IPRC HUYE

Integrated Polytechnic Regional College

P.o.Box 575 HUYE-Rwanda
Tel:+250255119248
Email: info@iprchuye.rp.ac.rw
www.iprchuye.rp.ac.rw

Module Title:PYTHON AND FUNDAMENTALS OF AI

Module Code: ITLPA701

RTQF Level: 7

Sector: ICT

Module Leader:NIYIGENA Claver&NDIZEYE Patrick.

MODULE NOTES

TABLE OF CONTENTS

LEARNING UNIT 1– PREPARE PYTHON ENVIRONMENT

1.1 Identify Python version

1.1.1 Definition of terms

1.1.1.1 Python

Python is a programming language commonly used for developing websites and software, task automation, data analysis, and data visualization. Since it's relatively easy to learn, Python has been adopted by many non-programmers such as accountants and scientists, for a variety of everyday tasks, like organizing finances. It was created by Guido van Rossum and first released in 1991.

Key features of Python include:

Readability: Python's syntax is designed to be clear and readable, which makes it easy for developers to write and maintain code.

Interpretation: Python is an interpreted language, which means that the source code is executed line by line by an interpreter. This makes it easier to debug and test code.

Object-Oriented: Python supports object-oriented programming, allowing developers to structure code using classes and objects.

Dynamic Typing: Python is dynamically typed, meaning that variable types are interpreted at runtime, making it more flexible.

Extensive Standard Library: Python comes with a large standard library that includes modules and packages for various purposes, reducing the need for developers to write code from scratch.

1.1.1.2 Programming language

A programming language is a formalized set of instructions that a computer can interpret and execute to perform specific tasks or operations. It serves as a medium for communication between a human programmer and a computer, enabling the creation of software and applications. Programming languages provide a way for humans to express algorithms, data structures, and logic in a format that can be understood and processed by a computer.

Key characteristics of programming languages include:

Syntax: Programming languages have a set of rules governing the structure and combination of symbols, keywords, and expressions. The syntax defines how the code should be written to be valid and understandable by the compiler or interpreter.

Semantics: Semantics refers to the meaning or interpretation of the code. It defines the behavior of the instructions and how they interact with each other to produce desired outcomes.

Abstraction: Programming languages provide abstraction mechanisms that allow developers to hide complex details and focus on higher-level concepts. This helps in managing the complexity of software development.

Variables and Data Types: Programming languages allow the declaration and manipulation of variables, which can store and represent data. Data types define the kind of data a variable can hold, such as integers, strings, or floating-point numbers.

Control Structures: Programming languages include constructs for controlling the flow of execution, such as loops (for, while) and conditional statements (if-else).

Functions or Procedures: Functions allow the decomposition of a program into smaller, reusable units. They take input, perform specific tasks, and return results.

Libraries and Frameworks: Many programming languages come with libraries or frameworks that provide pre-built functionalities, saving developers time and effort.

Examples of programming languages include Python, Java, C++, JavaScript, Ruby, and many more. Each language is designed with specific goals and use cases in mind, and different languages may be more suitable for certain types of applications or tasks. The choice of programming language often depends on factors such as project requirements, performance considerations, and personal or organizational preferences.

1.1.1.3 Use of python

Serve

Python is used for server-side scripting, allowing developers to embed Python code within web pages or server-side scripts to create dynamic content.

Software development

Python is often used to develop the back end of a website or application—the parts that a user doesn't see. Python's role in web development can include sending data to and from servers, processing data and communicating with databases, URL routing, and ensuring security. Python offers several frameworks for web development. Commonly used ones include Django and Flask.

Mathematics

Python is a powerful programming language that is widely used in various fields, and mathematics is no exception. Python provides extensive support for mathematical operations and computations through built-in functions, libraries, and modules. Here are some ways mathematics is used in Python: Arithmetic Operations, Math Module: Python's standard library includes a math module that provides a wide range of mathematical functions. This module includes functions for trigonometry, logarithms, exponentiation, rounding, and more, etc…

System scripting

A script, in the context of computer programming, refers to a set of instructions or commands written in a scripting language that is executed by a computer. This means that the script's instructions are executed line by line by an interpreter or runtime environment.

