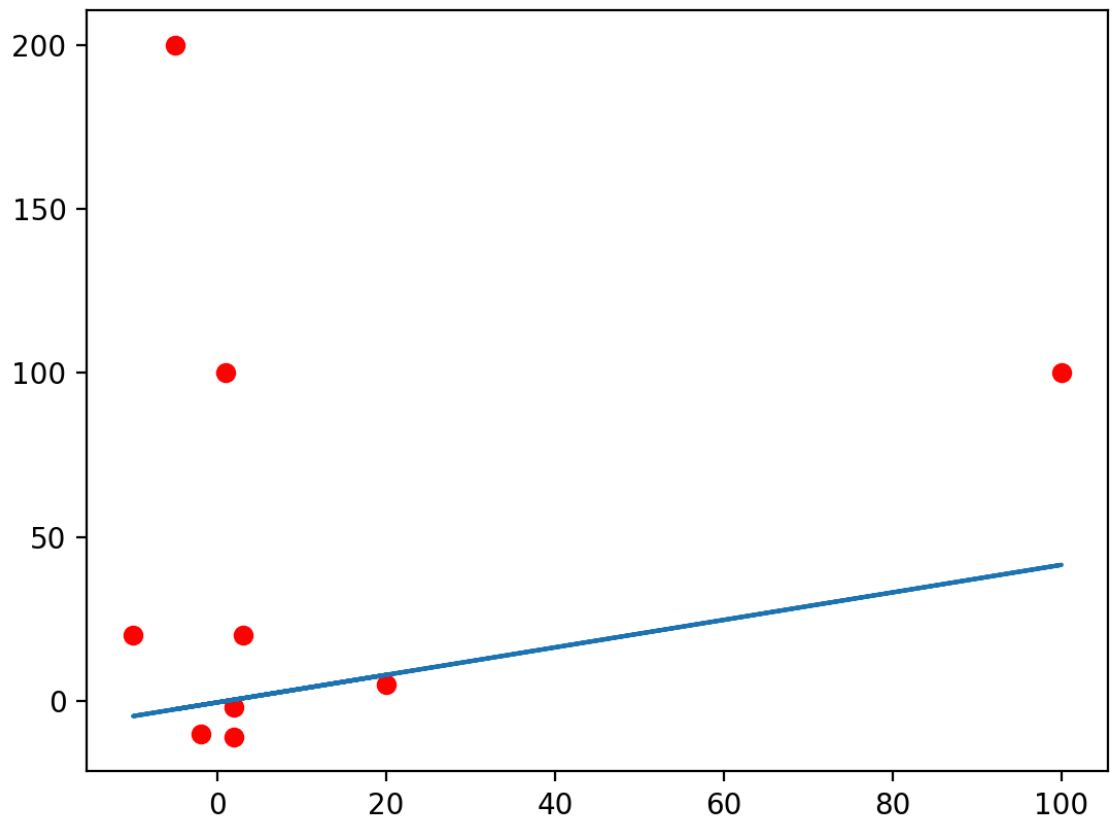

Figure 14-1

14.1.4 Polynomial Functions

The following Python program depicts a polynomial regression as depicted in Figure 14-2. Linear regression works on continuous data and when data are correlated. However, if the correlation is not linear, we can use polynomial regression that instead of a best-fit line will have a best-fit polynomial line. Polynomials fit a wide range of curvature.

```
import numpy

import matplotlib.pyplot as plt

x=[1,2,3,4,5,6,7,8,9,10]

y=[10,-2,3,4,5,-200,11,90,0,200]

pred=numpy.poly1d(numpy.polyfit(x,y,5))

line=numpy.linspace(1,10,100)

plt.scatter(x,y)

plt.plot(line,pred(line))

plt.show()
```

