Programming in Python

DECAP776

Edited by
Ajay Kumar Bansal

LOVELY
PROFESSIONAL
UNIVERSITY

LOVELY PROFESSIONAL UNIVERSITY

Programming in Python

Edited By:
Ajay Kumar Bansal

Title: PROGRAMMING IN PYTHON

Author's Name: Dr. Rajni Bhalla

Published By : Lovely Professional University

Publisher Address: Lovely Professional University, Jalandhar Delhi GT road, Phagwara - 144411

Printer Detail: Lovely Professional University

Edition Detail: (I)

ISBN: 978-81-19334-41-4

Content

Unit 01: Python Basics

CONTENTS

Objectives

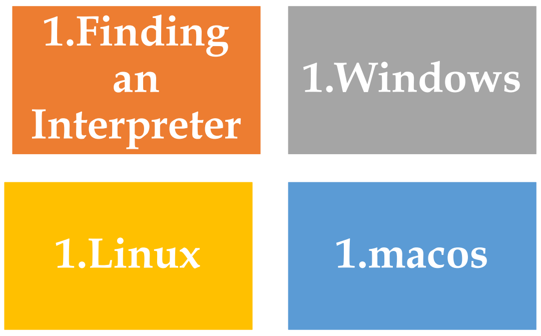
- After studying this unit, you will be able to:
- Understand basic concepts about python
- Learn installation steps in python
- Learn control statements
- Understand basic concepts of functions in python

Introduction

Python is a popular high-level, general-purpose programming language. Guido van Rossum invented it in 1991, and the Python Software Foundation continued to advance it. Programmers may convey their ideas in less code because to its syntax, which was created with code readability in mind.

Python is a programming language that enables quick work and more effective system integration.

Beginning with Python programming:



1. Finding an interpreter:
    a. ***Kaggle***: supports background processing in a manner comparable to Jupyter Notebook.
    b. ***Google Colab***: uses a google account solely, similar to Kaggle. both GPU and TPU compute, but only the pro version allows for background execution.
    c. ***Python.org***: like running Python from the command line
    d. ***Programiz***:Programmers are able to create and execute Python code online using this Python compiler (interpreter). It can use an IDLE-like Python Shell and accept user input.
    e. ***Online Python***: You can create, execute, and distribute Python code online for free using this online interpreter (compiler). It is the online Python interpreter that is the quickest, most trustworthy, and most potent.
    f. ***Online GDB***: An online IDE with a Python interpreter is called OnlineGDB. It is quick and simple to run a Python programme online with this interpreter. It works with Python 3.
    g. ***Replit:***The greatest website for Python online execution and interactive programming. The name of this terminal is derived from the Lisp and Python read-eval-print loop.
2. **Windows**: The Python software can be acquired from http://python.org/, and it comes with IDLE (Integrated Development Environment), one of many free interpreters that can be used to run Python scripts.
3. **Linux**:Popular Linux distributions like Fedora and Ubuntu include Python by default. Enter "python" in the terminal emulator to see what version of Python is currently running. When it launches, the interpreter should print the version number.
4. **macOS**: Python 2.7 is typically included with macOS. Python 3 must be manually installed from http://python.org.

1.1 What can Python do?

- Web applications can be developed on a server using Python.
- Workflows can be made with Python and other technologies.
- Database systems are connectable with Python. Files can also be read and changed by it.
- Big data management and advanced mathematical operations can both be done with Python.
- Python can be used to produce software that is ready for production or for rapid prototyping.

1.2 Why Python

- Python is cross-platform compatible (Windows, Mac, Linux, Raspberry Pi, etc).
- The syntax of Python is straightforward and resembles that of English.

- Python's syntax differs from various other programming languages in that it enables programmers to construct applications with fewer lines of code.
- Python operates on an interpreter system, allowing for the immediate execution of written code. As a result, prototyping can proceed quickly.
- Python can be used in a functional, object-oriented, or procedural manner.

1.3 Python Syntax compared to other Programming Languages

- With influences from mathematics and a focus on readability, Python shares several characteristics with the English language.
- In contrast to other programming languages, which frequently employ semicolons or parentheses, Python uses new lines to finish a command.
- Indentation, which utilises whitespace, is how Python defines scope, including the scope of loops, functions, and classes. Curly-brackets are frequently used for this in other computer languages.

1.4 Download and Installation of Python

A popular high-level programming language is Python. Installing Python on our machine is the initial step in writing and running Python code.

The process of installing Python on Windows is simple.

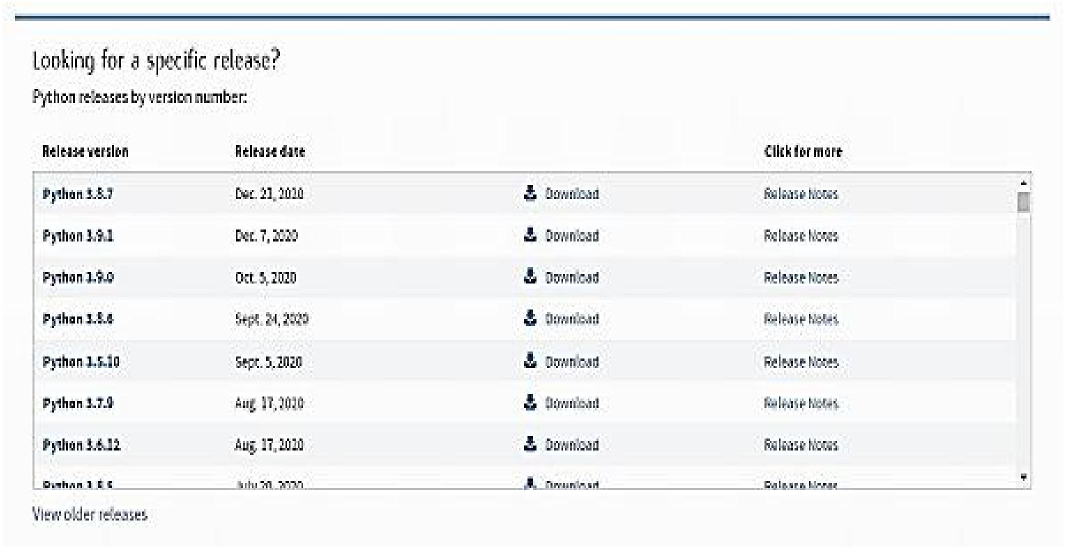
Step1: Select version of Python to Install

There are several versions of Python available, and each one has a different syntax and way of functioning. We must select the version that we want to utilise or that we require. There are numerous Python 2 and Python 3 iterations available.

Step2: Download Python Executable Intsaller

Navigate to the Download for Windows section on the Python website (www.python.org) using your web browser.

A list of every Python version will be provided. Choose the version you need, then click "Download." Let's say we go with Python version 3.9.1.



Looking for a specific release?

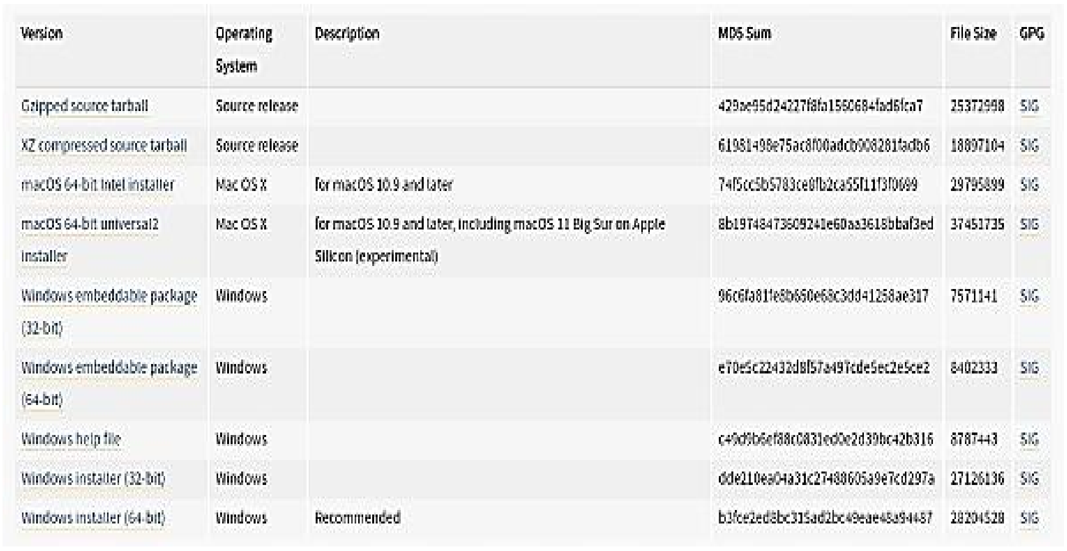
Python releases by version number:

Release version	Release date		Click for more
Python 3.8.7	Dec. 21, 2020	Download	Release Notes
Python 3.9.1	Dec. 7, 2020	Download	Release Notes
Python 3.9.0	Oct. 5, 2020	Download	Release Notes
Python 3.8.6	Sept. 24, 2020	Download	Release Notes
Python 3.5.10	Sept. 5, 2020	Download	Release Notes
Python 3.7.9	Aug. 17, 2020	Download	Release Notes
Python 3.6.12	Aug. 17, 2020	Download	Release Notes
Python 3.8.5	July 20, 2020	Download	Release Notes

View older releases

Upon clicking download, a variety of executable installers with varying operating system requirements will be made available. Select the installer that best fits your operating system and download it. Imagine that we choose the Windows installer (64 bits).

The download is less than 30MB in size.

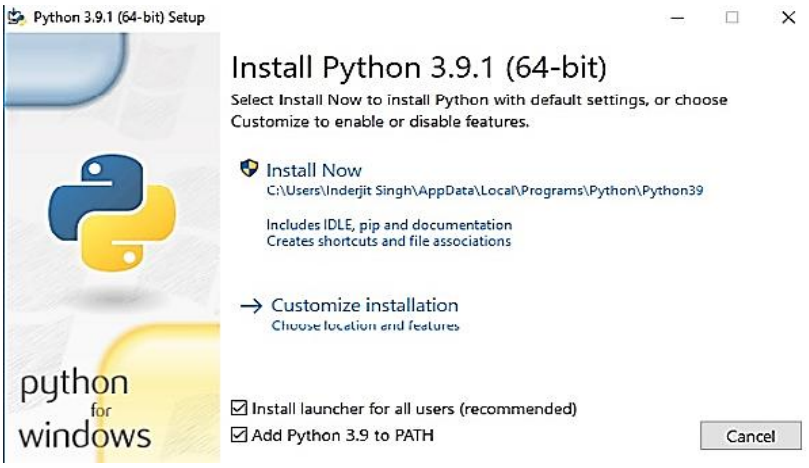


Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		429ae95d24227f8fa1560684fad6fca7	25372998	SIG
XZ compressed source tarball	Source release		61981498e75ac8f00adcb908281fadb6	18897104	SIG
macOS 64-bit Intel installer	Mac OS X	for macOS 10.9 and later	74f5cc5b5783ce8fb2ca55f11f3f0699	29795899	SIG
macOS 64-bit universal2 installer	Mac OS X	for macOS 10.9 and later, including macOS 11 Big Sur on Apple Silicon (experimental)	8b19748473609241e60aa3618bbaf3ed	37451735	SIG
Windows embeddable package (32-bit)	Windows		96c6fa81fe8b650e68c3dd41258ae317	7571141	SIG
Windows embeddable package (64-bit)	Windows		e70e5c22432d8f57a497cde5ec2e5ce2	8402333	SIG
Windows help file	Windows		c49d9b6ef88c0831ed0e2d39bc42b316	8787443	SIG
Windows installer (32-bit)	Windows		dde210ea04a31c27488605a9e7cd297a	27126136	SIG
Windows installer (64-bit)	Windows	Recommended	b3fce2ed8bc315ad2bc49eae48a94487	28204528	SIG

Step3: Run Executable Installer

The Python 3.9.1 Windows 64-bit installation was downloaded.

activate the installation. Click Install New after making sure that both of the checkboxes at the bottom are selected.



Python 3.9.1 (64-bit) Setup

Install Python 3.9.1 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

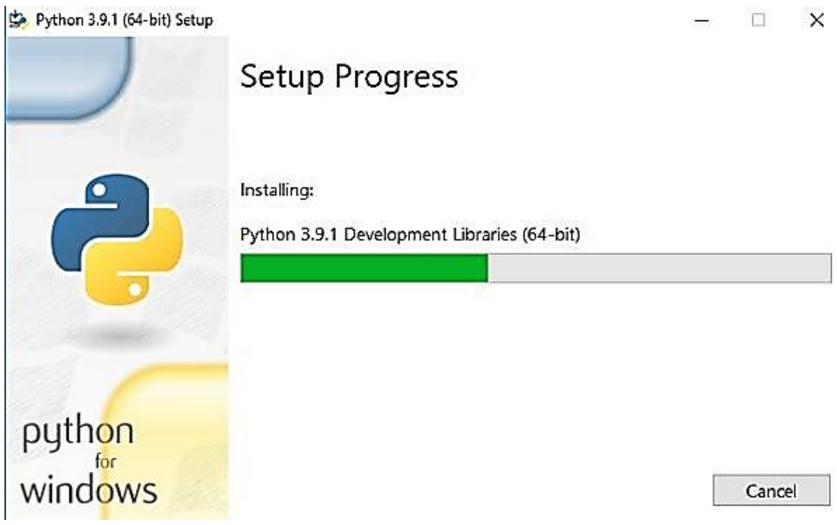
Install Now
C:\Users\Inderjit Singh\AppData\Local\Programs\Python\Python39

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.9 to PATH

The installation process begins when you click the Install Now button.

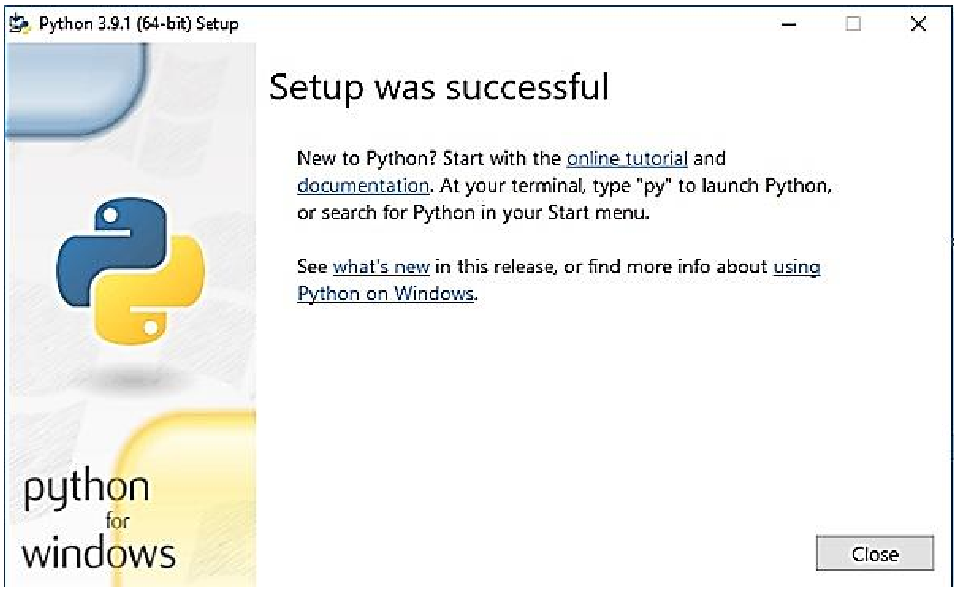


Setup Progress

Installing:

Python 3.9.1 Development Libraries (64-bit)

The installation procedure will take a few minutes to finish, and after it has, the screen below will appear.



Setup was successful

New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

See what's new in this release, or find more info about using Python on Windows.

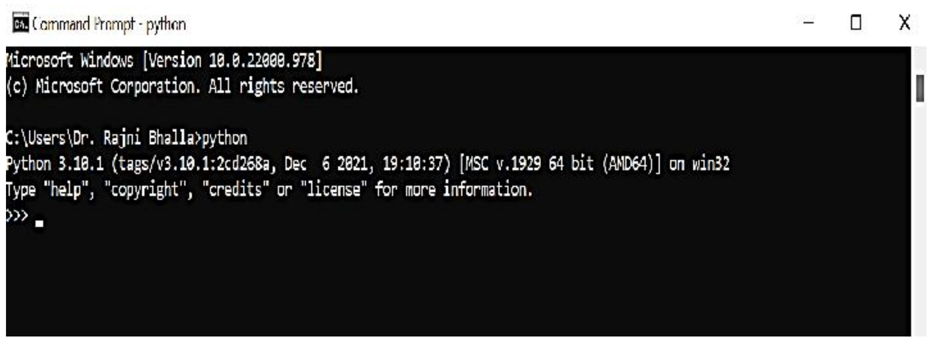
Step4: Verify python installed in windows

Verify that Python has been successfully installed on your system. Take the directions provided.

Launch the command window.

Enter "python" after you type it.

If Python is successfully installed on your Windows system, the version you have installed will be shown.



```
Microsoft Windows [Version 10.0.22000.978]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dr. Rajni Bhalla>python
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec  6 2021, 19:10:37) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Step5: Verify Pip was installed

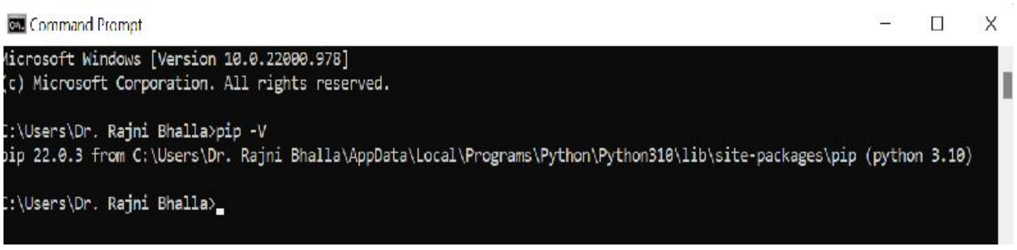
Pip is an effective framework for managing Python software packages. Therefore, confirm that it is set up.

To check if pip was installed, adhere to the instructions provided.

Launch the command window.

To see if pip was installed, type pip -V.

If pip is successfully installed, the output shown below occurs.



```
icrosoft Windows [Version 10.0.22000.978]
(c) Microsoft Corporation. All rights reserved.

:\Users\Dr. Rajni Bhalla>pip -V
pip 22.0.3 from C:\Users\Dr. Rajni Bhalla\AppData\Local\Programs\Python\Python310\lib\site-packages\pip (python 3.10)

:\Users\Dr. Rajni Bhalla>_
```

Python and pip have been successfully installed on our Windows PC.

1.5 Python Data Types

- ### Built-in Data Types

The concept of data type is crucial in programming.

Different forms of data can be stored in variables, and different types can perform various functions.

The following categories of data types are included by default in Python:

Text Type	**str**
Numeric Types	int, float, complex
Sequence Types	List, tuple, range
Mapping Type	dict
Set Types	Set, frozenset
Boolean Type	Bool
Binary Types	Bytes, bytearray, memoryview
None Type	NoneType

- ### Getting the Data Type

Using the type() method, you may determine the data type of any object:

```
x = 5
print(type(x))
```

Output

```
<class 'int'>
```

- ### Setting the Data Type

When you give a variable a value in Python, the data type is already determined:

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name”: "John", "age”: 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview
x = None	nonetype

Setting The Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

Example	Data Type
x = str("Hello World")	str
x = int (20)	int
x = float (20.5)	float
x = complex(1j)	complex
x = list (("apple", "banana", "cherry"))	list
x = tuple (("apple", "banana", "cherry"))	tuple
x = range (6)	range
x = dict (name="John", age=36)	dict
x = set (("apple", "banana", "cherry"))	set
x = frozenset (("apple", "banana", "cherry"))	frozenset

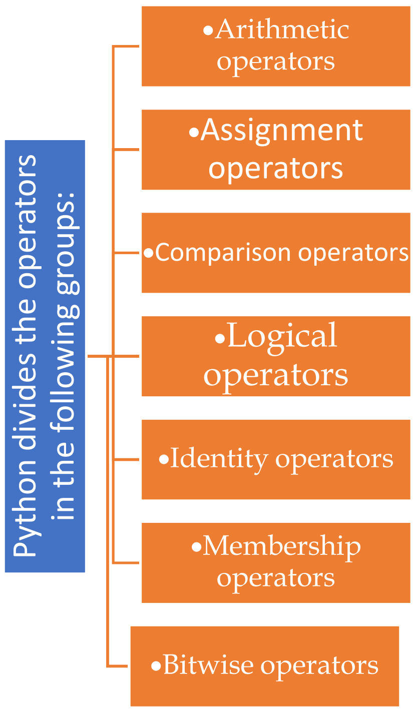
x = bool (5)	bool
x = bytes (5)	bytes
x = bytearray (5)	bytearray
x = memoryview (bytes (5))	memoryview

1.6 Python Operators

Operations on variables and values are carried out using operators.

The + operator is used to combine two values in the example below:

```
print(10+5)
```



Python Arithmetic Operators

Common mathematical procedures are carried out using arithmetic operators and numeric quantities.

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x – y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y

//	Floor division	x // y

Python Assignment Operators

In order to assign values to variables, assignment operators are used:

Operator	Example	Same As
=	X=5	X=5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Python Comparison Operators

To compare two values, comparison operators are employed.

Operator	Name	Example
==	Equal	x == y
!=	Not equal	X != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

Conditional statements are combined using logical operators:

Operator	Description	Example
And	Returns True if both statements are true	x < 5 and x < 10
Or	Returns True if one of the statements is true	x < 5 or x < 4
Not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Identity Operators

Identity operators are used to compare things to determine whether they are indeed the same object in the same memory address rather than whether they are equal:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
isnot	Returns True if both variables are not the same object	x is not y

Python Membership Operators

To determine whether a sequence is contained in an object, membership operators are used:

Operator	Description	Example
In	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

1.7 Control Statements in Python

Loops are used in Python to continually iterate over a section of code. Control statements are used to modify a loop's execution from its default behaviour. On the basis of a condition, control statements are used to alter how the loop executes. In Python, a variety of control statements are available.

Control Statements in Python are:

- Break Statement
- Continue Statement
- Pass Statement

Break Statement

In Python, the break statement is used to end or remove the control from the loop that contains the statement. It is used to end nested loops (a loop inside another loop), which are common with both while and for loops. The inner loop is ended, and control is transferred to the statement in the outside loop.

Continue statement

When a Python programme sees a continue statement, it skips the current iteration's execution when the condition is satisfied and instead allows the loop to carry on to the next iteration. It is used to keep the programme running even when it meets a break while being executed.

Pass statement

When the condition is met, the pass statement, a null operator, is used by the programmer to do nothing. Python's control statement simply moves on to the next iteration without stopping the execution or skipping any steps.

A programmer can use the pass statement to prevent the interpreter from throwing an error when a loop is left empty.

1.8 Python Functions

A function is a section of code that only executes when called.

You can supply parameters—data—to a function.

As a result, a function may return data.

Creating a Function

In Python a function is defined using the `def` keyword:

Example:

```
def my_function ():
print("Hello from a function")
```

Calling a Function

Use the function name in parenthesis to invoke the function:

```
def my_function ():
print("Hello from a function")

my_function()
```

Arguments

Functions accept arguments that can contain data.

The function name is followed by parenthesis that list the arguments. Simply separate each argument with a comma to add as many as you like.

The function in the following example only takes one argument (fname). A first name is passed to the function when it is called, and it is utilised there to print the whole name:

Example:

```
def my_function(fname):
print(fname + " Application")

my_function("Computer")
my_function("Science")
```

my_function("System")

Args is a common abbreviation for arguments in Python documentation.

Summary

- Python is a popular high-level, general-purpose programming language. Guido van Rossum invented it in 1991, and the Python Software Foundation continued to advance it
- Web applications can be developed on a server using Python.
- Python is cross-platform compatible (Windows, Mac, Linux, Raspberry Pi, etc).
- Indentation, which utilises whitespace, is how Python defines scope, including the scope of loops, functions, and classes. Curly-brackets are frequently used for this in other computer languages.
- Operations on variables and values are carried out using operators.
- Loops are used in Python to continually iterate over a section of code. Control statements are used to modify a loop's execution from its default behaviour.
- In Python, the break statement is used to end or remove the control from the loop that contains the statement
- When a Python programme sees a continue statement, it skips the current iteration's execution when the condition is satisfied and instead allows the loop to carry on to the next iteration
- When the condition is met, the pass statement, a null operator, is used by the programmer to do nothing.

Keywords

Python:The general-purpose, interactive, object-oriented, and high-level programming language Python is particularly well-liked.

Python path: It has a role similar to PATH. This variable tells the Python interpreter where to locate the module files imported into a program.

Python startup: It includes the location of a Python source code initialization file. Every time the interpreter is launched, it is executed.

Unix:The original Python IDE for Unix is called IDLE.

Windows: The first Windows interface for Python is called PythonWin, and it is an IDE with a GUI.

Macintosh:You can get the Macintosh version of Python and the IDLE IDE from the main website in MacBinary or BinHex format.

Reserved Words:You cannot use them as identifier names for constants, variables, or anything else.

Python Numbers: Number data types store numeric values.

Python Strings:Python defines strings as a contiguous group of characters that are enclosed in quotation marks.

Python Lists:Of all the compound data types in Python, lists are the most flexible. Items in a list are delimited by square brackets and separated by commas ([]).

Python Tuples: Another sequence data type that resembles the list is the tuple. A tuple is made up of several values that are separated by commas.

Python Dictionaries: The dictionaries used by Python are something like hash tables. They consist of key-value pairs and operate similarly to associative arrays or hashes seen in Perl.

Self Assessment

Q1: Who was the Python programming language's creator?

A. Mark
B. Guido can Rossum
C. Alfred novel
D. Ralf Kleinberg

Q2: What programming languages does Python support?

A. Structural programming
B. Object-oriented programming
C. Functional programming
D. All of the above

Q3. Select the correct extension of python file

A. .ps
B. .pyth
C. .py
D. .thon

Q4: Select true statement

A. Python code is both compiled and interpreted
B. Python code is neither compiled nor interpreted
C. Python code is only compiled
D. Python code is only interpreted

Q5: Which of the following is used in Python to define a block of code?

A. Key
B. Brackets
C. Indentation
D. All of the above

Q6: Which of the subsequent characters is used in Python to provide single-line comments?

A. //
B. #
C. /*
D. */

Q7: Which command is used to know the version of python

A. python -version
B. python -v
C. python -V
D. None of above

Q8: What is Python's precedence hierarchy?

A. Exponential, Parentheses, Multiplication, Division, Addition, Subtraction
B. Exponential, Parentheses, Division, Multiplication, Addition, Subtraction
C. Parentheses, Exponential, Multiplication, Division, Subtraction, Addition
D. Parentheses, Exponential, Multiplication, Division, Addition, Subtraction

Q9: PIP stands for

A. Program installer Preferrable
B. Preferred Installer Program
C. Parenthesis installer program
D. Program installer Parenthesis

Q10: In Python programming, which of the following is not a basic data type?

A. Dictionary
B. Class
C. Tuple
D. Lists

Q11: Which of the above statements in Python is used to generate an empty set?

A. Empty(a)
B. {}
C. set()
D. None of above

Q12: Which of the following describes how a function in Python is used?

A. For your application, functions don't improve modularity.
B. you can't also write your own functions.
C. Functions are reusable programme components
D. Functions are reusable programme components.

Q13: Which of the following sentences is utilized in Python's Exception Handling?

A. try
B. except
C. finally
D. All of the above

14: Which of the following list items is a legitimate Python escape sequence?

A. \n
B. \t
C. \\
D. All of the above

15. What language is written in Python?

A. C++
B. C
C. Java
D. None of these

Answers for Self Assessment

1.	B	2.	D	3.	C	4.	B	5.	C
6.	B	7.	C	8.	D	9.	B	10.	B
11.	C	12.	C	13.	D	14.	D	15.	C

Review Questions

1. Write down steps to download and install python.
2. Write down challenges used in installing python.
3. Explain all python data types.
4. Explain control statements in python.
5. What is the difference between list and tuples in python.
6. What is Python? What is the benefit of using a python.

FurtherReadings

- Mark Lutz,Programming Python: Powerful Object-Oriented Programming, OREILLY
- Wes McKinney, Python for data analysis, OREILLY
- David Ascher and Mark Lutz, Learning Python, OREILLY
- Eric Matthes, Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming, Starch Pres

Web Links

https://www.tutorialspoint.com/python/index.htm

https://www.python.org/downloads/

https://www.w3schools.in/python/data-types

https://www.programiz.com/python-programming/online-compiler/

https://www.codecademy.com/catalog/language/python

Unit 02: Python Data Structures

CONTENTS

Objectives

After this unit, student would be able to:

- understand basic concepts about strings
- learn about lists, tuples, sets, dictionaries

Introduction

In Python, strings are enclosed in either single or double quotation marks.

The same thing as "hello" is "hello".

With the print () method, a string literal can be shown:

Example:

```
#You can use double or single quotes:
print("Hello")
print('Hello')
```

2.1 Assign String to a Variable

The variable name, an equal sign, and the string are used to assign a string to the variable:

Example:

```
a = "Hello"
print(a)
```

2.2 Multiline Strings

A multiline string can be assigned to a variable by enclosing it in three quotes:

```
a = """Python is an interpreted, object-oriented, high-level,
dynamically semantic programming language.
It is particularly desirable for Rapid Application Development
as well as for usage as a scripting or glue language to tie
existing components together due to its high-level built-in
data structures, dynamic typing, and dynamic binding."""
print(a)
```

2.3 Strings are Arrays

Python's strings, like those of many other widely used programming languages, are collections of bytes that represent unicode characters.

Python does not, however, support character data types; instead, a single character is represented as a string with length 1.

To access the string's constituents, use square brackets.

Example:

Get the character at position 1 (remember that the first character has the position 0):

```
a = "Hello, World!"
print (a [1])
```

2.4 Looping Through a String

Strings are arrays;therefore, we can use a for loop to iterate through their characters.

Example: Loop through the letters in the word "apple":

```
for x in "apple":
    print(x)
```

2.5 String Length

Use the len() function to determine a string's length.

Example: The length of a string is returned by the len() function:

Example	Output

a = "Hello, World!"
print(len(a))
	13

2.6 Check String

The keyword in can be used to determine whether a specific word or character is present in a string.

Example: Check if "arrays" is present in the following text:

Example	Output
txt = "Like many other popular programming languages, strings in Python are arrays of bytes representing Unicode characters.!" print ("arrays" in txt)	True

It can be used in an if statement:

Example:

Example	Output
txt = "Like many other popular programming languages, strings in Python are arrays of bytes representing Unicode characters.!" If 'arrays' in txt: print ("Yes, 'arrays' is present.")	Yes, 'arrays' is present.

Check if NOT

The keyword not in can be used to determine whether a specific word or character DOES NOT exist in a string.

Example: Verify that the following text DOES NOT contain the word "file":

Example	Output

txt = "Like many other popular programming languages, strings in Python are arrays of bytes representing Unicode characters.!" print("file" not in txt)	True

It can be used in an if statement:

Example	Output
txt = "Like many other popular programming languages, strings in Python are arrays of bytes representing Unicode characters.!" If 'file' not in txt: print("No, 'file' is NOT present.")	No, 'file' is NOT present

2.7 Python Slicing Strings

Slicing

The slice syntax allows you to return a range of characters.

To return a portion of the string, enter the start index and the end index, separated by a colon.

Get the characters (not included) from positions 3 to 7:

Example	Output
b = "Hello, World!" print(b[3:7])	lo,

Note: Character one has index 0.

Slice From the Start

The range will begin at the first character if the start index is omitted:

Example*:* Get the characters (not included) from position 1 to position 5:

Example	Output
b = "Welcome, Students!" print (b [:5])	Welco

Slice To the End

The range will extend to the end if the end index is omitted.

Example: Get the characters starting at position 2 and continuing to the very end.

Example	Output
b = "Welcome, Students!" print(b[2:])	lcome, Students!

2.8 Negative Indexing

To begin the slice at the string's end, use negative indexes.

Example:

acquire the characters:

the letter "e" in "Students!" (Position -5)

"s" in "Students!" (position -2) is to be added but is excluded:

Example	Output

b = "Welcome, Students!"
print(b[-5:-2])
	ent

Python-Modify Strings

You can use a variety of built-in methods on strings in Python.

List of those methods are:

UpperCase

Lower Case

Remove Whitespace

Replace String

Split String

2.9 Uppercase

The string is returned by the upper() function in upper case.

Example	Output

a = "Welcome, Students!"
print(a.upper())
	WELCOME, STUDENTS!

2.10 Lowercase

Lowercase characters are returned by the lower() function.

Example	Output

a = "Welcome, Students!"
print(a.lower())
	welcome, students!

2.11 Remove Whitespace

Whitespace is the blank space that appears before and/or after the actual text, and you should usually eliminate it. Example

The strip() method eliminates any leading or trailing whitespace.

Example	Output

a = "   Welcome, Students! "
 print(a.strip())
	Welcome, Students!

2.12 Replace String

A string is replaced with another string using the replace() method.

Example:

Example	Output

a = "Welcome, Students!"
 print(a.replace("W", "J"))
	Jelcome, Students!

2.13 Split String

The text between the chosen separator is used as the list elements when the split() method returns a list.

Example	Output

a = "Welcome, Students!"
 b = a.split(",")
 print(b)
	['Welcome', ' Students!']

Python – String Concatenation

Use the + operator to concatenate, or merge, two strings.

Example	Output

a = "Welcome"
 b = "Students"
 c = a + b
 print(c)
	WelcomeStudents

Add a " " to create a pause between them:

Example	Output

a = "Welcome"
 b = "Students"
 c = a + " " + b
 print(c)
	Welcome Students

2.14 Python – Format – Strings

String Format

This combination of strings and numbers is not possible:

age = 36

txt = "This is my file, I am " + age

print(txt)

Output

Type Error: must be str, not int

But by utilising the format() technique, we can combine texts and numbers!

The given arguments are formatted using the format() method, which also inserts them into the string in the appropriate placeholders:

Example:

For strings, insert numbers using the format() method:

Example	Output

age = 36
txt = "This is my file, and I am {}"
print(txt.format(age))
	This is my file, and I am 36

There is no limit to the amount of arguments that can be passed to the format() method.

2.15 Python-Escape Characters

Use an escape character to inserted prohibited characters into a string.

Backslashes and the character you want to insert are considered escape characters.

A double quotation inside a string that is surrounded by double quotes is an illustration of an unlawful character.

Example	Output
txt = "String is a collection of "alphabets" words or other characters."	SyntaxError: invalid syntax #You will get an error if you use double quotes inside a string that are surrounded by double quotes:

Solution: Use the escape key to resolve this issue.

Example	Output
txt = "String is a collection of \" alphabets \" words or other characters."	String is a collection of "alphabets" words or other characters.

Escape Characters

Python also uses the following escape symbols:

Code	Result
\'	Single Quote
\\	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal value
\xhh	Hex value

2.16 Python-Lists

Lists

Multiple elements can be kept in a single variable by using lists.

One of the four built-in data types in Python for storing data collections is the list; the other three are the tuple, set, and dictionary, each of which has a unique purpose.

Square brackets are used to form lists.

Example:

Example	Output

thislist = ["onion", "tomato", "brinjal"]
print(thislist)
	['onion', 'tomato', 'brinjal']

List Items

List items can have duplicate values and are ordered and editable.

The first item in a list has the index [0], the second item has the index [1], et.

Ordered

When we refer to a list as being ordered, we indicate that the entries are in a specific order that will not alter.

The new things will be added at the end of the list if you add more items.

Changeable

The list is modifiable, which means that after it has been generated, we can edit, add, and remove entries from it.

Allow Duplicates

Lists can contain items with the same value since they are indexed.

Example:

Example	Output

thislist = ["onion", "tomato", "brinjal", "onion", "tomato"]
print(thislist)
	['onion', 'tomato', 'brinjal', 'onion', 'tomato']

List Length

The len() method can be used to count the number of elements in a list:

Example	Output

thislist = ["onion", "tomato", "brinjal”]
print(len(thislist))
	3

List Items - Data Types

Any data type can be used for list items.

Example: String, int and boolean data types:

Example	Output

list1 = ["onion", "tomato", "brinjal"]
list2 = [2, 3, 4, 5, 6]
list3 = [True, False, False]
print(list1)
print(list2)
print(list3)
	['onion', 'tomato', 'brinjal']
	[2, 3, 4, 5, 6]
	[True, False, False]

Example: A list can contain different data types. A list with strings, integers and boolean values shown in below table:

Example	Output

list1 = ["def", 36, False, 42, "male"]
print(list1)
	['def', 36, False, 42, 'male']

Accessing Values in Lists

Use the square brackets for slicing along with the index or indices to obtain the value located at that index to access values in lists. For instance,

Example	Output

list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7];
print "list1[0]: ", list1[0]
print "list2[1:5]: ", list2[1:5]
	list1[0]: physics
	list2[1:5]: [2, 3, 4, 5]

Updating Lists

By providing the slice on the left-hand side of the assignment operator, you can change one or more list elements. You can also add to list elements by using the append() method. For instance,

Example	Output

list = ['physics', 'chemistry', 1997, 2000];
print "Value available at index 2: "
print list [2]
list [2] = 2001;
print "New value available at index 2: "
print list [2]
	Value available at index 2:
	1997
	New value available at index 2:
	2001

Delete List Elements

If you are certain whose element(s) you are deleting, you can use the del statement to remove them; otherwise, you can use the remove() method. For instance

Example	Output
list1 = ['apple', 'orange', 1998, 2015]; print list1 del list1[2]; print "After deleting value at index 2: print list1	['apple', 'orange', 1998, 2015] After deleting value at index 2: ['apple', 'orange', 2015]

Basic List Operations

The + and * operators act on lists similarly to how they act on strings; they signify concatenation and repetition here as well, but the outcome is a new list rather than a string.

Python Expression	Result	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Indexing, Slicing, and Matrixes

Due to the fact that lists are sequences, indexing and slicing for lists function similarly to how they do for strings.

Assuming the data below

L = ['file', 'File', 'FILE!']

Python Expression	Result	Description
L[2]	FILE!	Offset starts at zero
L[-2]	File	Negative: Count from the right
L [1:]	['File', 'FILE!']	Slicing fetches section

Built-in Functions & Methods

Sr.no	Function with Description
1	cmp(list1, list2) Compares elements of both lists.
2	len(list) Gives the total length of the list.
3	max(list) Returns item from the list with max value.
4	min(list) Returns item from the list with min value.
5	list(seq) Converts a tuple into list.

Python includes the subsequent list of methods.

Sr.No	Methods and Descriptions
1	list.append(obj) Appends object obj to list
2	list.count(obj) Returns count of how many times obj occurs in list
3	list.extend(seq) Appends the contents of seq to list
4	list.index(obj) Returns the lowest index in list that obj appears
5	list.insert(index, obj)

	Inserts object obj into list at offset index
6	list.pop(obj=list[-1]) Removes and returns last object or obj from list
7	list.remove(obj) Removes object obj from list
8	list.reverse() Reverses objects of list in place
9	list.sort([func]) Sorts objects of list, use compare func if given

2.17 Python-Tuples

An ordered and unchangeable group of things is referred to as a tuple. Sequences are what tuples and lists both are. Tuples and lists vary in those tuples cannot be altered, although lists may, and because tuples use parentheses while lists use square brackets.

Simply placing various values separated by commas forms a tuple. You may also choose to enclose these comma-separated values in parenthesis. For instance:

```
tpl1 = (‘fruits’, ‘vegetables’, 2003, 2005);
tpl2 = (5,6,7,8,9);
tpl3 = “d”, “e”, “f”, “g”, “h”;
```

There are two parentheses surrounding the empty tuple, which contains nothing.

```
tpl1 = ();
```

Even though there is only one value in a tuple, you must still use a comma when writing it.