Examples of scripts include:

- **Shell Scripts:** Used for automating tasks in a Unix or Linux shell environment. Python Scripts: Python is often used for scripting tasks, thanks to its readability and versatility. i.e Automating backup
- Scripting is a very common practice among Python programmers. It's used for automation of daily tasks, reporting, server management, security, social media management, business growth and development, financial trading, automating software and many other intelligent solutions.

Server-side

- Web Development:

Django: A high-level web framework that follows the "don't repeat yourself" (DRY) principle. It facilitates rapid development of web applications with a clean and pragmatic design.

- **Flask:** A lightweight and flexible web framework that is easy to use and suitable for small to medium-sized web applications. Flask provides the essentials for building web applications without imposing too much structure.
- **API Development:**
- Python is often used to build RESTful APIs (Application Programming Interfaces) that allow communication between different software systems. Libraries like Flask and Django REST framework make it easy to develop robust APIs.
- Backend Services:
- Python is employed in building the backend logic for various services, handling tasks such as data processing, authentication, authorization, and business logic.
- Content Management Systems (CMS):
- Python can be used to create content management systems for websites. Examples include Wagtail, a Django-based CMS, and Plone, a content management system built on top of Zope.
- Data Processing and Analysis:
- Python is used for server-side data processing and analysis tasks. Libraries like NumPy, Pandas, and SciPy are commonly used for handling and analyzing large datasets.
- Task Automation:

- Python scripts running on servers are commonly used for task automation, whether it's periodic maintenance tasks, file processing, or system monitoring.
- Microservices Architecture:
- Python is often part of microservices architectures, where small, independent services communicate with each other to build scalable and maintainable applications.
- Server-side Scripting:
- Python is used for server-side scripting, allowing developers to embed Python code within web pages or server-side scripts to create dynamic content.
- Networking:
- Python is utilized for server-side networking tasks, such as building network servers, handling network protocols, and implementing network-related functionalities.
- Cloud Computing:
- Python is commonly used for interacting with cloud services and APIs. Platforms like AWS, Azure, and Google Cloud provide Python SDKs for managing resources and deploying applications.
- Machine Learning Deployment:
- Python is used for deploying machine learning models on the server side. Frameworks like Flask or FastAPI are often employed to create APIs for serving machine learning predictions.

1.1.2 Characteristic of python

Python is a versatile programming language known for its simplicity, readability, and flexibility. Here are some key characteristics of Python:

Readable and Simple Syntax:

- Python emphasizes readability and a clean syntax, making it easy for developers to express concepts with fewer lines of code. This readability is supported by the use of indentation to define code blocks.

Interpreted Language:

- Python is an interpreted language, meaning that the source code is executed line by line by an interpreter at runtime. This allows for quick testing and debugging.

High-Level Language:

- Python is a high-level programming language, abstracting low-level details and providing a more straightforward and human-readable syntax. This makes it accessible for beginners and efficient for experienced developers.

Object-Oriented Programming (OOP):

- Python supports object-oriented programming, allowing developers to structure code using classes and objects. Encapsulation, inheritance, and polymorphism are fundamental concepts in Python's OOP paradigm.

Dynamic Typing:

- Python is dynamically typed, meaning that the data type of a variable is interpreted at runtime. This provides flexibility but requires careful attention to variable types during development.

Cross-Platform Compatibility:

- Python code is generally platform-independent. A Python program written on one operating system (e.g., Windows) can run on another (e.g., Linux) with minimal or no modifications.

Extensive Standard Library:

- Python comes with a comprehensive standard library that includes modules and packages for various tasks. This eliminates the need for developers to write code from scratch for common functionalities.

Integration Capabilities:

- Python can easily integrate with other languages and technologies, making it a popular choice for building complex and integrated systems. It supports interfacing with C, C++, and Java, among others.