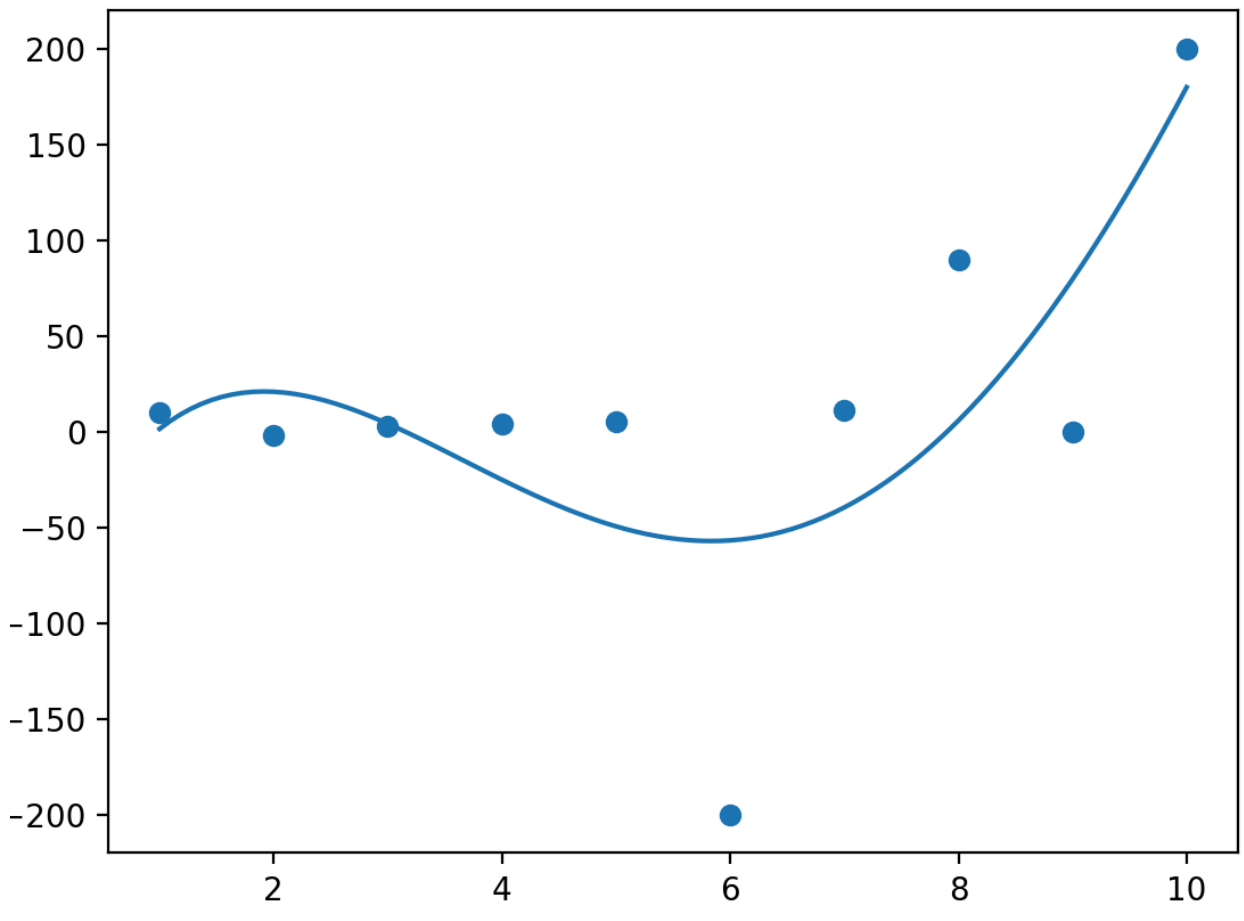

Figure 14-2

A linear regression model can determine the relationship between a variable and the response variable, where other variables are fixed. In polynomial regression, a nonlinear relationship between variables is modeled.

14.2. Unsupervised Learning

In unsupervised learning, input data variables are given with no corresponding output variables. Here the goal is to find patterns in the data. The machine must group information according to patterns and similarities but without any prior information.

- Clustering is one approach to unsupervised learning. This is where data is divided into several groups, and data with similarities will be placed in the same cluster. For example, emails that contain a set of specific words will be classified as spam. First, each data point is assigned to one cluster, then we calculate the center point of each cluster, reassign each data to the cluster with the closest centroid, and repeat until no other cluster assignment is possible.

The following Python program puts the data into two clusters, Figure 14–3 depicts the data:

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import AgglomerativeClustering

x=[1,-1,2,-2,3,-3,4,-4,5,-5]

y=[10,20,30,-10,-20,30,50,100,90,1]

clust=list(zip(x,y))

hierarchical_cluster=AgglomerativeClustering(n_clusters=2)

labels=hierarchical_cluster.fit_predict(clust)

plt.scatter(x,y,c=labels)