```
tpl1 = (50 , );
```

Tuple indices begin at 0 like string indices do, and they can be concatenated, sliced, and other operations.

Accessing Values in Tuples

Use the square brackets for slicing along with the index or indices to obtain the value located at that index to access values in tuples. For instance:

```
tpl1 = (‘fruits’, ‘vegetables’, 2003, 2005);
tpl2 = (5,6,7,8,9,10,11);
print "tpl1[0]: ", tpl1[0];
print "tpl2[1:5]: ", tpl2[1:5];
```

The outcome of running the aforementioned code is the following:

```
tpl1[0]: fruits
tpl2[1:5]: [6,7,8,9]
```

Updating Tuples

Due to their immutability, tuples cannot be updated or have their element values changed. The example that follows shows how to generate new tuples by using pieces of existing tuples:

```
tpl1 = (35, 45.56);
tpl2 = ('apple', 'orange');
# So, let's create a new tuple as follows
```

```
tpl3 = tpl1 + tpl2;
print (tpl3);
```

The outcome of running the aforementioned code is the following:

```
(35, 45.56, 'apple', 'orange')
```

Delete Tuple Elements

Individual tuple elements cannot be eliminated. Naturally, there is nothing wrong with creating another tuple after eliminating the undesirable components.

Simply use the del statement to specifically remove a whole tuple. For instance

Python Expression	Result

tpl1 = (35, 45.56);
tpl2 = ('apple', 'orange');
print (tpl1);
del (tpl1);
print("After deletion")
print(tpl1);
	Traceback (most recent call last):
	File "./prog.py", line 8, in <module>
	NameError: name 'tpl1' is not defined

Basic Tuple Operations

The + and * operators act on tuples similarly to how they act on strings; they signify concatenation and repetition here as well, but the outcome is a new tuple rather than a string.

In fact, all of the general sequence operations we used on strings in the previous chapter work on tuples as well.

Python Expression	Result	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!') * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Indexing, Slicing, and Matrixes

Tuples function similarly to strings in terms of indexing and slicing because they are sequences. assuming the data below

```
L = ('file', 'File', 'FILE!')
```

Python Expression	Result	Description
L[2]	‘FILE!’	Offset starts at zero
L[-2]	‘File’	Negative: Count from the right

L [1:]	['File', 'FILE!']	Slicing fetches section

No Enclosing Delimiters

As seen in these brief examples, tuples are the default for any group of numerous objects that are comma-separated and expressed without distinguishing symbols, such as brackets for lists and parentheses for tuples.

Python Expression	Result

print ('apple', -5.33e92, 19+5.6j, 'abc');
x, y = 3, 4;
print ("Value of x , y : ", x,y);
	apple -5.33e+92 (19+5.6j) abc
	Value of x , y : 3 4

2.18 Python-Dictionary

The items are separated by commas, each key is separated from its value by a colon (:), and the entire structure is contained in curly brackets. A dictionary that is completely empty of all words is written as follows:.

Values may not be unique within a dictionary, but keys always are. A dictionary's keys must be immutable data types like texts, integers, or tuples, while its values can be of any kind.

Accessing Values in a Dictionary

You can make use of the well-known square brackets and the key to access dictionary items. Here is a straightforward illustration:

Python Expression	Result
dict = {'Name': 'Alisha', 'Age': 11, 'Class': 'Fifth'} print ("dict['Name']: ", dict['Name']) print ("dict['Age']: ", dict['Age'])	dict['Name']: Alisha dict['Age']: 11

We see the following error if we try to retrieve a data item using a key that is not listed in the dictionary:

Python Expression	Result

dict = {'Name': 'Alisha', 'Age': 11, 'Class': 'Fifth'}
print("dict['Aalya']: ", dict['Aalya'])
	Traceback (most recent call last):
	File "./prog.py", line 2, in <module>
	KeyError: 'Aalya'

Updating Dictionary

As seen in the straightforward example below, you can change a dictionary by adding a new entry, a key-value pair, editing an existing entry, or deleting an existing entry.

Python Expression	Result

dict = {'Name': 'Alisha', 'Age': 11, 'Class': 'Fifth'} dict['Age'] = 8; # update existing entry dict['Class'] = "Sixth"; # Add new entry print ("dict['Age']: ", dict['Age']) print ("dict['Class']: ", dict['Class'])	dict['Age']: 8 dict['Class']: Sixth

Delete Dictionary Elements

You can either wipe a dictionary's complete contents, or you can remove certain dictionary entries. Additionally, you have the option to erase the entire lexicon at once.

Use the del statement to expressly erase an entire dictionary. Here is a straightforward illustration:

Python Expression	Result
dict = {'Name': 'Alisha', 'Age': 11, 'Class': 'Fifth'} del dict['Name']; # remove entry with key 'Name' dict.clear(); # remove all entries in dict del dict ; # delete entire dictionary print ("dict['Age']: ", dict['Age']) print ("dict['School']: ", dict['School'])	Traceback (most recent call last): File ". /prog.py", line 6, in <module> TypeError: 'type' object is not subscriptable

Properties of Dictionary Keys

There are no limitations on dictionary values. They may be any Python object, whether they are built-in or user-defined. The same is not applicable to keys, though.

There are two crucial things to keep in mind with dictionary keys.

(a) It is not permitted to use a key more than once. This implies that no duplicate keys are permitted. When using duplicate keys, the most recent assignment is chosen. For instance,

Python Expression	Result
dict = {'Name': 'Alisha', 'Age': 11, 'Name': 'Aalya'} print ("dict['Name']: ", dict['Name'])	dict['Name']: Aalya

(b) Keys must not be changeable. In other words, you can use dictionary keys like ['key'] but not strings, numbers, or tuples. Here is a straightforward illustration:

Python Expression	Result

dict = {['Name']: 'Alisha', 'Age': 11,} print ("dict['Name']: ", dict['Name'])	Traceback (most recent call last): File "./prog.py", line 1, in <module> TypeError: unhashable type: 'list'

Built-in Dictionary Functions & Methods

The following dictionary functions are available in Python.

Sr.No.	Function with Description
1	cmp (dict1, dict2) Compares elements of both dict.
2	len(dict) Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
3	str(dict) Produces a printable string representation of a dictionary
4	type(variable) Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

The following dictionary methods are available in Python.

Sr.No.	Function with Description
1	dict.clear() Removes all elements of dictionary *dict*
2	dict.copy() Returns a shallow copy of dictionary *dict*
3	dict.fromkeys() Create a new dictionary with keys from seq and values *set* to *value.*
4	dict.get(key, default=None) For *key* key, returns value or default if key not in dictionary
5	dict.has_key(key) Returns *true* if key in dictionary *dict*, *false* otherwise
6	dict.items() Returns a list of *dict*'s (key, value) tuple pairs
7	dict.keys() Returns list of dictionary dict's keys
8	dict.setdefault(key, default=None) Similar to get(), but will set dict[key]=default if *key* is not already in dict

9	dict.update(dict2) Adds dictionary *dict2*'s key-values pairs to *dict*
10	dict.values() Returns list of dictionary *dict*'s values

Summary

- Python's strings, like those of many other widely used programming languages, are collections of bytes that represent unicode characters.
- Use the len() function to determine a string's length.
- The keyword in can be used to determine whether a specific word or character is present in a string.
- The keyword not in can be used to determine whether a specific word or character DOES NOT exist in a string.
- Use the + operator to concatenate, or merge, two strings.
- Similar to dynamically scaled arrays specified in other languages (such as vector in C++ and ArrayList in Java), Python lists are similar to them. A list is a group of items that are denoted by the symbol [] and separated by commas.
- Refer to the index number to access the list entries. To retrieve a specific item in a list, type [] in the index operator.
- In Python, positions from the array's end are represented by negative sequence indexes.
- Use of the built-in append() function allows for the addition of elements to the List. The append() method can only add one element at a time to the list; loops must be used to add many elements using the append() method.
- Python is a popular high-level, general-purpose programming language. Guido van Rossum invented it in 1991, and the Python Software Foundation continued to advance it
- Web applications can be developed on a server using Python.

Keywords

Token:The smallest discrete unit in a Python programme is called a token. Tokens are used to construct each statement and instruction in a programme.

Keywords:In a computer language, keywords are words that have a particular importance or meaning. They cannot be utilised for any arbitrary reason, including as names for functions or variables.

Identifiers:The names assigned to any variable, function, class, list, methods, etc. for identification are known as identifiers.

Reserved Words:You cannot use them as identifier names for constants, variables, or anything else.

Python Numbers: Number data types store numeric values.

Python Strings:Python defines strings as a contiguous group of characters that are enclosed in quotation marks.

Literals or Values:The fixed values or data items used in a source code are known as literals.

Python Lists:Of all the compound data types in Python, lists are the most flexible. Items in a list are delimited by square brackets and separated by commas ([]).

Python Tuples: Another sequence data type that resembles the list is the tuple. A tuple is made up of several values that are separated by commas.

Python Dictionaries: The dictionaries used by Python are something like hash tables. They consist of key-value pairs and operate similarly to associative arrays or hashes seen in Perl.

Dictionary: Key-value pairs are stored in dictionaries. To make the dictionary more efficient, Key-Value is offered.

Review Questions

Q1. What will be the output of above Python code?

```
str1="8/2"
print("str1")
print(str1)
```

A. str1
B. str1 8/2
C. str1 4.0
D. str1

Q2. Which statement is not correct?

A. Strings cannot be changed.
B. The capitalization() function in the string type returns a string after changing the entire input string to uppercase.
C. The lower() function in the string language is utilised to return a string by lowercasing the entire input string.
D. None of these.

Q3. Select the correct output of the following program.

```
str1="Good Morning"
print(str1[2:8])
```

A. od Mor
B. od Morn
C. odMor
D. oodMor

Q4. What would be outout of following program.

```
str1="This Is my File"
str2=str1.replace('i','I')
print(str2)
```

A. This Is my File
B. ThIsis my File
C. ThIs Is my FIle
D. ThIs Is my File

Q5. Select the correct output for the following code.

list1=['1','3','6','7']

str1="8"

for i in list1:

str1=str1+i

print(str1)

A. 81367
B. Error
C. 1367
D. 8 Error

Q6. What will following Python code return?

str1="This is my file"

print(len(str1))

A. 13
B. 14
C. 15
D. 16

Q7. What will following python code returns?

x = str(4)

y = int(4)

z = float(4)

print(x)

print(y)

print(z)

A. 4 4 4.0
B. 4.0 4 4
C. 4 4.0 4
D. None of above

Q8. What will following python code returns?

thistuple = tuple(("abc", "def", "ghi"))

print(len(thistuple))

A. 4
B. 5
C. 3
D. 9

Q9. What will following python code returns?

tuple3 = (True, False, False)

print(type(tuple3))

A. <class 'list'>
B. <class 'tuple'>
C. <class 'boolean'>
D. None of above

Q10. What will following python code returns?

thisset = {"apple", "banana", "cherry", "apple"}

print(thisset)

A. {'apple','banana','cherry','apple'}
B. {'banana','cherry','apple','apple'}
C. {'banana','apple'}
D. {'banana','cherry','apple'}

Q11. What will following python code returns?

thisdict = {

"Section": "D1601",

"Class": "BCA",

"year": 1964,

"year": 2020

}

print(thisdict)

A. {'Section': 'D1601', 'Class': 'BCA', 'year': 1964}
B. {'Section': 'D1601', 'Class': 'BCA', 'year': 2020, 'year':1964}
C. {'Section': 'D1601', 'Class': 'BCA', 'year': 2020,1964}
D. {'Section': 'D1601', 'Class': 'BCA', 'year': 2020}

Q12. What will following python code returns?

thisdict = {

"Name": "Aalya",

"Section": "D1601",

"Class": "BCA"

}

x = thisdict.get("Section")

print(x)

A. D1601
B. Error
C. String not allowed
D. Type incompatible

Q13What will following python code returns?

colors = ["red", "green", "burnt sienna", "blue"]

colors[2]

A. Green
B. Burnt sienna
C. Blue
D. Red

Q14: Which syntax use to delete dictonary.

A. del dict
B. Del dictionary
C. rmvdict
D. remove dictionary

Q15: Select right function of len()

A. compares element and then calculate length
B. gives total length of dictionary
C. Both a and b
D. None of above

Answers for Self Assessment

1.	B	2.	B	3.	A	4.	D	5.	A
6.	C	7.	A	8.	C	9.	B	10.	D
11.	D	12.	A	13.	B	14.	A	15.	B

FurtherReadings

- Mark Lutz,Programming *Python: Powerful Object-Oriented Programming*, OREILLY
- Wes McKinney, *Python for data analysis*, OREILLY
- David Ascher and Mark Lutz, *Learning Python*, OREILLY
- Eric Matthes, Python Crash Course, 2nd Edition: *A Hands-On, Project-Based Introduction to Programming*, Starch Pres

Web Links

https://www.tutorialspoint.com/python/index.htm

https://www.python.org/downloads/

https://www.w3schools.in/python/data-types

https://www.programiz.com/python-programming/online-compiler/

https://www.codecademy.com/catalog/language/python

Unit 03: OOP Concepts

CONTENTS

Objectives

After studying this unit, you will be able to:

- understand features of OOPs
- learn basic concepts about encapsulation
- learn inheritance and its types

Introduction

The Python programming style known as object-oriented programming (OOPs) makes use of objects and classes. It seeks to incorporate in programming real-world concepts like inheritance, polymorphism, encapsulation, etc. The fundamental idea behind OOPs is to unite the data and the functions that use it such that no other portion of the code may access it.

Object-Oriented Programming's Core Ideas (OOPs) are:-

- Class
- Object
- Method
- Inheritance
- Polymorphism
- Data Abstraction
- Encapsulation

3.1 Class

A class is a group of related items. The models or prototypes used to generate objects are included in classes. It is a logical entity with a few methods and characteristics.Consider the following scenario to better appreciate the need for generating classes: Suppose you needed to keep track of the number of dogs that might have various characteristics, such as breed or age. If a list is utilised, the dog's breed and age might be the first and second elements, respectively. What if there were 100 different breeds of dogs? How would you know which ingredient should go where? What if you wanted to give these dogs additional traits? This is disorganised and just what courses need.

A few notes on the Python class:

- The keyword class is used to create classes.
- The variables that make up a class are known as attributes.
- With the dot (.) operator, attributes can always be retrieved and are always public. For example: Myclass.Myattribute

Class Definition Syntax

```
Class Classname
{
#Statement-1
.
.
.
#Statement-N
}
```

Example*:* Making a Python class that is empty

```
Class Dog:
        Pass
```

Using the class keyword, we built a class with the name dog in the example above.

3.2 Objects

The object is an entity that is connected to a state and activity. Any physical device, such as a mouse, keyboard, chair, table, pen, etc., may be used. Arrays, dictionaries, strings, floating-point numbers, and even integers are all examples of objects. Any single string or integer, more specifically, is an object. A list is an object that may house other things, the number 12 is an object, the text "Hello, world" is an object, and so on. You may not even be aware of the fact that you have been using items.

An Object consists of:

State:The properties of an object serve as a representation of it. Additionally, it reflects an object's characteristics.

Behavior:It is represented via an object's methods. It also shows how one object reacts to other objects.

Identity: It gives a thing a special name and makes it possible for objects to communicate with one another.

Let's look at the example of the class dog to better understand the state, behaviour, and identity (explained above).

- The identity may be regarded as the dog's name.

- Breed, age, and colour of the dog are examples of states or attributes.
- You may infer from the behaviour whether the dog is eating or sleeping.

Creating an object as an example

Obj=Dog()

This will produce an object with the class Dog, named obj, as stated above. Let's first grasp the fundamental terms that will be utilised while working with objects and classes before delving further into them.

a. The self

- An additional initial parameter in the method declaration is required for class methods. When we call the method, we don't supply a value for this parameter; Python does.
- Even if we have a method that doesn't require any parameters, we still need one.
- This is comparable to this Java reference and this C++ pointer.

This is the sole purpose of the special self. When we invoke a method of this object as myobject.method(arg1, arg2), Python automatically converts it to MyClass.method(myobject, arg1, arg2).

b. The __init__method

The constructors in Java and C++ are comparable to the __init__ method. As soon as a class object is created, it is executed. Any initialization you want to perform on your object can be done with the method.Let's build some objects utilising the self and __init__ methods after defining a class.

Example1*:* Class and object creation using class and instance properties

```
class Dog:
        # class attribute
        attr1 = "mammal"
        # Instance attribute
        def __init__(self, name):
                self.name = name

# Driver code
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")

# Accessing class attributes
print("Rodger is a {}".format(Rodger.__class__.attr1))
print("Tommy is also a {}".format(Tommy.__class__.attr1))

# Accessing instance attributes
print("My name is {}".format(Rodger.name))
print("My name is {}".format(Tommy.name))
```

Output
Rodger is a mammal Tommy is also a mammal My name is Rodger My name is Tommy

Example2: Class and object creation with methods

```
class Dog:

        # class attribute
        attr1 = "mammal"

        # Instance attribute
        def __init__(self, name):
                self.name = name

        def speak(self):
                print("My name is {}".format(self.name))
# Driver code
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")

# Accessing class methods
Rodger.speak()
Tommy.speak()
```

Output
My name is Rodger My name is Tommy

3.3 Methods

A function connected to an object is the method. A method is not specific to class instances in Python. Any sort of object may have methods.

3.4 Inheritance

The capacity of one class to derive or inherit properties from another class is known as inheritance. The class from which the properties are being derived is referred to as the base class or parent class, and the class from which the properties are being derived is referred to as the derived class or child class. The advantages of inheritance include:

- It accurately depicts relationships in the real world.
- It offers a code's reusability. We don't need to keep writing the same code. Additionally, it enables us to expand a class's features without changing it.
- Because of its transitive nature, if a class B inherits from a class A, then all of class B's subclasses will also automatically inherit from class A.

Types of Inheritance

Single Inheritance

A class can inherit properties from a single-parent class using single-level inheritance.

Multilevel Inheritance

A derived class can inherit properties from an immediate parent class, which in turn can inherit properties from his parent class, thanks to multi-level inheritance.

Hierarchical Inheritance

More than one derived class can inherit properties from a parent class thanks to hierarchical level inheritance.

Multiple Inheritance

One derived class may inherit properties from several different base classes thanks to multiple level inheritance.

Example: Python inheritance

```
# Python code to demonstrate how parent constructors
# are called.


# parent class
class Person(object):
        # __init__ is known as the constructor
        def __init__(self, name, idnumber):
                self.name = name
                self.idnumber = idnumber


        def display(self):
                print(self.name)
                print(self.idnumber)


        def details(self):
                print("My name is {}".format(self.name))
                print("IdNumber: {}".format(self.idnumber))
# child class
class Employee(Person):
        def __init__(self, name, idnumber, salary, post):
                self.salary = salary
                self.post = post
        # invoking the __init__ of the parent class
        Person.__init__(self, name, idnumber)

    def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))
        print("Post: {}".format(self.post))
```

# creation of an object variable or an instance

```
a = Employee('Rahul', 886012, 200000, "Intern")
```

# calling a function of the class Person using

# its instance

```
a.display()
a.details()
```

Output

```
Rahul
886012
My name is Rahul
IdNumber: 886012
Post: Intern
```

In the aforementioned article, two classes—Person (parent class) and Employee—have been established (Child Class). The Person class is an ancestor of the Employee class. As can be seen in the show function in the code above, we may use the methods of the person class through the employee class. The details() function shows how a child class can alter the parent class's behaviour.

3.5 Polymorphism

Simply put, polymorphism means having multiple forms. For instance, utilising polymorphism, we can answer the question of whether the given species of birds fly or not using just one function. ***Example:***Python's use of polymorphism

```
class Bird:
    def intro(self):
        print("There are many types of birds.")
    def flight(self):
        print("Most of the birds can fly but some cannot.")
class sparrow(Bird):
    def flight(self):
        print("Sparrows can fly.")
class ostrich(Bird):
        def flight(self):
                print("Ostriches cannot fly.")
obj_bird = Bird()
obj_spr = sparrow()
obj_ost = ostrich()

obj_bird.intro()
obj_bird.flight()

obj_spr.intro()
obj_spr.flight()

obj_ost.intro()
obj_ost.flight()
```

OUTPUT

```
There are many types of birds.
Most of the birds can fly but some cannot.
There are many types of birds.
Sparrows can fly.
There are many types of birds.
Ostriches cannot fly.
```

3.6 Data Abstraction

Both data abstraction and encapsulation are frequently used interchangeably. Since data abstraction is accomplished by encapsulation, the two terms are almost synonymous.

When using abstraction, internal details are hidden and only functionalities are displayed. Giving things names that capture the essence of what a function or an entire programme does is the process of abstracting something.

3.7 Encapsulation

One of the core ideas in object-oriented programming is encapsulation (OOP). It explains the concept of data wrapping and the techniques that operate on data as a single unit. This restricts direct access to variables and procedures and can avoid data alteration by accident. A variable can only be altered by an object's method in order to prevent inadvertent modification. These variables fall under the category of private variables.

A class, which encapsulates all the data that is contained in its member functions, variables, etc., is an example of encapsulation.

Table 1 Encapsulation in Python

Methods	Variables

```
# Python program to
# demonstrate private members

# Creating a Base class
class Base:
        def __init__(self):
                self.a = "EcontentOnline"
                self.__c = "EcontentOnline"

# Creating a derived class
class Derived(Base):
        def __init__(self):

                # Calling constructor of
                # Base class
                Base.__init__(self)
                print("Calling private member of base class: ")
                print(self.__c)
# Driver code
obj1 = Base()
print(obj1.a)

# Uncommenting print(obj1.c) will
# raise an AttributeError

# Uncommenting obj2 = Derived() will
# also raise an AtrributeError as
# private member of base class
# is called inside derived class
```

Output
EcontentOnline

The c variable was generated as the private attribute in the example above. We are unable to even directly read or modify the value of this attribute.

Difference between Object-Oriented vs. Procedure-Oriented Programming Languages.

Object-oriented Programming	Procedural Programming
The approach to addressing problems that uses objects for computation is called object-oriented programming.	A list of instructions is used in procedural programming to perform calculations in stages.
It makes development and upkeep simpler.	When a project grows in scope, maintaining the codes is difficult in procedural programming.
It replicates the thing in the actual world. Therefore, oops makes it simple to tackle difficulties in the actual world.	It doesn't represent reality in any way. It operates using detailed instructions broken down into smaller units called functions.
It offers data concealment. Consequently, it is safer than procedural languages. Private information is not accessible from anyplace.	Because procedural languages don't offer a suitable method for data binding, they are less secure.
C++, Java, .Net, Python, C#, and other object-oriented programming languages are examples.	Procedural languages include C, Fortran, Pascal, VB, and others.

Summary

- The Python programming style known as object-oriented programming (OOPs) makes use of objects and classes.
- A class is a group of related items. The models or prototypes used to generate objects are included in classes.
- The object is an entity that is connected to a state and activity. Any physical device, such as a mouse, keyboard, chair, table, pen, etc., may be used.
- The constructors in Java and C++ are comparable to the __init__ method. As soon as a class object is created, it is executed.
- A function connected to an object is the method. A method is not specific to class instances in Python. Any sort of object may have methods.
- The capacity of one class to derive or inherit properties from another class is known as inheritance.
- Simply put, polymorphism means having multiple forms. For instance, utilising polymorphism, we can answer the question of whether the given species of birds fly or not using just one function
- Both data abstraction and encapsulation are frequently used interchangeably. Since data abstraction is accomplished by encapsulation, the two terms are almost synonymous
- One of the core ideas in object-oriented programming is encapsulation (OOP). It explains the concept of data wrapping and the techniques that operate on data as a single unit.
- The approach to addressing problems that uses objects for computation is called object-oriented programming.
- A list of instructions is used in procedural programming to perform calculations in stages

Keywords

OOPS: Object-oriented programming is known as OOP. While object-oriented programming involves constructing objects that include both data and methods, procedural programming involves developing procedures or methods that perform actions on the data.

Class: Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods. A Class is like an object constructor, or a "blueprint" for creating objects

The __init__method: All classes have a function called __init__(), which is always executed when the class is being initiated.

The __str__function: What should be returned when the class object is rendered as a string is determined by the __str__() function.

Objects methods: Methods can also be found in objects. Object-specific functions are called methods in an object.

Self-Parameter: To access class-specific variables, use the self parameter, which is a reference to the currently running instance of the class.

Del:Using the del keyword, properties on objects can be deleted.

Pass statement: Although class definitions cannot be empty, if for some reason you have one that is empty, add the pass statement to prevent an error.

Inheritance: By using inheritance, we may create a class that has all the methods and attributes of another class.

Parent class:The class being inherited from, often known as the base class, is the parent class.

Child class:The class that inherits from another class is referred to as a child class or derived class.

Super Function:The super() function in Python allows a descendant class to inherit all of its parent's methods and properties.

Self Assessment

Q1. Which option best encapsulates inheritance?

A. Ability of a class to include methods from other classes in its definition
B. Techniques for grouping instance variables and methods to limit access to certain class members
C. A focus on variables and passing variables to functions
D. Enables the use of sophisticated software that is well-designed and flexible.

Q2. Which of the following claims about inheritance is false?

A. A class's protected members may be inherited.
B. The class that inherits is known as a subclass.
C. A class's private members can be accessed and inherited.
D. One characteristic of OOP is inheritance

.

Q3. What line of code should you write to activate the __init__ method in A from B if B is a subclass of A?

A. A.__init__(self)
B. B.__init__(self)
C. A.__init__(B)
D. B.__init__(A)

Q4: What function type is a built-in in the context of classes?

A. Identifies the name of any value's object.
B. Identifies any value's class name.
C. Determines a value's class description
D. Identifies any value's file name

Q5: What one of the following is not an inheritance type?

A. Double-level
B. Multi-level
C. Single-level
D. Multiple

Q6: Which of these is not one of OOP's core characteristics?

A. Encapsulation
B. Inheritance
C. Instantiation
D. Polymorphism

Q7: Which of the following definitions best describes encapsulation?

A. The capacity of a class to derive individuals from other classes as part of its own definition.
B. Techniques for combining instance variables and methods to limit access to specific class members
C. focuses on supplying parameters to functions and variables.
D. enables the use of sophisticated software that is well-designed and flexible.

Q8: Define Overriding.

A. Overriding can occur in the case of inheritance in class
B. It is a process of redefining inherited method in child class.
C. It is a magic method in python.
D. None of these

Q9: _____________ developed python language

A. Albert Einstein
B. Guido Van Rossum
C. Guido Evan
D. None of these

Q10: What year was the Python programming language created?

A. 1975
B. 1989
C. 1972
D. 1990

Q11: Which of the following commands the expression with the most precedence?

A. Addition
B. Subtraction
C. Parentheses
D. Power

Q12: Of the following, which best describes abstraction?

A. Hiding the execution
B. displaying crucial information
C. Hiding the important data
D. Hiding the implementation and showing only the features

Q13: A class is an ____________ abstraction.

A. Object
B. Logical
C. Real
D. Hypothetical

Q14: Abstraction can be used for _______________.

A. Control and data.
B. Only data
C. Only control
D. Classes

Q15: Which of the following can be considered a combination of data abstraction and programming?

A. Class
B. Object
C. Inheritance
D. Interfaces

Answer for Self Assessment

l.	A	2.	C	3.	A	4.	B	5.	A
6.	C	7.	B	8.	B	9.	B	10.	B
11.	C	12.	D	13.	B	14.	A	15.	B

Review Questions

1. What do you understand by OOPS? Write down the code to make a python class that si empty.
2. Define Objects. Write down example to create an object with methods.
3. What do you understand by inheritance and also define types of inheritance.
4. Write down difference between Single level inheritance, multilevel inheritance and multiple inheritance.
5. Define Polymorphism. Write down python code that define use of polymorphism.
6. What do you understand by Encapsulation? Write down python program to demonstrate private members.
7. Write down difference between object-oriented programming and procedural programming.

FurtherReadings

- Mark Lutz,Programming *Python: Powerful Object-Oriented Programming*, OREILLY
- Wes McKinney, *Python for data analysis*, OREILLY
- David Ascher and Mark Lutz, *Learning Python*, OREILLY
- Eric Matthes, Python Crash Course, 2nd Edition: *A Hands-On, Project-Based Introduction to Programming*, Starch Pres

Web Links

https://www.tutorialspoint.com/python/index.htm

https://www.python.org/downloads/

https://www.w3schools.in/python/data-types

https://www.programiz.com/python-programming/online-compiler/

https://www.codecademy.com/catalog/language/python

Unit 04: More on OOP Concepts

CONTENTS

Objectives

After this unit, student would be able to:

- learn various examples to elaborate upon the concepts of Python function overloading.
- understand operator overloading
- learn method overriding

Introduction

Function overloading is a phenomenon that occurs when several functions with the same name have different numbers of parameters. In contrast to other languages, Python does not provide function overloading, and the functional parameters lack a data type. Let's say we wish to leverage the functional overloading functionality. In that scenario, we can change the method's default values for arguments to None, which won't result in an error if that particular value isn't supplied as an argument when the function is called.

4.1 What is function overloading?

Function overloading, as the name suggests, is the practise of using the same function numerous times with various numbers of arguments. However, overloading of functions is not supported in Python. If we implement the function overloading code like we do in other languages, an error is thrown. Python lacks a data type for method parameters, which is the cause.

The new function with the same name replaces the prior function with the same name (but different parameters). Therefore, we now see an error if we attempt to call the original function, which had a different number of parameters, while passing a different number of arguments that are defined in the second function. Let's attempt to comprehend the same.

Table 1 Function Overloading

Program	Output

class sumClass:
def sum(self, a, b):
print("First method:",a+b)
def sum(self, a, b, c):
print("Second method:", a + b + c)
 	
obj=sumClass()
obj.sum(19, 8, 77) #correct output
obj.sum(18, 20) #throws error
	Second method: 104
	Traceback (most recent call last):
	File "<string>", line 9, in <module>
	TypeError: sum() missing 1 required positional argument: 'c'

We can see from the programme above that the second sum technique takes precedence over the first sum method. The function returns the output when we call it with three arguments, but an error is returned when we call it with with two arguments. Thus, function overloading is not supported by Python.

But does that imply there isn't another way to put this feature into practise? No, is the response. Function overloading in Python can be implemented in a variety of ways.

This can be done by declaring one or more parameters in the function declaration as None. In order to prevent an error from happening when calling a function that has a parameter set to None but no argument provided, we will also include a checking condition for None in the function body.

Syntax

When we define a function, we set one or more parameters to None and include a checking condition for None in the function body. This way, when we call the function, an error won't happen even if we don't pass the argument for a parameter that we have set to None.

We choose to invoke the function with or without the parameter by setting the parameters to None.

SYNTAX:

```
class name:
    def fun(self,p1=None,p2=None,...)
```

Table 2 Example of Function Overloading

Program	Output

class sumClass:
    def sum(self, a = None, b = None, c = None):
        if a != None and b == None or c == None:
            print("Provide more numbers") #if there is only 1 number as input
        else:
            print("First method:", a + b + c) #for calculating the sum
 	
obj=sumClass()
	First method: 104
	Provide more numbers

obj.sum(19, 8, 77)#104	
obj.sum(18)#Provide more numbers	

Here, we can see that function overloading may be implemented by changing the parameter default values to None and by including a few validations.

How function overloading works in Python?

In Python, the most recent definition of a function is taken into account for determining its validity. Setting the function's default settings for its parameters to None will allow us to apply the idea of function overloading. We have the choice of calling the function with or without the argument by setting the value of any functional parameters to None. Therefore, it won't throw an error if we don't include the parameter set as None.

Overloading Built-in Functions

We have a few special functions in the Python Data Model, and it gives us the ability to overload the built-in functions. The special function names all start with double underscores ( ).

By adding the special function to the len() method in our example, we will alter its standard behaviour. As a result, the interpreter calls the special function instead of the built-in function when a built-in function is declared as a special function inside a class. Let's look at an illustration utilising a unique Python function.

Program	Output
class items:	The total items are:
def __init__(self, cart):	4
self.cart= list(cart)	
#special function	
def __len__(self):	
print("The total items are:")	
return len(self.cart)#built-in function	
purchase = items(['apple', 'banana', 'mango','grapes'])	
print(len(purchase))#prints the body of the special function	

The __init__ method is invoked whenever an object derived from a class is created. We are attempting to alter Python's len() function's default behaviour, which merely shows the object's length. The specific definition we have created for the __len__() function will retrieve the desired results whenever we provide an object of our class to len().

We have inserted the desired code to our custom definition for __len__. Len() is overloaded by this.

Example:

Let's create some code in Python that calculates the area of figures using function overloading (triangle, rectangle, square). We will call the same function with different parameters while setting the default values of the parameters to None.

Program	Output
class areaClass:	Area of the triangle 76.0
def area(self,a,b=None,c=None,d=None):	Area of the square 324

`#when a and c are passed as arguments`
`if a!=None and b!=None and a!=b and a!=c:`
`print("Area of the triangle",(0.5*a*b))`
 	
`#when a,b,c and d are passed as arguments`
`elif(b!=None and c!=None and d!=None and a==b and a==c):`
`print("Area of the square",(a*c))`
 	
`elif(b==None and c==None and d==None):`
`print("Enter more numbers")`
`else:`
`if(a==c):`
`print("Area of the rectangle",(a*b))`
`else:`
`print("Area of the rectangle",(a*c))`
 	
`obj=areaClass()`
`obj.area(19,8,77)#Area of the triangle 76.0`
`obj.area(18,18,18,18)#Area of the square 324`
`obj.area(72,38,72,38)#Area of the rectangle 2736`
	Area of the rectangle 2736

Introduction

Depending on the operands used, an operator's meaning can change in Python.

4.2 Python Operator Overloading

Built-in classes are supported by Python operators. However, the same operator responds differently to several types. For instance, the + operator will combine two lists, concatenate two strings, or perform arithmetic addition on two numbers.

Operator overloading is a Python feature that enables the same operator to have several meanings depending on the context.

What transpires then when we utilize them with objects declared by a user? Let's have a look at the class below, which aims to imitate a point in a 2-D coordinate system.

Table 3 Python Operator Overloading

Program	Output

class Point:
def __init__(self, x=0, y=0):
self.x = x
	Traceback (most recent call last):
	File "<string>", line 9, in <module>
	print(p1+p2)
	TypeError: unsupported operand type(s) for +:

self.y = y **p1 = Point(1, 2)** **p2 = Point(2, 3)** **print(p1+p2)**	'Point' and 'Point'

Here, we can see that a TypeError was thrown because Python was unable to combine two Point objects.

However, using operator overloading in Python, we can complete this work. However, let's first gain some understanding of special functions.

Python Special Functions

Python refers to class functions that start with a double underscore as special functions.

These are not the usual class-defined functions that we define. One of these is the __init__() function that we previously defined. Every time we create a new object of that class, it is called.

Other unique functions are abundant in Python.We can integrate our class with built-in functions by using custom functions.

```
>>> p1 = Point(2,3)
>>> print(p1)
<__main__.Point object at 0x00000000031F8CC0>
```

Let's say that instead of printing what we got, we want the print() function to print the coordinates of the Point object. In our class, we may define a __str__() method that regulates how the item is printed. Let's examine how we can accomplish this:

```
class Point:
    def __init__(self, x = 0, y = 0):
self.x = x
self.y = y

    def __str__(self):
        return "({0},{1})".format(self.x,self.y)
```

Let's try the print() function once more right now.

Program	Output

class Point:
    def __init__(self, x=0, y=0):
self.x = x
self.y = y
 	
    def __str__(self):
        return "({0}, {1})".format(self.x, self.y)
	(2, 3)

p1 = Point(2, 3) **print(p1)**	

Better still. It turns out that when we utilise the built-in functions format () or str(), this procedure is also called ().

```
>>> str(p1)
'(2,3)'

>>> format(p1)
'(2,3)'
```

So, when you use str(p1) or format(p1), Python internally calls the p1. __str__ () method.

Overloading the + Operator

We will need to include the __add__() function in the class in order to overload the + operator. Great power entails enormous responsibility. Within this function, we are free to perform whatever we choose. But returning a Point object of the coordinate sum makes more sense.

Program	Output

```
class Point:
  def __init__(self, x=0, y=0):
self.x = x
self.y = y

  def __str__(self):
    return "({0},{1})".format(self.x, self.y)

  def __add__(self, other):
    x = self.x + other.x
    y = self.y + other.y
    return Point(x, y)


p1 = Point(1, 2)
p2 = Point(2, 3)

print(p1+p2)
```
	(3, 5)

In reality, when you use p1 + p2, Python actually calls p1. Point is obtained by adding __add (p2) (p1,p2). The addition operation is then completed in the manner that we instructed.

We can also overload other operators in a similar manner. Below is a summary of the special function that needs to be implemented.

Table 4 Overload Operator

Operator	Expression	Internally
Addition	p1 + p2	p1.__add__(p2)
Subtraction	p1 - p2	p1.__sub__(p2)
Multiplication	p1 * p2	p1.__mul__(p2)
Power	p1 ** p2	p1.__pow__(p2)
Division	p1 / p2	p1.__truediv__(p2)
Floor Division	p1 // p2	p1.__floordiv__(p2)
Remainder (modulo)	p1 % p2	p1.__mod__(p2)
Bitwise Left Shift	p1 << p2	p1.__lshift__(p2)
Bitwise Right Shift	p1 >> p2	p1.__rshift__(p2)
Bitwise AND	p1 & p2	p1.__and__(p2)
Bitwise OR	p1 | p2	p1.__or__(p2)
Bitwise XOR	p1 ^ p2	p1.__xor__(p2)
Bitwise NOT	~p1	p1.__invert__()

Overloading Comparison Operator

Operator overloading is not restricted to arithmetic operators in Python. Additionally, we can overload comparison operators.

Let's say we wanted to add the less than symbol to the Point class.

To achieve this, let's compare the distances between these places and the origin and then output the result. It may be carried out as follows.

Program

```
# overloading the less than operator
class Point:
  def __init__(self, x=0, y=0):
self.x = x
self.y = y

  def __str__(self):
    return "({0},{1})".format(self.x, self.y)

  def __lt__(self, other):
self_mag = (self.x ** 2) + (self.y ** 2)
```

Output

```
True
False
False
```

```
other_mag = (other.x ** 2) + (other.y ** 2)
        return self_mag<other_mag

p1 = Point(1,1)
p2 = Point(-2,-3)
p3 = Point(1,-1)

# use less than
print(p1<p2)
print(p2<p3)
print(p1<p3)
```

The special functions that must be implemented in order to overload other comparison operators are listed below.

Table 5 Overload Comparison Operator

Operator	Expression	Internally
Less than	p1 < p2	p1.__lt__(p2)
Less than or equal to	p1 <= p2	p1.__le__(p2)
Equal to	p1 == p2	p1.__eq__(p2)
Not equal to	p1 != p2	p1.__ne__(p2)
Greater than	p1 > p2	p1.__gt__(p2)
Greater than or equal to	p1 >= p2	p1.__ge__(p2)

4.3 Method Overriding in Python

Any object-oriented programming language has the capability of allowing a subclass or child class to offer a customised implementation of a method that is already supplied by one of its super-classes or parent classes. This capability is known as method overriding. The term "override" refers to a method in a subclass that replaces a method in a superclass when both methods share the same name, same parameters, same signature, and same return type (or sub-type).

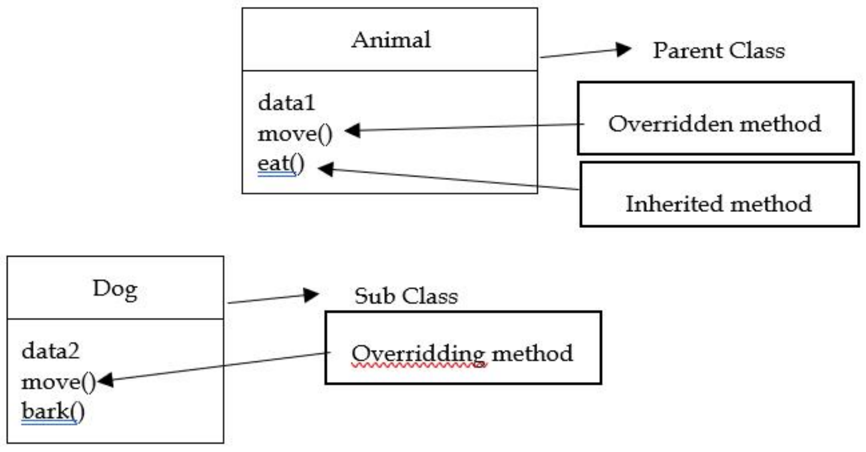


Figure 1 Method Overriding

The object that calls a method will determine which version of the method is executed. When a method is called from an object of a parent class, the method's parent class version is executed; however, when a method is called from an object of a subclass, the child class version is executed. In other words, the version of an overridden method that is run depends on the type of the object being referenced, not the type of the reference variable.

Program	Output

```
# Defining parent class
class Parent():

    # Constructor
    def __init__(self):
        self.value = "Inside Parent"

    # Parent's show method
    def show(self):
        print(self.value)

# Defining child class
class Child(Parent):

    # Constructor
    def __init__(self):
        self.value = "Inside Child"

    # Child's show method
    def show(self):
        print(self.value)


# Driver's code
```
	Inside Parent
	Inside Child

obj1 = Parent()
obj2 = Child()
 	
obj1.show()
obj2.show()	

Method overriding with multiple and multilevel inheritance

Multiple Inheritance: Many inheritance is the term used when a class derives from multiple base classes.

Example: Let's look at a scenario where we only wish to override methods from one parent class. The implementation is shown below.

Program	Output