1.1.3 Application of python

- Applications of Python:
- **Web Development:** Python is widely used for building web applications and websites. Frameworks like Django and Flask simplify the process of developing web applications.
- **Data Science and Machine Learning:** Python is a popular choice for data analysis, machine learning, and artificial intelligence. Libraries like NumPy, Pandas, Scikit-learn, and TensorFlow are widely used in these domains.
- **Automation and Scripting:** Python is commonly used for scripting and automation tasks. It's ideal for writing small utility scripts as well as large-scale automation systems.
- **Scientific Computing:** Python is used in scientific computing for tasks such as simulations, numerical analysis, and data visualization. Libraries like SciPy and Matplotlib are commonly employed for these purposes.
- **Game Development:** Python is used in game development, often with the help of libraries like Pygame.

- **Network Programming:** Python is used for network programming, including the development of networking tools and applications.
- **Desktop GUI Applications:** Python can be used to create desktop graphical user interface (GUI) applications using libraries such as Tkinter, PyQt, or Kivy.
- **Embedded Systems:** Python is used in developing software for embedded systems and IoT (Internet of Things) devices.
- The versatility of Python, combined with its simplicity and readability, has contributed to its widespread adoption across various industries and application domains.

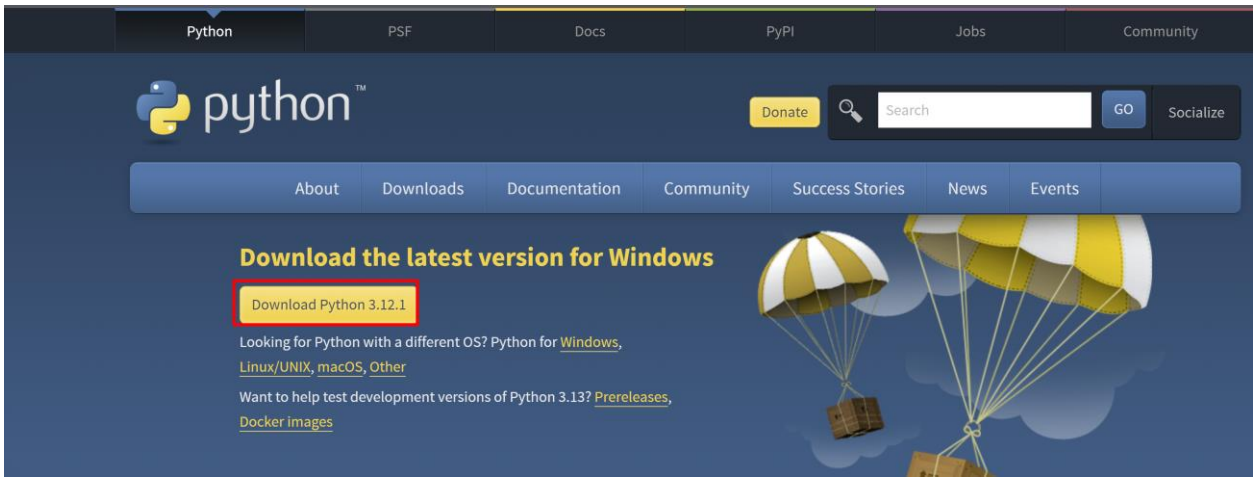
1.2 Add python Directory

1.2.1 Local Environment set up.

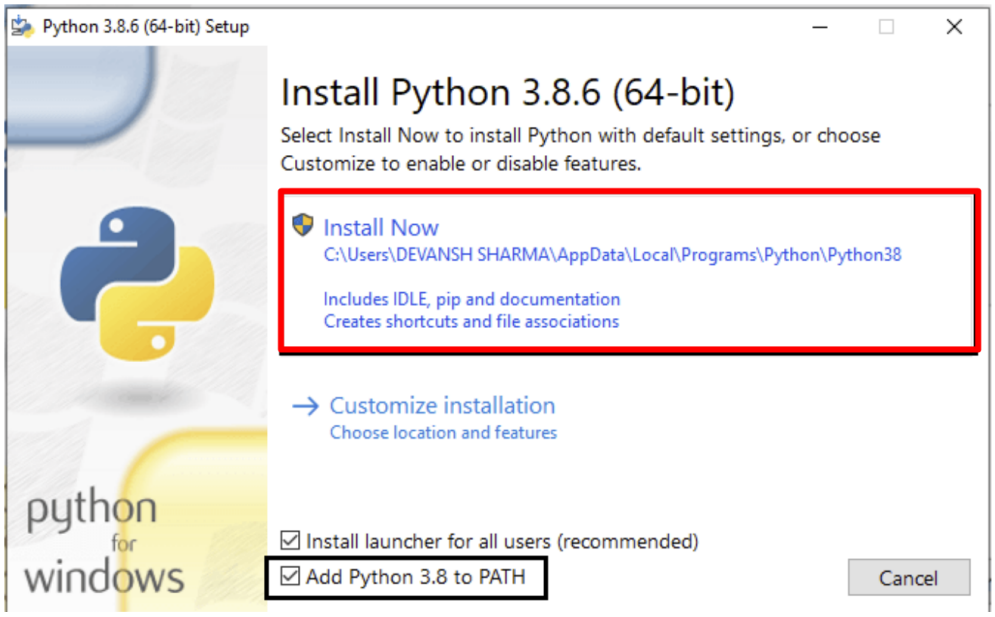
Setting up a local development environment for Python involves installing the necessary tools and libraries on your computer. Here are the general steps for setting up a Python environment on your local machine:

1.2.1.1 Windows

Step 1: Visit the official Python website (Download Python | Python.org) and download the latest version of Python. Follow the installation instructions for your operating system.

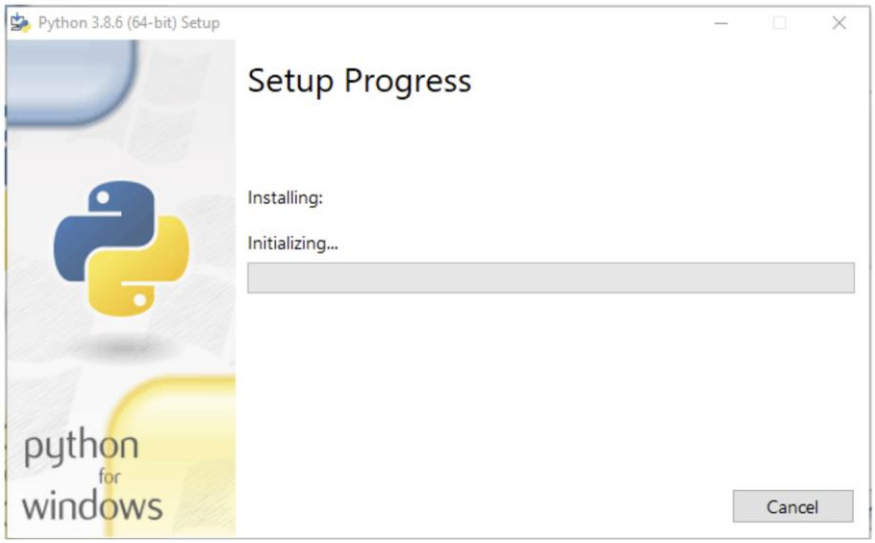


Step 2: Double-click the executable file, which is downloaded; the following window will open. Select Customize installation and proceed. Click on the Add Path check box, it will set the Python path automatically.



We can also click on the customize installation to choose desired location and features. Other important thing is install launcher for the all user must be checked.