plt.show()
```

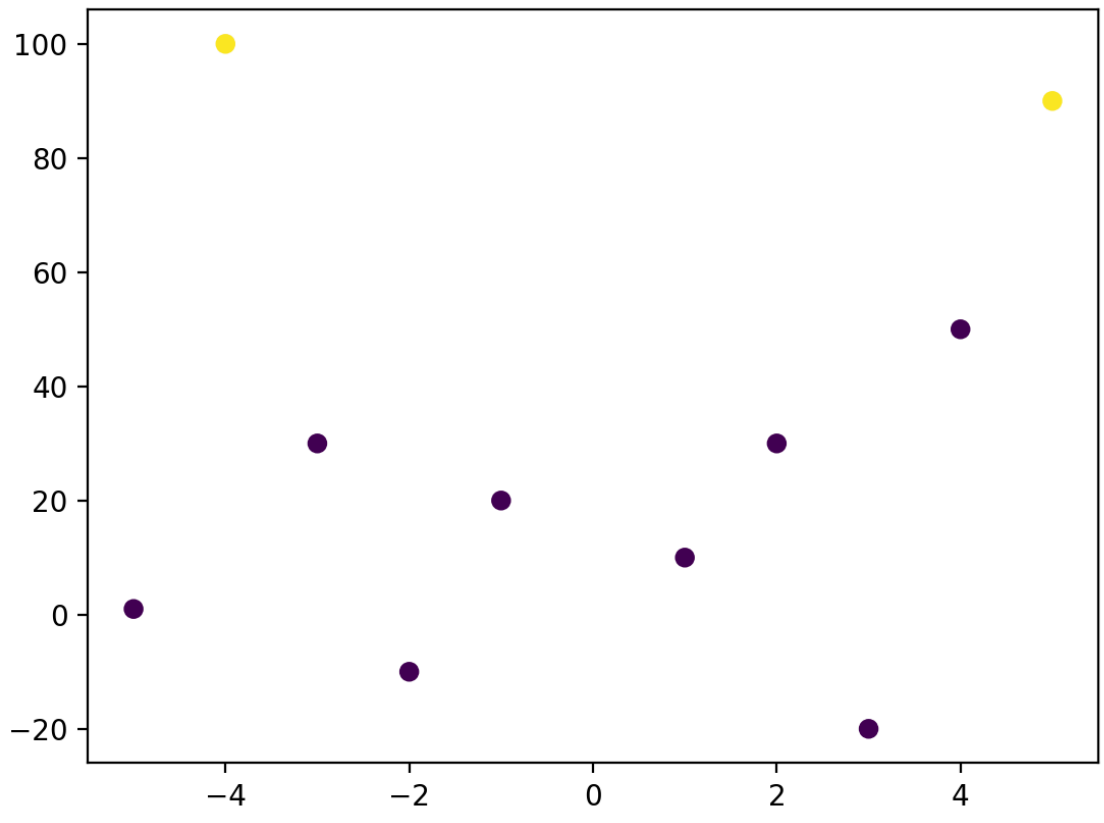

Figure 14-3

And the following Python program puts the same data into four clusters. Figure 14-4 depicts the results.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
x=[1,-1,2,-2,3,-3,4,-4,5,-5]
y=[10,20,30,-10,-20,30,50,100,90,1]
clust=list(zip(x,y))
hierarchical_cluster=AgglomerativeClustering(n_clusters=2)
labels=hierarchical_cluster.fit_predict(clust)
plt.scatter(x,y,c=labels)
plt.show()
```

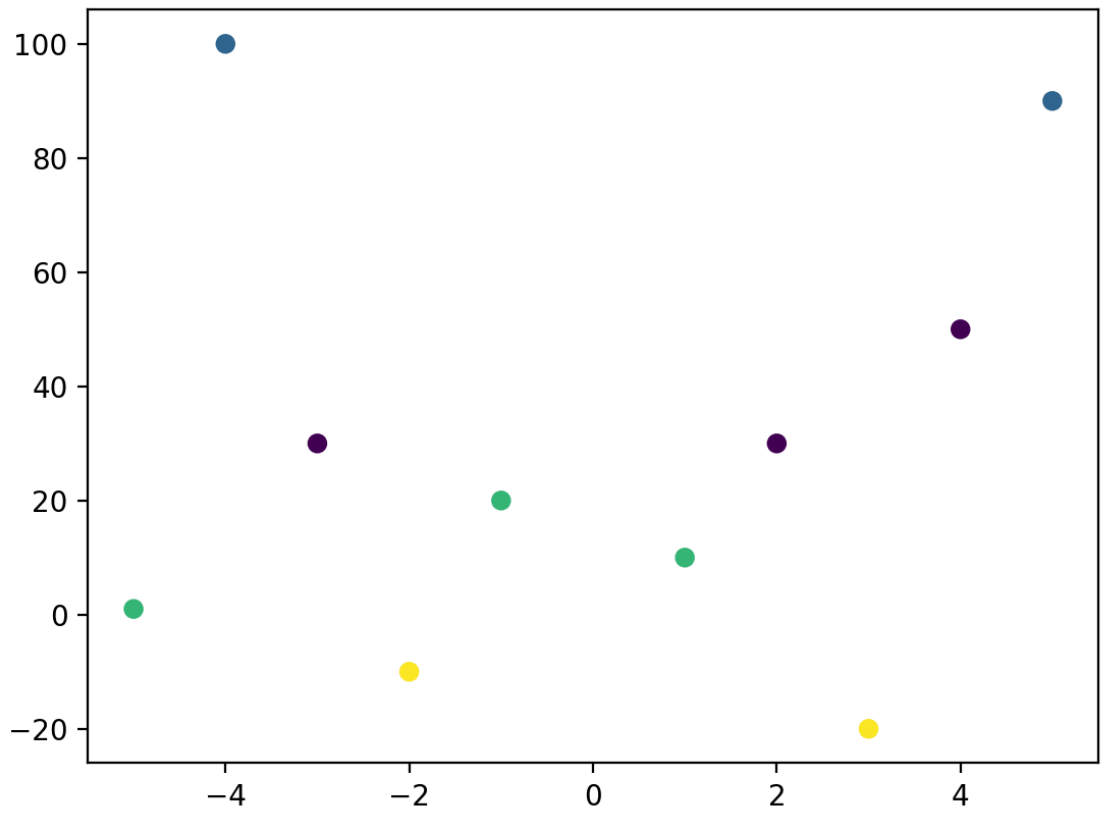

Figure 14-4

- Association is another approach to unsupervised learning, where you find rules that express your data. For example, people who are followers of person x will be followers of person y, based on the list of people whom they have followed in the past. Or, based on your past tweets, your future tweets will contain certain words.