```
# Python program to demonstrate
# overriding in multiple inheritance


# Defining parent class 1
class Parent1():

        # Parent's show method
        def show(self):
                print("Inside Parent1")

# Defining Parent class 2
class Parent2():

        # Parent's show method
        def display(self):
                print("Inside Parent2")

# Defining child class
class Child(Parent1, Parent2):

        # Child's show method
        def show(self):
                print("Inside Child")

# Driver's code
obj = Child()
```
	```
	Inside Child

	Inside Parent2
	```

obj.show()
obj.display()	

Summary

- Function overloading is a phenomenon that occurs when several functions with the same name have different numbers of parameters
- In Python, the most recent definition of a function is taken into account for determining its validity.
- Built-in classes are supported by Python operators. However, the same operator responds differently to several types.
- Python refers to class functions that start with a double underscore as special functions.
- Operator overloading is not restricted to arithmetic operators in Python. Additionally, we can overload comparison operators.
- Any object-oriented programming language has the capability of allowing a subclass or child class to offer a customised implementation of a method that is already supplied by one of its super-classes or parent classes.
- Many inheritances are the term used when a class derives from multiple base classes.

Keywords

Operator Overloading: When an operator in Python is overloaded, it signifies that it has additional meaning in addition to its usual operational meaning.

Multiple Inheritance: Many inheritance is the term used when a class derives from multiple base classes.

Multilevel Inheritance: When we have a child and grandchild relationship.

Function Overloading: When many functions share the same name but have different numbers of parameters, this is known as function overloading.

Built-in functions: We have a few special functions in the Python Data Model, and it gives us the ability to overload the built-in functions. The special function names all start with double underscores ( ).

Functions:A function is a segment of clean, reusable code that executes a single, connected operation. A higher level of code reuse and improved application modularity are provided via functions.

Operator Overloading: Built-in classes are supported by Python operators. However, the same operator responds differently to several types. For instance, the + operator will combine two lists, concatenate two strings, or perform arithmetic addition on two numbers.

Method Overriding: Any object-oriented programming language that supports method overriding enables a subclass or child class to provide a particular implementation of a method that is already supplied by one of its super-classes or parent classes.

Self Assessment

Q1. Which function overload the + operator?

A. __add__()
B. __plus__()
C. __sum__()

D. None of the above

Q2:Which function overload the == operator?

A. __eq__()
B. __equ__()
C. __equal__()
D. None of the above

Q3: Which function overloads the >> operator?

A. __more__()
B. __gt__()
C. __ge__()
D. None of the above

Q4: Which operator is overloaded by the __or__() function?

A. ||
B. |
C. //
D. /

Q5: Which function overloads the // operator?

A. __div__()
B. __ceildiv__()
C. __floordiv__()
D. __truediv__()

Q6 : Select the valid ways of overloading the operators.

A. Using friend function
B. Using member function
C. Either member function or friend function can be used
D. Operators can't be overloaded

Q7: Overloading a subprogram allows subprogram to

A. Operate on objects of different types
B. Operate on objects of same name
C. Operate on objects of different name
D. Operate on objects of same types

Q8: What is necessary condition to overload parameters type of a subprogram.

A. The base type of two parameter must differ.
B. The parameter can't of integer type
C. The parameter must have a different name

D. The base type of two parameter must be same.

Q9: By overloading + operator, it is possible to ________

A. Use binary addition
B. Use arithmetic addition
C. Use it a subtract operator
D. None of the above

Q10: What is correct about overloading + and – operators.

A. They can be defined as binary operators only.
B. They can be defined as unary operators only.
C. They can define as ternary operators only
D. They can be defined as either binary or unary operators.

Q11: Which of the subsequent claims about variable names in Python is true?

A. All variable names must begin with an underscore.
B. Unlimited length
C. The variable name length is a maximum of 2.
D. All of the above

Q12: Which of the following commands the expression with the most precedence?

A. Division
B. Subtraction
C. Power
D. Parentheses

Q13: Which of the following option is not a core data type in the python language?

A. Dictionary
B. Lists
C. Class
D. All of the above

Q14: How to set up Numpy on a computer.

A. install numpy
B. pip install python numpy
C. pip install numpy
D. pip install numpypython

Q15: Which option from the list below best demonstrates how to import the Numpy module into your programme?

A. import numpy
B. import numpy as np

C. from numpy import *
D. All of the above

Answers for Self Assessment

1.	A	2.	A	3.	D	4.	B	5.	C
6.	C	7.	A	8.	A	9.	A	10.	D
11.	B	12.	D	13.	C	14.	C	15.	D

Review Question

1. What is function overloading. Explain with example.
2. Explain operator overloading with example.
3. What do you understand by method overriding. Explain with example.
4. Explain function overloading with example.
5. What do you understand by type conversion and also explain difference between overloaded functions and overridden functions?

FurtherReadings

- Mark Lutz,Programming *Python: Powerful Object-Oriented Programming*, OREILLY
- Wes McKinney, *Python for data analysis*, OREILLY
- David Ascher and Mark Lutz, *Learning Python*, OREILLY
- Eric Matthes, Python Crash Course, 2nd Edition: *A Hands-On, Project-Based Introduction to Programming*, Starch Pres

Web Links

https://www.tutorialspoint.com/python/index.htm

https://www.python.org/downloads/

https://www.w3schools.in/python/data-types

https://www.programiz.com/python-programming/online-compiler/

https://www.codecademy.com/catalog/language/python

Unit 05: Exception Handling

CONTENTS

Objectives

After this unit, student would be able to:

- Learn how to handle exceptions in your program using try, except and finally statements with the help of example.

Introduction

In Python, there are two different sorts of errors: syntax errors and exceptions. Errors are issues in a programme that cause it to halt during execution. On the other hand, exceptions are raised when internal events take place that alter the program's usual course.

5.1 Exceptions in Python

Many built-in exceptions in Python are thrown when a problem is encountered by your programme (something in the programme goes wrong).

The Python interpreter stops the running process and passes control to the calling process until these exceptions are addressed. The software will crash if it is not addressed.

Let's take a software where function A calls function B, which then calls function C as an example. An exception that occurs in function C but isn't handled there is passed to B before being handled by A.

If the mistake is never addressed, a notification is shown and our software abruptly and unexpectedly stops.

Catching Exceptions in Python

A try statement in Python can be used to manage exceptions.

The try clause contains the crucial operation that can cause an exception. The except clause contains the code that manages exceptions.

Thus, once we have identified the exception, we may decide what actions to take. Here is an easy illustration.

Program	Output
# import module sys to get the type of	The entry is a

exception	Oops! <class 'ValueError'> occurred.
import sys	Next entry.
randomList = ['a', 0, 2]	
for entry in randomList:	The entry is 0
try:	Oops! <class 'ZeroDivisionError'>occured.
print("The entry is", entry)	Next entry.
r = 1/int(entry)	
break	The entry is 2
except:	The reciprocal of 2 is 0.5
print("Oops!", sys.exc_info()[0], "occurred.")	
print("Next entry.")	
print()	
print("The reciprocal of", entry, "is", r)	

We cycle through the values in the random Listlist in this application. The try block contains the code that, as was discussed earlier, can lead to an exception.

The unless block is bypassed and normal flow resumes if there is no exception (for last value). But the unless block handles any exceptions that do arise (first and second values).

Here, we use the excinfo () method in the sys module to print the name of the exception. As we can see, a results in ValueError whereas 0 results in ZeroDivisionError.

We may alternatively complete the aforementioned work in the following manner because every exception in Python inherits from the base Exception class:

Program

# import module sys to get the type of exception 	
import sys
randomList = ['a', 0, 2]
for entry in randomList:
  try:
print("The entry is", entry)
    r = 1/int(entry)
    break
  except Exception as e:
print("Oops!", e.__class__, "occurred.")
print("Next entry.")
print()
print("The reciprocal of", entry, "is", r)

Catching Specific Exceptions in Python

In the example given above, the except clause did not contain any specific exceptions.

This is not a suitable programming technique because it will catch every exception and treat each situation uniformly. Which exceptions an except clause should cover can be specified.

A try clause may contain any number of except clauses to address various exceptions, but only one will be carried out in the event of an exception.

Multiple exceptions can be specified in an except clause using a tuple of values. Here is an illustration of pseudocode.

Program

```
try:
   # do something
   pass
except ValueError:
   # handle ValueError exception
   pass
except (TypeError, ZeroDivisionError):
   # handle multiple exceptions
   # TypeError and ZeroDivisionError
   pass
except:
   # handle all other exceptions
   pass
```

Raising Exceptions in Python

Exceptions are produced in Python programming when runtime issues happen. Using the raise keyword, we can manually raise exceptions as well.

If we want to know why an exception was raised, we can optionally give values to the exception.

Program

```
>>> raise KeyboardInterrupt
Traceback (most recent call last):
...
KeyboardInterrupt
>>> raise MemoryError("This is an argument")
Traceback (most recent call last):
...
MemoryError: This is an argument
>>> try:
...     a = int(input("Enter a positive integer: "))
...     if a <= 0:
...         raise ValueError("That is not a positive number!")
... except ValueError as ve:
...     print(ve)
...
Enter a positive integer: -2
That is not a positive number!
```

Python try with else clause

If the code block inside attempt executed without any issues, you could occasionally want to run another section of code. You can pair the try statement with the optional else keyword in certain situations.

Note: The previous except clauses do not apply to exceptions in the else clause.

x	Output
# Program to print the reciprocal of even numbers	Enter a number: 1
	Not an even number!
try:	
num = int (input ("Enter a number: "))	
assert num % 2 == 0	
except:	
print("Not an even number!")	
else:	
reciprocal = 1/num	
print(reciprocal)	

If an even number is passed, the reciprocal is calculated and shown.

```
Enter a number: 4
0.25
```

If we do so, we receive a ZeroDivisionError since the previous except does not handle the code block inside else.

```
Enter a number: 0
Traceback (most recent call last):
File "<string>", line 7, in <module>
reciprocal = 1/num
ZeroDivisionError: division by zero
```

Python try...finally

In Python, a finally clause is an optional addition to the try statement. This phrase, which is typically used to release external resources, is always put into effect.

For instance, we might be utilizing a file or a Graphical User Interface while connecting via the network to a distant data center (GUI).

In any of these scenarios, whether the programme ran successfully or not, we must clear away the resource before it terminates. The finally clause carries out these operations (closing a file, shutting a GUI, or disconnecting from the network) to ensure execution.

Here is a file operation example to demonstrate this.

```
try:
    f = open("test.txt",encoding = 'utf-8')
    # perform file operations
finally:
f.close()
```

If an exception arises while the programme is running, this kind of construct ensures that the file is closed.

5.2 Python Custom Exceptions

Python comes with a variety of built-in exceptions that require your programme to print an error when something goes wrong.

But occasionally you might need to make your own special exclusions that are tailored to your needs.

Creating Custom Exceptions

Users in Python can define unique exceptions by developing new classes. This exception class must directly or indirectly derive from the default Exception class. This class is also where the majority of the built-in exceptions stem from.

```
>>> class CustomError(Exception):
...     pass
...


>>> raise CustomError
Traceback (most recent call last):
...
__main__.CustomError
>>> raise CustomError("An error occurred")
Traceback (most recent call last):
...
__main__.CustomError: An error occurred
```

This exception is user-defined and derives from the Exception class. Its name is CustomError. The raise statement with an optional error message can be used to raise this new exception as well as other exceptions.

It is best practice to save every user-defined exceptions that our application raises in a separate file when we are creating a large Python programme. Many common modules carry out this. As errors.py or exceptions.py, respectively, they declare their exceptions (generally but not always).

Although user-defined exception classes can implement anything a regular class can, we often keep them short and sweet. The majority of implementations declare a unique base class from which they derive all additional exception classes. This idea is shown in the example that follows.

Example: User-Defined Exception in Python

We'll show how user-defined exceptions may be used in a programme to raise and catch problems in this example.

Until the user correctly guesses a stored number, this programme will ask them to enter a number. If their guess is larger than or less than the stored amount, a clue is given to assist them in figuring it out.

```
# define Python user-defined exceptions
class Error(Exception):
   """Base class for other exceptions"""
   pass

class ValueTooSmallError(Error):
```

```
Enter a number: 12
This value is too large, try again!

Enter a number: 0
This value is too small, try again!
```

"""Raised when the input value is too small"""
pass
 	
class ValueTooLargeError(Error):
"""Raised when the input value is too large"""
pass
 	
# you need to guess this number 	
number = 10
 	
# user guesses a number until he/she gets it right 	
while True:
try:
i_num = int(input("Enter a number: "))
if i_num< number:
raise ValueTooSmallError
elifi_num> number:
raise ValueTooLargeError
break
except ValueTooSmallError:
print("This value is too small, try again!")
print()
except ValueTooLargeError:
print("This value is too large, try again!")
print()
 	
print("Congratulations! You guessed it correctly.")
	Enter a number: 8
	This value is too small, try again!
	
	Enter a number: 10
	Congratulations! You guessed it correctly.

Error is a base class that we've defined.

The other two exceptions that are raised by our software, ValueTooSmallError and ValueTooLargeError, are descended from this class. In Python programming, this is the typical technique to define user-defined exceptions, but there are other options as well.

Customizing Exception Classes

This class can be further modified to take other arguments as needed.

You must be familiar with the fundamentals of Object-Oriented programming in order to learn how to customize the Exception classes.

Program	Output
class SalaryNotInRangeError(Exception): """Exception raised for errors in the input salary.	Enter salary amount: 2000 Traceback (most recent call last):

Attributes:
salary -- input salary which caused the error
message -- explanation of the error
"""
def __init__(self, salary, message="Salary is not in (5000, 15000) range"):
self.salary = salary
self.message = message
super().__init__(self.message)
salary = int(input("Enter salary amount: "))
if not 5000 < salary < 15000:
raise SalaryNotInRangeError(salary)
	File "<string>", line 17, in <module>
	raise SalaryNotInRangeError(salary)
	__main__.SalaryNotInRangeError: Salary is not in (5000, 15000) range

Here, we have modified the Exception class's function Object () {[native code]} to accept the custom arguments message and salary. Super is then used to manually invoke the function Object () {[native code]} of the parent Exception class with the self. Message argument ().

It is defined to use the custom self. Salary attribute in the future.

When SalaryNot in RangeError is raised, the appropriate message is then shown using the inherited __str__ function of the Exception class.

By replacing it, we may also alter the __str__ method itself.

Program	Output

class SalaryNotInRangeError(Exception):
"""Exception raised for errors in the input salary.
Attributes:
salary -- input salary which caused the error
message -- explanation of the error
"""
def __init__(self, salary, message="Salary is not in (5000, 15000) range"):
self.salary = salary
self.message = message
super().__init__(self.message)
def __str__(self):
return f'{self.salary} -> {self.message}'
salary = int(input("Enter salary amount: "))
if not 5000 < salary < 15000:
raise SalaryNotInRangeError(salary)
	Enter salary amount: 2000
	Traceback (most recent call last):
	File "/home/bsoyuj/Desktop/Untitled-1.py", line 20, in <module>
	raise SalaryNotInRangeError(salary)
	__main__.SalaryNotInRangeError: 2000 -> Salary is not in (5000, 15000) range

Summary

- When you have finished testing the software, you can turn an assertion on or off as a sanity check.
- An event that occurs during the execution of a programme and obstructs the regular flow of the program's instructions is an exception.
- Python uses try and except statements to catch and deal with exceptions.
- Using the raise statement, you can raise exceptions in a number of different ways.
- By deriving classes from the typical built-in exceptions, Python also lets you design your own exceptions.
- To provide handlers for various exceptions, a try statement may contain more than one except clause.
- The finally keyword in Python is always invoked following the try and except sections.
- The raise statement enables the programmer to compel the occurrence of a particular exception. raise's lone argument identifies the exception that should be raised.
- When your programme encounters a problem, Python automatically throws a number of built-in exceptions (something in the programme goes wrong).
- The finally keyword is available in Python, and it is always used after the try and except blocks.

Keywords

Exceptions: The finally keyword is available in Python, and it is always used after the try and except blocks.

try: A try statement in Python can be used to manage exceptions

try catch:The try clause contains the crucial operation that can cause an exception

Arithmetic error: Raised when an error occurs in numeric calculations

Keyerror: Raised when a key does not exist in a dictionary

Runtimeerror: Raised when an error occurs that do not belong to any specific expectations.

Unboundlocalerror: Raised when a local variable is referenced before assignment

Zerodivisionerror: Raised when the second operator in a division is zero

Identation error: Raised when indendation is not correct

Importerror: Raised when an imported module does not exist

AssertionError: Raised when an assert statement fails

AttributeError: Raised when attribute reference or assignment fails

EOFError: Raised when the input() method hits an "end of file" condition (EOF)

NameError: Raised when a variable does not exist

OverflowError: Raised when the result of a numeric calculation is too large

TabError: Raised when indentation consists of tabs or spaces

Self Assessment

1. A try-except block can only contain so many except statements.

A. Zero

B. One

C. More than one

D. More than zero

2. Is the Python code below correct?

```
try:
        #Do something
except:
        #Do Something
else:
        #Do something
```

A. No, there is no such thing lese

B. No, else cannot be used with except.

C. No, else must come before except

D. Yes

3. The finally block is executed when?

A. there is no exception

B. there is an exception

C. only if some condition that has been specified is satisfied

D. Always

4. What would the following Python code produce?

```
def foo ():
  try:
print (1)
  finally:
print (2)
foo ()
```

A. 12

B. 1

C. 2

D. 11

5. What occurs when '1' == 1 is put into action?

A. getting a True

B. we receive a False

C. a TypeError happens

D. There is a ValueError.\

6. What type of error is produced in python?

A. Compile time error

B. Run time error

C. Both a and b

D. None of these

7. When a ________ error occurs, the interpreter refuses to run the programme until the error is fixed; instead, we must save and rerun the programmes.

A. Syntax errors
B. Logical error
C. Runtime error
D. All of the above

8. A Python object called _________ stands for an error.

A. Interpreter
B. Compiler
C. Exception
D. Module

9. An exception is said to have been made when a programme fails to run as intended___________

A. Created
B. Asserted
C. Triggered
D. Raised

10. When a local or global variable is not defined, it gets raised.

A. NameError
B. ValueError
C. TypeError
D. ZeroDivisionError

11. It is raised when a calculation's outcome is greater than the numeric data type's upper limit.

A. ZeroDivisonError
B. OverFlowError
C. TypeError
D. ValueError

12. A rule is considered to be broken when

A. Error encountered and exception object is created
B. Runtime system searches for appropriate exception handler
C. Code that is designed to handle exception is executed
D. None of these

13. Exception handling is done in

A. try block
B. except block
C. finally, block
D. else block

14. Which of the following keywords doesn't fall within the category of exception handling?
A. try
B. catch
C. except
D. finally

15. When managing exceptions, certain keywords are utilised.
A. raise, assert, throw, catch
B. try, except, throw, catch
C. try, except, else, finally
D. throw, catch, finally, assert

Answers for SelfAssessment

1.	D	2.	D	3.	D	4.	A	5.	B
6.	C	7.	A	8.	C	9.	D	10.	A
11.	B	12.	C	13.	C	14.	B	15.	C

Review Questions

1. Describe how actually does the python try except clause works.
2. Give example how to catch specific exceptions in python.
3. With example explain raising exception in python.
4. Differentiate between try with else cluse and try with finally clause with example.
5. In Python can a try have multiple except?

Further Readings

- Mark Lutz,Programming Python: Powerful Object-Oriented Programming, OREILLY
- Wes McKinney, Python for data analysis, OREILLY
- David Ascher and Mark Lutz, Learning Python, OREILLY
- Eric Matthes, Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming, Starch Pres

Web Links

- https://www.tutorialspoint.com/python/index.htm
- https://www.python.org/downloads/
- https://www.w3schools.in/python/data-types
- https://www.programiz.com/python-programming/online-compiler/
- https://www.codecademy.com/catalog/language/python
- https://www.programiz.com/python-programming/user-defined-exception

Unit 06: Introduction to Numpy

CONTENTS

Objectives

- Learn basic concepts about arrays and lists.
- Learn to differentiate between array and list.
- Learn several array creation routines in Numpy which are used to create Ndarray objects.

Introduction

Python uses the data structures list and array to store many elements. Let's examine some key distinctions between lists and arrays in Python.

6.1 What is List in Python?

In Python, a list is a group of things that can include items of different data types, such as numeric, character, logical values, etc. It is a collection that supports negative indexing and is ordered. [] can be used to generate a list with data values.

Using Python's built-in methods, it is simple to merge and copy the contents of lists.

Program	Output

# creating a list containing elements 	
# belonging to different data types 	
sample_list = [1,"ABC",['k','j']]
print(sample_list)
	[1, 'ABC', ['k', 'j']]

An integer is the first element, a string is the second, and a list of characters is the third.

6.2 What is Array in Python?

A vector with members that are homogenous, or of the same data type, is referred to as an array. Contiguous memory regions are used to allocate elements, making it simple to modify them by adding or removing them or accessing them. To declare arrays in Python, we must utilize the array module. An exception with the message "Incompatible data types" is raised if an array's items have distinct data types.

Program	Output
# creating an array containing same # data type elements import array sample_array = array.array('i', [4, 5, 6]) # accessing elements of array for i in sample_array: print(i)	4 5 6

Difference between List and Array

List	Array
can include components from several data kinds.	only includes elements of the same data type.
No need to import a module specifically for declaration	A module must be imported specifically for a declaration.
cannot do mathematical calculations directly	may do mathematical calculations directly
may hold several types of items by being nested.	Contains either all identically sized nested elements
preferred for shorter data item sequences	preferred for longer data item sequences
Greater flexibility makes it simple to change (add or remove) data.	Less flexibility because addition and deletion must be done in terms of elements
Without using any explicit looping, the complete list can be printed.	To print or retrieve the array's component values, a loop must be created.
greater memory use for simple element addition	smaller in size when compared to memory

Array Creation routines

There are numerous ways to generate Numpy arrays.

To construct ndarray objects, Numpy provides a number of array creation functions.

numpy.empty()	builds an array with the provided form and datatype that is uninitialized.	numpy.empty(shape_of_array, dtype)
		>>>a = np.empty([2,2], dtype=int)
		>>>print(a)
			[[1 2]
			[3 4]]

numpy.zeros	produces an array with just zeros	numpy.zeros(shape, dtype)	[[0 0]

()	in it.	x = np.zeros([2,2], dtype=int)
		print(x)
			[0 0]]

numpy.ones( )	produces a single-item array.	numpy.ones(shape,dtype)
		x = np.ones([2,2])
		print(x)
			[[1. 1.]
			[1. 1.]]

numpy.eye()	produces a two-dimensional array with one on each diagonal and zeros everywhere else.	numpy.eye(N,M,k=0,dtype) where N → no. of rows M → no. of columns k → index of the diagonal >>>x = np.eye(3,k=0, dtype=int) >>>print(x)	[[1 0 0] [0 1 0] [0 0 1]]

numpy.identity()	gives the identity array with a primary diagonal of 1.	#numpy.identity(n,dtype)
		>>>x = np.identity(4, dtype=int)
		>>>print(x)
			[[1 0 0 0]
			[0 1 0 0]
			[0 0 1 0]
			[0 0 0 1]]

numpy.arange()	makes an array with the given range	#numpy.arange(value,<start><stop><step>):
		>>>x = np.arange(10)
		>>>print(x)
		>>>y = np.arange(0,10)
		>>>print(y)
		>>>z = np.arange(0,10,2)
		>>>print(z)
			[0 1 2 3 4 5 6 7 8 9]
			[0 1 2 3 4 5 6 7 8 9]
			[0 2 4 6 8]

numpy.mat()	Consider the input to be a matrix.	#numpy.mat(data, dtype)
		x = np.mat([(1,2,3),(4,5,6),(7,8,9)], dtype=int)
		print(x)
			[[1 2 3]
			[4 5 6]
			[7 8 9]]

numpy.full()	produces a new array containing a value for the fill.	#np.full(shape, fill_value)
		>>>x = np.full(shape=(2,3), fill_value=7)
		>>>print(x)
			[[7 7 7]
			[7 7 7]]

numpy.asarray()	creates a numpy array from a seque nce type.	#numpy.asarray(data, dtype) where a --> i/p data [lists, tuples, ndarrays] that can be	[[1 2] [3 4]]

		converted to an array. >>>x = np.asarray([[1,2],[3,4]], dtype=int) >>>print(x)	

6.3 How to Create An Array from Existing Data

numpy.asarray

With the exception of the fact that it has fewer parameters, this function is comparable to numpy.array. This procedure can be used to translate a Python sequence into a ndarray.

```
numpy.asarray(a, dtype = None, order = None)
```

These following parameters are passed to the function Object () {[native code]}.

Sr.No.	Parameter & Description
a	Any type of data input is acceptable, including list, list of tuples, tuples, tuples of tuples, and tuples of lists.
dtype	By default, the generated ndarray is affected by the data type of the incoming data.
order	F or C (row major) (column major). defaults to C.

Examples	Output
Example1 # convert list to ndarray import numpy as np x = [1,2,3] a = np.asarray(x) print a	[1 2 3]
Example2 # dtype is set import numpy as np x = [1,2,3] a = np.asarray(x, dtype = float) print a	[1. 2. 3.]

numpy.frombuffer

A buffer is treated as a one-dimensional array by this function. To return an ndarray, any object that exposes the buffer interface is used as an argument.

numpy.frombuffer(buffer, dtype = float, count = -1, offset = 0)

The following parameters are passed to the constructor

Sr.No.	Parameter & Description

1	buffer whatever object offers the buffer interface
2	dtype The returned data is a ndarray. by default, is float
3	count the amount of data to read; a default value of -1 indicates all data
4	offset the place to start reading from. default is zero.

Example	Output

import numpy as np
s = 'Hello World'
a = np.frombuffer(s, dtype = 'S1')
print a
	['H' 'e' 'l' 'l' 'o' ' ' 'W' 'o' 'r' 'l' 'd']

numpy.fromiter

This function converts any iterable object into an ndarray object. This function returns a brand-new one-dimensional array.

numpy.fromiter(iterable, dtype, count = -1)

Sr.No.	Parameter & Description
1	**iterable** Any iterable object
2	**dtype** Data type of resultant array
3	**count** The number of items to be read from iterator. Default is -1 which means all data to be read

The range () function, which may be used to return a list object, is demonstrated in the following examples. An ndarray object is created using an iterator of this list.

Examples	Output
Example1 # create list object using range function import numpy as np list = range(5) print list	[0, 1, 2, 3, 4]
Example2 # obtain iterator object from list import numpy as np	[0. 1. 2. 3. 4.]

list = range(5)
it = iter(list)
# use iterator to create ndarray 	
x = np.fromiter(it, dtype = float)
print x	

Indexing and Slicing

Only sequence data types are capable of indexing and slicing. Sequence types maintain the order in which elements are added, allowing us to retrieve their elements through indexing and slicing.

List, tuple, string, range, byte, and byte arrays are all sequence types in Python. Additionally, all of these kinds are compatible with indexing and slicing.

6.4 Indexing

An element of an iterable is said to be "indexed" if it is based on where it is located inside the iterable. Indexing starts at position 0. Index 0 represents the initial element in the sequence. Indexing in the negative starts at 1. Index -1 serves as a representation of the final element in the sequence. Each character in a string has a corresponding index number that can be used to access that character. Characters in a String can be accessed in two different ways.

- Using positive indexing to access characters in strings
- Utilising negative indexing to access characters in strings

	C O M P U T E R
Positive Indexing Negative indexing	0 1 2 3 4 5 6 7 9 -8 -7 -6 -5 -4 -3 -2 -1

Accessing string characters using positive indexing

We pass a Positive index (that we want to access) in square brackets in this situation. Index number zero is the first in the list of index numbers. (depicts the start of a string's characters).

Program	**Output**

# input string 	
inputString = "Hello tutorialspoint python"
 	
print("0th index character:", inputString[0])
print("7th index character", inputString[7])
 	
print("12th index character:", inputString[12])
('0th index character:', 'H')
('7th index character', 'u')
('12th index character:', 'a')
	0th index character: H
	7th index character u
	12th index character: a

Accessing string characters using negative indexing

In this kind of indexing, the negative index that we want to access is passed in square brackets. In this instance, the index number starts at -1. (that represents the last character of a string).

Program	**Output**

input string inputString = "Hello tutorialspoint python" print("last index character:", inputString[-1]) print("6th index character from last:", inputString[-6]) ('last index character:', 'n') ('6th index character from last:', 'p')	last index character: n 6th index character from last: p

Indexing in List

Program	Output
# input list inputList=[1, 4, 8, 6, 2] print ("Element at index 2:", inputList[2]) print ("last element of an input list:", inputList[-1]) ('Element at index 2:', 8) ('last element of an input list:', 2)	Element at index 2: 8 last element of an input list: 2

6.5 Slicing

Getting a subset of elements from an iterable based on their indices is referred to as "slicing."

By slicing a string, which is essentially a string contained within another string, we can produce a substring. When we only require a section of the string and not the complete string, we use slicing.

Syntax:

string [start:end: step]

Parameters

start - index from where to start

end - ending index

step - numbers of jumps/increment to take between i.estepsize

Program	Output
# input string inputString = "Hello tutorialspoint python" print("First 4 characters of the string:", inputString[: 4]) print("Alternate characters from 1 to 10 index(excluded):", inputString[1 : 10 : 2]) print("Alternate characters in reverse order from 1 to 10 index(excluded):", inputString[-1 : -10 : -2]) ('First 4 characters of the string:', 'Hell') ('Alternate characters from 1 to 10 index(excluded):', 'eluo') ('Alternate characters in reverse order from 1	First 4 characters of the string: Hell Alternate characters from 1 to 10 index(excluded): eluo Alternate characters in reverse order from 1 to 10 index(excluded): nhy n

to 10 index(excluded):', 'nhy n')	

6.6 Tuple Slicing

Tuple slicing is an option. It functions similarly to how lists and strings do. Several elements can be obtained through tuple slicing. The slicing operator is also used to accomplish tuple slicing. The syntax can be used to express the slicing operator.

Syntax:

[start:stop:step]

6.7 Difference between Indexing and Slicing

Indexing	Slicing
It only produces 1 item.	A new list or tuple is returned.
If you try to utilise an index that is too big, an IndexError will be thrown.	Out-of-range indices are handled kindly when used for slicing.
The list's length cannot be altered by item assignment during indexing.	By designating objects to slicing, we can modify the list's length or even remove items from it.
Indexing can be given a single element or an iterable.	A Type Error occurs when we assign a single element to slicing. It only permits iterables.

Summary

- In Python, a list is a group of things that can include items of different data types, such as numeric, character, logical values, etc
- A vector with members that are homogenous, or of the same data type, is referred to as an array
- With the exception of the fact that it has fewer parameters, this function is comparable to numpy.array.
- A buffer is treated as a one-dimensional array by this function. To return an ndarray, any object that exposes the buffer interface is used as an argument
- This function converts any iterable object into an ndarray object. This function returns a brand-new one-dimensional array.
- Only sequence data types are capable of indexing and slicing. Sequence types maintain the order in which elements are added, allowing us to retrieve their elements through indexing and slicing.
- An element of an iterable is said to be "indexed" if it is based on where it is located inside the iterable.
- We pass a Positive index (that we want to access) in square brackets in this situation. Index number zero is the first in the list of index numbers. (depicts the start of a string's characters).
- In this kind of indexing, the negative index that we want to access is passed in square brackets. In this instance, the index number starts at -1.
- Getting a subset of elements from an iterable based on their indices is referred to as "slicing."
- Tuple slicing is an option. It functions similarly to how lists and strings do. Several elements can be obtained through tuple slicing.

Keywords

append ():Adds an element at the end of the list

clear ():Removes all the elements from the list

copy ():Returns a copy of the list

count ():Returns the number of elements with the specified value

extend ():Add the elements of a list (or any iterable), to the end of the current list

index ():Returns the index of the first element with the specified value

insert ():Adds an element at the specified position

pop ():Removes the element at the specified position

remove ():Removes the first item with the specified value

reverse ():Reverses the order of the list

sort ():Sorts the list

SelfAssessment

1. Which of the ensuing commands will result in the creation of a list?

A. list1 = list()
B. list1 = []
C. list1 = list([1, 2, 3])
D. all of the mentioned

2. What is the output when we execute list("hello")?

A. ['h', 'e', 'l', 'l', 'o']
B. ['hello']
C. ['llo']
D. ['olleh']

3. What is max(list1) if list1 is [2445,133,12454,123]?

A. 2445
B. 133
C. 12454
D. 123

4. What function do we use to shuffle the list, let's say list1?

A. list1.shuffle()
B. shuffle(list1)
C. random.shuffle(list1)
D. random.shuffleList(list1)

5. What is list1[-1] if list1 is [2, 33, 222, 14, 25]?

A. Error
B. None

C. 25

D. 2

6. What does Python's NumPy function do?

A. To do numerical calculations

B. To do scientific computing

C. Both A and B

D. None of the mentioned above

7. What other Python libraries are comparable to Pandas?

A. NPy

B. RPy

C. NumPy

D. none of the mentioned above

8. Which of the following statements about Python's Pip is true?

A. Pip is a standard package management system

B. It is used to install and manage the software packages written in Python

C. Pip can be used to search a Python package

D. All of the mentioned above

9. Arrays in NumPy can be.

A. Indexed

B. Sliced

C. Iterated

D. All of the mentioned above

10. What will happen if you observe the following code and predict the outcome?

```
import numpy as np
a=np.array([1,2,3,4,5,6])
print(a)
```

A. [1 2 3 4 5]

B. [1 2 3 4 5 6]

C. [0 1 2 3 4 5 6]

D. None of the mentioned above

11. What will happen if you observe the following code and predict the outcome?

```
import numpy as np
a = np.array([10, 20, 30, 40])
b = np.array([18, 15, 14])
c = np.array([25, 24, 26, 28, 23])
x, y, z = np.ix_(a, b, c)
```

A. [[10]]
   [[20]]
   [[30]]
   [[40]]]

B. [[[1]]
   [[2]]
   [[3]]
   [[4]]

C. [[5]]]
   [[[18]]
   [[15]]
   [[[14]]]

D. None of the mentioned above

12. ndim allows us to find:

A. We can determine the array's dimension.

B. array size

C. Matrix operational activities

D. None of the previously mentioned

13. What will the following Python code produce?

```
import numpy as np
a = np.array([(10,20,30)])
print(a.itemsize)
```

A. 10

B. 9

C. 8

D. All of the mentioned above

14. Which of the following is the truncation division operator in Python?

A. |

B. //

C. /

D. %

15. Which one of the following is not a keyword in Python language?

A. pass

B. eval

C. assert

D. nonlocal

Answers for SelfAssessment

1.	D	2.	A	3.	C	4.	A	5.	C

6.	C	7.	C	8.	D	9.	D	10.	B
11.	A	12.	A	13.	C	14.	B	15.	B

Review Questions

1. Explain difference between indexing and slicing
2. Differentiate between Arrays and lists with examples.
3. Write down advantages of NumPy arrays compared with lists.
4. Explain using arrays in python with example.
5. Explain copy (), extend (), index (), pop () and remove () method with example.

Further Readings

- Mark Lutz, Programming Python: Powerful Object-Oriented Programming, OREILLY
- Wes McKinney, Python for data analysis, OREILLY
- David Ascher and Mark Lutz, Learning Python, OREILLY
- Eric Matthes, Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming, Starch Pres

Web Links

- https://www.tutorialspoint.com/python/index.htm
- https://www.python.org/downloads/
- https://www.w3schools.in/python/data-types
- https://www.programiz.com/python-programming/online-compiler/
- https://www.codecademy.com/catalog/language/python
- https://www.programiz.com/python-programming/user-defined-exception

Unit 07: Operations on NumPy Arrays

CONTENTS

Objectives

After this unit, student would be able to:

- learn basic manipulation operations on arrays.
- learn using broadcasting term how NumPy handles arrays of differing dimensions
- learn different types of binary operators with examples

Introduction

Arrays are not natively supported by Python, although Python Lists can be used in their place. You can utilize LISTS as ARRAYS as demonstrated on this unit, but in order to interact with arrays in Python, you must import a library like the NumPy library.

7.1 Arrays

Multiple values can be stored in an array in a single variable as shown in

Table 1 Creating an array containing colors name

Program	Output

colors = ["Red", "Green", "Yellow"]
print(colors)
	['Red', 'Green', 'Yellow']

What is an Array?

A unique type of variable called an array has the capacity to store several values at once.

If you have a list of objects, such as a list of automobile names, you might store the colors in separate variables as follows:

```
colors1 = "Red"
colors2 = "Green"
colors3 =  "Yellow"
```

How would you discover a specific color if you wanted to loop through all of the colors? What if you had 300 colors instead of only three?

A system is the answer!

You can use an index number to access the values in an array, which can store numerous values under a single name.

Access the Elements of an Array

An array element is referred to by its index number.

Program	Output

colors = ["Red", "Green", "Yellow"]
x = colors[0]
print(x)
	Red

Example: Change the first array item's value:

Program	Output

colors = ["Red", "Green", "Yellow"]
colors [0] = “Blue”
print(colors)
	['Blue', 'Green', 'Yellow']

The Length of an Array

Return the length of an array using the len() function (the number of elements in an array).

Example:Give the array of colors' element count back:

Program	Output

colors = ["Red", "Green", "Yellow"]
x = len(colors)
print(x)
	3

An array's length is always one greater than its topmost array index.

Looping Array Elements

To iterate over each element of an array, use the for in loop.

Example: Print each element in the array of colours:

Program	Result

colors = ["Red", "Green", "Blue"]
for x in colors:
print(x)
	Red
	Green
	Blue

Adding Array Elements

The append () method can be used to include an element in an array.

Example: Add this element to the array of colours:

Program	Result

colors = ["Red", "Green", "Yellow"]
colors.append("Blue")
print(colors)
	['Red', 'Green', 'Yellow', 'Blue']

Removing Array Elements

To iterate over each element of an array, use the for in loop.

Example:Subtract the second color from the array:

Program	Output

colors = ["Red", "Green", "Blue"]
colors.pop(1)
print(colors)
	['Red', 'Blue']

The remove () method can also be used to delete an element from an array.

Example: Eliminate the element whose value is "Green"

Program	Output

colors = ["Red", "Green", "Blue"]
colors.remove("Green")
print(colors)
	['Red', 'Blue']

The remove () method of the list simply eliminates the first instance of the entered value.

Array Methods

Method	Description
append()	adds a new element to the list's end
clear()	removes all of the list's elements.
copy()	gives a copy of the list back
count()	returns the quantity of elements that have the given value.
extend()	To finish the current list, append the entries of another list (or any iterable)
index()	gives back the position of the first element with the given value.
insert()	adds a component in the designated location.
pop()	removes the component from the designated place.
remove()	the first item with the required value is eliminated.

reverse()	reverses the list sort's original order ()
sort()	lists are sorted

Arrays are not supported by default in Python, however Python Lists can be used in their place.

7.2 Broadcasting with NumPy Arrays

The term "broadcasting" describes how Numpy handles arrays of differing dimensions when performing operations that result in restrictions; the smaller array is broadcast across the bigger array to ensure that they have similar forms.

As we know that Numpy is built in C, broadcasting offers a way to vectorize array operations so that looping happens in C rather than Python. This results in effective algorithm implementations without the requirement for extra data duplication. In some circumstances, broadcasting is a negative idea because it results in memory usage that slows down computation.

Program	Output

import numpy as np
a = np.array([5, 7, 3, 1])
b = np.array([90, 50, 0, 30])
# array are compatible because of same Dimension 	
c = a * b
print (c)
	[450 350 0 30]

Broadcasting Rules:

1. Prepend the shape of the lower rank array with 1s until both shapes have the same length if the arrays don't have the same rank.

2. If the two arrays in a dimension have the same size or if one of the arrays has size 1 in that dimension, the two arrays are compatible in that dimension.

3. If the arrays are consistent with all dimensions, they can be broadcast together.

4. Each array acts as though it has a shape equal to the maximum element-wise shape of the two input arrays after broadcasting.

5. The first array acts as if it were copied along any dimension where one array had size 1 and the other array had size larger than 1.

7.3 Binary Operators

The binary operators fall into the following categories:

a. Arithmetic Operators

Common mathematical procedures are carried out using arithmetic operators and numeric values:

Operator	Name	Example
+	Addition	x + y
_	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y

%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor Division	x // y

b. Bitwise Operators

To compare (binary) numbers, use the following bitwise operators:

Operator	Name	**Description**
&	AND	if both bits are 1, sets each bit to 1.
|	OR	if one of two bits is 1, it sets each bit to 1.
^	XOR	if just one of two bits is 1, it sets each bit to 1.
~	NOT	all the bits are inverted.
<<	Zero fill left shift	Pushing zeros in from the right causes a shift to the left, causing the last few bits to disappear.
>>	Signed right shift	Push copies of the leftmost bit in from the left to shift right whil e letting the rightmost bits fall off.

c. Relational Operators

The primary purpose of the relational operators, commonly referred to as comparison operators, is to return either true or false depending on the value of the operands.

The relational operators are listed as follows:

1. <
2. >
3. <=
4. >=
5. ==
6. !=

Summary

- Arrays are not natively supported by Python, although Python Lists can be used in their place
- A unique type of variable called an array has the capacity to store several values at once.
- An array element is referred to by its index number.
- Return the length of an array using the Len () function (the number of elements in an array).
- To iterate over each element of an array, use the for in loop.
- The append () method can be used to include an element in an array.
- The term "broadcasting" describes how Numpy handles arrays of differing dimensions when performing operations that result in restrictions; the smaller array is broadcast across the bigger array to ensure that they have similar forms.
- The primary purpose of the relational operators, commonly referred to as comparison operators, is to return either true or false depending on the value of the operands.
- Python's array module can be imported to generate an array. An array can be created by using the syntax array (data type, value list), which takes two arguments: a data type and a value list.

- The Array can have elements added to it by using the built-in insert () function.
- (1)/O Time Complexity (n) (O(1) for inserting elements at the array's end, O(n) for inserting elements at its start, and O(n) for inserting elements throughout the entire array.

Keywords

append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Self Assessment

1. What does the following code produce as output?

   L = ['d','e','f','g']

   print("".join(L))

A. Error
B. None
C. defg
D. ‘d’,’e’,’f’,’g’

2. What does the following code produce as output?

   print type(type(int))

A. <type ‘type’>
B. type ‘int’
C. integer
D. 0

3. When a function is declared inside a class, what is called?

A. Function
B. Module
C. Class function
D. Method

4. Which of the following describes how Python's id() function is used?

A. Id returns the identity of the object

B. Id returns the first number in list

C. Id returns the last number in list

D. None

5. To declare an array, you must use

A. Parenthesis ( )

B. Curly brackets { }

C. Pipes | |

D. Brackets [ ]

6. An item's position in an array is known as its

A. Value

B. Location

C. Index

D. Position

7. What else is said to be a FOR-NEXT loop?

A. condition-controlled

B. count-controlled

C. Uncontrolled loop

D. Controlled loop

8. What is the index of the following array's lower bound?

   Dim student (24) as string

A. 24

B. 25

C. 23

D. 0

9. In names=[Red, Green, Blue] Index value 1 is?

A. Red

B. Blue

C. 0

D. Green

10. len(names)

A. Finds the length of the list called names

B. Finds the length of the list called len

C. Finds the length of the list called Length

D. None

11. colors = ["Red", "Green", "Blue"]

    for x in colors:

```
print(x)
```

A. Red

B. Green

C. Red Green Blue

D. Blue

12. Use of pop () method

A. To add an element to an array

B. To remove an element from an array

C. To remove the first occurrence of the specified value

D. None of above.

13. Use of clear () method

A. Remove only first element from the list

B. Removes all the elements from the list

C. Reverse the list

D. None of above

14. What does the following code produce as output?