The following Python program uses a frequent pattern growth algorithm for finding associations, which is a data-mining technique. Here it finds the frequent items in a dataset, where the minimum support is 0.8, and the minimum confidence for generating association rule is 0.5.

```
from fpgrowth_py import fpgrowth

car= [['Ferrari','Tesla','Ford'],['Ford','Tesla','Toyota']]

freqItemSet,rules=fpgrowth(car,minSupRatio=0.8, minConf=0.5)

print(freqItemSet)
```

The output is depicted below, as out of six data items, there are two “Ford” and two “Tesla”; here the minimum support is 0.8:

```
[{'Tesla'}, {'Ford'}]
[{'Tesla'}, {'Ford'}]
```

15. Python and Statistics

Statistics is the science of collecting, organizing, and analyzing data. In statistics, we frequently use these values:

- Mean: The average value of a data set is calculated by adding all numbers in the data set and then dividing it by the number of values in the set.
  - ✦ You can import Numpy and use numpy.mean()
- Median: The mid-point value is calculated by sorting all the data and picking the one in the middle.
  - ✦ Data should be sorted and then use numpy.median()
- Mode: The most common value is the number that occurs the highest number of times.
  - ✦ Use stats.mode(), which returns mode and how many times the mode appeared.
- Standard deviation: This term refers to how much the data deviates from the typical values.
  - ✦ Returns the standard deviation of data, where a low number is an indication that most numbers are close to the mean. Use numpy.std().
- Variance: This term refers to how the values are spread out, and it shows how data points differ from the mean.
  - ✦ Returns the variance of data. Use numpy.var().
- Quantiles: This term is used to divide a population according to a distribution.
  - ✦ Divides the data into intervals with equal probability. Use numpy.quantiles().

The following Python program utilizes all the above functions.

```
import numpy

from scipy import stats

A=[10,2,3,4,5,6,7,8,9,10]

x=numpy.mean(A)

print("mean",x)

x=numpy.median(A)

print("median",x)

x=stats.mode(A)

print("mode",x)

x=numpy.std(A)

print("std",x)

x=numpy.var(A)

print("var",x)

x=numpy.quantile(A,0.75)

print("75%",x)
```

And the output is as following:

```
mean 6.4
median 6.5
mode ModeResult(mode=10, count=2)
std 2.727636339397171
var 7.439999999999995
75% 8.75
```

15.1 Standardizing Data by Scaling

Sometimes, your data has different numerical values and may even be in different units of measurement, so it is hard to compare them. Imagine the number of retweets and the number of followers of someone; both are numerical values but in different scales. You can scale data, which means data is standardized by scaling data to fit a normal distribution with a mean of zero and a standard deviation of one. If a is the original value, m is the mean and s is the standard deviation, a new standard value of a can be calculated as $\frac{a-m}{s}$. In Python, you can use StandardScaler() to do so.

The following Python program converts an array of old numbers into a scaled array of new numbers.

```
from numpy import asarray

from sklearn.preprocessing import StandardScaler

old=asarray([[1,0.1],[2, 0.05],[10, 0.2],[5, 0.1],[-1, 0.01]])

print(old)

print("the new scaled data")

scaler=StandardScaler()

new=scaler.fit_transform(old)

print(new)
```

And the output is as the following:

```
[[ 1.    0.1 ]
 [ 2.    0.05]
 [10.    0.2 ]
 [ 5.    0.1 ]
 [-1.    0.01]]
the new scaled data
[[-0.62725005  0.12561486]
 [-0.36589586 -0.65947801]
 [ 1.72493763  1.69580059]
 [ 0.4181667   0.12561486]
 [-1.14995842 -1.2875523 ]]
```

15.2 T-Test

Many times, in statistics, we need to compare several samples to conclude the given data. A t-test is used to see if there is a significant difference between two groups of data. A t-test checks to see if the two groups of data are significantly different or if the difference is just due to random variation of data, and, hence, they are representing the same fact.

Assumption means that the data in each group follows the normal distribution and that the observation of data in the two groups is independent of each other.

In a t-test, we compare two means while considering a signification level, which is a predefined threshold used to decide whether to accept or reject results to be statistically significant. The degree of freedom gives us the number of independent variables used to calculate the estimate between two sample groups. In the following Python program, the significance level is 5%, and the degree of freedom is five, which depends on the size of the sample and how many are independent of each other.

```
alpha=0.05

stats.t.ppf(1-alpha,5)
```

Stats contains a large number of probability distributions, statistical tests, and estimations, which you can import into your Python program, with a command such as the following:

```
import scipy.stats as stats.from scipy import stats

import numpy as np

A=np.array([1,2,3,4])

B=np.array([-12,100,8])

ts,pv=stats.ttest_ind(A,B)

alpha=0.05

df=len(A)+len(B)

ct=stats.t.ppf(1-alpha/4,df)

print(ts,"****",pv,"*****",ct)

print("With T-value")
```

```
if (ts>ct):
    print("significant difference")
else:
    print("No significant difference”)
```

References

1. David Amos, Dan Bader, Joanna Jablonski, and Fletcher Heisler. *A practical introduction to Python 3*, ISBN: 9781775093336 (electronic).

2. Brian Heinold and John Prexy. *A Practical Introduction to Python Programming*, ISBN: 979-8848271577.

3. https://docs.python.org/3/ Python 3.12 documentation

4. https://www.python.org

5. https://en.wikipedia.org/wiki/Python_Software_Foundation

6. Tony Gaddis. *Starting out with Python*, 5th edition. Pearson, ISBN: 978-0-13-592903-2.

7. Douglas B. West. *Introduction to Graph Theory*, 2nd edition. Pearson, ISBN:978-0-13-1437371.

8. https://en.wikipedia.org/wiki/List_of_Unicode_characters

9. https://beginnersbook.com/2018/03/python-tutorial-learn-programming/

10. https://www.w3resource.com/python/python-tutorial.php

11. Wes McKinney. *Python for Data Analysis*, O'Reilly, ISBN: 978-1-449-31979-3.

12. Alberto Boschetti and Luca Massaron. *Python Data Science Essentials*, Packt Publishing Ltd., ISBN: 978-1-78953-786-4.

13. Samir Madhavan. *Mastering Python for Data Science,* Packt Publishing Ltd., ISBN: 978-1-78439-015-0.

14. Andreas C. Müller and Sarah Guido. *Introduction to Machine Learning with Python, A Guide for Data Scientists*, O'Reilly, ISBN: 978-1-449-36941-5.

15. Frank Kane. *Hands-On Data Science and Python Machine Learning,* Packt Publishing Ltd., ISBN: 978-1-78728-074-8.

16. Joel Grus.*Data Science from Scratch*, O'Reilly, ISBN:978-1-492-04113-9.

Appendix

Solutions for Practice Questions (2.4)

1. Write a Python program to get a number, then calculate its squared and cubed value.

```
number = int (input("Enter a number"))

print (number *number, "***", number*number*number)
```

2. Write a Python program to get a number such as n as input and then calculate n^n.

```
number= int (input("Enter a number"))

print (number ** number)
```

3. Write a Python program to get a number that contains three digits and then display each digit on a single line.

```
number=int (input("Enter a number"))

d1 = number // 100

d2 = (number // 10) % 10

d3 = (number % 100) % 10

print (d1,"***", d2,"***", d3)
```

4. Write a Python program to get a circle's radius and calculate its area.

```
R= float (input("Enter a radius"))

Area=3.14*R*R

print ("The area is ", Area)
```

5. Write a Python program to find the roots of a quadratic equation.

```
a = float (input ("Enter a"))

b = float (input ("Enter b"))

c = float (input ("Enter c"))

d = (b**2) - (4*a*c)

d1 = d**0.5

sol1 = ((-1*b) - d1)/(2*a)

sol2 = ((-1*b) + d1)/(2*a)

print (sol1,"***",sol2 )
```

6. Write a Python program to get a name as input and display it five times.

```
name= input ("Enter a name")

print (name*5)
```

7. Write a Python program to get today's date as three integers and display it as m/day/year.

```
month= int(input("Enter a month"))

day= int(input("Enter a day"))

year= int(input("Enter a year"))

print(month,'/', day, '/', year)
```

8. Write a Python program to simulate the roll of a die.

```
import random

dice=random.randint(1,6)

print (dice)
```

9. Write a Python program to simulate the roll of a pair of dice.

```
import random

dice1=random.randint(1,6)

dice2=random.randint(1,6)

print (dice1,"****",dice2)
```

10. Write a Python program to generate an even random number between one and one hundred.

```
import random

m=random.randint(1,50)

m=2*m

print (m)
```

Return to Section 2.4

Solutions for Practice Questions (3.2)

1. Write a Python program to get a number and verify if it is divisible by five.

```
number= int(input("Enter a number"))

if (number %5==0):

    print("It is divisible by 5")

else:

    print("It is not divisible by 5")
```

2. Write a Python program to get a number and verify if it is divisible by five or by three.

```
number= int(input( "Enter a number"))

if (number %5==0 or number %3==0):

    print ("It is divisible either by 3 or by 3")

else:

    print ("It is not divisible by 3 and 5")
```

3. Write a Python program to get a year and verify whether it is a leap or a common year. A year is a leap year if it is divisible by four, except those years divisible by a hundred are not leap years unless they are also divisible by four hundred.

```
year= int (input ( "Enter a year" ))

if ((year%4==0 and year%100!=0)or(year %400==0)):

    print ("It is a leap year")

else:

    print ("It is a common year")
```

4. Write a Python program to get a date and check if it is a magic date. A date is magical when you multiply month by day and get the year. For example, February 12th of 2024 is a magical date as 2*12=24.

```
month= int(input("Enter a month"))

day= int(input("Enter a day"))

year= int(input("The last two digits of a year?”))

if (month*day ==year):

    print ("It is a magic date")

else:

    print ("It is NOT a magic date")
```

5. Write a Python program to get a number between one and five and convert it to a Roman numeral.

```
number= int(input ("Enter a number"))

if (number==1):

    print ("I")

elif (number==2):

    print ("II")

elif (number==3):

    print ("III")

elif (number==4):

    print ("IV")

elif (number==5):

    print ("V")

else:

    print ("Invalid input")
```

6. Write a Python program to check if a given input is odd or even without using remainder (%).

```
number=int (input("Enter a number"))

if ((number//2)*2==number):

    print ("Even")

else:

    print ("Odd")
```