```
val1 = 4
val2 = 4
 res = val1 + val2
print(res)
```

A. 88

B. 44

C. 8

D. 4+4

15. What does the following code produce as output?

```
val1 = 3
val2 = 2
res = val1 // val2
print(res)
```

A. 1

B. 2

C. 3

D. 0

16. Explain all arithmetic operatorswith example.
17. Explain bitwise operators with example.
18. Explain relational operators with example.
19. What do you understand by Arrays? Explain difference between array and lists.

20. The term "broadcasting" describes how NumPy handles arrays of differing dimensions when performing operations that result in restrictions. Explain with example.

Answers for Self Assessment

1.	C	2.	A	3.	D	4.	A	5.	D
6.	C	7.	B	8.	B	9.	D	10.	A
11.	C	12.	B	13.	B	14.	C	15.	A

Further Readings

- Mark Lutz,Programming Python: Powerful Object-Oriented Programming, OREILLY
- Wes McKinney, Python for data analysis, OREILLY
- David Ascher and Mark Lutz, Learning Python, OREILLY
- Eric Matthes, Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming, Starch Pres

Web Links

- https://www.tutorialspoint.com/python/index.htm
- https://www.python.org/downloads/
- https://www.w3schools.in/python/data-types
- https://www.programiz.com/python-programming/online-compiler/
- https://www.codecademy.com/catalog/language/python

Unit 08: NumPy Functions

CONTENTS

Objectives

- learn built-in module to run mathematical functions
- learn built-in module to run statistical functions
- learn basic concepts pf sort, search and count Functions

Introduction

Mathematical calculations may occasionally be required when working on certain types of business or scientific tasks. Python has a math module that can handle these calculations. Basic operations like addition, subtraction, multiplication, and division as well as more complex ones like trigonometric, logarithmic, and exponential functions are handled by the math module's functions. With the aid of a sizable dataset comprising functions that are described with the aid of useful examples, we learn about the math module from fundamentals to advanced concepts.

We must import the module into our code in order to utilize it.

```
import math
```

8.1 Constants Provided by the Math Module

The value of numerous constants, including pi and tau, is provided via the math module. The usage of such constants eliminates the need to precisely and repeatedly write down the value of each constant. These constants are offered by the math module:

Some Consonants

Sr.no	Consonants & Descriptions
1	pi Return the value of pi: 3.141592

2	E Return the value of natural base e. e is 0.718282
3	tau Returns the value of tau. tau = 6.283185
4	inf Returns the infinite
5	nan Not a number type

8.2 Numbers and Numeric Representation

Numbers are represented using these functions in a variety of ways. The techniques are as follows:

Sr.No	Function & Description
1	Ceil(x) Give the Ceiling value back. It is the smallest integer that is either greater than or equal to x.
2	copysign(x,y) It copies the sign of y to x and returns the number x.
3	fbas(x) gives x's absolute value back.
4	factorial(x) returns the x factorial, where x >=0.
5	floor(x) Give the Floor value back. It is the greatest integer that is less than or equal to x.
6	fsum(iterable) Calculate the elemental sum of an iterable object.
7	gcd(x, y) returns the x and y's greatest common divisor.
8	isfinite(x) determines whether x is neither a nan nor an infinite.
9	isinf(x) determines if x is infinite
10	isnan(x) determines whether or not x is a number.
11	remainder(x,y) Calculate the leftover after dividing x by y.

Example Code

```
import math
print('The Floor and Ceiling value of 23.56 are: ' + str(math.ceil(23.56)) + ', ' + str(math.floor(23.56)))
x = 10
y = -15
print('The value of x after copying the sign from y is: ' + str(math.copysign(x, y)))
print('Absolute value of -96 and 56 are: ' + str(math.fabs(-96)) + ', ' + str(math.fabs(56)))
my_list = [12, 4.25, 89, 3.02, -65.23, -7.2, 6.3]
print('Sum of the elements of the list: ' + str(math.fsum(my_list)))
print('The GCD of 24 and 56 : ' + str(math.gcd(24, 56)))
x = float('nan')
if math.isnan(x):
print('It is not a number')
x = float('inf')
y = 45
if math.isinf(x):
print('It is Infinity')
print(math.isfinite(x)) #x is not a finite number
print(math.isfinite(y)) #y is a finite number
```

Output

```
The Floor and Ceiling value of 23.56 are: 24, 23
The value of x after copying the sign from y is: -10.0
Absolute value of -96 and 56 are: 96.0, 56.0
Sum of the elements of the list: 42.13999999999999
The GCD of 24 and 56 : 8
It is not a number
It is Infinity
False
True
```

8.3 Power and Logarithmic Functions

These functions are used to do various power- and logarithmic-related calculations.

Sr.No	Function & Description
1	pow(x,y) the value of x raised to the power of y
2	sqrt(x) determines x's square root

3	exp(x) Finds xe, where e = 2.718281
4	log(x[, base]) provides the base and returns the Log of x. The standard base is e.
5	log2(x) gives the Log of x with base 2 as a result.
6	log10(x) provides the Log of x with a base of 10.

Example Code

```
import math
print('The value of 5^8: ' + str(math.pow(5, 8)))
print('Square root of 400: ' + str(math.sqrt(400)))
print('The value of 5^e: ' + str(math.exp(5)))
print('The value of Log(625), base 5: ' + str(math.log(625, 5)))
print('The value of Log(1024), base 2: ' + str(math.log2(1024)))
print('The value of Log(1024), base 10: ' + str(math.log10(1024)))
```

Output

```
The value of 5^8: 390625.0
Square root of 400: 20.0
The value of 5^e: 148.4131591025766
The value of Log(625), base 5: 4.0
The value of Log(1024), base 2: 10.0
The value of Log(1024), base 10: 3.010299956639812
```

8.4 Trigonometric & Angular Conversion Functions

Different trigonometric operations are computed using these functions.

Sr.No.	Function & Description
1	sin(x) Specify x's sine in radians.
2	cos(x) Specify x's cosine in radians.
3	tan(x) Give back x's tangent in radians.
4	asin(x) This is the sine's inverse operation; the other two are acos and atan.
5	degrees(x) Change angle x's radian value to a degree.

6	radians(x) x-angle conversion from degrees to radians

Example Code
import math print('The value of Sin(60 degree): ' + str(math.sin(math.radians(60)))) print('The value of cos(pi): ' + str(math.cos(math.pi))) print('The value of tan(90 degree): ' + str(math.tan(math.pi/2))) print('The angle of sin(0.8660254037844386): ' + str(math.degrees(math.asin(0.8660254037844386))))
Output
The value of Sin(60 degree): 0.8660254037844386 The value of cos(pi): -1.0 The value of tan(90 degree): 1.633123935319537e+16 The angle of sin(0.8660254037844386): 59.99999999999999

8.5 Statistical Function in Python

By importing the statistic keyword, Python has the capacity to solve mathematical expressions and statistical data. Numerous statistical and mathematical calculations can be performed using Python.

These procedures determine the sample or population's average value.

mean()	Arithmetic mean value (average) of data.
harmonic_mean()	Harmonic mean value of data.
median()	Median value (middle value) of data.
median_low()	Low median value of data.
median_high()	High median value of data.
median_grouped()	Median of the grouped data and also calculate the 50th percentile of the grouped data.
mode()	Maximum number of occurrence of data.

a. mean()

With the use of an iterator or series, this function determines the arithmetic mean or average value of the sampled data.

Example	Output
list = [1, 2, 3,3,4,5,] print ("The mean values is : ",end="") print (statistics.mean(list))	The mean value is : 3

b. harmonic_mean()

This function computes a real-valued number's harmonic mean sequentially or iteratively.

Example	Output
list = [1,2,3] print ("The harmonic _mean values is : ",end="") print (statistics.harmonic_mean(list))	The harmonic _mean values is :1.6

c. median()

The middle value of the arithmetic data is calculated using this function iteratively.

Example	Output
list= [1, 3,5,7] print ("The median values is : ",end="") print (statistics.median(list))	The median values is :4.0

d. median_low()

When there are more even than odd components in the data, this function calculates the lower of the two middle elements, rather than the median.

Example	Output
list = [1,2,2,3,3,3] print ("The median_low values is : ",end="") print (statistics.median_low(list))	The median_low values is :2

e. median_high

If there are even numbers of items, this function calculates the higher of the two middle elements in the data; otherwise, it calculates the median of the data.

Example	Output
list = [1,2,2,3,3,3] print ("The median_high values is : ",end="") print (statistics.median_high(list))	The median_high values is :3

f. median_grouped

Example	Output
list = [2,2,3,4] print ("The median_grouped values is : ",end="") print (statistics.median_grouped(list))	The median_grouped values is : 2.5

g. mode()

This method returns the data point with the greatest number of occurrences from nominal or discrete data.

Example	Output
list = [2,2,3,4,4,1,2] print ("The mode values is : ",end="") print (statistics.mode(list))	The mode values are: 2

8.6 Sort, Search and Count Function

NumPy has a number of functions that are relevant to sorting. These sorting functions implement many sorting algorithms, each of which is distinguished by its execution speed, worst-case performance, required workspace, and algorithmic stability. Three sorting algorithms are compared in the table below.

Kindl	Speed	Worst Case	Work Space	Stable
'quicksort'	1	O(n^2)	0	no
'mergesort'	2	O(n*log(n))	~n/2	yes
'heapsort'	3	O(n*log(n))	0	no

numpy.sort()

A sorted version of the input array is returned by the sort () function. The following characteristics apply.
numpy.sort(a, axis, kind, order)

where,

Sr.No	Parameter & Description
1	A Array to be sorted
2	Axis the direction that the array should be sorted along. If none, sorting on the last axis flattens the array.
3	Kind default is quicksort
4	Order If the array has fields, specify how they should be sorted.

Example

```
import numpy as np
a = np.array([[3,7],[9,1]])
print 'Our array is:'
print a
print '\n'
print 'Applying sort() function:'
print np.sort(a)
print '\n'
print 'Sort along axis 0:'
print np.sort(a, axis = 0)
print '\n'
```

```
# Order parameter in sort function
dt = np.dtype([('name', 'S10'),('age', int)])
a = np.array([("raju",21),("anil",25),("ravi", 17), ("amar",27)], dtype = dt)
print 'Our array is:'
print a
print '\n'
print 'Order by name:'
print np.sort(a, order = 'name')
```

Output

```
Our array is:
[[3 7]
 [9 1]]
Applying sort() function:
[[3 7]
 [1 9]]
Sort along axis 0:
[[3 1]
[9 7]]
Our array is:
[('raju', 21) ('anil', 25) ('ravi', 17) ('amar', 27)]
Order by name:
[('amar', 27) ('anil', 25) ('raju', 21) ('ravi', 17)]
```

numpy.argsort()

The numpy.argsort() function returns an array of data indices by performing an indirect sort on the input array along the specified axis. The sorted array is built using this indices array.

Example

```
import numpy as np
x = np.array([3, 1, 2])
print 'Our array is:'
print x
print '\n'
print 'Applying argsort() to x:'
y = np.argsort(x)
print y
print '\n'
print 'Reconstruct original array in sorted order:'
print x[y]
print '\n'
print 'Reconstruct the original array using loop:'
```

```
for i in y:
  print x[i],
```

Output

```
Our array is:
[3 1 2]
Applying argsort() to x:
[1 2 0]
Reconstruct original array in sorted order:
[1 2 3]
Reconstruct the original array using loop:
1 2 3
```

numpy.lexsort()

Function uses a series of keys to conduct an indirect sort. The keys can be thought of as a spreadsheet column. The function provides a list of indices that can be used to access the sorted data. Keep in mind that the sort's primary key just so happens to be the last key.

Example

```
import numpy as np
nm = ('raju','anil','ravi','amar')
dv = ('f.y.', 's.y.', 's.y.', 'f.y.')
ind = np.lexsort((dv,nm))
print 'Applying lexsort() function:'
print ind
print '\n'
print 'Use this index to get sorted data:'
print [nm[i] + ", " + dv[i] for i in ind]
```

Output

```
Applying lexsort() function:
[3 1 0 2]
Use this index to get sorted data:
['amar, f.y.', 'anil, s.y.', 'raju, f.y.', 'ravi, s.y.']
```

Numerous functions for searching inside an array are available in the NumPy module. There are functions for determining the maximum, minimum, and items satisfying a particular criterion.

numpy.argmax() and numpy.argmin()

These two operations give back the indices of the highest and lowest elements along the specified axis.

```
import numpy as np
a = np.array([[30,40,70],[80,20,10],[50,90,60]])
print 'Our array is:'
print a
print '\n'
print 'Applying argmax() function:'
print np.argmax(a)
print '\n'
print 'Index of maximum number in flattened array'
print a.flatten()
print '\n'
print 'Array containing indices of maximum along axis 0:'
maxindex = np.argmax(a, axis = 0)
print maxindex
print '\n'
print 'Array containing indices of maximum along axis 1:'
maxindex = np.argmax(a, axis = 1)
print maxindex
print '\n'
print 'Applying argmin() function:'
minindex = np.argmin(a)
print minindex
print '\n'
print 'Flattened array:'
print a.flatten()[minindex]
print '\n'
print 'Flattened array along axis 0:'
minindex = np.argmin(a, axis = 0)
print minindex
print '\n'
print 'Flattened array along axis 1:'
minindex = np.argmin(a, axis = 1)
print minindex
```

Output

```
Our array is:
[[30 40 70]
 [80 20 10]
 [50 90 60]]
Applying argmax() function:
7
Index of maximum number in flattened array
[30 40 70 80 20 10 50 90 60]
Array containing indices of maximum along axis 0:
```

```
[1 2 0]
Array containing indices of maximum along axis 1:
[2 0 1]
Applying argmin() function:
5
Flattened array:
10
Flattened array along axis 0:
[0 1 1]
Flattened array along axis 1:
[0 2 0]
```

numpy.nonzero()

The output of the numpy.nonzero() function is the indexes of the array's non-zero members.

Example

```
import numpy as np
a = np.array([[30,40,0],[0,20,10],[50,0,60]])
print 'Our array is:'
print a
print '\n'
print 'Applying nonzero() function:'
print np.nonzero (a)
```

Output

```
Our array is:
[[30 40 0]
 [ 0 20 10]
 [50 0 60]]
Applying nonzero() function:
(array([0, 0, 1, 1, 2, 2]), array([0, 1, 1, 2, 0, 2]))
```

numpy.where()

The where() function returns the indices of input array members that satisfy the specified criterion.

Example

```
import numpy as np
x = np.arange(9.).reshape(3, 3)
print 'Our array is:'
print x
print 'Indices of elements > 3'
y = np.where(x > 3)
```

```
print y
print 'Use these indices to get elements satisfying the condition'
print x[y]
```

Output

```
Our array is:
[[ 0. 1. 2.]
 [ 3. 4. 5.]
 [ 6. 7. 8.]]
Indices of elements > 3
(array([1, 1, 2, 2, 2]), array([1, 2, 0, 1, 2]))
Use these indices to get elements satisfying the condition
[ 4. 5. 6. 7. 8.]
```

numpy.extract()

The elements satisfying any criterion are returned by the extract() function.

Example

```
import numpy as np
x = np.arange(9.).reshape(3, 3)
print 'Our array is:'
print x
# define a condition
condition = np.mod(x,2) == 0
print 'Element-wise value of condition'
print condition
print 'Extract elements using condition'
print np.extract(condition, x)
```

Output

```
Our array is:
[[ 0. 1. 2.]
 [ 3. 4. 5.]
 [ 6. 7. 8.]]
Element-wise value of condition
[[ True False True]
 [False True False]
 [ True False True]]
Extract elements using condition
[ 0. 2. 4. 6. 8.]
```

Summary

- Mathematical calculations may occasionally be required when working on certain types of business or scientific tasks
- The value of numerous constants, including pi and tau, is provided via the math module
- By importing the statistic keyword, Python has the capacity to solve mathematical expressions and statistical data.
- With the use of an iterator or series, this function determines the arithmetic mean or average value of the sampled data.
- When there are more even than odd components in the data, this function calculates the lower of the two middle elements, rather than the median.
- If there are even numbers of items, this function calculates the higher of the two middle elements in the data; otherwise, it calculates the median of the data.
- NumPy has a number of functions that are relevant to sorting. These sorting functions implement many sorting algorithms, each of which is distinguished by its execution speed, worst-case performance, required workspace, and algorithmic stability
- A sorted version of the input array is returned by the sort() function.
- The numpy.argsort() function returns an array of data indices by performing an indirect sort on the input array along the specified axis.
- Function uses a series of keys to conduct an indirect sort. The keys can be thought of as a spreadsheet column.

Keywords

numpy.sort(): A sorted version of the input array is returned by the sort() function.

numpy.argsort(): The numpy.argsort() function returns an array of data indices by performing an indirect sort on the input array along the specified axis. The sorted array is built using this indices array.

numpy.lexsort(): function uses a series of keys to conduct an indirect sort. The keys can be thought of as a spreadsheet column. The function provides an array of indices that can be used to retrieve the sorted data.

numpy.argmax() and numpy.argmin(): These two operations give back the indices of the highest and lowest elements along the specified axis.

numpy.nonzero():The output of the numpy.nonzero() function is the indexes of the array's non-zero members**.**

numpy.where():The where() function returns the indices of input array members that satisfy the specified criterion.

numpy.extract():The function extract() returns elements that meet any requirement.

Self Assessment

1. What does the following code produce as output?

```
# Import math Library
import math
print(math.acos(0.65)
```

A. 0.863211890069541

B. 2.15316056466364

C. 1.5707963267948966

D. 0.0

2. What does the following code produce as output?

```
# Import statistics Library
import statistics
print(statistics.mean([2, 4, 8, 16, 9, 11, 13]))
```

A. 7

B. 9

C. 8

D. 0

3. What does the following code produce as output?

```
# Import statistics Library
import statistics
print(statistics.median([2, 4, 6, 8, 16, 11, 13]))
```

A. 7

B. 9

C. 8

D. 8.5

4. What does the following code produce as output?

```
print(math.ceil(2.4))
print(math.ceil(6.3))
print(math.ceil(-4.3))
print(math.ceil(21.6))
print(math.ceil(15.0))
```

A. 4,7,-4,22,17

B. 4,7,-4,21,15

C. 4,7,-4,22,15

D. 3,7,-4,22,15

5. What does the following code produce as output?

```
# import math Library
import math
print(math.isfinite(5670))
print(math.isfinite(-46.33))
print(math.isfinite(math.inf))
```

A. True, True, False

B. False, True, False

C. True, False, False

D. True, True, True

6. What does the following code produce as output?

```
# Import statistics Library
import statistics
# Calculate the variance from a sample of data
print(statistics.variance([2, 4, 6, 8, 8, 10]))
```

A. 7.6666666
B. 8.6666666
C. 9.6666667
D. 7.456789

7. What does the following code produce as output?

```
# Import statistics Library
import statistics
# Calculate the standard deviation from a sample of data
print(statistics.stdev([2, 4, 6, 8, 10, 12]))
```

A. 3.7416573867739413
B. 3.8516573867739413
C. 4.7416573867739413
D. 3.9426673867739413

8. What does the following code produce as output?

```
# Import statistics Library
import statistics
# Calculate the mode
print(statistics.mode([2, 4, 4, 5, 6, 8, 8, 10]))
```

A. 3
B. 6
C. 5
D. 4

9. What does the following code produce as output?

```
# Import math Library
import math
# Initialize the number of items to choose from
n = 5
# Initialize the number of possibilities to choose
k = 4
# Print total number of possible combinations
print (math.comb(n, k))
```

A. 6
B. 7
C. 5
D. 4

10. What does the following code produce as output?

```
#Import math Library
import math
#Convert angles from radians to degrees:
print (math.degrees(7.90))
print (math.degrees(-12))
```

A. 852.63665815335037, -687.5493541569879
B. 352.63665815335037, -687.5493541569879
C. 462.63665815335037, -887.5493541569879
D. 452.63665815335037, -687.5493541569879

11. What does the following code produce as output?

```
#Import math Library
import math
#Remove - sign of given number
print(math.fabs(-55.34))
print(math.fabs(-6))
```

A. 55.34, 6.0
B. 54.34,6.0
C. 55.24,6.0
D. None of above
E. Red Green Blue
F. Blue

12. What does the following code produce as output?

```
# Import math Library
import math
# Return the base-10 logarithm of different numbers
print(math.log10(1.8183))
```

A. 0.359665538729672
B. 0.269665538729672
C. 0.459665538729672
D. 0.259665538729672

13. What does the following code produce as output?

```
# Import statistics Library
import statistics
# Calculate the variance of an entire population
print(statistics.pvariance([2, 4, 6, 8, 9, 11]))
```

A. 10.222222222222223
B. 11.222222222222223
C. 9.222222222222223

D. 12.222222222222223

14. What does the following code produce as output?

```
#Import math Library
import math
#find the  the greatest common divisor of the two integers
print (math.gcd(4, 5))
```

A. 1
B. 2
C. 3
D. 4+4

15. What does the following code produce as output?

```
# Import math Library
import math
# Return the value of 9 raised to the power of 3
print(math.pow(4, 3))1
```

A. 12
B. 64
C. 444
D. None of above

16. Explain with example function of ceil () and copysign ()
17. Explain power and logarithmic functions
18. Explain trigonometric and angular conversion functions.
19. Explain statistical functions in python.
20. Explain sort, search and count function in python

Answers for Self Assessment

1.	A	2.	B	3.	C	4.	D	5.	A
6.	A	7.	A	8.	D	9.	C	10.	D
11.	A	12.	D	13.	C	14.	A	15.	B

FurtherReadings

- Mark Lutz,Programming Python: Powerful Object-Oriented Programming, OREILLY
- Wes McKinney, Python for data analysis, OREILLY
- David Ascher and Mark Lutz, Learning Python, OREILLY
- Eric Matthes, Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming, Starch Pres

Web Links

- https://www.tutorialspoint.com/python/index.htm
- https://www.python.org/downloads/
- https://www.w3schools.in/python/data-types
- https://www.programiz.com/python-programming/online-compiler/
- https://www.codecademy.com/catalog/language/python

Unit 09: Handling with Pandas

CONTENTS

Objectives

- Series, Dataframe, Sorting, Working with Csv Files
- Operations Using Dataframe

Introduction

Python's Pandas package is used to manipulate data sets.

It offers tools for data exploration, cleaning, analysis, and manipulation.

Wes McKinney came up with the name "Pandas" in 2008, and it refers to both "Panel Data" and "Python Data Analysis."

9.1 Why use Pandas?

With the aid of Pandas, we can examine large data sets and draw conclusions based on statistical principles.

Pandas can organizedisorganized data sets, making them readable and useful.

In data science, relevant data is crucial.

Data Science is a subfield of computer science that focuses on the storage, utilization, and analysis of data with the goal of extracting knowledge from it.

What can Pandas Do?

Pandas provides you with information on the data. Like:

Does a relationship exist between two or more columns?

What is the median value?

The maximum?

Minimum value

Rows that are irrelevant or contain incorrect data, such as empty or NULL values, can also be deleted by Pandas. This process is known as data cleaning.

Installation of Pandas

Pandas installation is fairly simple if Python and PIP are already installed on a machine.

Use this command to install it:

```
pip install pandas
```

Use a Python distribution with Pandas already installed, such as Anaconda, Spyder, etc., if this command fails.

Import Pandas

Once Pandas is installed, import it by adding the import keyword to your applications:

```
Import pandas
```

Pandas has been imported and is now ready for usage.

Example	Output

import pandas as pd
mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}myvar = pd.DataFrame(mydataset)
print(myvar)
	cars passings
	0 BMW 3
	1 Volvo 7
	2 Ford 2

Pandas Series

9.2 What is a Series

A Pandas Series resembles a table's column.

It is a one-dimensional array that can hold any kind of data.

Program	Output

import pandas as pd
a = [3, 6, 1]
myvar = pd.Series(a)
print(myvar)
	0 3
	1 6
	2 1
	dtype: int64

Labels

The values are identified with their index number if nothing else is supplied. The index of the first item is 0, that of the second is 1, etc.

To access a certain value, use this label.

Program	Output

import pandas as pd
a = [3, 6, 1]
myvar = pd.Series(a)
	3

print(myvar[0])	

Create Labels

You are able to name your own labels using the index option.

Program	Output

import pandas as pd
a = [3, 6, 1]
myvar = pd.Series(a, index = ["a", "b", "c"])
print(myvar)
	a 3
	b 6
	c 1
	dtype: int64

When you create labels, you can use the label to get to an item.

Program	Output

import pandas as pd
a = [3, 6, 1]
myvar = pd.Series(a, index = ["a", "b", "c"])
print(myvar["b"])
	6

Key/Value Objects as Series

When constructing a Series, you can also utilise a key/value object like a dictionary.

Program	Output

import pandas as pd
calories = {"day1": 330, "day2": 420, "day3": 300}
myvar = pd.Series(calories)
print(myvar)
	day1 330
	day2 420
	day3 300
	dtype: int64

Note: The labels are changed into the dictionary's keys.

Use the index option to specify only the words you wish to be included in the Series, leaving out the rest of the words in the dictionary.

Program	Output

import pandas as pd
calories = {"day1": 400, "day2": 320, "day3": 300}
myvar = pd.Series(calories, index = ["day1", "day2"])
print(myvar)
	day1 400
	day2 320
	dtype: int64

9.3 <u>DataFrames</u>

Known as DataFrames in Pandas, data sets are often multidimensional tables.

A DataFrame is the entire table, but a Series is similar to a column.

Program	Output

import pandas as pd
data = {
  "calories": [320, 480, 590],
  "duration": [30, 50, 55]
}
myvar = pd.DataFrame(data)
print(myvar)
	calories duration
	0 320 30
	1 480 50
	2 590 55

9.4 Python List Sort () Method

Program	Output

colors = ['Red', 'Green', 'Blue']
colors.sort()
print(colors)
	['Blue', 'Green', 'Red']

Definition and Usage

The list is automatically sorted ascending using the sort () method. Making a function to select the sorting criteria is another option (s).

Syntax

list.sort(reverse=True|False, key=myFunc)

Parameter Values

Parameter	Description
reverse	Optional. reverse=True will sort the list descending. Default is reverse=False
Key	Optional. A function to specify the sorting criteria(s)

Sort the list descending	
Program	Output

colors = ['Blue', 'Green', 'Red']
colors.sort(reverse=True)
print(colors)
	['Red', 'Green', 'Blue']

Sort the list by length of values:	
Program	Output

# A function that returns the length of the value: 	
def myFunc(e):
  return len(e)
cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']
cars.sort(key=myFunc)
print(cars)
	['VW', 'BMW', 'Ford', 'Mitsubishi']

Sort a list of dictionaries based on the "year" value of the dictionaries	
Program	Output

def myFunc(e):
  return e['year']
cars = [
  {'car': 'Ford', 'year': 2005},
  {'car': 'Mitsubishi', 'year': 2000},
  {'car': 'BMW', 'year': 2019},
  {'car': 'VW', 'year': 2011}
]
cars.sort(key=myFunc)
print(cars)
	[{'car': 'Mitsubishi', 'year': 2000}, {'car': 'Ford', 'year': 2005}, {'car': 'VW', 'year': 2011}, {'car': 'BMW', 'year': 2019}]

9.5 Working with CSV Files for Data Science

What is CSV?

The term "Comma Separated Values" or CSV. It is the most basic way to save tabular data as plain text. Because we as data scientists usually use CSV data in our daily work, it is crucial to know how to work with it.

Structure of CSV

Year	Experience	Salary
2001	1	39343
2004	5	40000
2015	6	70000

The document is called "Salary Data.csv." The header line of a CSV file contains the names of the fields and features.

Reading a CSV

Python offers a variety of CSV file handling options.

Using csv.reader

Using the csv.reader object, the Python language's built-in module for reading CSV files.

Steps to Read a CSV File

1. Import the csv library

   import csv

2. Open the csv file

   file = open('Salary_Data.csv')

   type(file)

3. Use the csv.reader object to read the csv file

   csvreader = csv.reader(file)

4. Extract the field names

   Make a header list that is empty. To get the header, use the next() method.

   The.next() method advances to the following row while returning the current row.

   When you call next() for the first time, it returns the header; the second time, it returns the first record; and so on.

   header = []

   header = next(csvreader)

   header

5. Extract the rows/records

   As you iterate over the csvreader object, create an empty list called rows and append each row to the rows list.

   rows = []

   for row in csvreader:

   rows.append(row)

   rows

6. Close the file

   The opened file is closed using the.close() method. Once closed, we are unable to operate on it in any way.

   file.close()

Steps of reading CSV files using pandas

1. Import pandas library

   import pandas as pd

2. Load CSV files to pandas using read_csv()

   Basic Syntax: pandas.read_csv(filename, delimiter=',')

   data= pd.read_csv("Salary_Data.csv")

3. Extract the field names

   .columns is used to obtain the header/field names.

   data.columns

4. Extract the rows

   All the data of a data frame can be accessed using the field names.

   data.Salary

9.6 Operations Using Data Frame

The datasets will be loaded into a Pandas Data Frame in the real world from existing storage, which may be a SQL database, a CSV file, or an Excel file. You can generate a Pandas Data Frame from lists, dictionaries, and lists of dictionaries, for example. A data frame can be constructed in a variety of ways. The following are some examples:

Creating Data Frame using List: You can generate a Data Frame from a single list or from a list of lists.

Program	Output
# import pandas as pd	0
import pandas as pd	0 This
	1 is
# list of strings	2 my
lst = ['This', 'is', 'my', 'File',	3 File
'read', 'it', 'carefully']	4 read
	5 it
# Calling DataFrame constructor on list	6 carefully
df = pd.DataFrame(lst)	
print(df)	

Creating Data Frame from dict of ndarray/lists:

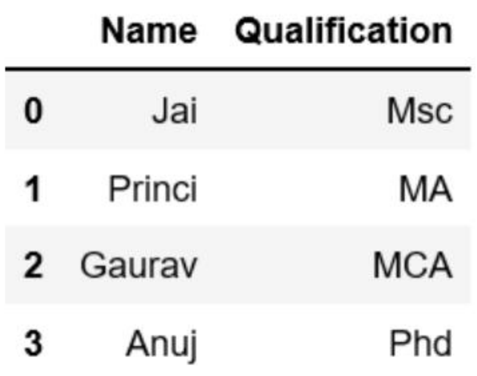
The length of each narray must be the same in order to generate a DataFrame from a dict of narray/list. If index is supplied, the length index must match the length of the arrays. If no index is provided, range(n), where n is the length of the array, will be used as the default index.

Program	Output
# Python code demonstrate creating	Name Age
# DataFrame from dictnarray / lists	0 Tom 20
# By default addresses.	1 nick 21
	2 krish 19
import pandas as pd	3 jack 18
# intialise data of lists.	
data = {'Name':['Tom', 'nick', 'krish', 'jack'],	
'Age':[20, 21, 19, 18]}	
# Create DataFrame	
df = pd.DataFrame(data)	
# Print the output.	
print(df)	

Data is arranged in rows and columns in a data frame, which is a two-dimensional data structure. Basic operations like selecting, removing, adding, and renaming can be done on rows and columns.

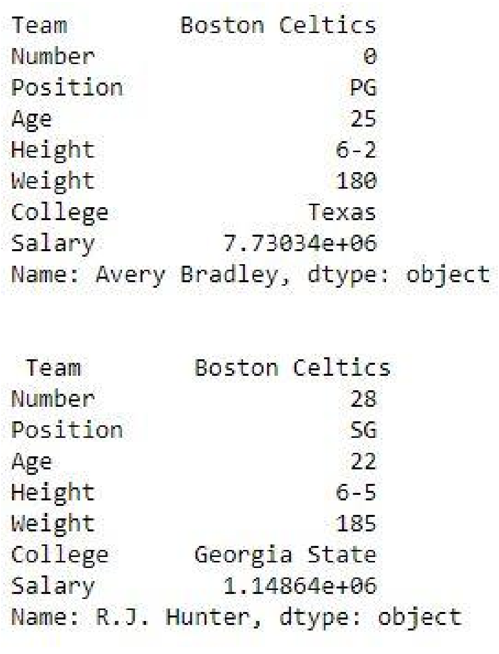
Column selection: We have two options for accessing columns in a Pandas DataFrame in order to choose one of them.

Program	Output

# Import pandas package 	
import pandas as pd
 	
# Define a dictionary containing employee data 	
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}
 	
# Convert the dictionary into DataFrame 	
df = pd.DataFrame(data)
 	
# select two columns 	
print(df[['Name', 'Qualification']])
	
	  Name Qualification
	0 Jai Msc
	1 Princi MA
	2 Gaurav MCA
	3 Anuj Phd

Row Selection:Rows can be retrieved from a Data frame using a special mechanism that Pandas offers. Rows from a Pandas DataFrame are retrieved using the DataFrame.loc method. Additionally, rows can be chosen by giving an integer location to the iloc[] method.

Program	Output

# importing pandas package 	
import pandas as pd
 	
# making data frame from csv file 	
data = pd.read_csv("nba.csv", index_col ="Name")
 	
# retrieving row by loc method 	
first = data.loc["Avery Bradley"]
second = data.loc["R.J. Hunter"]
 	
print(first, "\n\n\n", second)
	
	Team Boston Celtics
	Number 0
	Position PG
	Age 25
	Height 6-2
	Weight 180
	College Texas
	Salary 7.73034e+06
	Name: Avery Bradley, dtype: object
	
	 Team Boston Celtics
	Number 28
	Position SG
	Age 22
	Height 6-5
	Weight 185
	College Georgia State
	Salary 1.14864e+06
	Name: R.J. Hunter, dtype: object

Indexing and Selecting Data

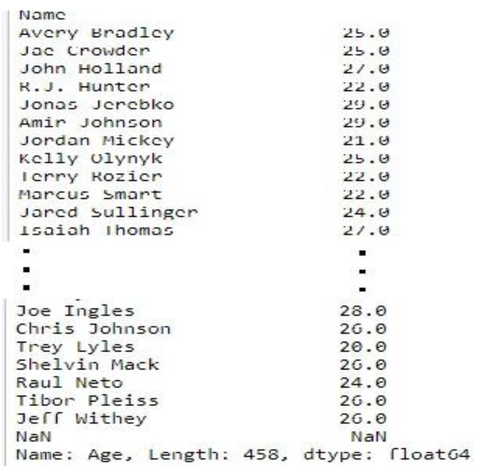
In Pandas, picking specific rows and columns of data from a DataFrame constitutes indexing. Selecting all the rows and part of the columns, some of the rows and all the columns, or a portion of each row and each column is what is referred to as indexing. Another name for indexing is subset selection.

Indexing a Data frame using indexing operator []:

The square brackets that come after an item are referred to by the indexing operator. The indexing operator is also used by the.locand.iloc indexers to make selections. To use the indexing operator df[in this sentence.

Selecting a single column

Simply place the name of the column between the brackets to pick only that one.

Program	Output
# importing pandas package import pandas as pd # making data frame from csv file data = pd.read_csv("nba.csv", index_col ="Name") # retrieving columns by indexing operator first = data["Age"] print(first)	

```
Name
Avery Bradley        25.0
Jae Crowder          25.0
John Holland         27.0
R.J. Hunter          22.0
Jonas Jerebko        29.0
Amir Johnson         29.0
Jordan Mickey        21.0
Kelly Olynyk         25.0
Terry Rozier         22.0
Marcus Smart         22.0
Jared Sullinger      24.0
Isaiah Thomas        27.0
.                     .
.                     .
.                     .
Joe Ingles           28.0
Chris Johnson        26.0
Trey Lyles           20.0
Shelvin Mack         26.0
Raul Neto            24.0
Tibor Pleiss         26.0
Jeff Withey          26.0
NaN                   NaN
Name: Age, Length: 458, dtype: float64
```

Indexing a DataFrame using. loc[ ]:

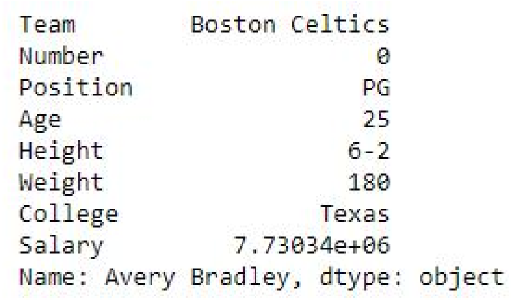
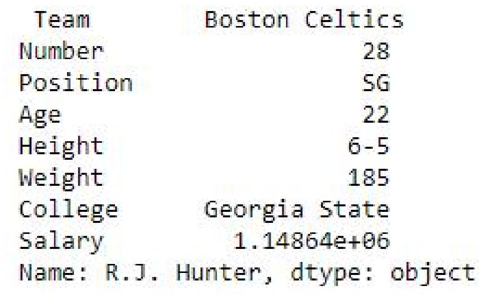
The label of the rows and columns is used to pick data in this function. Data is chosen by the df.loc indexer in a different way than by the indexing operator alone. Subsets of rows or columns may be chosen. Additionally, it can choose a subset of both rows and columns at once.

Selecting a single row

We provide a single row label in a.loc function in order to select a single row utilising that function.

Output:

Since there was only one parameter both times, two series were returned, as seen in the output image.

# importing pandas package 	
import pandas as pd
 	
# making data frame from csv file 	
data = pd.read_csv("nba.csv", index_col ="Name")
 	
# retrieving row by loc method 	
first = data.loc["Avery Bradley"]
second = data.loc["R.J. Hunter"]
 	
print(first, "\n\n\n", second)
	
	Team Boston Celtics
	Number 0
	Position PG
	Age 25
	Height 6-2
	Weight 180
	College Texas
	Salary 7.73034e+06
	Name: Avery Bradley, dtype: object
	
	
	Team Boston Celtics
	Number 28
	Position SG
	Age 22
	Height 6-5
	Weight 185
	College Georgia State
	Salary 1.14864e+06
	Name: R.J. Hunter, dtype: object

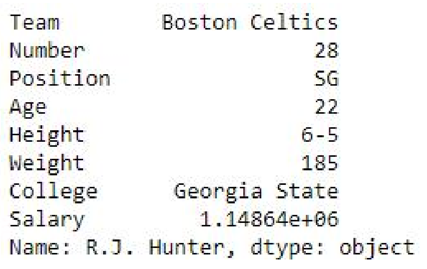
Indexing a DataFrameusing .iloc[ ] :

We can retrieve rows and columns by position using this function. We must give both the desired places for the desired rows and columns in order to accomplish that. While the df.iloc indexer is quite similar to df.loc, it only selects integer locations.

Selecting a single row

We can supply a single integer to the.iloc[] function in order to use it to choose a single row.

Program	Output

import pandas as pd
# making data frame from csv file 	
data = pd.read_csv("nba.csv", index_col ="Name")
# retrieving rows by iloc method 	
row2 = data.iloc[3]
print(row2)
	
	Team Boston Celtics
	Number 28
	Position SG
	Age 22
	Height 6-5
	Weight 185
	College Georgia State
	Salary 1.14864e+06
	Name: R.J. Hunter, dtype: object

Summary

- A one-dimensional labelled array called a series can store any kind of data (integer, string, float, python objects, etc.). The term index refers to all of the axis labels.
- The index passed must have the same length as data if the data is an ndarray. If no index is provided, range(n), where n is the array length, will be used as the default index, which is [0,1,2,3.. range(len(array))-1].
- The values are identified with their index number if nothing else is supplied. The index of the first item is 0, that of the second is 1, etc.
- You are able to name your own labels using the index option.
- The term "Comma Separated Values" or CSV. It is the most basic way to save tabular data as plain text. Because we as data scientists usually use CSV data in our daily work, it is crucial to know how to work with it.

- The list is automatically sorted ascending using the sort() method. Making a function to select the sorting criteria is another option (s).
- A 2-dimensional labelled data structure called a "DataFrame" has columns that could be of many sorts. It can be compared to a spreadsheet, SQL table, or dictionary of Series objects. It is typically the pandas object that is used the most.
- Data is arranged in rows and columns in a data frame, which is a two-dimensional data structure.

Keywords

Dataframe:A Pandas DataFrame is a two-dimensional data structure having rows and columns, similar to a two-dimensional array

Series: A Pandas Series resembles a table's column. It is a one-dimensional array that can hold any kind of data.

Labels:The values are identified with their index number if nothing else is supplied. The index of the first item is 0, that of the second is 1, etc.

Key/Value Object: When constructing a Series, you can also utilise a key/value object like a dictionary.

DataFrames:In Pandas, data sets are often multidimensional tables, or "DataFrames."

CSV Files: Using CSV files is an easy approach to store large data sets (comma separated files).

Max_rows:The Pandas option settings control how many rows are returned.

Column_Selection: We have two options for accessing the columns in a Pandas DataFrame in order to choose one of them.

Row_Selection: Rows from a Pandas DataFrame are retrieved using the DataFrame.loc method.

Indexing Operator: The indexing operator is also used by the.locand.iloc indexers to make selections.

Dropna(): We used the dropna() method to remove null values from a dataframe. This function removes rows and columns of datasets containing null values in several ways.

Self Assessment

1. Which of the following is not true about DataFrame?

A. A dataframe can be created by passing dictionaries
B. A dataframe is size immutable
C. A dataframe index can be string
D. A column of dataframe can have different types

2. In Pndas_____________ is used to store data in multiple columns.

A. Series
B. DataFrame
C. Both of the above
D. None of the above

3. A ___________ is a two-dimensional labelled data structure.

A. DataFrame
B. Series
C. List

D. None of the above

4. ____________data structure has both a row and column index

A. DataFrame

B. Series

C. List

D. None of the above

5. Which library is imported for creating DataFrame?

A. Python

B. DataFrame

C. Pandas

D. Random

6. What does the following code produce as output?

Import pandas as pd

D1=pd.DataFrame([1,2,3])

A. 1

B. 4

C. 3

D. 2

7. We can create a DataFrame from ____________

A. Numpy Arrays

B. List of Dictionaries

C. Dictionaries of Lists

D. All of the above

8. Which of the following is used to give user defined column index in DataFrame?

A. index

B. column

C. columns

D. colindex

9. What does the following code produce as output?

#import pandas as pd

LoD = [{'a':10, 'b':20}, {'a':5, 'b':10, 'c':20}]

D1=pd.DataFrame(LoD)

A. 1

B. 2

C. 3

D. 4

10. In regards to separated value files such as .csv and. tsv, what is the delimiter?

A. Delimiters are not used in separated value files

B. Any character such as the comma (,) or tab (\t) that is used to separate the column data.

C. Anywhere the comma (,) character is used in the file

D. Any character such as the comma (,) or tab (\t) that is used to separate the row data

11. In separated value files such as .csv and .tsv, what does the first row in the file typically contain?

A. The source of the data

B. The column names of the data

C. Notes about the table data

D. The author of the table data

12. When iterating over an object returned from csv.reader(), what is returned with each iteration?

For example, given the following code block that assumes csv_reader is an object returned from csv.reader(), what would be printed to the console with each iteration?