7. Write a Python program to get a word that contains three symbols and display it in reverse order.

```
name= input ("Enter a name")

print (name[2], name[1], name[0])
```

8. Write a Python program to get a word with a length of four as input and then display each string element on one line.

```
name= input ("Enter a name")

print(name[3],"\n",name[2],"\n",name[1],"\n",name[0],end="")
```

9. Write a Python program to get a word with a length of four as input, then switch the first and last elements and redisplay the new word.

```
name= input ("Enter a name")

print (name[3], name[1],name[2],name[0],end= "" )
```

10. Write a Python program to get a word with a length of four as input and verify if it is a palindrome.

```
name= input ("Enter a name")

if (name[3]==name[0] and name[2]==name[1]):

    print ("It is a palindrome")

else:

    print ("It is not a palindrome")
```

11. Write a Python program to get a word with a length of four as input and then reverse it.

```
name= input ( "Enter a name" )

print (name[3],name[2],name[1],name[0], sep= "" )
```

12. Write a Python program to get a word with a length of four as input, and if it starts with ‘a’, replace ‘a’ with ‘A’ and redisplay the name; otherwise, display the last two elements of the name.

```
name= input ("Enter a name")

if (name[0]== 'a' ):

    print ('A' ,name[1],name[2],name[3], sep="" )

else:

    print (name[2],name[3], sep="" )
```

13. Write a Python program to get a word, and if the length of the word is even, extract the first element and display it; otherwise, display the last element of the string.

```
word = input ("Enter a name")

if (len (word) %2 ==0):

    print (word[0])

else:

    print (word[ len ( word) -1] )
```

14. Write a Python program to get a word with a length of four as input and count the total number of times it contains ‘e’ or ‘E.’

```
word= input ("Enter a word")

total=0

if (word[0]== 'e' or word[0]== 'E' ):

    total=total+1

if (word[1]== 'e' or word[1]== 'E' ):

    total=total+1

if (word[2]== 'e' or word[2]== 'E' ):

    total=total+1

if (word[3]== 'e' or word[3]== 'E' ):

    total=total+1

print ("The total number of E's " ,total)
```

15. Write a Python program to get a word with a length of four as input and then replace every ‘e’ with ‘X.’

```
name= input ("Enter a name")
if (name[0]== 'e' ):
     temp1= 'x'
else :
     temp1=name[0]
if (name[1]== 'e' ):
     temp2= 'x'
else :
     temp2=name[1]
if (name[2]== 'e' ):
     temp3= 'x'
else :
     temp3=name[2]
if (name[3]== 'e' ):
     temp4= 'x'
else :
     temp4=name[3]
print (temp1,temp2,temp3,temp4,sep= "")
```

Return to Section 3.2

Solutions for Practice Questions (4.3)

1. Write a Python program to display numbers between one and five.

```
n=1
while (n<6):
    print (n)
    n=n+1
```

2. Write a Python program to display numbers between five and one.

```
n=5
while (n>0):
    print (n)
    n=n-1
```

3. Write a Python program to show at which temperature Fahrenheit and Centigrade have the same reading.

```
F=-100
C=-99
while (F!=C):
    print (F, C)
    C=(F-32)*5//9
    F=F+1
```

4. Write a Python program to calculate the following sum: $1+2+3+\ldots+10$.

```
n=1
total=0
while (n<=10):
    total=total+n
    n=n+1
print (total)
```

5. Write a program to calculate the following sum: $\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{10}$.

```
n=1
total=0
while (n<=10):
    total=total+(1/n)
    n=n+1
print (total)
```

6. Write a program to display numbers between one and five using a for loop.

```
for n in [1,2,3,4,5]:
    print (n)
```

7. Write a program to display numbers between five and one using a for loop.

```
for n in [5,4,3,2,1]:
    print (n)
```

8. Write a program to simulate rolling a pair of dice and show how many tries it takes to get a pair.

```
import random
total=0
flag= True
while (flag):
    dice1=random.randint(1,6)
    dice2=random.randint(1,6)
    total=total+1
    print (total, "***",dice1, "****",dice2)
    if (dice1 ==dice2):
        flag= False
```

9. Write a program to get five numbers as input and calculate their average.

```
total=0

for n in range (1,6):

    number=float (input ("Please enter a number"))

    total=total+number

print (total/5)
```

10. Write a program to get inputs from the keyboard, where "-1" indicates the end of inputs, and then calculate their average.

```
total=0

counter=0

flag= True

while (flag):

    number=float(input("Enter a number,-1 to exit"))

    if (number !=-1):

        total=total+number

        counter=counter+1

    else :

        flag= False

print (total/counter)
```

11. Write a program to check whether a given word is a palindrome.

```
flag= True

word= input ("please input a word")

for n in range (0, len (word)):

    if (word[n] !=word[ len (word)-1-n]):

        flag= False

if (flag== True):

    print ("It is a palindrome")

else:

    print ("It is NOT a palindrome")
```

12. Write a program to get a word as input and replace every string character with its first character; for example,“abcd” would change to “aaaa”.

```
word= input ("please input a word")

for n in range (0, len(word)):

    print (word[0],end= "" )
```

Return to Section 4.3

Solutions for Practice Questions (4.5)

1. Write a Python program to display the pattern in Figure 4-2.

```
for i in range (1,5):
    for j in range (1,i+1):
        print ( "*", end=" " )
    print ( "\n" )
```

2. Write a Python program to display the pattern in Figure 4-3.

```
for i in range (1,5):
    for j in range (1,i+1):
        print (i, end= " ")
    print ("\n")
```

3. Write a Python program to display the pattern in Figure 4-4.

```
for i in range (1,5):
    for j in range (1,i+1):
        print (j, end= " ")
    print("\n")
```

4. Write a Python program to display the pattern in Figure 4-5.

```
for i in range (5,1,-1):
    for j in range (1, i):
        print (j, end= " ")
    print ("\n")
```

5. Write a Python program to display the pattern in Figure 4-6.

```
for i in range (1,5):
    for j in range (5, i,-1):
        print (" " , end= " ")
    for k in range (1,j):
        print (k, end= " ")
    print ("\n" ,end= "" )
```

6. Write a Python program to display the pattern in Figure 4-7.

```
for n in range (0,5):
    for m in range (n, 4):
        print (" " , end= "")
    for m in range (n + 1):
        print ("* " , end="")
    print ()
```

7. Write a Python program to display the pattern as depicted in Figure 4-8.

```
for n in range (0,5):
    for m in range (n, 4):
        print (" ", end= "")
        for m in range (n + 1):
            print (n, end="")
    print()
```

8. Write a Python program to display the pattern as depicted in Figure 4-9.

```
for n in range (0,5):
        for m in range (n, 4):
            print (" ", end= "")
        for m in range (n + 1):
            print (n, end=" ")
        print ()
```

9. Write a Python program to display the pattern depicted in Figure 4-10.

```
for n in range (0,5):
    for m in range (n, 4):
        print (" ", end= "")
    for m in range (n + 1):
        print (m, end=" ")
    print ()
```

10. Write a Python program to display the pattern depicted in Figure 4-11.

```
for n in range (0,5):
    for m in range (n, 4):
        print (" ", end= "")
    for m in range (n+1,0,-1):
        print (m, end=" ")
    print ()
```

Return to Section 4.5

Solutions for Practice Questions (5.4)

1. Write a Python program to generate a random number between zero and five, then simulate a magic ball. You may select your message from these options: without a doubt, better not tell you now, my sources say no, ask again later, outlook hazy.

```
import random

def magic(rand1):

    if (rand1==1):

        print (" Reply Hazy")

    if (rand1==2):

        print (" Without a doubt")

    if (rand1==3):

        print (" Better not tell you now")

    if (rand1==4):

        print (" My sources say no")

    if (rand1==5):

        print (" Ask again later")

rand1=random.randint(1,5)

magic(rand1)
```

2. Write a Python program to get a word from a user through the keyboard and then display the first character of that word.

```
word= input("Please enter a word:")

def Extract(word):

    print(word[0])

Extract(word)
```

3. Write a Python program to get a word from a user through the keyboard and then display the first character of that word. Extracting the character should be done in a function, but displaying the character should be done in the main program.

```
word= input ("Please enter a word:")

def Extract(word):

    return(word[0])

print (Extract(word))
```

4. Write a Python program to get two numbers as input and calculate their average.

```
val1= float(input("1st number?"))

val2= float(input("2nd number?"))

def Avg (num1,num2):

    return ((num1+num2)/2)

print (Avg(val1,val2))
```

5. Write a Python program to get a degree in Fahrenheit and convert it to Celsius.

```
F= float (input("Please input a degree in F:"))

def FtoC(F):

    C=(F-32)*5/9

    return C

print (FtoC(F))
```

6. Write a Python program to get an input (integer) and calculate its factorial, where the factorial of a number is denoted by n! and is the product of all positive integers less than or equal to n. For example, $5! = 5 \times 4 \times 3 \times 2 \times 1$.

```
number=int (input("Input a value:"))

def Fact (m):

    fact=1

    for i in range (1,m+1):

        fact=fact*i

    print (m, "!=" ,fact)

Fact(number)
```

7. Write a Python program to generate the first ten Fibonacci numbers, where each is the sum of the two preceding ones. These are numbers in the Fibonacci series: $1, 1, 2, 3, 5, 8, \ldots$

```
number= int(input("How many elements of Fibonacci should I display?"))

def Fib (m):

    i=0

    fib1=1

    fib2=1

    total=0

    print("Fibonacci series: ",fib1,",",fib2,end= "")

    for i in range (1,m+1):

        total=fib1+fib2

        print (",", total,end= "" )

        fib1=fib2

        fib2=total

Fib(number)
```

8. Write a Python program to get a word from a user through the keyboard and then reverse the word through the use of a function.

```
word= input ("Please enter a word:")

def Rev(word):

     newname=""

     for i in range (0, len(word)):

          newname=word[m]+newname

     return(new name)

print (Rev(word))
```

Return to Section 5.4

Solutions for Practice Questions (7.4)