```
for item in csv_reader:
print(item)
```

A. The full line of the file as a string

B. The column data as a list

C. The row data as a list

D. The individual value data that is separated by the delimiter.

13. When we create Data Frame from Dictionary of List then Keys becomes the ___________

A. Row Labels

B. Column Labels

C. Both of the above

D. None of the above

14. Data Frame created from a single Series has ______ column

```
# Import math Library
import math
# Return the value of 9 raised to the power of 3
print (math.pow (4, 3))1
```

A. 1

B. 2

C. n (n is the number of elements in the series)

D. None of above

15. What do you understand by pandas? Explain use of pandas with example along with installation procedure.
16. What do you understand by CSV file? Explain the steps to read a CSV file.
17. Explain with example creation of DataFrame from dict of ndarrays.
18. Explain column selection and row selection in a DataFrame with examples.
19. Difference between. loc and. iloc function using example.

Answers for Self Assessment

1.	B	2.	B	3.	A	4.	A	5.	A
6.	C	7.	A	8.	A	9.	C	10.	C
11.	B	12.	B	13.	C	14.	B	15.	A

Further Readings

- Mark Lutz,Programming Python: Powerful Object-Oriented Programming, OREILLY
- Wes McKinney, Python for data analysis, OREILLY
- David Ascher and Mark Lutz, Learning Python, OREILLY
- Eric Matthes, Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming, Starch Pres

Web Links

- https://www.tutorialspoint.com/python/index.htm
- https://www.python.org/downloads/
- https://www.w3schools.in/python/data-types
- https://www.programiz.com/python-programming/online-compiler/
- https://www.codecademy.com/catalog/language/python

Unit 10: Data Cleanup

CONTENTS

Objectives

After this unit, student would be able to:

- learn basic concepts about data cleanup

Introduction

Missing data is a constant issue in real-world situations. The accuracy of model predictions in fields like machine learning and data mining is severely hampered by the poor quality of the data that missing values produce. To improve the accuracy and validity of their models in these fields, missing value treatment is a prominent area of focus.

10.1 When and Why Do Records Get Lost?

Let's have a look at an online product survey. People frequently don't disclose all the information pertaining to them. Few people disclose their experience, but not the length of time they have been using the product; few people disclose their experience, but not their contact information. As a result, some data is always missing in some form, and this occurs frequently in real time.

Let's now examine how Pandas can be used to handle missing values, such as NA or NaN.

Program	Output

```
# import the pandas library
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',
```
	```
	  one       two       three
	a  0.077988  0.476149  0.965836
	b       NaNNaNNaN
	c -0.390208 -0.551605 -2.301950
	d       NaNNaNNaN
	e -2.000303 -0.788201  1.510072
	```

'h'],columns=['one', 'two', 'three'])	f -0.930230 -0.670473 1.146615
	g NaNNaNNaN
df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])	h 0.085100 0.532791 0.887415
print df	

We have produced a DataFrame with missing values using reindexing. NaN stands for Not a Number in the output.

Check for Missing Values

The isnull() and notnull() functions, which are also methods on Series and DataFrame objects, are provided by Pandas to make identifying missing values more straightforward (and across various array dtypes).

Program	Output
import pandas as pd	a False
import numpy as np	b True
df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',	c False
	d True
'h'],columns=['one', 'two', 'three'])	e False
df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])	f False
print df['one'].isnull()	g True
	h False
	Name: one, dtype: bool

Cleaning / Filling Missing Data

Pandas offers several cleaning techniques for the missing values. The following sections provide examples of how the fillna function can "fill in" NA values with non-null data in a few different ways.

Replace NaN with a Scalar Value

The software that follows demonstrates how to change "NaN" to "0."

Program	Output
import pandas as pd	one two three
import numpy as np	a -0.576991 -0.741695 0.553172
df = pd.DataFrame(np.random.randn(3, 3), index=['a', 'c', 'e'],columns=['one',	b NaNNaNNaN
	c 0.744328 -1.735166 1.749580
'two', 'three'])	
df = df.reindex(['a', 'b', 'c'])	NaN replaced with '0':
print df	one two three
print ("NaN replaced with '0':")	a -0.576991 -0.741695 0.553172
print df.fillna(0)	b 0.000000 0.000000 0.000000
	c 0.744328 -1.735166 1.749580

The value zero is being filled in here, but any other value may be used.

Fill NA Forward and Backward

We will fill in the missing data using the filling principles covered in the ReIndexing.

Method	Action
Pad/fill	fill methods forward
Bfill/backfill	fill methods backward

Program	Output
import pandas as pd **import numpy as np** **df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',** **'h'],columns=['one', 'two', 'three'])** **df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])** **print df.fillna(method='pad')**	one two three a 0.077988 0.476149 0.965836 b 0.077988 0.476149 0.965836 c -0.390208 -0.551605 -2.301950 d -0.390208 -0.551605 -2.301950 e -2.000303 -0.788201 1.510072 f -0.930230 -0.670473 1.146615 g -0.930230 -0.670473 1.146615 h 0.085100 0.532791 0.887415

10.2 Drop Missing Values

Use the dropna method and the axis argument if you just want to ignore the missing values. A row is removed from consideration if any value within it is NA by default because axis=0, or along row.

Program	Output
import pandas as pd **import numpy as np** **df = pd.DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e', 'f',** **'h'],columns=['one', 'two', 'three'])** **df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])** **print df.dropna()**	one two three a 0.077988 0.476149 0.965836 c -0.390208 -0.551605 -2.301950 e -2.000303 -0.788201 1.510072 f -0.930230 -0.670473 1.146615 h 0.085100 0.532791 0.887415

10.3 Replace Missing (or) Generic Values

We frequently need to swap out a generic value with a different value. Applying the replace approach will allow us to accomplish this.

The fillna() function behaves in a manner analogous to replacing NA with a scalar value.

import pandas as pd import numpy as np df = pd.DataFrame({'one':[10,20,30,40,50,2000], 'two':[1000,0,30,40,50,60]}) print df.replace({1000:10,2000:60})	one two 0 10 10 1 20 0 2 30 30 3 40 40 4 50 50 5 60 60

10.4 Data Cleaning with Python

We will now guide you through the set of activities indicated below using Pandas and NumPy. We'll offer a very brief overview of the assignment before describing the required code using the terms INPUT (what you should enter) and OUTPUT (what you should see as a result).

The basic data cleansing chores that we'll take on are as follows:

1. Importing Libraries
2. Input Customer Feedback Dataset
3. Locate Missing Data
4. Check for Duplicates
5. Detect Outliers
6. Normalize Casing

1. Importing Libraries

Let's get your Python script going with NumPy and Pandas installed.

```
import pandas as pd
import numpy as np
```

In this situation, the libraries ought to be loaded into your

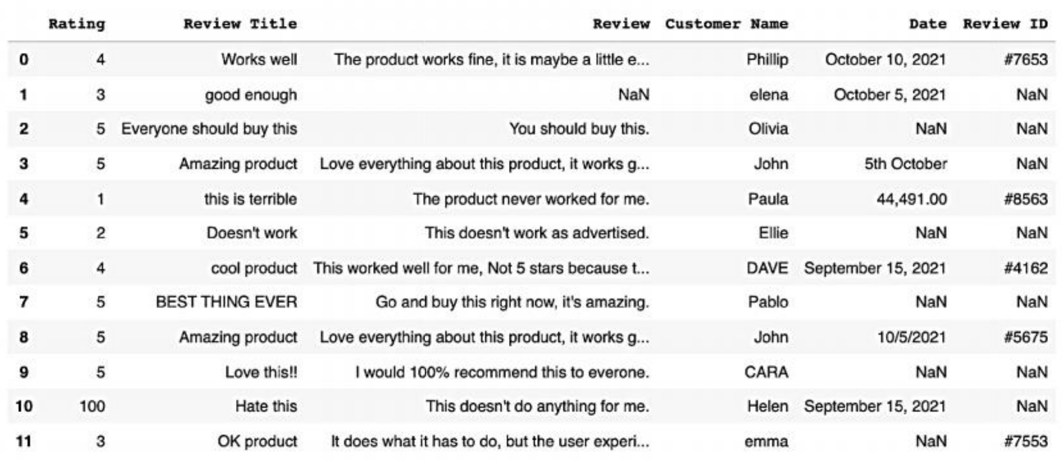
2. Input Customer Feedback Dataset

The feedback dataset is then read by our libraries. Let's have alook at that.

Input

```
data = pd.read_csv('feedback.csv')
```

Output:



	Rating	Review Title	Review	Customer Name	Date	Review ID
0	4	Works well	The product works fine, it is maybe a little e...	Phillip	October 10, 2021	#7653
1	3	good enough	NaN	elena	October 5, 2021	NaN
2	5	Everyone should buy this	You should buy this.	Olivia	NaN	NaN
3	5	Amazing product	Love everything about this product, it works g...	John	5th October	NaN
4	1	this is terrible	The product never worked for me.	Paula	44,491.00	#8563
5	2	Doesn't work	This doesn't work as advertised.	Ellie	NaN	NaN
6	4	cool product	This worked well for me, Not 5 stars because t...	DAVE	September 15, 2021	#4162
7	5	BEST THING EVER	Go and buy this right now, it's amazing.	Pablo	NaN	NaN
8	5	Amazing product	Love everything about this product, it works g...	John	10/5/2021	#5675
9	5	Love this!!	I would 100% recommend this to everone.	CARA	NaN	NaN
10	100	Hate this	This doesn't do anything for me.	Helen	September 15, 2021	NaN
11	3	OK product	It does what it has to do, but the user experi...	emma	NaN	#7553

As you can see, the dataset you wish to look at is "feedback.csv". And in this instance, we know we are utilizing the Pandas library to read our dataset as we see "pd.read_csv" as the prior function.

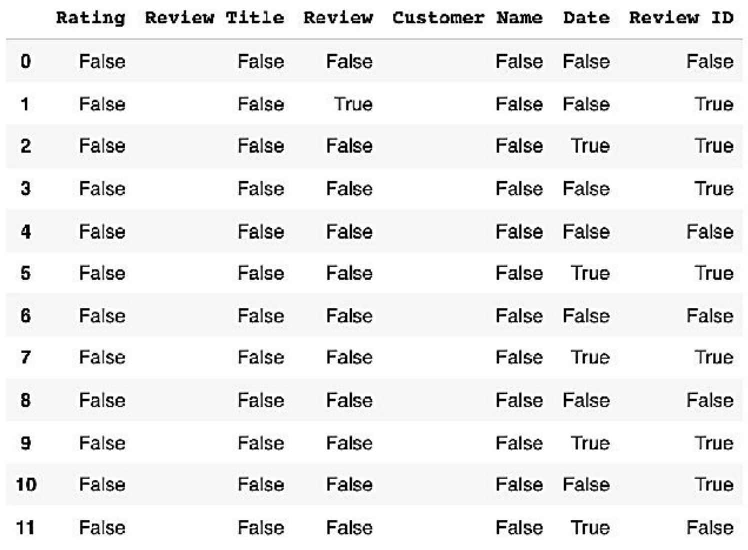
3. Locate Missing Data

The isnull function, a sneaky Python exploit, will then be used to find our data. Actually a common function, "isnull" aids in locating missing items in our collection. This information is helpful since it shows what has to be fixed throughout the data cleaning process.

Input

data.isnull()

Output

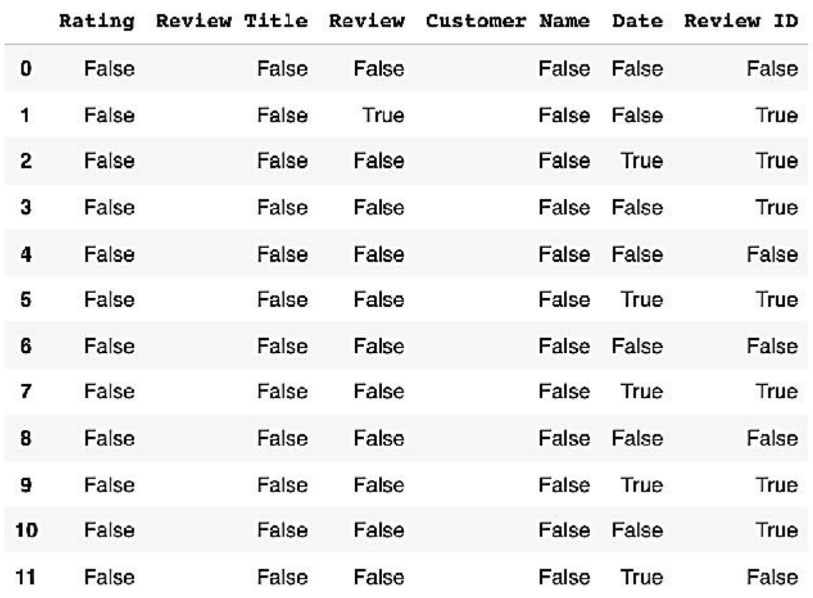


	Rating	Review Title	Review	Customer Name	Date	Review ID
0	False	False	False	False	False	False
1	False	False	True	False	False	True
2	False	False	False	False	True	True
3	False	False	False	False	False	True
4	False	False	False	False	False	False
5	False	False	False	False	True	True
6	False	False	False	False	False	False
7	False	False	False	False	True	True
8	False	False	False	False	False	False
9	False	False	False	False	True	True
10	False	False	False	False	False	True
11	False	False	False	False	True	False

We get a collection of boolean values as our output result.

The list can provide us with a variety of insights. The first thing to consider is where the missing data is; any column with a 'True' reading denotes that the data file's category for that column contains missing data.

Datapoint 1 has missing information in its Review section and Review ID section, for instance (both are marked true).



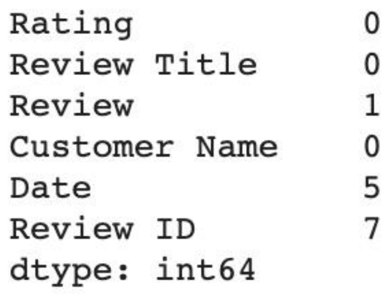
	Rating	Review Title	Review	Customer Name	Date	Review ID
0	False	False	False	False	False	False
1	False	False	True	False	False	True
2	False	False	False	False	True	True
3	False	False	False	False	False	True
4	False	False	False	False	False	False
5	False	False	False	False	True	True
6	False	False	False	False	False	False
7	False	False	False	False	True	True
8	False	False	False	False	False	False
9	False	False	False	False	True	True
10	False	False	False	False	False	True
11	False	False	False	False	True	False

Each feature's missing data can be expanded further by coding:

INPUT:

data.isnull().sum()

OUTPUT:



```
Rating           0
Review Title     0
Review           1
Customer Name    0
Date             5
Review ID        7
dtype: int64
```

From here, we really sanitise the data using code. There are only two primary alternatives here. Either remove the data or enter the blanks. If you decide to:

a. *Drop the data*

Another choice will need to be made: to maintain the data in the set while simply dropping the missing values, or to completely remove the feature (the entire column) because there are so many missing datapoints that it is unusable for analysis.

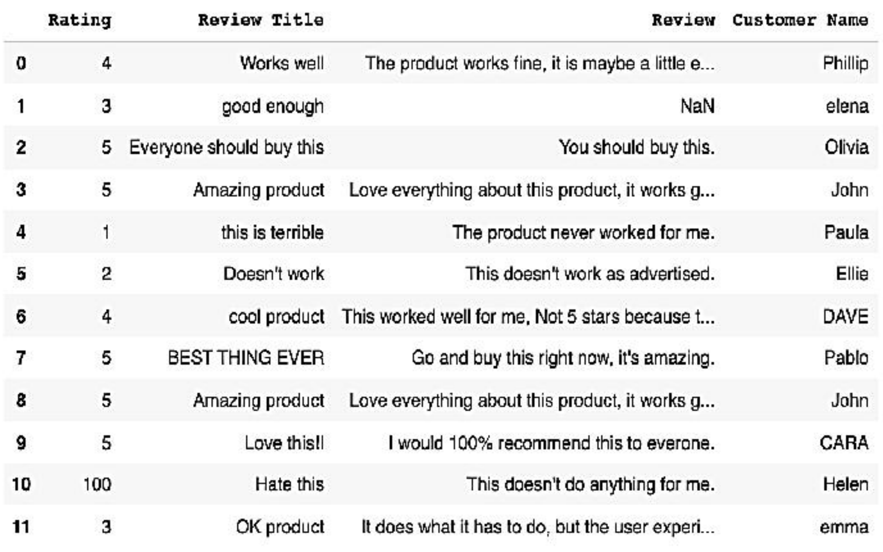
You must go in and label the missing values as void in accordance with Pandas or NumBy standards if you want to remove them (see section below). However, this is the code to remove the full column:

INPUT:

remove = ['Review ID','Date']

data.drop(remove, inplace =True, axis =1)

OUTPUT:



	Rating	Review Title	Review	Customer Name
0	4	Works well	The product works fine, it is maybe a little e...	Phillip
1	3	good enough	NaN	elena
2	5	Everyone should buy this	You should buy this.	Olivia
3	5	Amazing product	Love everything about this product, it works g...	John
4	1	this is terrible	The product never worked for me.	Paula
5	2	Doesn't work	This doesn't work as advertised.	Ellie
6	4	cool product	This worked well for me, Not 5 stars because t...	DAVE
7	5	BEST THING EVER	Go and buy this right now, it's amazing.	Pablo
8	5	Amazing product	Love everything about this product, it works g...	John
9	5	Love this!!	I would 100% recommend this to everone.	CARA
10	100	Hate this	This doesn't do anything for me.	Helen
11	3	OK product	It does what it has to do, but the user experi...	emma

Now, let's examine our other option.

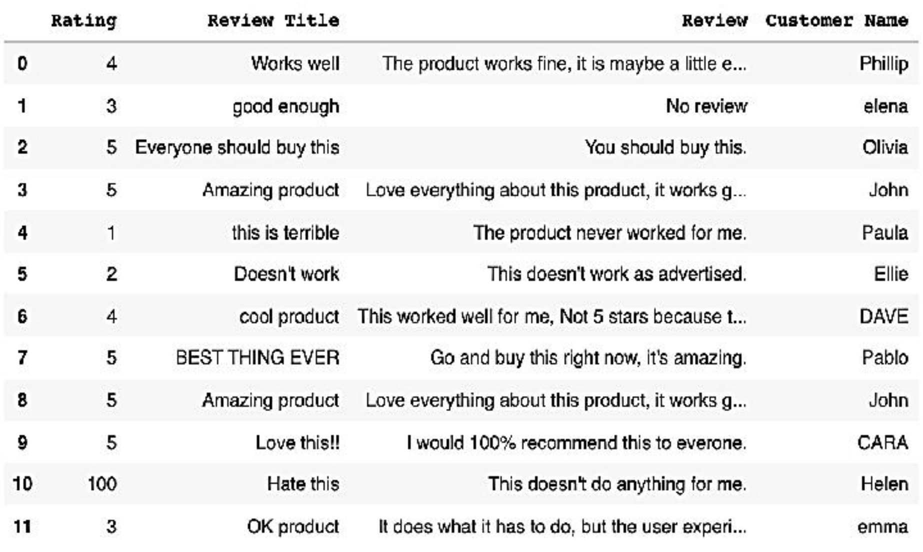
b. *Input missing data*

Technically speaking, adding individual values using Pandas or NumBy standards is the same as adding missing data; we refer to it as adding "No Review." When entering missing data, you have two options: manually enter the right information or add "No Review" using the code below.

INPUT

```
data['Review'] = data['Review'].fillna('No review')
```

OUTPUT



	Rating	Review Title	Review	Customer Name
0	4	Works well	The product works fine, it is maybe a little e...	Phillip
1	3	good enough	No review	elena
2	5	Everyone should buy this	You should buy this.	Olivia
3	5	Amazing product	Love everything about this product, it works g...	John
4	1	this is terrible	The product never worked for me.	Paula
5	2	Doesn't work	This doesn't work as advertised.	Ellie
6	4	cool product	This worked well for me, Not 5 stars because t...	DAVE
7	5	BEST THING EVER	Go and buy this right now, it's amazing.	Pablo
8	5	Amazing product	Love everything about this product, it works g...	John
9	5	Love this!!	I would 100% recommend this to everone.	CARA
10	100	Hate this	This doesn't do anything for me.	Helen
11	3	OK product	It does what it has to do, but the user experi...	emma

As you can see, data point 1 has been successfully designated.

4. Check for Duplicates

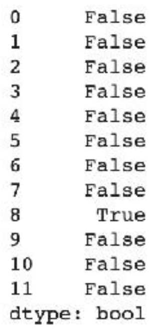
Similar to missing data, duplicates are problematic and choke analytics tools. Let's find them and get rid of them.

In order to find duplicates, we start with:

INPUT:

data.duplicated()

OUTPUT:



```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8      True
9     False
10    False
11    False
dtype: bool
```

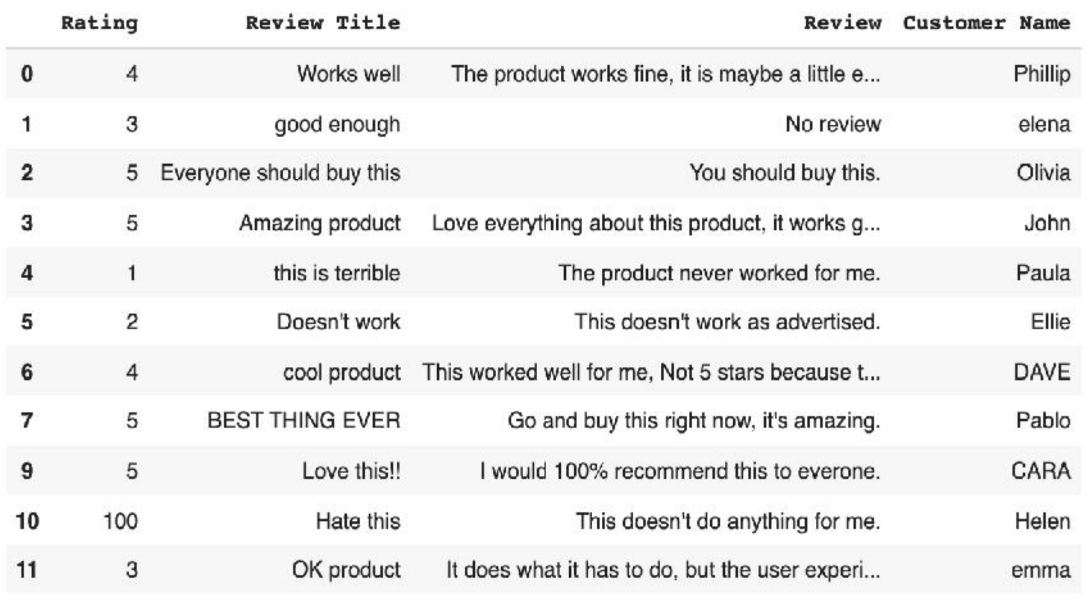
Also known as a list of boolean values with duplicate values indicated by a 'True' reading.

Let's move forward and eliminate that duplicate (datapoint 8).

INPUT:

data.drop_duplicates()

OUTPUT:



	Rating	Review Title	Review	Customer Name
0	4	Works well	The product works fine, it is maybe a little e...	Phillip
1	3	good enough	No review	elena
2	5	Everyone should buy this	You should buy this.	Olivia
3	5	Amazing product	Love everything about this product, it works g...	John
4	1	this is terrible	The product never worked for me.	Paula
5	2	Doesn't work	This doesn't work as advertised.	Ellie
6	4	cool product	This worked well for me, Not 5 stars because t...	DAVE
7	5	BEST THING EVER	Go and buy this right now, it's amazing.	Pablo
9	5	Love this!!	I would 100% recommend this to everone.	CARA
10	100	Hate this	This doesn't do anything for me.	Helen
11	3	OK product	It does what it has to do, but the user experi...	emma

Our dataset with our duplicate deleted is now available. Onwards.

5. Detect Outliers

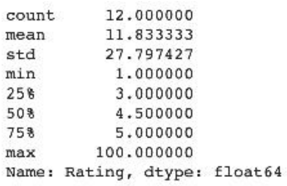
Numerical values that are significantly beyond the statistical norm are considered outliers. They are data points that are sufficiently out of range that they are probably misreads, to cut down on superfluous science jargon.

They must be eliminated, just like duplicates. Pulling up our dataset first, let's look for an outlier.

INPUT:

data['Rating'].describe()

OUTPUT:



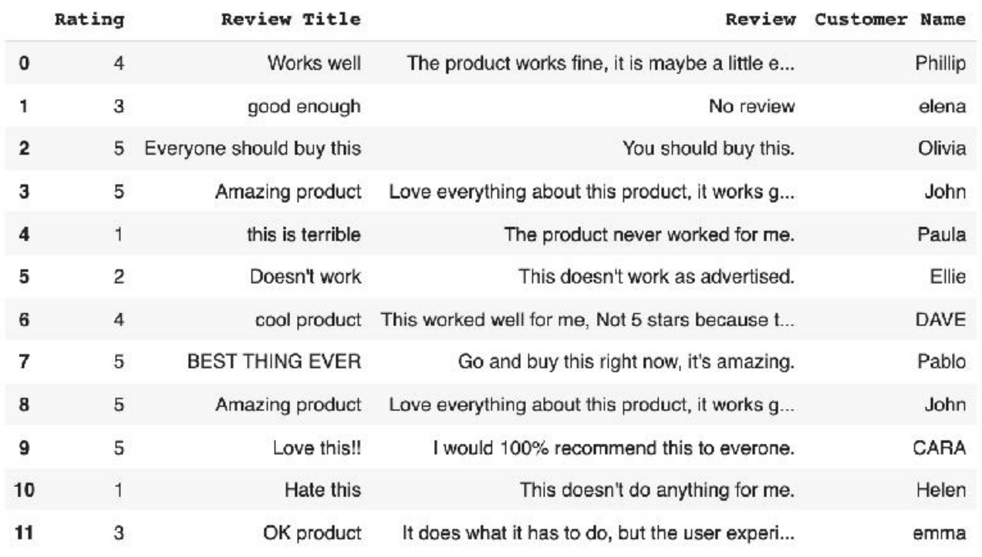
```
count     12.000000
mean      11.833333
std       27.797427
min        1.000000
25%        3.000000
50%        4.500000
75%        5.000000
max      100.000000
Name: Rating, dtype: float64
```

Look at that "max" value; none of the other values, including the mean (average), are even close to 100. Your understanding of your dataset will now determine how you will address outliers. The data scientists who entered the knowledge in this instance are aware that they meant to enter a value of 1, not 100. In order to correct our data, we may safely delete the outlier.

INPUT:

data.loc[10,'Rating'] = 1

OUTPUT:



	Rating	Review Title	Review	Customer Name
0	4	Works well	The product works fine, it is maybe a little e...	Phillip
1	3	good enough	No review	elena
2	5	Everyone should buy this	You should buy this.	Olivia
3	5	Amazing product	Love everything about this product, it works g...	John
4	1	this is terrible	The product never worked for me.	Paula
5	2	Doesn't work	This doesn't work as advertised.	Ellie
6	4	cool product	This worked well for me, Not 5 stars because t...	DAVE
7	5	BEST THING EVER	Go and buy this right now, it's amazing.	Pablo
8	5	Amazing product	Love everything about this product, it works g...	John
9	5	Love this!!	I would 100% recommend this to everone.	CARA
10	1	Hate this	This doesn't do anything for me.	Helen
11	3	OK product	It does what it has to do, but the user experi...	emma

Now that our dataset only contains ratings between 1 and 5, there won't be any big distortion caused by a single errant 100.

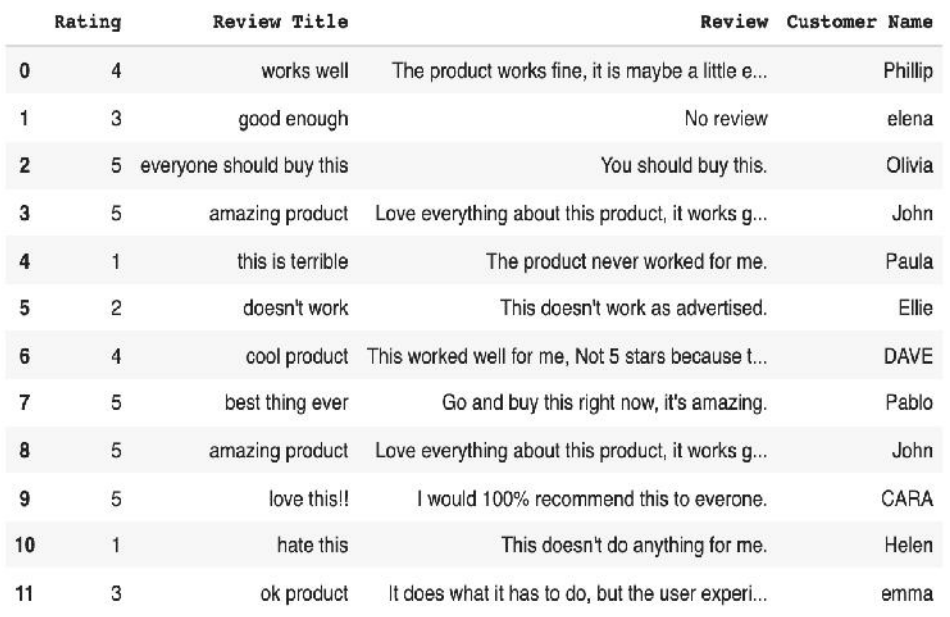
6. Normalize Casing

Last but not least, we'll cross our ts and dot our i's. Meaning that we will uppercase Customer Names so that our algorithms can recognise them as variables and standardise (lowercase) all review titles to prevent confusing our algorithms.Here’s how to make every review title lowercase:

INPUT

data['Review Title'] = data['Review Title'].str.lower()

OUTPUT



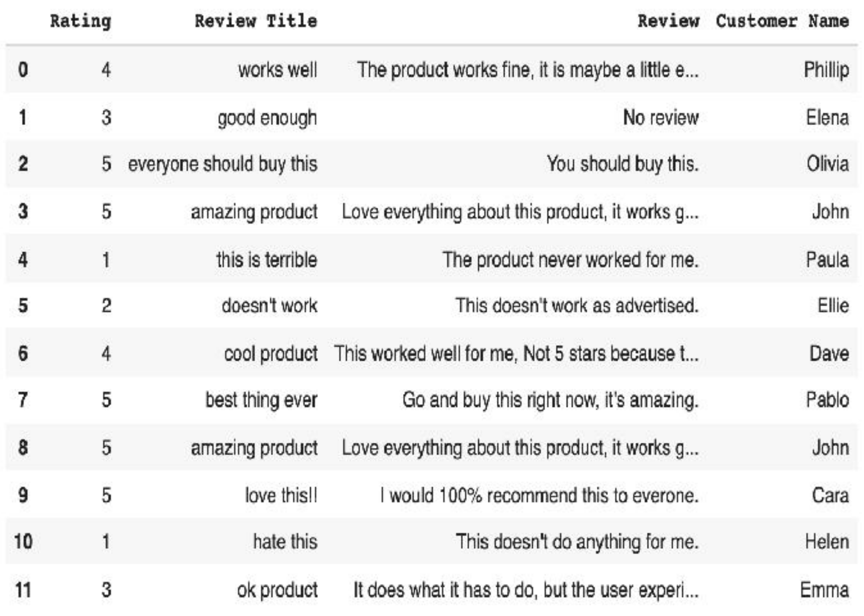
	Rating	Review Title	Review	Customer Name
0	4	works well	The product works fine, it is maybe a little e...	Phillip
1	3	good enough	No review	elena
2	5	everyone should buy this	You should buy this.	Olivia
3	5	amazing product	Love everything about this product, it works g...	John
4	1	this is terrible	The product never worked for me.	Paula
5	2	doesn't work	This doesn't work as advertised.	Ellie
6	4	cool product	This worked well for me, Not 5 stars because t...	DAVE
7	5	best thing ever	Go and buy this right now, it's amazing.	Pablo
8	5	amazing product	Love everything about this product, it works g...	John
9	5	love this!!	I would 100% recommend this to everone.	CARA
10	1	hate this	This doesn't do anything for me.	Helen
11	3	ok product	It does what it has to do, but the user experi...	emma

Looks fantastic! Now let's make sure that none of our sophisticated software misclassifies a customer name since it isn't capitalised. How to capitalise "Customer Name" correctly is as follows:

INPUT:

```
data['Customer Name'] = data['Customer Name'].str.title()
```

OUTPUT:



	Rating	Review Title	Review	Customer Name
0	4	works well	The product works fine, it is maybe a little e...	Phillip
1	3	good enough	No review	Elena
2	5	everyone should buy this	You should buy this.	Olivia
3	5	amazing product	Love everything about this product, it works g...	John
4	1	this is terrible	The product never worked for me.	Paula
5	2	doesn't work	This doesn't work as advertised.	Ellie
6	4	cool product	This worked well for me, Not 5 stars because t...	Dave
7	5	best thing ever	Go and buy this right now, it's amazing.	Pablo
8	5	amazing product	Love everything about this product, it works g...	John
9	5	love this!!	I would 100% recommend this to everone.	Cara
10	1	hate this	This doesn't do anything for me.	Helen
11	3	ok product	It does what it has to do, but the user experi...	Emma

And there it is—our data collection complete with all the fixings. Or, more accurately, with all the fixes: To find and remove inaccurate data and normalise the remaining data, we made good use of logical Python packages.

Summary

- The practise of correcting or deleting inaccurate, damaged, improperly formatted, duplicate, or incomplete data from a dataset is known as data cleaning.

- The process of converting data from one format or structure to another is known as data transformation.
- Remove duplicate or pointless observations as well as undesirable observations from your dataset. The majority of duplicate observations will occur during data gathering.
- When you measure or transfer data and discover odd naming conventions, typos, or incorrect capitalization, those are structural errors.
- There will frequently be isolated findings that, at first look, do not seem to fit the data you are evaluating.
- The main measure of how well-founded and likely accurate a concept, conclusion, or measurement is called validity.
- Similar to missing data, duplicates are problematic and choke analytics tools. Let's find them and get rid of them.
- Numerical values that are significantly beyond the statistical norm are considered outliers.
- Another choice will need to be made: to maintain the data in the set while simply dropping the missing values, or to completely remove the feature (the entire column) because there are so many missing datapoints that it is unusable for analysis.

Keywords

Data type constraints: Each column's values must belong to a specific data type, such as Boolean, numeric (integer or real), date, etc.

Range Constraints: Most of the time, dates or numbers must fall inside a specified range. In other words, they have minimum and/or maximum values that are acceptable.

Unique Constraints: A field, or a group of fields, must be distinct throughout a dataset. The same social security number cannot be shared by two people, for instance.

Set-Membership Constraints: A set of discrete values or codes is used to generate the values for each column. A person's sex, for instance, may be Female, Male, or Non-Binary.

Foreign-key Constraints: The more typical case of set membership is this. One table's column that has distinct values defines the set of values in another table's column. For instance, the "state" column in a US taxpayer database must be one of the US's recognized states or territories; the list of acceptable states and territories is kept in a separate State table. Foreign key is a word that was adopted from relational database terminology.

Regular expression patterns: It may occasionally be necessary to validate text fields in this manner. For instance, it might be necessary for phone numbers to follow the pattern (999) 999-9999.

Cross-field validation: A certain set of multi-field conditions must be true. For instance, in laboratory medicine, the differential white blood cell count's component parts must add up to 100. (Since they are all percentages). A patient's date of discharge from the hospital cannot be earlier than the date of admission in a hospital database.

Accuracy:The degree of conformity of a measure to a standard or a true value.

Completeness: The extent to which all necessary actions are known. Data cleansing techniques are usually never able to completely correct incompleteness since they cannot be used to infer information that was not originally recorded in the data.

Consistency: A set of measures' degree of system-to-system equivalence.

Uniformity: The extent to which a set of data measures are defined across all systems using the same units of measurement.

Duplicate Detection:An algorithm is needed for duplicate detection in order to determine whether the same thing is represented twice in the data.

Parsing:A parser determines whether a string of data complies with the specification for permitted data. This is comparable to how a parser deals with languages and grammars.

Statistical Methods:An expert may discover values that are unexpected and thus incorrect by analyzing the data using the mean, standard deviation, range, or clustering algorithms.

Self Assessment

1. Which of the following phrases describes the challenge of identifying abstract patterns (or structures) in unlabeled data?

A. Supervised learning
B. Unsupervised learning
C. Hybrid learning
D. Reinforcement learning

2. Which clustering method calls for the merging approach?

A. Partitioned
B. Naïve Bayes
C. Hierarchical
D. Both A and C

3. Self-organizing maps are another example of a ________ style of learning.

A. Supervised learning
B. Unsupervised learning
C. Missing data imputation
D. Both A & C

4. The total number of neonates in the example of predicting the number of births can be thought of as the ______________.

A. Features
B. Observation
C. Attribute
D. Outcome

5. Which of the following claims about the classification is accurate?

A. It is a measure of accuracy
B. It is a subdivision of a set
C. It is the task of assigning a classification
D. None of the above

6. How many different sorts of functions are there in data mining?

A. 5
B. 4
C. 2
D. 3

7. The ________________ is the analysis carried out to find the intriguing statistical correlation between associated -attributes value pairs.

A. Mining of association
B. Mining of correlation
C. Mining of clusters
D. All of the above

8. Which of the following can be characterized as a data object that deviates from the norm (or the model of available data)?
A. Evaluation Analysis
B. Outliner Analysis
C. Classification
D. Prediction

9. Which of the following statements about data cleaning is untrue?
A. It refers to the process of data cleaning
B. It refers to the transformation of wrong data into correct data
C. It refers to correcting inconsistent data
D. All of the above

10. The data mining system is categorised using:
A. Database technology
B. Information Science
C. Machine learning
D. All of the above

11. How many different types of data warehousing approaches are there to integrate heterogeneous databases?
A. 3
B. 4
C. 5
D. 2

12. Select different types of attributes.
A. Nominal
B. Ordinal
C. Interval
D. All of the above

13. Select the correct examples for nominal
A. ID Numbers, eye color, zip codes
B. Rankings, taste of potato chips, grades, height
C. Calendar dates, temperature in Celsius or Fahrenheit
D. The temperature in Kelvin, length time, counts

14. Examples of Ordinal can be
A. ID Numbers, eye color, zip codes
B. Rankings, taste of potato chips, grades, height

C. Calendar dates, temperature in Celsius or Fahrenheit
D. The temperature in Kelvin, length time, counts

15. Example of structured data are
A. Generally
B. Dimensionality
C. Resolution
D. All of the Above

Answers for Self Assessment

1.	B	2.	C	3.	B	4.	D	5.	B
6.	C	7.	B	8	B	9	D	10	D
11	A	12	D	13	A	14	B	15	D

Review Questions

1. What do you understand by data cleaning? explain best practices for data cleanig.
2. Explain with code how null values stored in pandas data frames.
3. Difference between structured and unstructured data.
4. What are the effect of missing values in prediction and also explain functions that are used to handle missing values.
5. Explain following with example.
   a. How to see first five rows of Data Frame in python
   b. Define data profiling.
   c. How t check the class of each variable in pandas DataFrame
   d. Write code to see the dimensions of a DataFrame in python.
   e. Explain data mining.

FurtherReadings

- Mark Lutz,Programming*Python: Powerful Object-Oriented Programming*, OREILLY
- Wes McKinney, *Python for data analysis*, OREILLY
- David Ascher and Mark Lutz, *Learning Python*, OREILLY
- Eric Matthes, Python Crash Course, 2nd Edition: *A Hands-On, Project-Based Introduction to Programming*, Starch Pres

Web Links

https://www.tutorialspoint.com/python/index.htm

https://www.python.org/downloads/

https://www.w3schools.in/python/data-types https://www.programiz.com/python-programming/online-compiler/

https://www.codecademy.com/catalog/language/python

Unit 11: Data Visualization

CONTENTS

Objectives

After this unit, the student would be able to:

- learn basic concepts of data visualizations
- understand how to draw line plots and multiple subplots
- understand matplotlib, bar chart, histogram, box, and whisker plot.

Introduction

When working with data, it can be challenging to fully comprehend your data if it is just presented in tabular form. We must visualize or represent our data visually to fully comprehend what it means, to properly clean it, and to choose the best models for it. This makes patterns, correlations, and trends more obvious that cannot be seen in data that is presented as a table or CSV file.

Data visualization is the act of using visual representations of our data to identify trends and relationships. We can utilize a variety of Python data visualization libraries, like Matplotlib, Seaborn, Plotly, etc., to do data visualization.

11.1 What is Data Visualization?

The study of how to visually represent data is known as data visualization. It effectively communicates findings from data by graphically plotting the data.

We can obtain a visual summary of our data via data visualization. The human mind processes and comprehends any given data more easily when it is presented with images, maps, and graphs. Both small and large data sets benefit from data visualization, but large data sets are where it shines because it is impossible to manually see, let alone process, and comprehend, all of our data.

Data Visualization in Python

Figure 1 Data Visualization

Python provides several plotting libraries, including Matplotlib, Seaborn, and many other data visualization tools with a variety of features for building educational, unique, and visually appealing plots to present data most simply and powerfully.

11.2 Matplotlib and Seaborn

But when ought either of us to be used? Let's do a comparative study to better comprehend this. The two well-known visualization libraries for Python, Matplotlib, and Seaborn, are compared in the table below.

Matplotlib	Seaborn
It is used to plot simple graphs like line charts, bar graphs, and so forth.	It can carry out complicated visualizations with fewer commands and is primarily used for statistics visualization.
It primarily utilizes datasets and arrays.	It is compatible with whole datasets.
Compared to Matplotlib, Seaborn is significantly more organized and practical and treats the entire dataset as a single entity.	With data arrays and frames, Matplotlib functions effectively. The figures and aces are seen as objects.
Seaborn is primarily used for statistical analysis and has more built-in themes.	For exploratory data analysis, Matplotlib is more customizable and works well with Pandas and Numpy.

Figure 2 Matplotlib vs Seaborn

11.3 Line Charts

An informational graph called a line chart shows data as a collection of dots connected by straight lines. Each marker or data point in a line chart is drawn and connected by a line or curve.

Let's think about the Kanto apple yield (tonnes per hectare). Using this information, let's create a line graph to show how the apple yield has changed over time. We begin by importing Seaborn and Matplotlib.

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Figure 3 Importing necessary modules

Using Matplotlib

To depict the yield of apples, we are utilizing arbitrary data points.

```
yield_apples = [0.895, 0.91, 0.919, 0.926, 0.929, 0.931]
plt.plot(yield_apples)
```

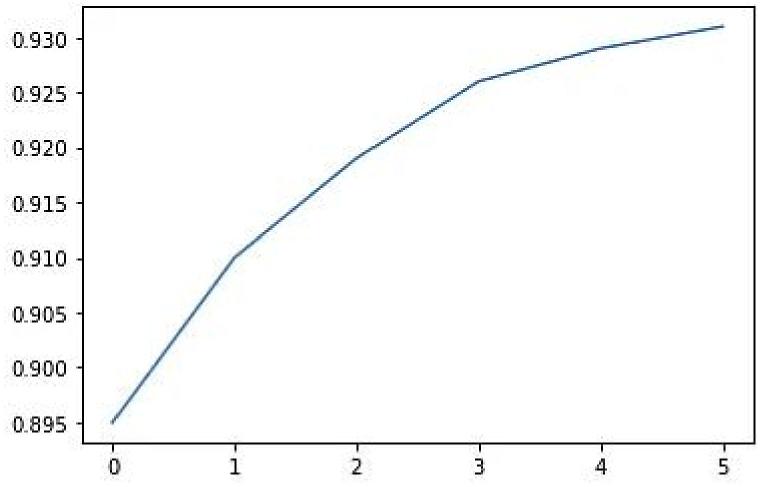


Figure 4 Plotting Apple Yield

We can also include the values for the x-axis to clarify the graph's meaning.

```
years=[2010, 2011, 2012, 2013, 2014, 2015]
yield_apples = = [0.895, 0.91, 0.919, 0.926, 0.929, 0.931]
plt.plot(years,yield_apples)
```

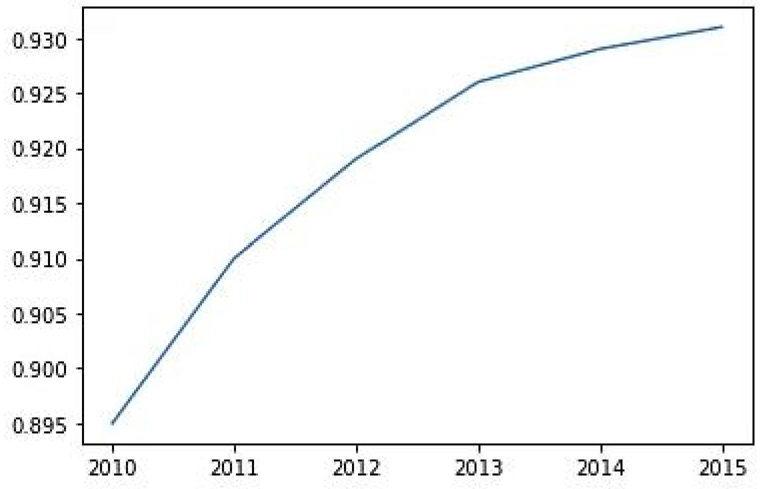


Figure 5 Axis Values

Let's give the axes labels so we can demonstrate what each axis represents.

plt.plot(years, yield_Apples)

plt.xlabel('Year')

plt.ylabel('Yield(tons per hectare)');

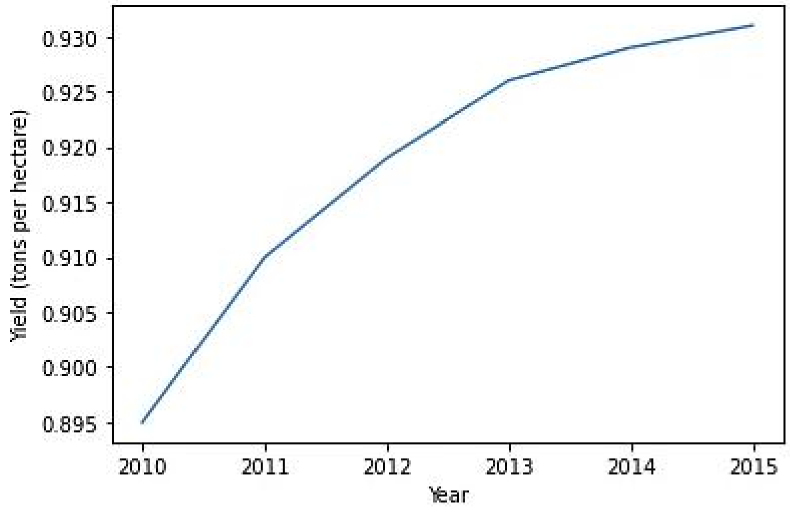


Figure 6 Axis with Labels

Simply use the plt.plot method once for each dataset to plot numerous datasets on the same graph. On the same graph, let's utilise this to compare the yields of apples and oranges.

years=range(2000,2012)

apples=[0.895, 0.91, 0.919, 0,926, 0.929, 0.931, 0.934, 0.936, 0.937, 0.9375, 0.9372, 0.939]

oranges=[0.962, 0.941, 0.930, 0.923, 0.918, 0908, 0.907, 0.904, 0.901, 0.898, 0.9, 0.896]

plt.plot(years, apples)

plt.plot(years, oranges)

plt.xlabel('Year')

plt.ylabel('Yield(tons per hectare)');

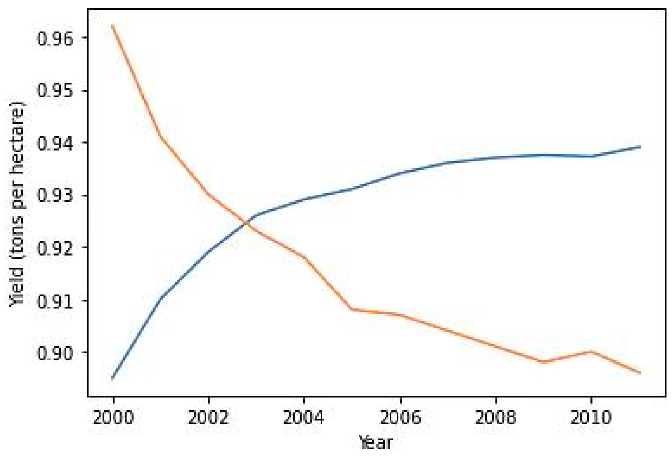


Figure 7 Plotting multiple graphs

Simply use the plt.plot method once for each dataset to plot numerous datasets on the same graph. On the same graph, let's utilise this to compare the yields of apples and oranges.

plt.plot(years, apples)

plt.plot(years, oranges)

plt.xlabel('year')

plt.ylabel('yield(tons per hectare)')

plt.title("crop yields in kanto')

plt.legend(['apples', 'oranges'])

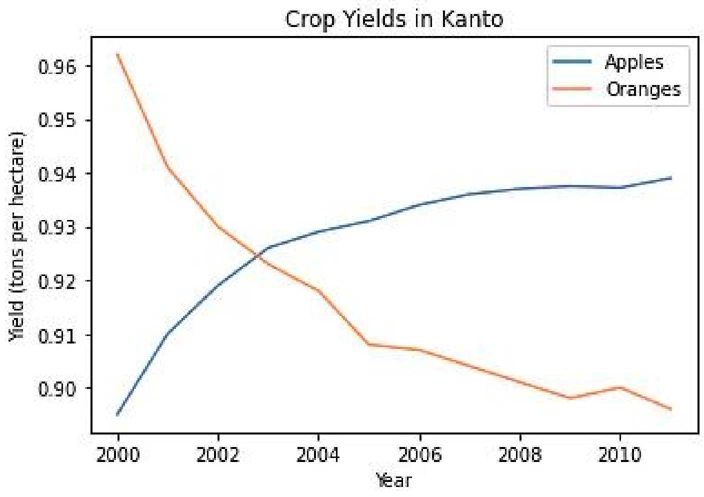


Figure 8 Plotting Multiple Graph

With the help of the marker parameter, we can use markers to show each data point on our graph. Matplotlib offers a wide variety of marker shapes, including a circle, cross, square, diamond, etc.

```
plt.plot(years, apples, marker='o')
plt.plot(years, oranges, marker='x')

plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')

plt.title("Crop Yields in Kanto")
plt.legend(['Apples', 'Oranges'])
```

```
<matplotlib.legend.Legend at 0x2054a8940d0>
```

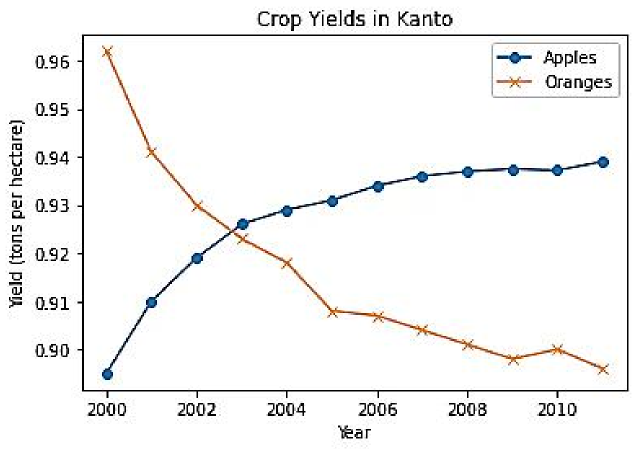
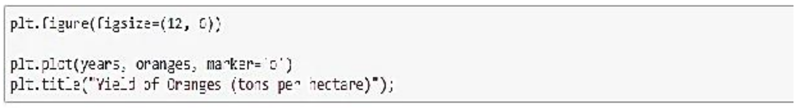


Figure 9 Using Markers

To alter the size of the figure, use the plt.figure function.



```
plt.figure(figsize=(12, 6))

plt.plot(years, oranges, marker='o')
plt.title("Yield of Oranges (tons per hectare)");
```

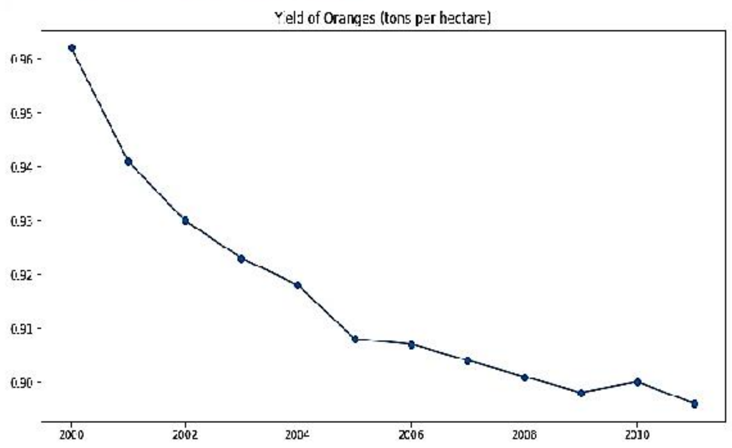


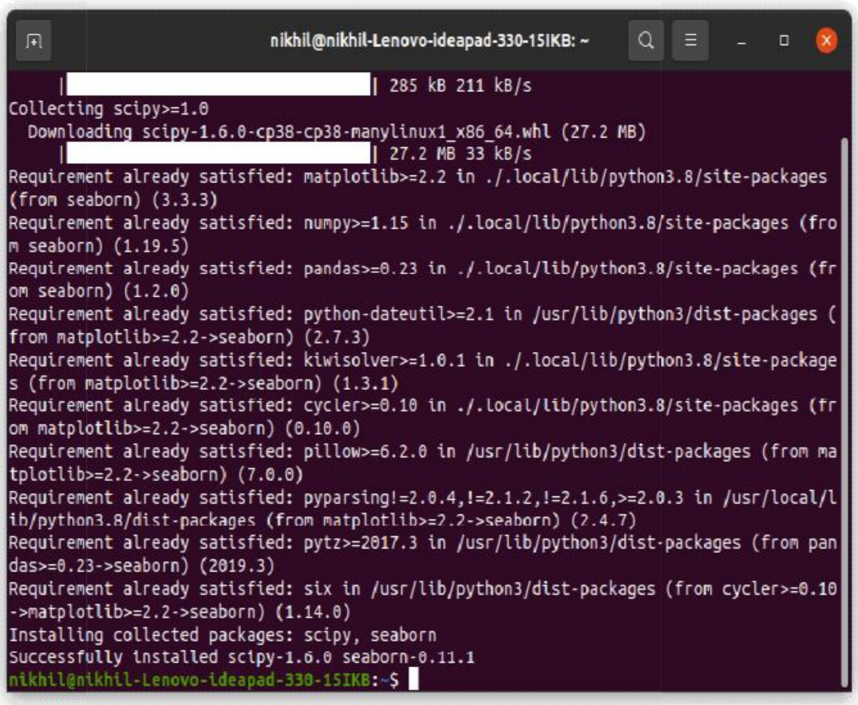
Figure 10 Changing Graph Size

11.4 Seaborn

A high-level interface called Seaborn was constructed on top of Matplotlib. It offers stunning design themes and colour schemes to create graphs that are more appealing.

Enter the following command in the terminal to install Seaborn.

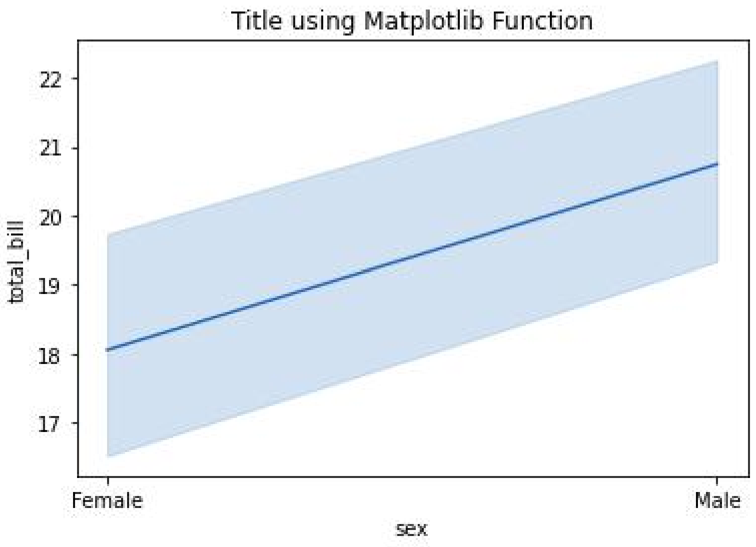
pip install seaborn



```
     |████████████████████████████████| 285 kB 211 kB/s
Collecting scipy>=1.0
  Downloading scipy-1.6.0-cp38-cp38-manylinux1_x86_64.whl (27.2 MB)
     |████████████████████████████████| 27.2 MB 33 kB/s
Requirement already satisfied: matplotlib>=2.2 in ./.local/lib/python3.8/site-packages
(from seaborn) (3.3.3)
Requirement already satisfied: numpy>=1.15 in ./.local/lib/python3.8/site-packages (fro
m seaborn) (1.19.5)
Requirement already satisfied: pandas>=0.23 in ./.local/lib/python3.8/site-packages (fr
om seaborn) (1.2.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/lib/python3/dist-packages (
from matplotlib>=2.2->seaborn) (2.7.3)
Requirement already satisfied: kiwisolver>=1.0.1 in ./.local/lib/python3.8/site-package
s (from matplotlib>=2.2->seaborn) (1.3.1)
Requirement already satisfied: cycler>=0.10 in ./.local/lib/python3.8/site-packages (fr
om matplotlib>=2.2->seaborn) (0.10.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/lib/python3/dist-packages (from ma
tplotlib>=2.2->seaborn) (7.0.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in /usr/local/l
ib/python3.8/dist-packages (from matplotlib>=2.2->seaborn) (2.4.7)
Requirement already satisfied: pytz>=2017.3 in /usr/lib/python3/dist-packages (from pan
das>=0.23->seaborn) (2019.3)
Requirement already satisfied: six in /usr/lib/python3/dist-packages (from cycler>=0.10
->matplotlib>=2.2->seaborn) (1.14.0)
Installing collected packages: scipy, seaborn
Successfully installed scipy-1.6.0 seaborn-0.11.1
nikhil@nikhil-Lenovo-ideapad-330-15IKB:~$ 
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
 # reading the database
data = pd.read_csv("tips.csv")
# draw lineplot
sns.lineplot(x="sex", y="total_bill", data=data)
# setting the title using Matplotlib
plt.title('Title using Matplotlib Function')
plt.show()
```

Output



11.5 Scatter Plot

Python's Matplotlib toolkit provides a complete tool for building static, animated, and interactive visualisations. It is used to create a variety of Python graphs, including scatter plots, 3-D plots, histograms, bar charts, pie charts, and line plots. The Matplotlib library's information on scatter plots will be used here.

matplotlib.pyplot.scatter()

Dots are used in scatter plots to show the relationship between variables, which are used to observe relationships between variables. To create a scatter plot, use the matplotlib library's scatter() method. Most often, scatter plots are used to show the relationship between variables and how changing one affects the other.

Syntax

The syntax for scatter() method is given below:

matplotlib.pyplot.scatter(x_axis_data, y_axis_data, s=None, c=None, marker=None, cmap=None, vmin=None, vmax=None, alpha=None, linewidths=None, edgecolors=None)

The following parameters are passed to the scatter() method:

An array of **x-axis** data is called x axis data.

y axis data: An array of y-axis information

marker size **s** (can be scalar or array of size equal to size of x or y)

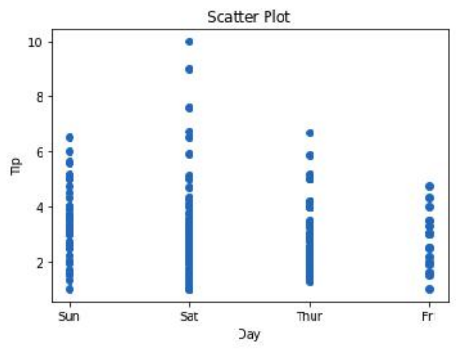
Color **C** of the Marker Color Sequence

linewidths, marker- **marker style**, and **cmap**- cmap name

- marker border width - marker border colour - marker border **alpha**

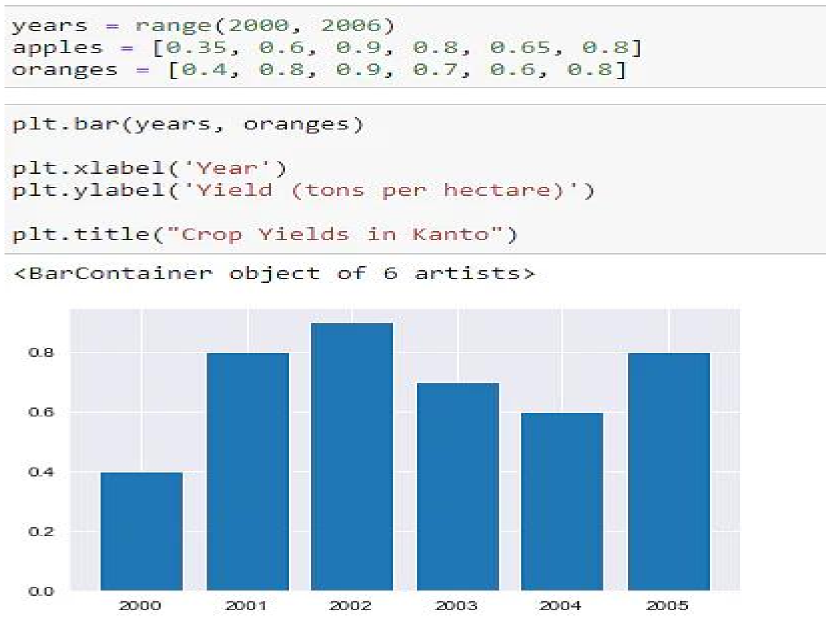
- blending value, which ranges from 0 (transparent) to 1. (opaque)

Program	Output

import pandas as pd
import matplotlib.pyplot as plt
# reading the database 	
data = pd.read_csv("tips.csv")
# Scatter plot with day against tip 	
plt.scatter(data['day'], data['tip'])
# Adding Title to the Plot 	
plt.title("Scatter Plot")
# Setting the X and Y labels 	
plt.xlabel('Day')
plt.ylabel('Tip')
plt.show()
	

11.6 Bar Graphs

When you have categorical data, a bar graph can be used to display it. A bar graph uses bars to indicate value on the y-axis and category on the x-axis to plot data. Bar graphs display data that falls into a particular category using bars of varying heights.



```
years = range(2000, 2006)
apples = [0.35, 0.6, 0.9, 0.8, 0.65, 0.8]
oranges = [0.4, 0.8, 0.9, 0.7, 0.6, 0.8]
```

```
plt.bar(years, oranges)

plt.xlabel('Year')
plt.ylabel('Yield (tons per hectare)')

plt.title("Crop Yields in Kanto")
```

```
<BarContainer object of 6 artists>
```

Figure 11 Plotting Bar Graph

Bars can also be stacked on top of one another. The data for apples and oranges should be plotted.

```
plt.bar(years, apples)
plt.bar(years, oranges, bottom=apples)
```

```
<BarContainer object of 6 artists>
```

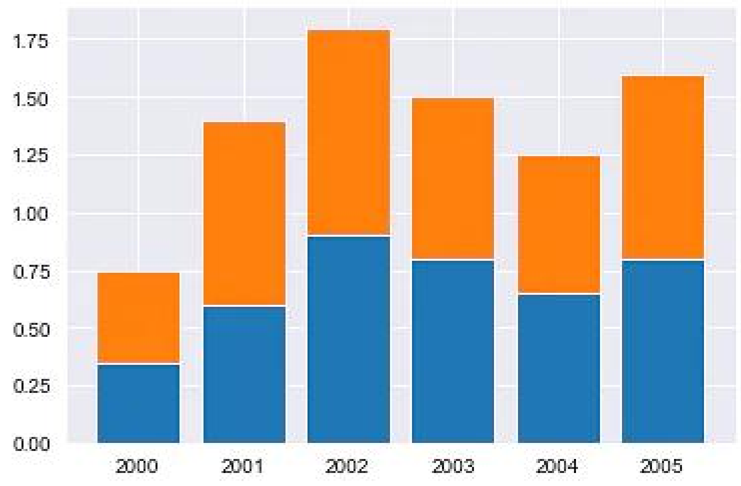


Figure 12 Plotting Stacked Bar Graphs

Next, let's use Seaborn's tips dataset. The dataset includes:

- Information about the sex (gender)
- Time of day
- Total bill
- Tips given by customers visiting the restaurant for a week

```
tips_df = sns.load_dataset("tips")
tips_df
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

To see how the average bill amount varies on various days of the week, we can create a bar chart. By calculating the day-wise averages and utilising plt.bar afterwards, we can accomplish this. Additionally, a barplot function that can compute averages automatically is offered by the Seaborn library.

```
sns.barplot(x='day', y='total_bill', data=tips_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2054afa7370>
```

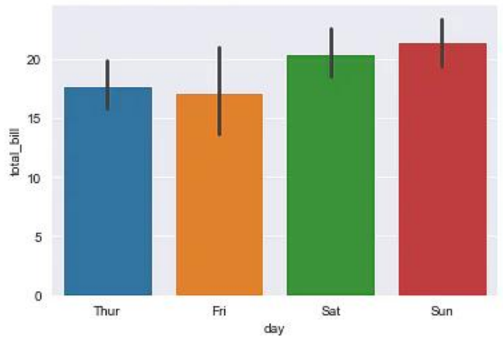


Figure 13 Plotting Averages of Each Bar

The hue option can be used to compare bar charts side by side. Based on the third feature mentioned in this argument, a comparison will be made.

```
sns.barplot(x='day', y='total_bill', hue='sex', data=tips_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2054b046700>
```

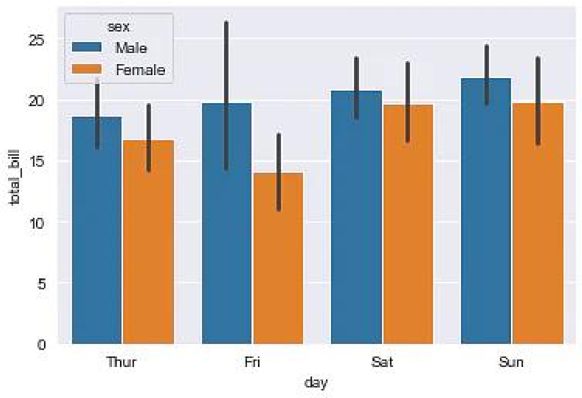


Figure 14 Plotting multiple bar graphs

By changing the axis, the bars can be made horizontal.

```
sns.barplot(x='total_bill', y='day', hue='sex', data=tips_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2054af1b250>
```

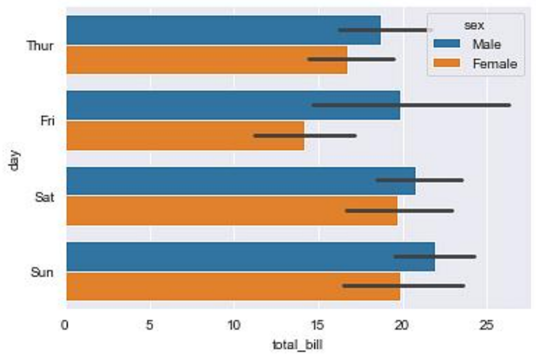
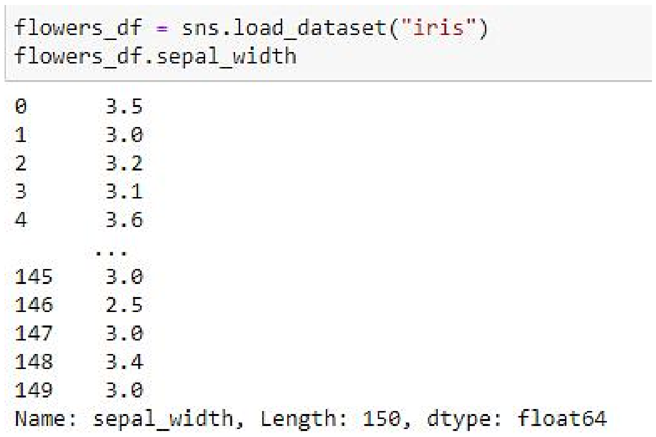


Figure 15 Plotting horizontal bar graphs

11.7 Histograms

A histogram is a bar graph that shows how data changes over time. The range is plotted along the x-axis, and the height of the data pertaining to a range is plotted along the y-axis. Data are plotted using histograms over a range of values. To display the data corresponding to each range, they employ a bar representation. Let's once more plot histograms using the "Iris" data, which provides details about flowers.

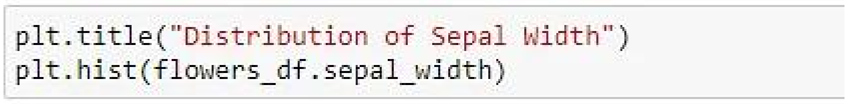


```
flowers_df = sns.load_dataset("iris")
flowers_df.sepal_width
```

```
0      3.5
1      3.0
2      3.2
3      3.1
4      3.6
      ...
145    3.0
146    2.5
147    3.0
148    3.4
149    3.0
Name: sepal_width, Length: 150, dtype: float64
```

Figure 16 Iris Dataset

Now, let’s plot a histogram using the hist() function.



```
plt.title("Distribution of Sepal Width")
plt.hist(flowers_df.sepal_width)
```

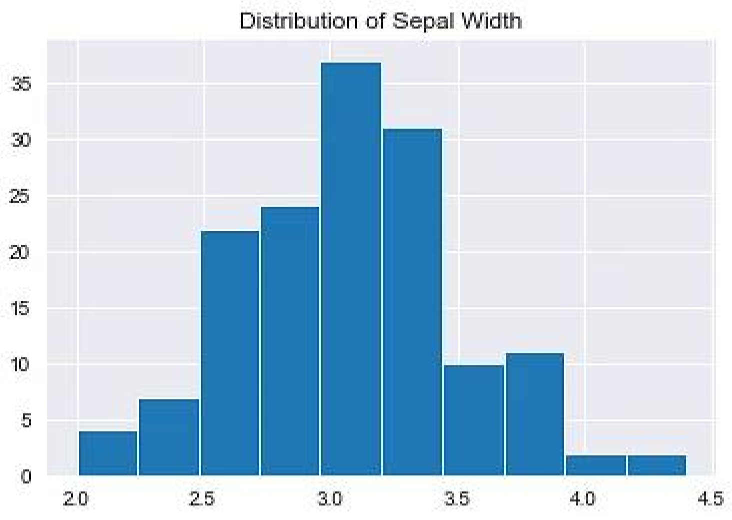


Figure 17 Plotting Histogram

Numpy also allows us to modify the number and size of bins.

```
import numpy as np

# Specifying the boundaries of each bin
plt.hist(flowers_df.sepal_width, bins=np.arange(2, 5, 0.25))
```

```
(array([ 4.,  7., 22., 24., 50., 18., 13.,  8.,  3.,  1.,  0.]),
 array([2.  , 2.25, 2.5 , 2.75, 3.  , 3.25, 3.5 , 3.75, 4.  , 4.25, 4.5 ,
       4.75]),
 <a list of 11 Patch objects>)
```

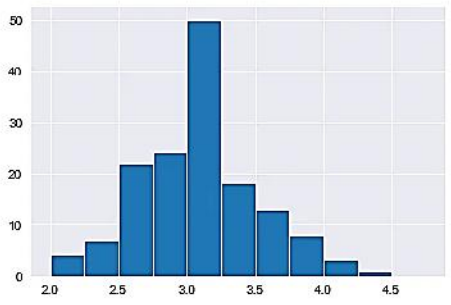


Figure 18 Changing number and size of bins

We can create bins of unequal size too.

```
# Bins of unequal sizes
plt.hist(flowers_df.sepal_width, bins=[1, 3, 4, 4.5])
```

```
(array([57., 89.,  4.]),
 array([1. , 3. , 4. , 4.5]),
 <a list of 3 Patch objects>)
```

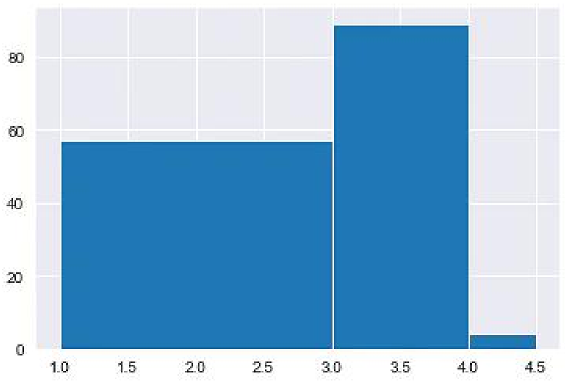


Figure 19 Bins of Unequal Size

We can include several histograms in a single chart, just like we can with line charts. So that the bars of one histogram don't obscure those of the others, we can make each histogram less opaque. Let's create distinct histograms for every type of flower.

```
setosa_df = flowers_df[flowers_df.species == 'setosa']
versicolor_df = flowers_df[flowers_df.species == 'versicolor']
virginica_df = flowers_df[flowers_df.species == 'virginica']
```

```
plt.hist(setosa_df.sepal_width, alpha=0.4, bins=np.arange(2, 5, 0.25));
plt.hist(versicolor_df.sepal_width, alpha=0.4, bins=np.arange(2, 5, 0.25));
```

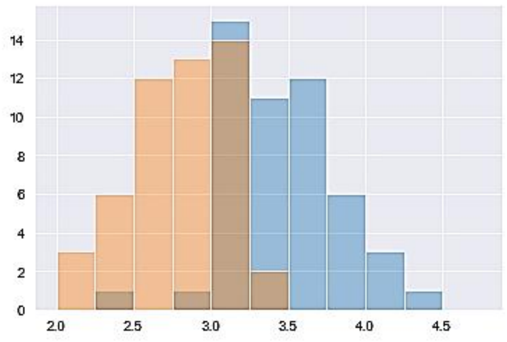


Figure 20 Multiple Histogram

If the stacked parameter is set to True, then many histograms can be piled on top of one another.

```
plt.title('Distribution of Sepal Width')

plt.hist([setosa_df.sepal_width, versicolor_df.sepal_width, virginica_df.sepal_width],
         bins=np.arange(2, 5, 0.25),
         stacked=True);

plt.legend(['Setosa', 'Versicolor', 'Virginica']);
```

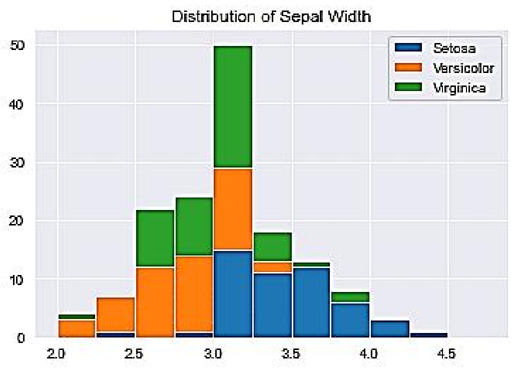


Figure 21 Stacking Histogram

Summary

- The human mind processes and comprehends any given data more easily when it is presented with images, maps, and graphs
- It is used to plot simple graphs like line charts, bar graphs, and so forth.
- It can carry out complicated visualisations with fewer commands and is primarily used for statistics visualisation.
- An informational graph called a line chart shows data as a collection of dots connected by straight lines.
- Use the plt.plot method once for each dataset to plot numerous datasets on the same graph.
- A high-level interface called Seaborn was constructed on top of Matplotlib. It offers stunning design themes and colour schemes to create graphs that are more appealing.
- Python's Matplotlib toolkit provides a complete tool for building static, animated, and interactive visualisations.

- Dots are used in scatter plots to show the relationship between variables, which are used to observe relationships between variables.
- When you have categorical data, a bar graph can be used to display it. A bar graph uses bars to indicate value on the y-axis and category on the x-axis to plot data.
- A histogram is a bar graph that shows how data changes over time. The range is plotted along the x-axis, and the height of the data pertaining to a range is plotted along the y-axis.

Keywords

Seaborn:Python has a dataset-oriented library called Seaborn that can be used to create statistical representations.

Bokeh: For contemporary web browsers, there is a visualisation library called Bokeh.

Altair: A declarative statistical visualisation library for Python is called Altair. The Vega-Lite JSON specification served as the foundation for Altair's user-friendly, dependable API.

Plotly:A high-level, declarative, interactive, open-source, and browser-based visualisation toolkit for Python is called plotly.py.

Ggplot: The graphics grammar is implemented in Python by ggplot.

Bar Chart:When comparing metric values between various data subsets, a bar chart is utilised.

Column Chart: When comparing a single category of data between specific sub-items, such as when comparing revenue between areas, column charts are typically utilised.

Stacked Bar Chart: When comparing the sums of the available groups and the makeup of the various subgroups, a stacked bar chart is employed.

Pie Chart: Pie charts can be used to determine how much of each component there is in a given whole.

Area Chart:To monitor changes over time for one or more groups, area charts are employed.

Column Histogram: To view the distribution for a single variable with few data points, column histograms are utilised.

Scatter Plot:It is possible to use scatter plots to determine the relationships between two variables.

Box Plot:The form of the distribution, its central value, and its variability are displayed using a box plot.

Waterfall Chart:A waterfall chart can be used to illustrate how a variable's value gradually changes as a result of increments or decrements.

Venn Diagrams: To visualise the connections between two or three sets of items, utilise Venn diagrams.

Self Assessment

Q1. Select those which does not visualize the data

A. Charts
B. Shapes
C. Graphs
D. Maps

Q2: Which of the following type of chart is not supported by pyplot?

A. Histogram
B. Boxplot

C. Pie
D. All of the above

Q3: plot which is used to given statistical summary is

A. Bar
B. Line
C. Histogram
D. Box Plot

Q4: To compare data we can use _________ chart

A. Line
B. Bar
C. Pie
D. Scatter

Q5: To import pyplot module we can write

A. Import pyplot as plt
B. Import matplotlib.pyplot
C. Import matplotlib.pyplot as plt
D. Both b and c

Q6: Matplotlib is a ____________ plotting library

A. 1D
B. 2D
C. 3D
D. All of above

Q7: Data __________ refers to graphical representation of data.

A. Visualization
B. Analysis
C. Plotting
D. Handling

Q8: The interface of Matplotlib used for data visualization is

A. Seaborn
B. Anaconda
C. MATLAB
D. Pyplot

Q9: which library is the most used visualization library in python?

A. visual
B. matlibplot

C. matplotlib
D. matlab

Q10: Which function of matplotlib can be used to create a line chart?

A. line
B. plot
C. graph
D. bar

Q11: Which graph should be used if we want to show distribution of elements?

A. pie
B. basemap
C. bar
D. histogram

Q12: which graph should be used If we want to find patterns in data?

A. bar
B. histogram
C. scatterplots
D. basemap

Q13: Which of the following command is correct to install matplotlib?

A. Pip install matplot
B. Pipe install matplot
C. Pip install matplotlib
D. None of the above

Q14: __________ function of the pyplot is used to create a figure/chart/plot.

A. show()
B. plotting()
C. plot()
D. plots()

Q15: A figure/chart contains

A. Plotting area
B. Legend
C. Axis labels
D. All of the above

Answers for Self Assessment

1.	B	2.	D	3.	D	4.	B	5.	D
6.	B	7.	A	8	D	9	C	10	D
11	D	12	C	13	C	14	C	15	D

Review Questions

1. What is data visulaizations? Write down benefits of Data Visualization.
2. Write down difference between Matplotlib and Seaborn.
3. Explain Line chart with example.
4. What do you understand by seaborn. Write down command to install seaborn. Explain use of seaborn with example.
5. Explain difference between scatter plot, bar graph and histogram.

FurtherReadings

- Maheshwari, Anil. *Big Data*. McGraw-Hill Education, 2019.
- Mayer-Schonberger, Viktor; Cukier, Kenneth (2013). Big Data: A Revolution That Will Transform How We Live, Work, and Think . Houghton Mifflin Harcourt.
- McKinsey Global Institute Report (2011). Big Data: The Next Frontier For Innovation, Competition, and Productivity. Mckinsey.com
- Marz, Nathan, and James Warren (2015). Big Data: Principles and Best Practices of Scalable Realtime Data Systems. Manning Publications.
- Sandy Ryza, Uri Laserson et.al (2014). Advanced-Analytics-with-Spark. OReilley. White, Tom (2014). Mastering Hadoop. OReilley.