1. Write a Python program to read a file and print every line of the file on the screen by using a for loop.

```
myfile=open("output11.txt",'r')

for n in myfile:

    str1=int(n)

    print(str1)

myfile.close()
```

2. Write a Python program to read a file and print every line of the file on the screen by using a while loop.

```
myfile=open("output11.txt",'r')

str1=myfile.readline()

while str1 !='':

    str1=myfile.readline()

    print(str1)

myfile.close()
```

3. Write a Python program to read a file and print the total number of lines in the file.

```
myfile=open("output11.txt",'r')

str1=myfile.readline()

m=0

while str1 !='' :

    print(str1.rstrip( '\n' ))

    m=m+1

    str1=myfile.readline()

myfile.close()

print("The total number of lines is :",m)
```

4. Write a Python program to read a file and print every word in the file that begins with ‘A’.

```
myfile=open("output2.txt",'r')

line=myfile.readline()

while(line !=''):

    if(line[0] =="A"):

        print(line)

        line=myfile.readline()

myfile.close()
```

5. Write a Python program to get an input from the keyboard and then add it to an existing file.

```
myfile=open("output2.txt",'a')

line=input("please input the word to be added to file")

myfile.write(line)

myfile.close()
```

6. Write a Python program to generate five random numbers and write them in a file.

```
import random

randfile=open( "output2.txt",'w')

for n in range(1,6):

    rand1=random.randint(-10,10)

    randfile.write(f'{rand1} \n')

randfile.close()
```

7. Write a Python program to read a file and check if it contains a specific word.

```
myfile=open("input00.txt",'r')
flag=True
while(flag or line !='' ):
    line=myfile.readline()
    if line.find('Chester') != -1:
        flag=False
    if(line =='' ):
        break
myfile.close()
if(flag==False):
    print("Found")
else:
    print("Not Found")
```

8. Write a Python program to read a file and count the total number of e's in the file.

```
myfile=open("input00.txt",'r')

counter=0

line="..."

while(line !='' ):

     line=myfile.readline()

     for n in range(0,len(line)):

          if(line[n] =="e"):

               counter=counter+1

myfile.close()

print(counter)
```

9. Write a Python program to write five numbers on a file and then display their summation.

```
myfile=open("data.txt","w")

for n in range(1,6):

     myfile.write(str(n))

     myfile.write("\n")

myfile.close()

myfile=open("data.txt","r")

sum=0

for n in myfile:

     sum=sum+int(n)

print(sum)

myfile.close()
```

Return to Section 7.4

Solutions for Practice Questions (8.1)

1. Write a Python program to replace every element of a list with its square.

```
for i in range(len(num)):

    num[i]=num[i]**2
```

2. Write a Python program to find the smallest and the second smallest elements in a list.

```
num=[12,-3,100,-67]

num.sort()

print(num[0],num[1])
```

3. Write a Python program to remove the first and last elements of a list and redisplay the new list.

```
num=[12,-3,100,-67,200,5]

m=len(num)

num.remove(num[0])

num.remove(num[len(num)-1])

print(num)
```

4. Write a Python program to calculate the average of the elements in a list.

```
num=[-5,4,5,1,-4]

print(sum(num)/len(num))
```

5. Write a Python program to replace every even element of a list with "*".

```
num=[12,-6,3,200,11,21,0]

for n in range(0,len(num)):

    if(num[n] %2==0):

        num[n]="*"

print(num)
```

Return to Section 8.1

Solutions for Practice Questions (9.1)

1. Write a Python program to generate an array and reverse it, without changing the order of elements in the original array.

```
def rev(A):
    NewA=A[::-1]
    print("Reversed Array is: ",end=" ")
    for i in NewA:
        print(i, end=" ")
A=[1,2,3,4,5]
rev(A)
```

2. Write a Python program to generate an array as input and reverse it. The original array will be reversed.

```
A=[1,2,3,4,5,6]
print(A)
def rev(A,start,end):
    while start < end:
        A[start],A[end]=A[end],A[start]
        start=start+1
        end=end-1
rev(A,0,5)
print("Reversed array is: ")
print(A)
```

3. Write a Python program to find the common elements in two arrays.

```
A=[1,-1,0,100,2]

B=[100,12,-1]

def com(A,B):

    com=[i for i in A if i in B]

    return com

print(com(A,B))
```

4. Write a Python program to find repeated items in an array.

```
A=[1,-1,0,1,2,200,1,0]

def Repeat(A):

    size=len(A)

    rep=[]

    for i in range(size):

        k=i+1

        for j in range(k,size):

            if (A[i]==A[j] and A[i] not in rep):

                rep.append(A[i])

        return rep

print(Repeat(A))
```

5. Write a Python program to find the second largest value in an array.

```
A=[1,0,4,-1,100]

n=len(A)

def Max(A,size):

    A.sort(reverse=True)

    for i in range(1,size):

        if (A[i] != A[0]):

            print("The 2nd largest ",A[i])

            return

Max(A,n)
```

6. Write a Python program to check if an array contains a value.

```
A=[1,-1,100,2,-100]

flag=False

for i in A:

    if(i==100):

        flag=True

        break

if( flag==True):

    print("Found")

else:

    print("!Found")
```

Return to Section 9.1