Web Links

1. Apache Hadoop resources: https://hadoop.apache.org/docs/r2.7.2/
2. Apache HDFS: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
3. Hadoop API site: http://hadoop.apache.org/docs/current/api/
4. NoSQL databases: http://nosql-database.org/
5. Apache Spark: http://spark.apache.org/docs/latest/
6. Tutorials on Big Data technologies: https://www.tutorialspoint.com/
7. https://www.tutorialspoint.com/hadoop/hadoop_multi_node_cluster.htm

Unit 12: Data Visualization

CONTENTS

Objectives

After this unit, student would be able to

- Understand basic concepts of seaborn
- Learn basic difference between seaborn and matplotlib
- Know how data visualization perform using seaborn

Introduction

Python's Seaborn visualization module is fantastic for plotting statistical visualizations. It offers lovely default styles and color schemes to enhance the appeal of statistics charts. It is constructed on top of the Matplotlib toolkit and is tightly integrated with the Pandas data structures.

With Seaborn, visualization will be at the heart of data exploration and comprehension. For a better comprehension of the dataset, it offers dataset-oriented APIs that allow us to switch between various visual representations for the same variables.

12.1 Different categories of plot in Seaborn

Plots are mostly used to show how different variables relate to one another. These variables may be entirely numerical or may represent a category, such as a group, class, or division. Seaborn categorizes the plot into the following groups.

Relational plots: This type of graphic is used to see how two variables are related.

Categorical plots: This graphic discusses categorical variables and the visualization of them.

Distribution Plots: Plots used to examine univariate and bivariate distributions include distribution plots.

Regression plots: The main purpose of the regression plots in Seaborn is to provide a visual aid that highlights patterns in a dataset during exploratory data analysis.

Scatter Plots: Plots in a matrix An array of scatterplots makes up a matrix plot.

Multi-plot grids: Drawing numerous instances of the same plot on various subsets of the dataset is a helpful strategy.

12.2 Installation

For python environment

pip install seaborn

For conda environment

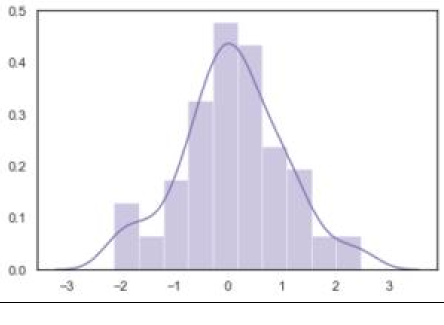
conda install seaborn

Some basic plots using seaborn

Dist Plots: Histograms are plotted using the seaborn dist plot, as well as the kdeplot and rugplot variants.

Program

# Importing libraries 	
importnumpy as np
importseaborn as sns
# Selecting style as white, 	
# dark, whitegrid, darkgrid 	
# or ticks 	
sns.set(style="white")
# Generate a random univariate 	
# dataset 	
rs=np.random.RandomState(10)
d =rs.normal(size=100)
 	
# Plot a simple histogram and kde 	
# withbinsize determined automatically 	
sns.distplot(d, kde=True, color="m")

Output


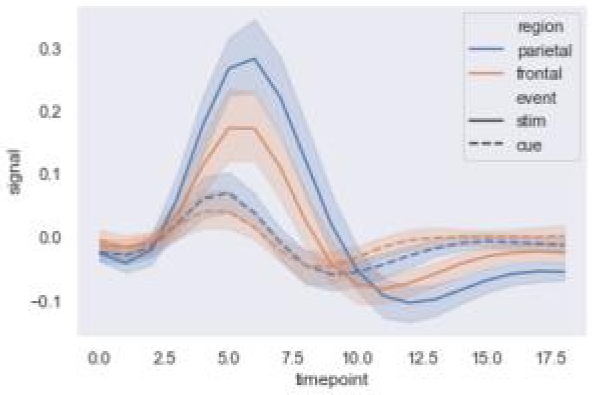
Line Plot:One of the most fundamental plots in the Seaborn Library is the line plot. This graphic is mostly used to depict continuous data in the form of a time series.

Program

importseaborn as sns
sns.set(style="dark")
fmri=sns.load_dataset("fmri")

```
# Plot the responses for different\
# events and regions
sns.lineplot(x="timepoint",
y="signal",
hue="region",
style="event",
data=fmri)
```

Output

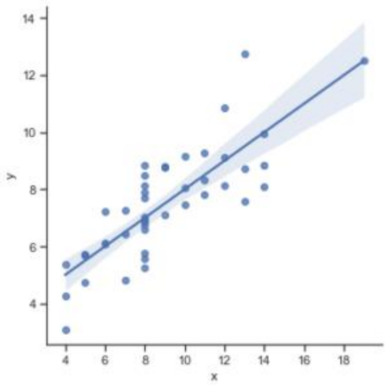


Lmplot:Another very simple plot is the lmplot. It displays a line denoting a linear regression model together with data points in a 2D space, and the labels x and y can be set to represent the horizontal and vertical axes, respectively.

Program

```
importseaborn as sns

sns.set(style="ticks")

# Loading the dataset
df=sns.load_dataset("anscombe")

# Show the results of a linear regression
sns.lmplot(x="x", y="y", data=df)
```

Output



12.3 Difference between Matplotlib vs Seaborn

Data is graphically represented in data visualization. It facilitates data analysis and forecasting by breaking down a large dataset into manageable graphs. It is a crucial component of data science that simplifies and expands the accessibility of complex data.

The foundation of Python-based data visualization is made up of Matplotlib and Seaborn. With the aid of additional libraries like NumPy and Pandas, Matplotlib is a Python library that is used to plot graphs. It is an effective Python tool for data visualization. It is used to plot 2D graphs of arrays and make static conclusions. John D. Hunter originally mentioned it in 2002.

It makes use of Pyplot to offer a free and open-source MATLAB-like interface. It can work with different operating systems and their graphical front ends. Seaborn: Additionally, it is a Python library that utilises Matplotlib, Pandas, and NumPy to plot graphs.

It is a superset of the Matplotlib library and is constructed on top of Matplotlib. It aids in the visualization of single- and two-variate data. It embellishes Matplotlib visuals with lovely themes. It serves as a valuable tool for visualizing linear regression models. It is used to create static Time-Series data graphs. It also helps to make graphs more attractive by removing overlap.

Table of difference between Matplotlib and Seaborn

Features	Matplotlib	Seaborn
Functionality	It is used to create simple graphs. Bargraphs, histograms, pie charts, scatter plots, lines, and other visual representations of data are used to visualise datasets.	Data visualisation patterns and graphs can be found throughout Seaborn. Interesting themes are employed. It aids in assembling all of the data into a single plot. It also offers data distribution.
Syntax	It employs syntax that is relatively intricate and extensive. Example: Matplotlib.pyplot.bar(x axis, y axis) is theis the syntax for a bar graph.	It has relatively simple syntax, making it simpler to learn and comprehend. Example: The seaborn.barplot(x axis, y axis) syntax for a bar graph.
Dealing multiple figures	We can open and work with many figures at once. They are clearly closed, though. One figure can be closed at a time using the syntax matplotlib.pyplot.close (). Close all the figures using this syntax: matplotlib.pyplot.close("all")	Each figure's creation is given a specific time by Seaborn. But it might result in (OOM) out of memory problems.
Visualization	Matplotlib serves as a graphics package for data visualisation in Python and integrates nicely with Numpy and Pandas. Similar capabilities and syntax are available in Pyplot as in MATLAB. Users of MATLAB can therefore readily examine	With Pandas data frames, Seaborn is more at ease. Beautiful graphics are provided in Python by using simple sets of functions.

	it.	
Pliability	Matplotlib is a powerful and highly customisable	With the aid of its default themes, Seaborn prevents the overlapping of plots.
Data Frames and Arrays	When dealing with data frames and arrays, Matplotlib performs well. It views axes and figures as objects. There are several stateful plotting APIs in it. Thus, methods similar to plot() can operate without parameters.	Compared to Matplotlib, Seaborn is a lot more useful and organised and treats the entire dataset as a single entity. Because Seaborn is not very stateful, parameters are needed when calling methods like plot ()
Use Cases	Matplotlib uses Numpy and Pandas to plot a variety of graphs.	The enhanced version of Matplotlib, known as Seaborn, plots graphs using Matplotlib, Numpy, and Pandas.

12.4 Data Visualization with Seaborn

The visual presentation of data is known as data visualisation. Because of the excellent ecosystem of Python packages focused on data, it is crucial for data analysis. By summarising and presenting a large quantity of data in a straightforward and understandable format, it also helps to grasp the data, no matter how complex it may be, as well as the value of the data. It also aids in the effective and clear transmission of information.

Pandas and Seaborn

One of those packages, Pandas and Seaborn, makes importing and analysing data more simpler. Pandas and Seaborn will be used to examine the data in this article.

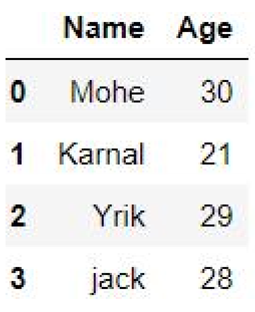
Pandas

Pandas provide tools for processing and cleaning up your data. It is the most widely used Python data analysis library. A data table is referred to as a dataframe in pandas.

So, let's start with creating Pandas data frame:

Program

```
# Python code demonstrate creating

importpandas as pd

# initialise data of lists.
data ={'Name':[ 'Mohe', 'Karnal', 'Yrik', 'jack'],
'Age':[ 30, 21, 29, 28]}

# Create DataFrame
df=pd.DataFrame( data )

# Print the output.
df
```

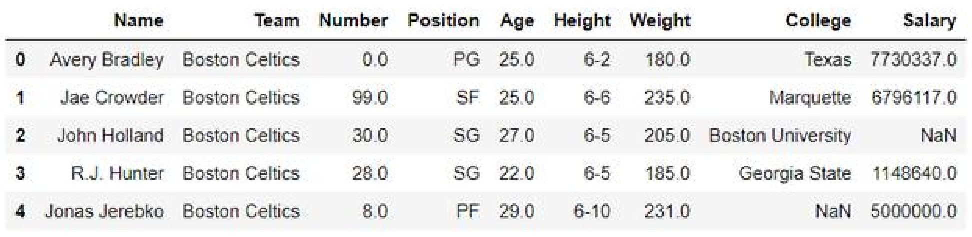
Output


```
     Name  Age
0    Mohe   30
1  Karnal   21
2    Yrik   29
3    jack   28
```

Example2: : Load the CSV data from the system and display it through pandas.

Program

# import module 	
importpandas
 	
# load the csv 	
data =pandas.read_csv("nba.csv")
 	
# show first 5 column 	
data.head()

Output


```
            Name            Team  Number  Position   Age  Height  Weight            College     Salary
0  Avery Bradley  Boston Celtics     0.0        PG  25.0     6-2   180.0              Texas  7730337.0
1    Jae Crowder  Boston Celtics    99.0        SF  25.0     6-6   235.0          Marquette  6796117.0
2   John Holland  Boston Celtics    30.0        SG  27.0     6-5   205.0  Boston University        NaN
3    R.J. Hunter  Boston Celtics    28.0        SG  22.0     6-5   185.0      Georgia State  1148640.0
4  Jonas Jerebko  Boston Celtics     8.0        PF  29.0    6-10   231.0                NaN  5000000.0
```

Seaborn

Python's Seaborn visualisation module is fantastic for plotting statistical visualisations. It is constructed on top of the Matplotlib toolkit and is tightly integrated with the Pandas data structures.

Installation
For python environment :

```
pip install seaborn
```

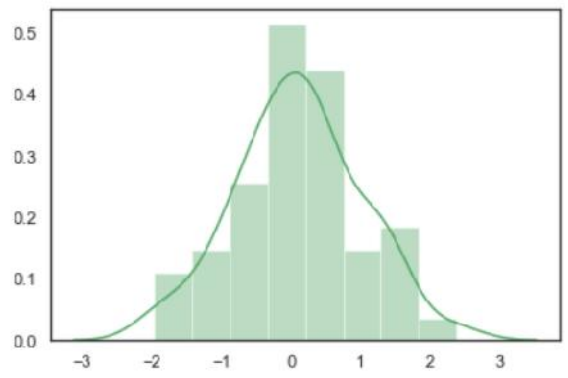
For condaenvironment :

```
conda install seaborn
```

Some basic plots using seaborn:
Importing libraries import numpy as np

```
import seaborn as sns

# Selecting style as white,
# dark, whitegrid, darkgrid
# or ticks
sns.set( style = "white" )

# Generate a random univariate
# dataset
rs = np.random.RandomState( 10 )
d = rs.normal( size = 50 )

# Plot a simple histogram and kde
# with binsize determined automatically
sns.distplot(d, kde = True, color = "g")
```

Output

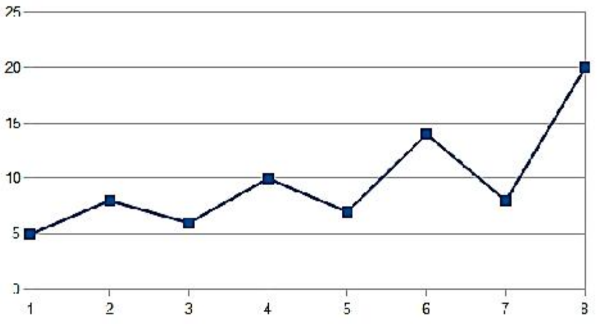


12.5 Seaborn: Statistical Data Visualization

Seaborn makes the statistical linkages easier to see. We use statistical analysis to determine the relationships between variables in a dataset and how those relationships depend on other factors. The statistical analysis used here makes it easier to see trends and spot different patterns in the dataset. These are the plot will help to visualize:

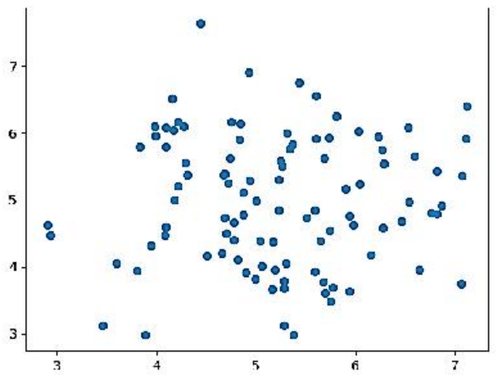
- Line Plot
- Scatter Plot
- Box plot
- Point plot
- Count plot
- Violin plot
- Swarm plot
- Bar plot
- KDE Plot

Line Plot:



Although scatter plots are quite successful, there is no one form of visualisation that is always the best. Instead, the visual representation should be customised for the unique characteristics of the dataset and the plot's intended purpose.You might want to comprehend how variations in one variable as a function of time, or a similarly continuous variable, in various datasets. Making a line plot in this case is a wise decision. By setting kind=" line, the lineplot() function in Seaborn can carry out this task either directly or in conjunction with relplot().

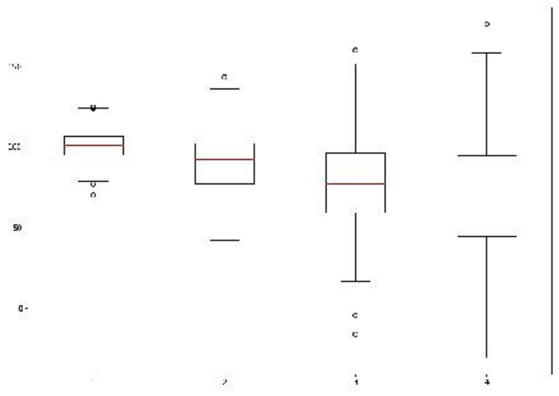
Scatter Plot



A scatterplot can be used in conjunction with several semantic groups to aid in clear understanding of a graph. They can use the semantics of colour, size, and style parameters to plot two-dimensional visuals that can be improved by mapping up to three additional variables. Each parameter controls the visual and semantic features used to distinguish the various subsets. Making graphics more accessible can be achieved by using redundant meanings.

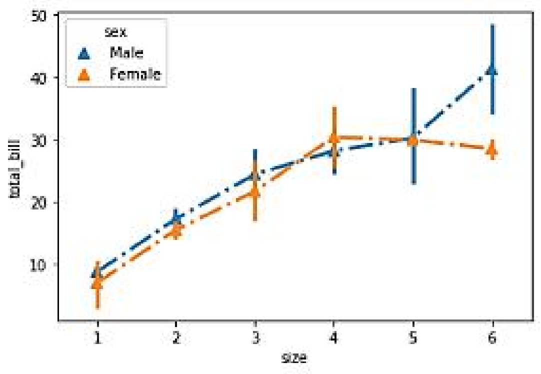
Syntax: seaborn.scatterplot(x=None, y=None, hue=None, style=None, size=None, data=None, palette=None, hue_order=None, hue_norm=None, sizes=None, size_order=None, size_norm=None, markers=True, style_order=None, x_bins=None, y_bins=None, units=None, estimator=None, ci=95, n_boot=1000, alpha='auto', x_jitter=None, y_jitter=None, legend='brief', ax=None, **kwargs)
Parameters:
x, y: Input data variables that should be numeric.
data: Dataframe where each column is a variable and each row is an observation.
size: Grouping variable that will produce points with different sizes.
style: Grouping variable that will produce points with different markers.
palette: Grouping variable that will produce points with different markers.
markers: Object determining how to draw the markers for different levels.
alpha: Proportional opacity of the points.
Returns: This method returns the Axes object with the plot drawn onto it.

Box Plot:



The seaborn boxplot has a very simple structure. Distributions are represented visually using boxplots.That is incredibly helpful when comparing data between two groups. A boxplot may also be referred to as a box-and-whisker plot. Any box displays the dataset's quartiles, and the whiskers extend to display the remainder of the distribution.The boxplot plot is related with the boxplot() method.

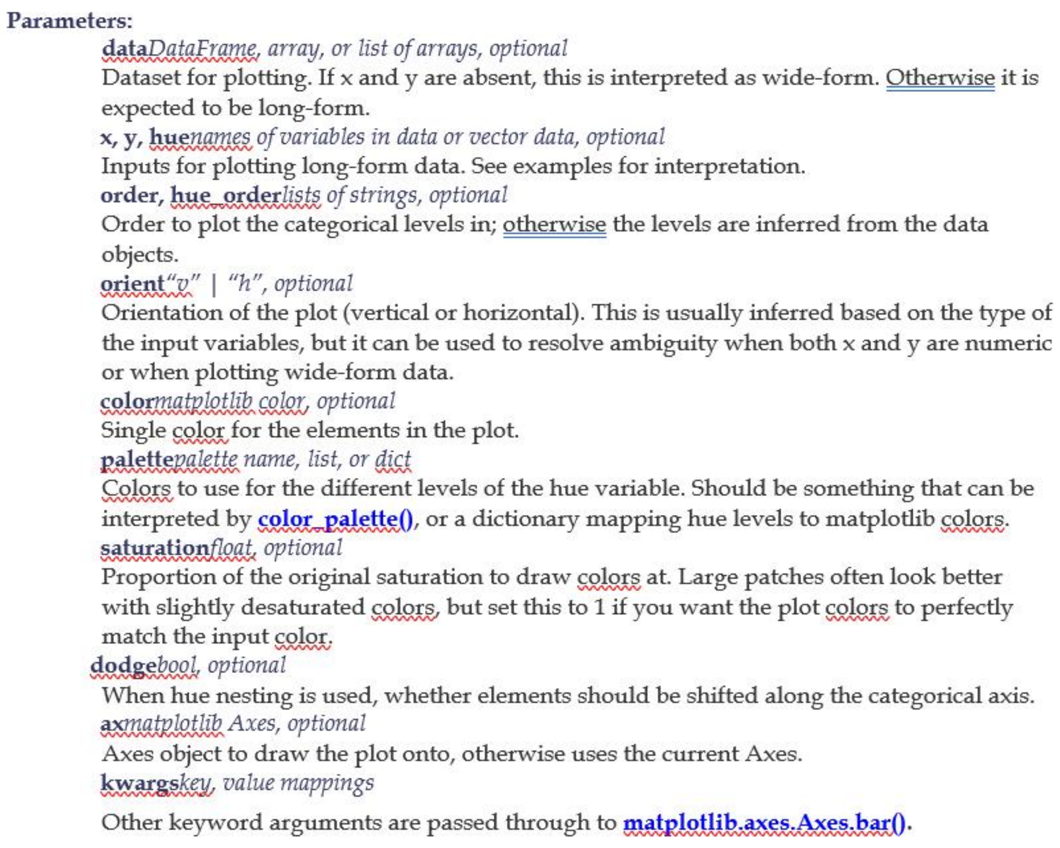
Point Plot:



A point plot uses the position of the dot to indicate an estimate of the central tendency for a numerical variable, and error bars are used to show the degree of uncertainty surrounding that estimate.For comparisons between various levels of one or more categorical variables, point plots may be more helpful than bar plots. They excel in demonstrating interactions, or how the connection between levels of one category variable alters as levels of a second categorical variable are added. It is simpler for the eyes to detect interactions by differences in slope rather than by comparing the heights of various groupings of points or bars thanks to the lines that connect each point from the same hue level.

Count Plot:seaborn.**countplot**(data=None, *, x=None, y=None, hue=None, order=None, hue_order=None, orient=None, color=None, palette=None, saturation=0.75, width=0.8, dodge=True, ax=None, **kwargs)

A count plot resembles a histogram over a categorical variable as opposed to a quantitative one. You can compare counts across nested variables because the fundamental API and settings are the same as those for barplot().The more recent histplot() function, despite it has slightly different default behaviour, offers greater capabilities.



Parameters:

data*DataFrame, array, or list of arrays, optional*

Dataset for plotting. If x and y are absent, this is interpreted as wide-form. Otherwise it is expected to be long-form.

x, y, hue*names of variables in data or vector data, optional*

Inputs for plotting long-form data. See examples for interpretation.

order, hue_order*lists of strings, optional*

Order to plot the categorical levels in; otherwise the levels are inferred from the data objects.

orient*"v" | "h", optional*

Orientation of the plot (vertical or horizontal). This is usually inferred based on the type of the input variables, but it can be used to resolve ambiguity when both x and y are numeric or when plotting wide-form data.

color*matplotlib color, optional*

Single color for the elements in the plot.

palette*palette name, list, or dict*

Colors to use for the different levels of the hue variable. Should be something that can be interpreted by **color_palette()**, or a dictionary mapping hue levels to matplotlib colors.

saturation*float, optional*

Proportion of the original saturation to draw colors at. Large patches often look better with slightly desaturated colors, but set this to 1 if you want the plot colors to perfectly match the input color.

dodge*bool, optional*

When hue nesting is used, whether elements should be shifted along the categorical axis.

ax*matplotlib Axes, optional*

Axes object to draw the plot onto, otherwise uses the current Axes.

kwargs*key, value mappings*

Other keyword arguments are passed through to **matplotlib.axes.Axes.bar()**.

Violin Plot:seaborn.violinplot(data=None, *, x=None, y=None, hue=None, order=None, hue_order=None, bw='scott', cut=2, scale='area', scale_hue=True, gridsize=100, width=0.8, inner='box', split=False, dodge=True, orient=None, linewidth=None, color=None, palette=None, saturation=0.75, ax=None, **kwargs)

A violin plot and a box and whisker plot serve the same purpose. In order to allow for comparison, it displays the distribution of quantitative data across a number of levels of one (or more) categorical variables. The violin plot includes a kernel density estimation of the underlying distribution as opposed to a box plot, in which all of the plot elements correspond to actual datapoints.This can be a useful and appealing technique to display numerous data distributions at once, but take in note that the estimate procedure is affected by the sample size and so violins for small samples may appear deceptively smooth.

Swarm Plot: The Seaborn swarmplot is presumably similar to the stripplot, with the exception that the points are adjusted to avoid overlap in order to better depict the distribution of values. A swarm plot can be created independently, but it also works well in conjunction with a box, which is desirable since the names linked with the names will be used to annotate the axes. This plot style is commonly referred to as a "beeswarm."

Syntax: seaborn.swarmplot(x=None, y=None, hue=None, data=None, order=None, hue_order=None, dodge=False, orient=None, color=None, palette=None, size=5, edgecolor='gray', linewidth=0, ax=None, **kwargs)

Parameters:
x, y, hue: Inputs for plotting long-form data.
data: Dataset for plotting.
color: Color for all of the elements
size: Radius of the markers, in points.

Bar Plot: A bar plot, often known as a bar chart, is a graph that uses rectangular bars with lengths and heights proportionate to the values they represent to depict a category of data. Both horizontal and vertical graphs of the bars are possible. The comparisons between the distinct categories are shown in a bar chart. The exact categories under comparison are shown by one of the plot's axes, while the measured values associated with those categories are represented by the other axis.

KDEPLOT: Kdeplot, also known as a Kernel Distribution Estimation Plot, is a graphical representation of the probability density function of continuous or non-parametric data variables; it can be used to plot either a single variable or a number of variables simultaneously. We may create a Kdeplot with various capabilities added to it using the Python Seaborn module.

Summary

- A package called Seaborn uses Matplotlib as its foundation to plot graphs. In order to see random distributions, it will be used.
- The statistical link between the data points is depicted using relational graphs. Because it enables humans to recognise trends and patterns in data, visualisation is essential.
- Histograms are plotted using the seaborn dist plot, as well as the kdeplot and rugplot variants.
- Another very simple plot is the lmplot. It displays a line denoting a linear regression model together with data points in a 2D space, and the labels x and y can be set to represent the horizontal and vertical axes, respectively.
- Data is graphically represented in data visualisation. It facilitates data analysis and forecasting by breaking down a large dataset into manageable graphs.
- Matplotlib is used to create simple graphs. Bar graphs, histograms, pie charts, scatter plots, lines, and other visual representations of data are used to visualize datasets.
- Data visualisation patterns and graphs can be found throughout Seaborn. Interesting themes are employed.
- The visual presentation of data is known as data visualisation. . Because of the excellent ecosystem of Python packages focused on data, it is crucial for data analysis.
- Pandas provide tools for processing and cleaning up your data. It is the most widely used Python data analysis library.
- Python's Seaborn visualisation module is fantastic for plotting statistical visualisations. It is constructed on top of the Matplotlib toolkit and is tightly integrated with the Pandas data structures.
- The seaborn boxplot has a very simple structure. Distributions are represented visually using boxplots.
- A point plot uses the position of the dot to indicate an estimate of the central tendency for a numerical variable, and error bars are used to show the degree of uncertainty surrounding that estimate. For comparisons between various levels of one or more categorical variables, point plots may be more helpful than bar plots.
- A violin plot and a box and whisker plot serve the same purpose. In order to allow for comparison, it displays the distribution of quantitative data across a number of levels of one (or more) categorical variables.
- The Seaborn swarmplot is presumably similar to the stripplot, with the exception that the points are adjusted to avoid overlap in order to better depict the distribution of values.
- The KNN algorithm, also referred to as K-nearest neighbor, is a non-parametric algorithm that groups data points according to their proximity and association with other pieces of available information.

Keywords

Relational plots: This type of graphic is used to see how two variables are related.

Categorical plots: This graphic discusses categorical variables and the visualization of them.

Distribution Plots: Plots used to examine univariate and bivariate distributions include distribution plots.

Regression plots: The main purpose of the regression plots in Seaborn is to provide a visual aid that highlights patterns in a dataset during exploratory data analysis.

Scatter Plots: Plots in a matrix An array of scatterplots makes up a matrix plot.

Multi-plot grids: Drawing numerous instances of the same plot on various subsets of the dataset is a helpful strategy.

Visualizations:Data is graphically represented in data visualisation. It facilitates data analysis and forecasting by breaking down a large dataset into manageable graphs.

Pandas and Seaborn: Pandas and Seaborn, makes importing and analysing data more simpler.

Scatter:A scatterplot can be used in conjunction with several semantic groups to aid in clear understanding of a graph.

Box Plot:A boxplot may also be referred to as a box-and-whisker plot. Any box displays the dataset's quartiles, and the whiskers extend to display the remainder of the distribution.The boxplot plot is related with the boxplot() method.

Point plot: A point plot uses the position of the dot to indicate an estimate of the central tendency for a numerical variable, and error bars are used to show the degree of uncertainty surrounding that estimate

Self Assessment

Q1. Series and DataFrame's plot function is only a basic wrapper over _____________

A. gplt.plot()
B. plt.plot()
C. plt.plotgraph()
D. none of the mentioned

Q2. Please specify the ideal kind keyword combination for graph plotting.

A. ‘hist’ for histogram
B. ‘box’ for boxplot
C. ‘area’ for area plots
D. all of the mentioned

Q3. Which of the following values does the kind barplot keyword provide?

A. Bar
B. Kde
C. Hexbin
D. none of the mentioned

Q4. By utilising the _________ method in pandas.tools.plotting, you may produce a scatter plot matrix.

A. sca_matrix
B. scatter_matrix
C. DataFrame.plot
D. all of the mentioned

Q5: Indicate the incorrect kind keyword combination for graph plotting.

A. For scatter plots, use "scatter"
B. "kde" for bin plots with hexagonal axes
C. "pie" for plots of pie
D. None of the previously listed

Q6. Which of the following plots are used to check if a data set or time series is random?

A. Lag
B. Random
C. Lead
D. None of the mentioned

Q7: Which of the following does not visualize data.

A. Charts
B. Maps
C. Shapes
D. Graphs

Q8. Which of the chart is not supported by pyplot?

A. Histogram
B. Boxplot
C. Pie
D. All are correct

Q9: To display histogram with well-defined edge we can write

A. df.plot(type='hist', edge='red')
B. df.plot(type='hist', edgecolor='red')
C. df.plot(type='hist', line='red')
D. df.plot(type='hist', linecolor='red')

Q10: Plot which is used to given statistical summary is
A. Bar
B. Line
C. Histogram
D. Box Plot

Q11: Which of the following is not the parameter of pyplot's plot() method.
A. Marker

B. Lineheight
C. Linestyle
D. Color

Q12: To compare data, we can use ________ chart.

A. Line
B. Bar
C. Pie
D. Scatter

Q13: Which of the following chart element is used to identify data series by its color patterns.

A. Chart title
B. Legend
C. Marker
D. Data Labels

Q14: Matplotlib is _____________ plotting library.

A. 1D
B. 2D
C. 3D
D. All of the above

Q15: Data ________ refers to graphical representation of data.

A. Visualisation
B. Analysis
C. Plotting
D. Handling

Answers for Self Assessment

1.	B	2.	D	3.	A	4.	D	5.	B
6.	A	7.	C	8.	C	9.	B	10.	D
11.	B	12.	B	13.	B	14.	B	15.	A

Review Questions

FurtherReadings

- Mark Lutz,Programming *Python: Powerful Object-Oriented Programming*, OREILLY

- Wes McKinney, *Python for data analysis*, OREILLY
- David Ascher and Mark Lutz, *Learning Python*, OREILLY
- Eric Matthes, Python Crash Course, 2nd Edition: *A Hands-On, Project-Based Introduction to Programming*, Starch Pres

Web Links

https://www.tutorialspoint.com/python/index.htm

https://www.python.org/downloads/

https://www.w3schools.in/python/data-types

https://www.programiz.com/python-programming/online-compiler/

https://www.codecademy.com/catalog/language/python

Unit 13: OOP Concepts

CONTENTS

Objectives

After studying this unit, you will be able to:

- understand features of OOPs
- learn basic concepts about encapsulation
- learn inheritance and its types

Introduction

The Python programming style known as object-oriented programming (OOPs) makes use of objects and classes. It seeks to incorporate in programming real-world concepts like inheritance, polymorphism, encapsulation, etc. The fundamental idea behind OOPs is to unite the data and the functions that use it such that no other portion of the code may access it.

Object-Oriented Programming's Core Ideas (OOPs) are:-

- Class
- Object
- Method
- Inheritance
- Polymorphism
- Data Abstraction
- Encapsulation

13.1 Class

A class is a group of related items. The models or prototypes used to generate objects are included in classes. It is a logical entity with a few methods and characteristics.Consider the following scenario to better appreciate the need for generating classes: Suppose you needed to keep track of the number of dogs that might have various characteristics, such as breed or age. If a list is utilised, the dog's breed and age might be the first and second elements, respectively. What if there were 100 different breeds of dogs? How would you know which ingredient should go where? What if you wanted to give these dogs additional traits? This is disorganised and just what courses need.

A few notes on the Python class:

- The keyword class is used to create classes.
- The variables that make up a class are known as attributes.
- With the dot (.) operator, attributes can always be retrieved and are always public. For example: Myclass.Myattribute

Class Definition Syntax

```
Class Classname
{
#Statement-1
.
.
.
#Statement-N
}
```

Example*:* Making a Python class that is empty

```
Class Dog:
        Pass
```

Using the class keyword, we built a class with the name dog in the example above.

13.2 Objects

The object is an entity that is connected to a state and activity. Any physical device, such as a mouse, keyboard, chair, table, pen, etc., may be used. Arrays, dictionaries, strings, floating-point numbers, and even integers are all examples of objects. Any single string or integer, more specifically, is an object. A list is an object that may house other things, the number 12 is an object, the text "Hello, world" is an object, and so on. You may not even be aware of the fact that you have been using items.

An Object consists of:

State:The properties of an object serve as a representation of it. Additionally, it reflects an object's characteristics.

Behavior:It is represented via an object's methods. It also shows how one object reacts to other objects.

Identity: It gives a thing a special name and makes it possible for objects to communicate with one another.

Let's look at the example of the class dog to better understand the state, behaviour, and identity (explained above).

- The identity may be regarded as the dog's name.

- Breed, age, and colour of the dog are examples of states or attributes.
- You may infer from the behaviour whether the dog is eating or sleeping.

Creating an object as an example

Obj=Dog()

This will produce an object with the class Dog, named obj, as stated above. Let's first grasp the fundamental terms that will be utilised while working with objects and classes before delving further into them.

a. The self

- An additional initial parameter in the method declaration is required for class methods. When we call the method, we don't supply a value for this parameter; Python does.
- Even if we have a method that doesn't require any parameters, we still need one.
- This is comparable to this Java reference and this C++ pointer.

This is the sole purpose of the special self. When we invoke a method of this object as myobject.method(arg1, arg2), Python automatically converts it to MyClass.method(myobject, arg1, arg2).

b. The __init__method

The constructors in Java and C++ are comparable to the __init__ method. As soon as a class object is created, it is executed. Any initialization you want to perform on your object can be done with the method.Let's build some objects utilising the self and __init__ methods after defining a class.

Example1*:* Class and object creation using class and instance properties

```
class Dog:
        # class attribute
        attr1 = "mammal"
        # Instance attribute
        def __init__(self, name):
                self.name = name

# Driver code
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")

# Accessing class attributes
print("Rodger is a {}".format(Rodger.__class__.attr1))
print("Tommy is also a {}".format(Tommy.__class__.attr1))

# Accessing instance attributes
print("My name is {}".format(Rodger.name))
print("My name is {}".format(Tommy.name))
```

Output
Rodger is a mammal Tommy is also a mammal My name is Rodger My name is Tommy

Example2: Class and object creation with methods

```
class Dog:

        # class attribute
        attr1 = "mammal"

        # Instance attribute
        def __init__(self, name):
                self.name = name

        def speak(self):
                print("My name is {}".format(self.name))
# Driver code
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")

# Accessing class methods
Rodger.speak()
Tommy.speak()
```

Output
My name is Rodger My name is Tommy

13.3 Methods

A function connected to an object is the method. A method is not specific to class instances in Python. Any sort of object may have methods.

13.4 Inheritance

The capacity of one class to derive or inherit properties from another class is known as inheritance. The class from which the properties are being derived is referred to as the base class or parent class, and the class from which the properties are being derived is referred to as the derived class or child class. The advantages of inheritance include:

- It accurately depicts relationships in the real world.
- It offers a code's reusability. We don't need to keep writing the same code. Additionally, it enables us to expand a class's features without changing it.
- Because of its transitive nature, if a class B inherits from a class A, then all of class B's subclasses will also automatically inherit from class A.

Types of Inheritance

Single Inheritance

A class can inherit properties from a single-parent class using single-level inheritance.

Multilevel Inheritance

A derived class can inherit properties from an immediate parent class, which in turn can inherit properties from his parent class, thanks to multi-level inheritance.

Hierarchical Inheritance

More than one derived class can inherit properties from a parent class thanks to hierarchical level inheritance.

Multiple Inheritance

One derived class may inherit properties from several different base classes thanks to multiple level inheritance.

Example: Python inheritance

```
# Python code to demonstrate how parent constructors
# are called.

# parent class
class Person(object):
        # __init__ is known as the constructor
        def __init__(self, name, idnumber):
                self.name = name
                self.idnumber = idnumber

        def display(self):
                print(self.name)
                print(self.idnumber)

        def details(self):
                print("My name is {}".format(self.name))
                print("IdNumber: {}".format(self.idnumber))
# child class
class Employee(Person):
        def __init__(self, name, idnumber, salary, post):
                self.salary = salary
                self.post = post
                # invoking the __init__ of the parent class
                Person.__init__(self, name, idnumber)

        def details(self):
                print("My name is {}".format(self.name))
                print("IdNumber: {}".format(self.idnumber))
                print("Post: {}".format(self.post))

# creation of an object variable or an instance
a = Employee('Rahul', 886012, 200000, "Intern")
# calling a function of the class Person using
# its instance
a.display()
a.details()
```

Output

```
Rahul
886012
My name is Rahul
IdNumber: 886012
Post: Intern
```

In the aforementioned article, two classes—Person (parent class) and Employee—have been established (Child Class). The Person class is an ancestor of the Employee class. As can be seen in the show function in the code above, we may use the methods of the person class through the employee class. The details() function shows how a child class can alter the parent class's behaviour.

13.5 Polymorphism

Simply put, polymorphism means having multiple forms. For instance, utilising polymorphism, we can answer the question of whether the given species of birds fly or not using just one function. ***Example:***Python's use of polymorphism

```
class Bird:
        def intro(self):
                print("There are many types of birds.")
        def flight(self):
                print("Most of the birds can fly but some cannot.")
class sparrow(Bird):
        def flight(self):
                print("Sparrows can fly.")
class ostrich(Bird):
        def flight(self):
                print("Ostriches cannot fly.")
obj_bird = Bird()
obj_spr = sparrow()
obj_ost = ostrich()

obj_bird.intro()
obj_bird.flight()

obj_spr.intro()
obj_spr.flight()

obj_ost.intro()
obj_ost.flight()
```

OUTPUT

```
There are many types of birds.
Most of the birds can fly but some cannot.
There are many types of birds.
Sparrows can fly.
There are many types of birds.
Ostriches cannot fly.
```

13.6 Data Abstraction

Both data abstraction and encapsulation are frequently used interchangeably. Since data abstraction is accomplished by encapsulation, the two terms are almost synonymous.

When using abstraction, internal details are hidden and only functionalities are displayed. Giving things names that capture the essence of what a function or an entire programme does is the process of abstracting something.

13.7 Encapsulation

One of the core ideas in object-oriented programming is encapsulation (OOP). It explains the concept of data wrapping and the techniques that operate on data as a single unit. This restricts direct access to variables and procedures and can avoid data alteration by accident. A variable can only be altered by an object's method in order to prevent inadvertent modification. These variables fall under the category of private variables.

A class, which encapsulates all the data that is contained in its member functions, variables, etc., is an example of encapsulation.

Table 1 Encapsulation in Python

Methods	Variables

```
# Python program to
# demonstrate private members

# Creating a Base class
class Base:
        def __init__(self):
                self.a = "EcontentOnline"
                self.__c = "EcontentOnline"

# Creating a derived class
class Derived(Base):
        def __init__(self):

                # Calling constructor of
                # Base class
                Base.__init__(self)
                print("Calling private member of base class: ")
                print(self.__c)
# Driver code
obj1 = Base()
print(obj1.a)

# Uncommenting print(obj1.c) will
# raise an AttributeError

# Uncommenting obj2 = Derived() will
# also raise an AtrributeError as
# private member of base class
# is called inside derived class
```

Output
EcontentOnline

The c variable was generated as the private attribute in the example above. We are unable to even directly read or modify the value of this attribute.

Difference between Object-Oriented vs. Procedure-Oriented Programming Languages.

Object-oriented Programming	Procedural Programming
The approach to addressing problems that uses objects for computation is called object-oriented programming.	A list of instructions is used in procedural programming to perform calculations in stages.
It makes development and upkeep simpler.	When a project grows in scope, maintaining the codes is difficult in procedural programming.
It replicates the thing in the actual world. Therefore, oops makes it simple to tackle difficulties in the actual world.	It doesn't represent reality in any way. It operates using detailed instructions broken down into smaller units called functions.
It offers data concealment. Consequently, it is safer than procedural languages. Private information is not accessible from anyplace.	Because procedural languages don't offer a suitable method for data binding, they are less secure.
C++, Java, .Net, Python, C#, and other object-oriented programming languages are examples.	Procedural languages include C, Fortran, Pascal, VB, and others.

Summary

- The Python programming style known as object-oriented programming (OOPs) makes use of objects and classes.
- A class is a group of related items. The models or prototypes used to generate objects are included in classes.
- The object is an entity that is connected to a state and activity. Any physical device, such as a mouse, keyboard, chair, table, pen, etc., may be used.
- The constructors in Java and C++ are comparable to the __init__ method. As soon as a class object is created, it is executed.
- A function connected to an object is the method. A method is not specific to class instances in Python. Any sort of object may have methods.
- The capacity of one class to derive or inherit properties from another class is known as inheritance.
- Simply put, polymorphism means having multiple forms. For instance, utilising polymorphism, we can answer the question of whether the given species of birds fly or not using just one function
- Both data abstraction and encapsulation are frequently used interchangeably. Since data abstraction is accomplished by encapsulation, the two terms are almost synonymous
- One of the core ideas in object-oriented programming is encapsulation (OOP). It explains the concept of data wrapping and the techniques that operate on data as a single unit.
- The approach to addressing problems that uses objects for computation is called object-oriented programming.
- A list of instructions is used in procedural programming to perform calculations in stages

Keywords

OOPS: Object-oriented programming is known as OOP. While object-oriented programming involves constructing objects that include both data and methods, procedural programming involves developing procedures or methods that perform actions on the data.

Class: Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods. A Class is like an object constructor, or a "blueprint" for creating objects

The __init__method: All classes have a function called __init__(), which is always executed when the class is being initiated.

The __str__function: What should be returned when the class object is rendered as a string is determined by the __str__() function.

Objects methods: Methods can also be found in objects. Object-specific functions are called methods in an object.

Self-Parameter: To access class-specific variables, use the self parameter, which is a reference to the currently running instance of the class.

Del:Using the del keyword, properties on objects can be deleted.

Pass statement: Although class definitions cannot be empty, if for some reason you have one that is empty, add the pass statement to prevent an error.

Inheritance: By using inheritance, we may create a class that has all the methods and attributes of another class.

Parent class:The class being inherited from, often known as the base class, is the parent class.

Child class:The class that inherits from another class is referred to as a child class or derived class.

Super Function:The super() function in Python allows a descendant class to inherit all of its parent's methods and properties.

Self Assessment

Q1. Which option best encapsulates inheritance?

A. Ability of a class to include methods from other classes in its definition
B. Techniques for grouping instance variables and methods to limit access to certain class members
C. A focus on variables and passing variables to functions
D. Enables the use of sophisticated software that is well-designed and flexible.

Q2. Which of the following claims about inheritance is false?

A. A class's protected members may be inherited.
B. The class that inherits is known as a subclass.
C. A class's private members can be accessed and inherited.
D. One characteristic of OOP is inheritance

.

Q3. What line of code should you write to activate the __init__ method in A from B if B is a subclass of A?

A. A.__init__(self)
B. B.__init__(self)
C. A.__init__(B)
D. B.__init__(A)

Q4: What function type is a built-in in the context of classes?

A. Identifies the name of any value's object.
B. Identifies any value's class name.
C. Determines a value's class description
D. Identifies any value's file name

Q5: What one of the following is not an inheritance type?

A. Double-level
B. Multi-level
C. Single-level
D. Multiple

Q6: Which of these is not one of OOP's core characteristics?

A. Encapsulation
B. Inheritance
C. Instantiation
D. Polymorphism

Q7: Which of the following definitions best describes encapsulation?

A. The capacity of a class to derive individuals from other classes as part of its own definition.
B. Techniques for combining instance variables and methods to limit access to specific class members
C. focuses on supplying parameters to functions and variables.
D. enables the use of sophisticated software that is well-designed and flexible.

Q8: Define Overriding.

A. Overriding can occur in the case of inheritance in class
B. It is a process of redefining inherited method in child class.
C. It is a magic method in python.
D. None of these

Q9: _____________ developed python language

A. Albert Einstein
B. Guido Van Rossum
C. Guido Evan
D. None of these

Q10: What year was the Python programming language created?

A. 1975
B. 1989
C. 1972
D. 1990

Q11: Which of the following commands the expression with the most precedence?

A. Addition
B. Subtraction
C. Parentheses
D. Power

Q12: Of the following, which best describes abstraction?

A. Hiding the execution
B. displaying crucial information
C. Hiding the important data
D. Hiding the implementation and showing only the features

Q13: A class is an ____________ abstraction.

A. Object
B. Logical
C. Real
D. Hypothetical

Q14: Abstraction can be used for _______________.

A. Control and data.
B. Only data
C. Only control
D. Classes

Q15: Which of the following can be considered a combination of data abstraction and programming?

A. Class
B. Object
C. Inheritance
D. Interfaces

Answer for Self Assessment

1.	A	2.	C	3.	A	4.	B	5.	A
6.	C	7.	B	8.	B	9.	B	10.	B
11.	C	12.	D	13.	B	14.	A	15.	B

Review Questions

1. What do you understand by OOPS? Write down the code to make a python class that si empty.
2. Define Objects. Write down example to create an object with methods.
3. What do you understand by inheritance and also define types of inheritance.
4. Write down difference between Single level inheritance, multilevel inheritance and multiple inheritance.
5. Define Polymorphism. Write down python code that define use of polymorphism.
6. What do you understand by Encapsulation? Write down python program to demonstrate private members.
7. Write down difference between object-oriented programming and procedural programming.

FurtherReadings

- Mark Lutz,Programming *Python: Powerful Object-Oriented Programming*, OREILLY
- Wes McKinney, *Python for data analysis*, OREILLY
- David Ascher and Mark Lutz, *Learning Python*, OREILLY
- Eric Matthes, Python Crash Course, 2nd Edition: *A Hands-On, Project-Based Introduction to Programming*, Starch Pres

Web Links

https://www.tutorialspoint.com/python/index.htm

https://www.python.org/downloads/

https://www.w3schools.in/python/data-types

https://www.programiz.com/python-programming/online-compiler/

https://www.codecademy.com/catalog/language/python

Unit 14: Machine Learning Algorithms

CONTENTS

Objectives

After this unit, student would be able to learn:

- basic concepts of linear regression.
- how decision tree is used in python.
- basic concepts about random forests and k-means clustering.

Introduction

Programs that use machine learning algorithms are able to discover hidden patterns in data, forecast results, and enhance performance based on past performance. In machine learning, various algorithms can be used for various tasks, such as simple linear regression for prediction issues like stock market forecasting and the KNN algorithm for classification issues.

14.1 Types of Machine Learning Algorithms

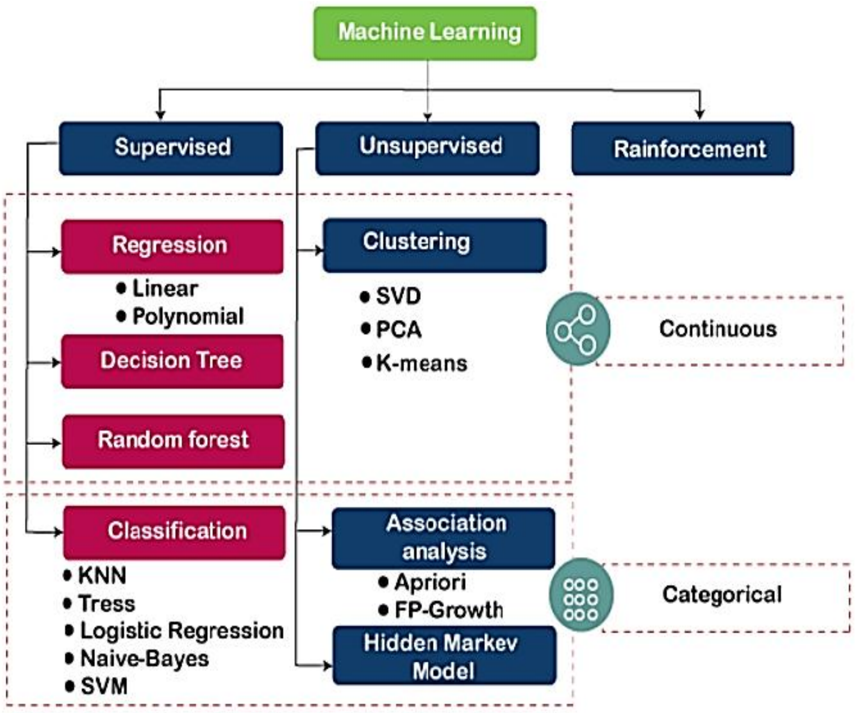


Figure 1 Types of Machine Learning Algorithms

Supervised Learning Algorithms

A type of machine learning called supervised learning requires outside supervision for the machine to learn. The labelled dataset is used to train the supervised learning models. After training and processing, the model is put to the test by being given a sample set of test data to see if it predicts the desired result.

In supervised learning, mapping input and output data is the main objective. It is the same as when a student is studying under the teacher's supervision because supervised learning is dependent on supervision. Spam filtering is a prime example of supervised learning.

The method of supervised learning involves giving the machine learning model the right input data as well as the output data. Finding a mapping function to link the input variable (x) with the output variable is the goal of a supervised learning algorithm (y).

Supervised learning has applications in the real world such as risk assessment, image categorization, fraud detection, spam filtering, etc.

Unsupervised Learning Algorithms

In supervised machine learning, models are trained on labelled data while being watched over by training data. However, there may be several instances where we lack labelled data and must instead identify hidden patterns in the supplied dataset. Therefore, we need unsupervised learning strategies to handle these kinds of problems in machine learning.

Unsupervised learning is a type of machine learning in which models are not supervised using training datasets, as the name implies. Instead, models themselves decipher the provided data to reveal hidden patterns and insights. It is comparable to the learning process that occurs in the human brain while learning something new. It is characterized as:

Unsupervised learning is a subcategory of machine learning in which models are trained using unlabeled datasets and are free to operate on the data without being checked by a human observer.

Because unlike supervised learning, we have the input data but no corresponding output data, unsupervised learning cannot be used to solve a regression or classification problem directly.

Finding the underlying structure of a dataset, classifying the data into groups based on similarities, and representing the dataset in a compressed format are the objectives of unsupervised learning.

Difference between Supervised and Unsupervised Learning Algorithms

Supervised Learning Algorithms	Unsupervised Learning Algorithms
Using labelled data, supervised learning algorithms are taught.	Unlabeled data is used to train algorithms for unsupervised learning.
A supervised learning model uses direct feedback to determine whether or not it is foretelling the correct outcome.	A model of unsupervised learning does not incorporate feedback.
A model of supervised learning forecasts the results.	Unsupervised learning models uncover data's buried patterns.
In supervised learning, the model receives input data in addition to output.	In unsupervised learning, the model receives only input data.
The objective of supervised learning is to develop the model's capacity to forecast output in the presence of novel data.	Unsupervised learning aims to extract hidden patterns and insightful information from an unknown dataset.
To train the model in supervised learning, supervision is required.	The model can be trained without any supervision using unsupervised learning
Classification and regression issues can be grouped under supervised learning.	The model can be trained without any supervision using unsupervised learning.
When both the input and the associated output are known, supervised learning may be applied.	Unsupervised learning issues fall under the categories of clustering and associations.
A supervised learning model yields reliable results.	When we only have input data and no corresponding output data, unsupervised learning can be applied.
Supervised learning falls short of true artificial intelligence because we must first train the model for each set of data before it can accurately predict the outcome.	In comparison to supervised learning, an unsupervised learning model could produce less accurate results.
It includes a variety of algorithms, including Bayesian logic, decision trees, support vector machines, multi-class classification, linear regression, and logistic regression.	Unsupervised learning is more in line with actual artificial intelligence because it acquires knowledge by experience, much like a kid does when learning daily tasks
It comprises a variety of procedures, including Bayesian logic, decision trees, support vector machines, multi-class classification, linear regression, and logistic regression.	It contains a number of algorithms, including the Apriori algorithm, KNN, and Clustering.

14.2 Linear Regression

One of the simplest and most widely used Machine Learning techniques is linear regression. It is a statistical technique for performing predictive analysis. For continuous/real/numeric variables like sales, salary, age, and product price, among others, linear regression makes predictions. The linear regression algorithm, often known as linear regression, demonstrates a linear relationship between a dependent (y) and one or more independent (y) variables. Given that linear regression demonstrates a linear relationship, it may be used to determine how the dependent variable's value changes as a function of the independent variable's value. The link between the variables is represented by a sloping straight line in the linear regression model. Think on the photo below:

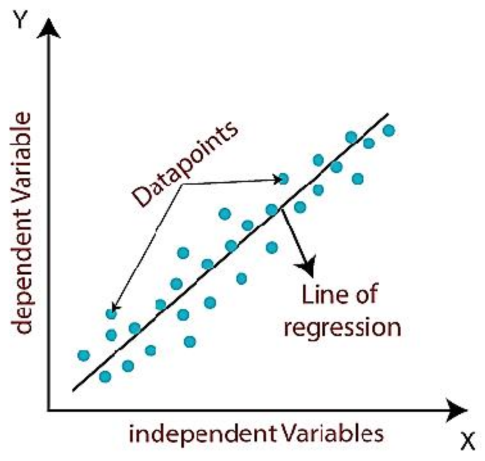


Figure 2 Linear Regression

A linear regression can be conceptualized mathematically as:

$$y= a_0+a_1x+ \varepsilon$$

Y=Dependent Variable (Target Variable)

X=Independent Variable (predictor Variable)

a0=intercept of the line (Gives an additional degree of freedom)

a1=Linear regression coefficient (scale factor to each input value).

ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

Linear Regression in Python

Step1*:* **Import Python Packages***: To begin, we must import a few packages required for linear regression*:

- **Numpy** – fundamental package for scientific computing to create the example dataset.
- **Pandas** – a powerful tool for data analysis and manipulation.
- **Scikit Learn** (sklearn) – tools for predictive data analysis, including linear regression.
- **Matplotlib:** plotting library for visualization.

```
import numpy as np

import pandas as pd

from sklearn.linear_model import LinearRegression #only importing the linear_model function

import matplotlib.pyplot as plt

%matplotlib inline
```

Step2: Generate Random Training Dataset

We will create a random sample dataset as the training set since we want to present linear regression in a straightforward manner.

With x1 serving as the only input variable, we first create a randomly generated dataset of size 50. The output, y, is then set up to have an approximately linear relationship with x1. To introduce noise to the dataset, the random variable noise is added.

Now we have a simple linear regression problem.

```
num_obs = 50

x1 = np.random.uniform(low=-10.0, high=10.0, size=num_obs)

noise = np.random.normal(loc=0.0, scale=5.0, size=num_obs)

y = 10 + 2*x1 + noise
```

By using a scatterplot to represent the relationship between x1 and y, we can see that it is roughly linear.

```
plt.figure(figsize=(20,7))

plt.plot(x1, y, 'o')
```

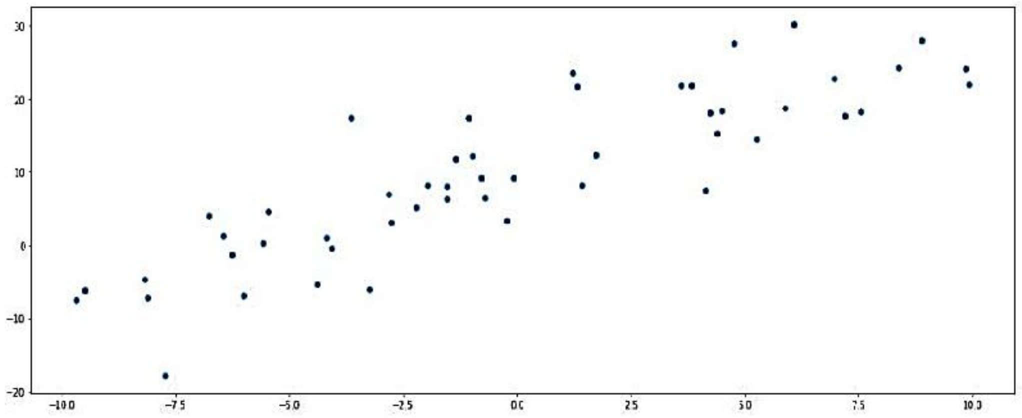


Figure 3 Scatter Plot

Step #3: Create and Fit Linear Regression Models

Let's now build a model using the scikit learn package's linear regression approach. The default approach is Ordinary Least Squares.

Recall that:

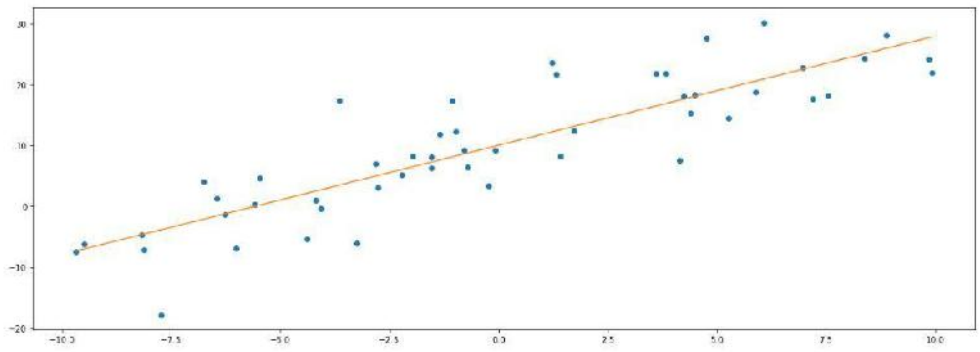
- The numpy array x1 is converted to a matrix because the sklearn package requires it.
- reshape(-1,1): - 1 represents 1 column and instructs NumPy to retrieve the number of rows from the original x1.
- We begin by making an instance of the class LinearRegression, abbreviated lr.
- The fit method's input parameters can vary, but we'll leave them at their default values.

```
features = x1.reshape(-1, 1)

target = y

lr = LinearRegression()

lr.fit(features, target)
```

Step #4: Check the Result Model: coefficients and plot

After fitting the model, we can call its attributes to look at the results. We can print out the coefficients.

```
print (lr.intercept_)
```



```
print(lr.coef_)
```

We can see that while.coef_ returns an array,.intercept_ returns a scalar. These coefficients' values are fairly close to their actual values (y = 10 + 2*x1). The regression line and training dataset can both be seen together in a visualization.

```
plt.figure(figsize=(20,7))
plt.plot(x1, y, 'o')x_chart = np.linspace(x1.min(), x1.max(), num=100)
plt.plot(x_chart, lr.intercept_ + lr.coef_[0]*x_chart)
```

Step #5: Make Predictions with Linear Regression!

The most thrilling part is this. Using our new model, let's make a prediction. Assuming we have four new x1 input values (0, 1, 2, 3), we have two options:

Use the scikit-learn predict method or manually enter the values into the equation to predict.

```
#use the equation to predict
x_new = np.array([0, 1, 2, 3])
y_prediction = lr.intercept_ + x_new*lr.coef_[0]
y_prediction
# use model to predict
lr.predict(x_new.reshape(-1,1))
```

Both methods return the same predicted values.

Output:

```
array([10.06295511, 11.85833473, 13.65371434, 15.44909395])
```

14.3 K-Nearest Neighbor

One of the simplest machine learning algorithms, based on the supervised learning method, is K-Nearest Neighbor. The K-NN algorithm makes the assumption that the new case and the existing cases are comparable, and it places the new instance in the category that is most like the existing categories. A new data point is classified using the K-NN algorithm based on similarity after all the existing data has been stored. This means that utilizing the K-NN method, fresh data can be quickly and accurately sorted into a suitable category. Although the K-NN approach is most frequently employed for classification problems, it can also be utilized for regression. Since K-NN is a non-parametric technique, it makes no assumptions about the underlying data. It is also known as a lazy learner algorithm since it saves the training dataset rather than learning from it immediately. Instead, it uses the dataset to perform an action when classifying data. The KNN algorithm simply stores the dataset during the training phase, and when it receives new data, it categorizes it into a category that is very similar to the new data. Consider the following scenario: We have an image of a creature that resembles both cats and dogs, but we are unsure of its identity.

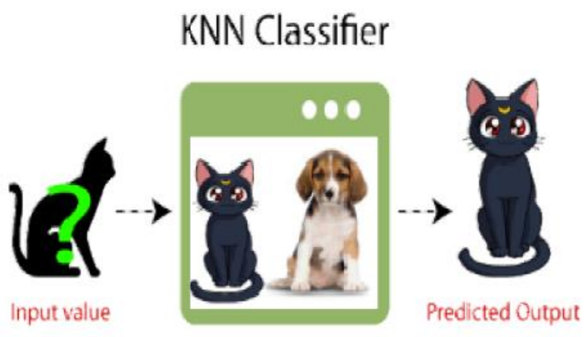
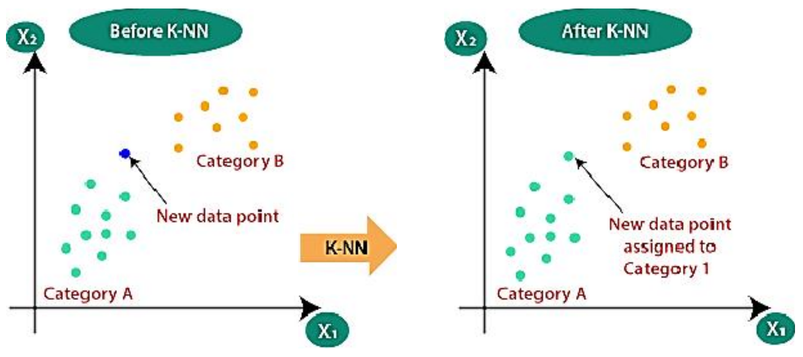


Figure 4 K-NN Classifier

Why do we need a K-NN Algorithm?

If there are two categories, Category A and Category B, and we have a new data point, x1, which category does this data point belong in? We require a K-NN algorithm to address this kind of issue. K-NN makes it simple to determine the category or class of a given dataset. Take a look at the diagram below:



How does K-NN work?

The following algorithm can be used to describe how the K-NN works:

Step 1: Decide on the neighbors' K-numbers.

Calculate the Euclidean distance between K neighbors in step two.

Step 3: Based on the determined Euclidean distance, select the K closest neighbors.

Step 4: Count the number of data points in each category among these k neighbors.

Step 5: Assign the fresh data points to the category where the neighbor count is highest.

Step 6: Our model is complete.

14.4 <u>Decision Trees</u>

A non-parametric supervised learning technique for classification and regression is called a decision tree (DT). The objective is to learn straightforward decision rules derived from the data features in order to build a model that predicts the value of a target variable. A piecewise constant approximation of a tree can be thought of.

For instance, in the example below, using a set of if-then-else decision rules, decision trees learn from data to approximate a sine curve. The decision rules are more complex and the model is more accurate the deeper the tree is.

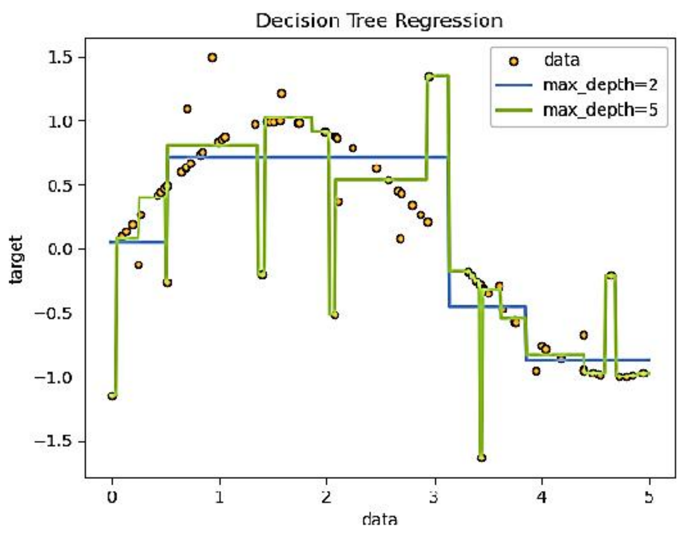


Figure 5 Decision Trees Regression

Advantages of Decision Trees

- Easy to comprehend and interpret. One can picture trees.
- Little data preparation is necessary. Data normalization, the creation of dummy variables, and the elimination of blank values are frequently necessary for other procedures. The module does not, however, support missing values.
- As more data points are utilized to train the tree, the cost of using it to forecast data increases exponentially.
- Capable of working with both categorical and numerical data. Categorical variables are not currently supported by the Scikit-Learn implementation. Other strategies are mainly specialized in studying datasets that have only one sort of variable. For more details, refer to algorithms.
- able to manage issues with several outputs.
- using the white box model. Boolean logic makes it simple to explain a condition if it can be observed in a model for a given situation. Results may be more challenging to interpret in a black box model, such as an artificial neural network.
- It is possible to use statistical tests to verify a model. This enables the model's dependability to be taken into account.
- performs well even if the underlying model from which the data were created slightly violates some of its basic assumptions.

Disadvantages of Decision Trees

- The too complicated trees that decision-tree learners can produce do not effectively generalize the input. Overfitting is the term for this. To prevent this issue, mechanisms like pruning, defining the minimum number of samples needed at a leaf node, or establishing the maximum depth of the tree are required.
- Because even slight changes in the data could produce an entirely different tree, decision trees can be unstable. The solution to this issue is to employ decision trees as part of an ensemble.
- As can be seen in the above graphic, decision tree predictions are piecewise constant approximations rather than smooth or continuous predictions. They therefore struggle with extrapolation.

- It is well known that learning an optimum decision tree under various conditions of optimality, even for straightforward notions, is an NP-complete issue. Because each node makes judgments that are locally optimal, heuristic algorithms like the greedy algorithm serve as the foundation for practical decision-tree learning algorithms. Such algorithms cannot promise to return the decision tree that is globally optimal. Multi-tree training in an ensemble learner with replacement sampling for the features and samples can help to mitigate this.
- Certain concepts, like XOR, parity, or multiplexer difficulties, are challenging to understand because decision trees do not simply describe them.
- If some classes predominate, decision tree learners will produce biased trees. As a result, it is advised to balance the dataset before fitting it to the decision tree.

14.5 Random Forest

Every decision tree has a significant variance, but when we mix them all in parallel, the variance is reduced since each decision tree is perfectly trained using that specific sample of data, so the output is dependent on numerous decision trees rather than just one. The majority voting classifier is used to determine the final output in a classification challenge. The final output in a regression problem is the mean of every output. Aggregation describes this section.

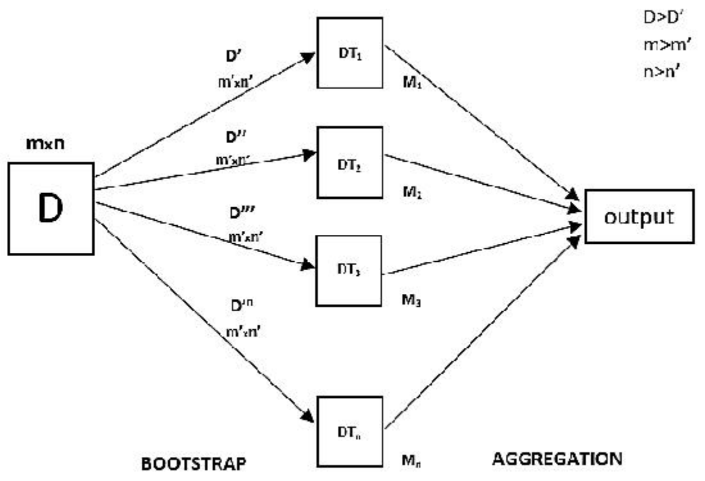


Figure 6 Random Forest

With the aid of several decision trees and a method known as Bootstrap and Aggregation, also referred to as bagging, Random Forest is an ensemble methodology capable of handling both regression and classification tasks. This method's fundamental principle is to integrate several decision trees to get the final result rather than depending solely on one decision tree.

Multiple decision trees serve as the fundamental learning models in Random Forest. We create sample datasets for each model by randomly selecting rows and features from the dataset. This component is known as Bootstrap.

The Random Forest regression technique must be approached similarly to other machine learning techniques.

- Create a specific query or set of data, then ask the source to provide the needed information.
- Make that the data is in a format that can be accessed; if not, convert it to the necessary format.
- List any obvious abnormalities and missing data that may be needed to obtain the desired data.
- Establish a machine learning model.
- Decide on the baseline model you wish to accomplish.
- train the machine learning model with the data.
- Using test data, provide insight into the model.

- Compare the test data and the model's projected data's performance metrics now.
- You can try updating your model accordingly, dating your data, or using another data modelling technique if it doesn't meet your expectations.
- You now interpret the information you have learned and report accordingly.
- In the example below, you will apply a similar sampling technique.

Here is a detailed example of how Random Forest Regression is implemented.

Importing the libraries

Step 1: Import the required libraries.

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

Step 2: Import and print the dataset

data = pd.read_csv('Salaries.csv')

print(data)

Step 3: Select all rows and column 1 from dataset to x and all rows and column 2 as y

# the coding was not shown which is like that

x= df.iloc [:, : -1] # ” : ” means it will select all rows,   “: -1 ” means that it will ignore last column

y= df.iloc [:, -1 :] # ” : ” means it will select all rows,   “-1 : ” means that it will ignore all columns except the last one

the “iloc()” function enables us to select a particular cell of the dataset, that is, it helps us select a value that belongs to a particular row or column from a set of values of a data frame or dataset.

Step 4: Fit Random Forest regressor to the dataset

# Fitting Random Forest Regression to the dataset

# import the regressor

from sklearn.ensemble import RandomForestRegressor

# create regressor object

regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)

# fit the regressor with x and y data

regressor.fit(x, y)

Step 5: Predicting a new result

p.array([6.5]).reshape(1, 1)) # test the output by changing values

Step 6: Visualising the result

# Visualising the Random Forest Regression results

# arrange for creating a range of values

# from min value of x to max

value of x with a difference of 0.01

# between two consecutive values

X_grid = np.arrange(min(x), max(x), 0.01)

# reshape for reshaping the data into a len(X_grid)*1 array,

# i.e. to make a column out of the X_grid value

X_grid = X_grid.reshape((len(X_grid), 1))

# Scatter plot for original data

```
plt.scatter(x, y, color = 'blue')
# plot predicted data
plt.plot(X_grid, regressor.predict(X_grid), color = 'green')
plt.title('Random Forest Regression')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

Output

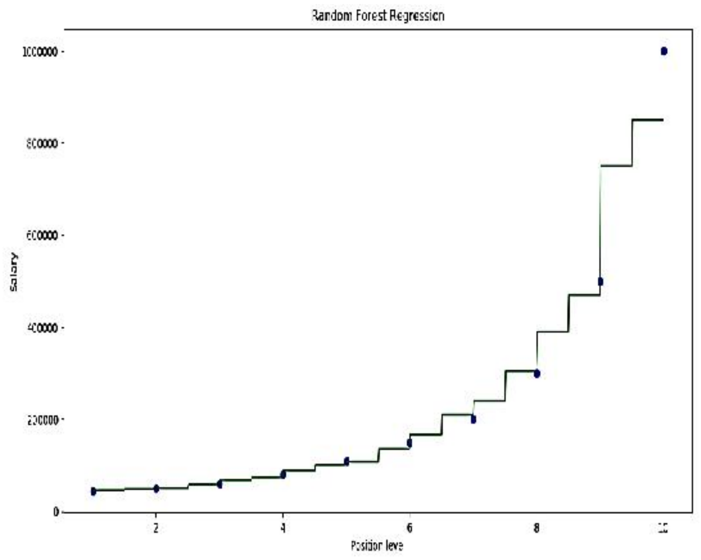


Figure 7 Random Forest Regression

14.6 K-Means Clustering Algorithm

The clustering issues in machine learning or data science are resolved using the unsupervised learning algorithm K-Means Clustering. In this chapter, we will learn what the K-means clustering algorithm is, how it operates, and how to implement it in Python.

What is K-Means Algorithm?

Unsupervised learning algorithm K-Means Clustering divides the unlabeled dataset into various clusters. Here, K specifies how many pre-defined clusters must be produced as part of the process; for example, if K=2, there will be two clusters, if K=3, there will be three clusters, and so on.

It is an iterative technique that separates the unlabeled dataset into k distinct clusters, with each dataset belonging to just one group with identical characteristics.

It gives us the ability to divide the data into various groups and provides a practical method for automatically identifying the groups in the unlabeled dataset without the need for any training.

Each cluster has a centroid assigned to it because the algorithm is centroid-based. This algorithm's primary goal is to reduce the total distances between each data point and its corresponding clusters.

It allows us to categories the data into different groups and offers a workable technique for quickly and accurately determining the groups in the unlabeled dataset without the need for any training.

The technique is centroid-based, and each cluster has a centroid given to it. Reducing the overall distances between each data point and its matching clusters is the main objective of this technique.

The algorithm starts with an unlabeled dataset as its input, separates it into k clusters, and then continues the procedure until it runs out of clusters to use. In this algorithm, the value of k should be predetermined.

The two major functions of the k-means clustering algorithm are:

- uses an iterative technique to choose the best value for K center points or centroids.
- each data point is matched with the nearest k-center. A cluster is formed by the data points that are close to a specific k-center.

As a result, each cluster is distinct from the others and contains data points with some commonality.

The K-means Clustering Algorithm is explained in the diagram below:

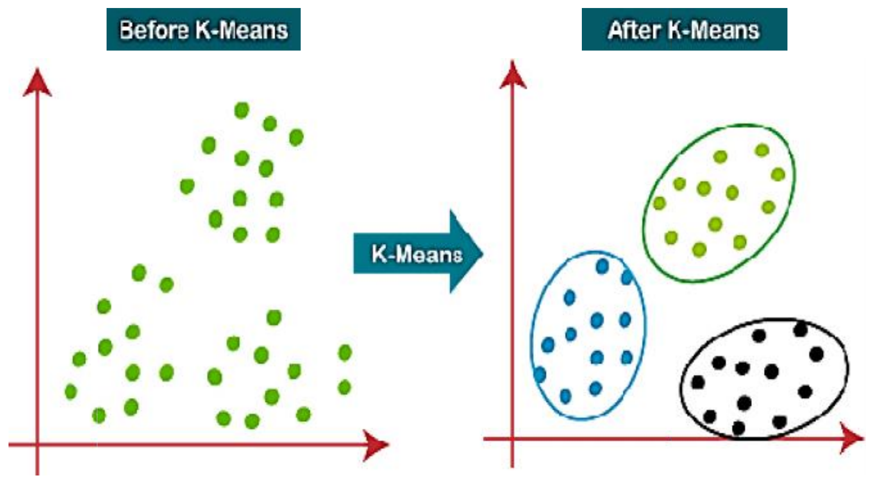


Figure 8 K-Means Algorithm

How Does K-Means Algorithm Work?

The following stages illustrate how the K-Means algorithm functions:

Step 1: To determine the number of clusters, choose K.

Step 2: Pick K locations or centroids at random. (It might not be the supplied dataset.)

Step 3: Assign each data point to its nearest centroid, which will create the K clusters that have been predetermined.

Step 4: Determine the variance and relocate each cluster's centroid.

Step 5: Re-assign each data point to the new centroid of each cluster by repeating the third step.

Step 6: Move to step 4 if there is a reassignment; otherwise, go to FINISH.

Step 7: The finished model

Summary

- The field of study known as machine learning enables computers to learn without being explicitly programmed.
- Astrategy for predicting a response based on a single feature is simple linear regression.
- A subset of machine learning and artificial intelligence is supervised learning, commonly referred to as supervised machine learning. It is distinguished by the way it trains computers to accurately classify data or predict outcomes using labelled datasets.
- In order to accurately classify test data into different categories, classification uses an algorithm.
- To comprehend the relationship between dependent and independent variables, regression is used.
- Neural networks process training data by simulating the connectivity of the human brain through layers of nodes, which is mostly used for deep learning algorithms.

- A classification method known as Naive Bayes adopts the idea of Class Conditional Independence from the Bayes Theorem.
- In order to anticipate future results, linear regression is frequently employed to determine the relationship between a dependent variable and one or more independent variables.
- While logistical regression is used when the dependent variable is categorical, or has binary outputs, such as "true" and "false" or "yes" and "no," linear regression is used when the dependent variable is continuous.
- Vladimir Vapnik created the well-known supervised learning model known as the support vector machine, which is used for both data classification and regression.
- The KNN algorithm, also referred to as K-nearest neighbor, is a non-parametric algorithm that groups data points according to their proximity and association with other pieces of available information.

Keywords

Linear Regression: When modelling the relationship between a scalar answer and one or more explanatory variables in statistics, linear regression is a linear method.

Linearity:This indicates that the parameters (regression coefficients) and predictor variables are combined linearly to produce the mean of the response variable.

Constant variance: This translates to the fact that the variance of the errors is independent of the values of the predictor variables. Therefore, regardless of how big or small the responses are, the variability of the responses for given fixed values of the predictors is the same.

Independence of Errors:This assumes that the errors of the response variables are uncorrelated with each other.

K Nearest Neighbor: One of the simplest Machine Learning algorithms based on the Supervised Learning technique is K-Nearest Neighbor.

Lazy Learner Algorithm: It is also known as a lazy learner algorithm since it saves the training dataset rather than learning from it immediately. Instead, it uses the dataset to perform an action when classifying data.

Euclidean Algorithm:The distance between two points, which we have already examined in geometry, is known as the Euclidean distance.

Decision Trees:A non-parametric supervised learning technique for classification and regression is called a decision tree (DT). The objective is to learn straightforward decision rules derived from the data features in order to build a model that predicts the value of a target variable.

Random Forests: Leo Breiman and Adele Cutler are the creators of the widely used machine learning technique known as random forest, which mixes the output of various decision trees to produce a single outcome.

K Means Clustering: The goal of k-means clustering, a vector quantization technique that originated in signal processing, is to divide n observations into k clusters, where each observation belongs to the cluster that has the closest mean (also known as the cluster centroid or cluster centre), which serves as a prototype for the cluster.

K Medoids: K-medoids, also known as Partitioning Around Medoids, or PAM, minimises the sum of distances for any given distance function by using the medoid rather than the mean.

Principal Component Analysis:Principal component analysis provides the k-means clustering's relaxed solution, which is determined by the cluster indicators (PCA)

K

Self Assessment

1. Choose the item from the list below that is not a kind of learning.

A. Semi supervisedunsupervisedlearning

B. Supervised learning
C. Unsupervised Learning
D. Reinforcement Learning

2. What is the term for the application of machine learning techniques to a big database?
A. Supervised learning
B. Unsupervised Learning
C. Reinforcement Learning
D. Data mining

3. Machine learning approaches can be traditionally categorized into ______ categories
A. 2
B. 3
C. 4
D. 5

4. The following effects occur as the number of features increases:
A. longer computation times
B. more complex models
C. worse learning accuracy
D. all of the above.

5. The k-means algorithm is a
A. Supervised learning algorithm
B. Unsupervised learning algorithm
C. Semi-supervised learning algorithm
D. Weakly supervised learning algorithm

6. We have a model for unsupervised learning called.
A. interactive
B. predictive
C. descriptive
D. prescriptive

7. Any machine learning model's success Hinges on the engineering of a good feature space.
A. Pre-requisite
B. Process
C. Objective
D. None of the above

8. An example would be determining whether a tumour is benign or malignant.
A. Unsupervised Learning
B. Supervised Regression Problem
C. Supervised Classification Problem
D. Categorical Attribute

9. This is a reference to the adjustments made to the detected data before to feeding it to the algorithm.
A. Problem Identification
B. Identification of Required Data
C. Data Pre-processing
D. Definition of Training Data Set

10. Which of the following is true about SVM?
A. It is useful only in high-dimensional spaces
B. It requires less memory
C. SVM does not perform well when we have a large data set
D. SVM performs well when we have a large data set

11. Which of the following kNN choices would you take into account if there are a lot of sounds in the data?
A. Increase the value of k
B. Decrease the value of k
C. Noise does not depend on k
D. k = 0

12. Decision tree defined as
A. Flow-Chart
B. Structure in which internal node represents test on an attribute, each branch represents outcome of test and each leaf node represents class label.
C. Flow-Chart & Structure in which internal node represents test on an attribute, each branch represents outcome of test and each leaf node represents class label.
D. None of the mentioned

13. Choose from the following that are Decision Tree nodes?
A. Decision Nodes
B. End Nodes
C. Chance Nodes
D. All of the mentioned

14. Decision Nodes are depicted by
A. Disks
B. Squares
C. Circles
D. Triangles

15. Which benefit(s) of the following apply to decision trees?
A. Possible Scenarios can be added
B. Use a white box model, If given result is provided by a model
C. Worst, best and expected values can be determined for different scenarios
D. All of the mentioned

Answers for Self Assessment

1.	A	2.	D	3.	B	4.	D	5.	B
6.	C	7.	A	8.	C	9.	C	10.	D
11.	A	12.	C	13.	D	14.	B	15.	D

Review Question

Further Readings

- Mark Lutz,Programming Python: Powerful Object-Oriented Programming, OREILLY
- Wes McKinney, Python for data analysis, OREILLY
- David Ascher and Mark Lutz, Learning Python, OREILLY
- Eric Matthes, Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming, Starch Pres

Web Links

- https://www.tutorialspoint.com/python/index.htm
- https://www.python.org/downloads/
- https://www.w3schools.in/python/data-types
- https://www.programiz.com/python-programming/online-compiler/
- https://www.codecademy.com/catalog/language/python

LOVELY PROFESSIONAL UNIVERSITY
Jalandhar-Delhi G.T. Road (NH-1)
Phagwara, Punjab (India)-144411
For Enquiry: +91-1824-521360
Fax.: +91-1824-506111
Email: odl@lpu.co.in

ISBN 978-81-19334-41-4
9 788119 334414