Step 3. Installation in Process



Step 4: Verify Python Installation:

Open a terminal or command prompt and run the following commands to verify that Python has been installed correctly:

python --version # or python -V

Install a Text Editor or Integrated Development Environment (IDE):

Step 5: Choose a text editor or an integrated development environment for writing and running Python code. Some popular choices include:

Visual Studio Code

PyCharm

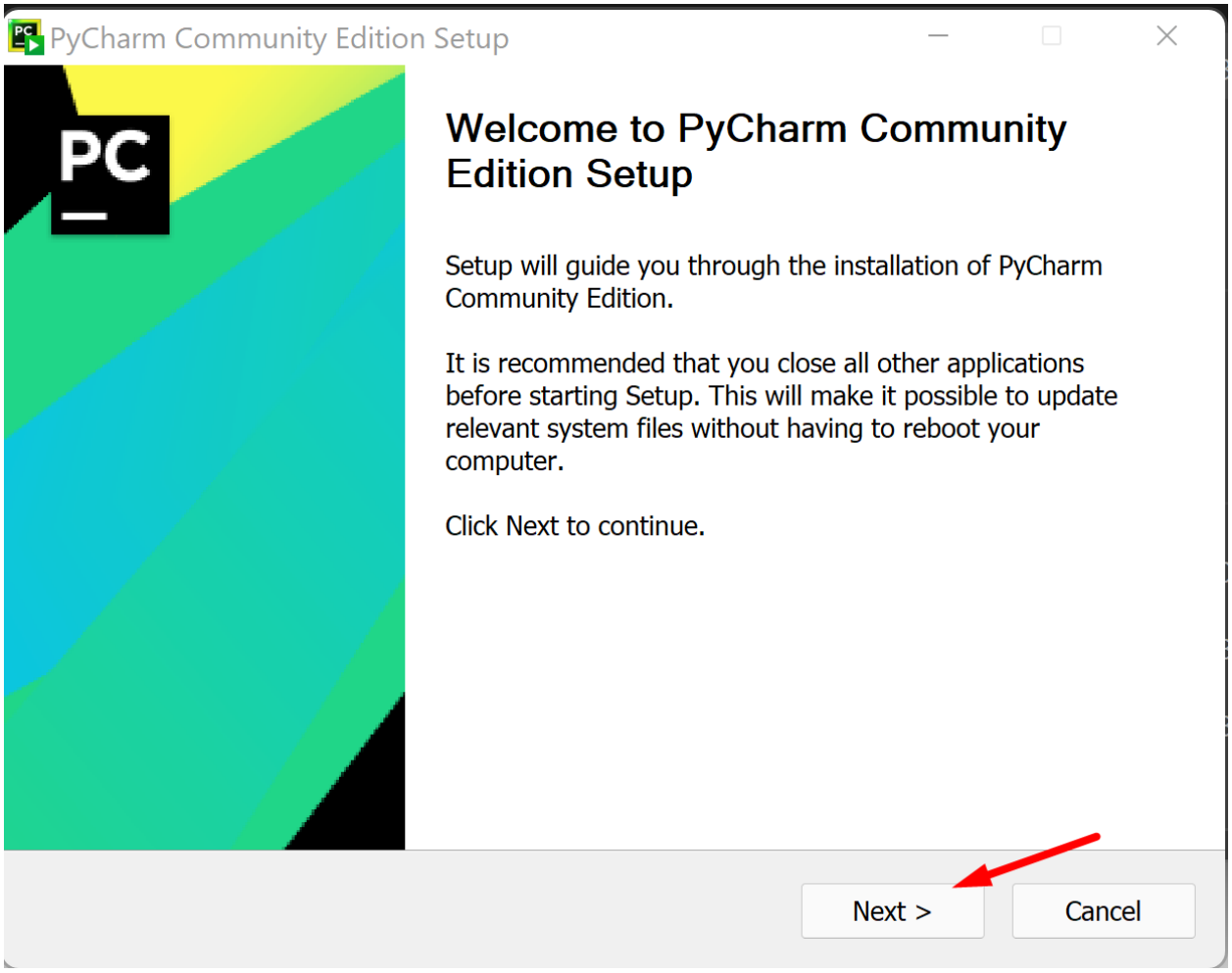
Atom

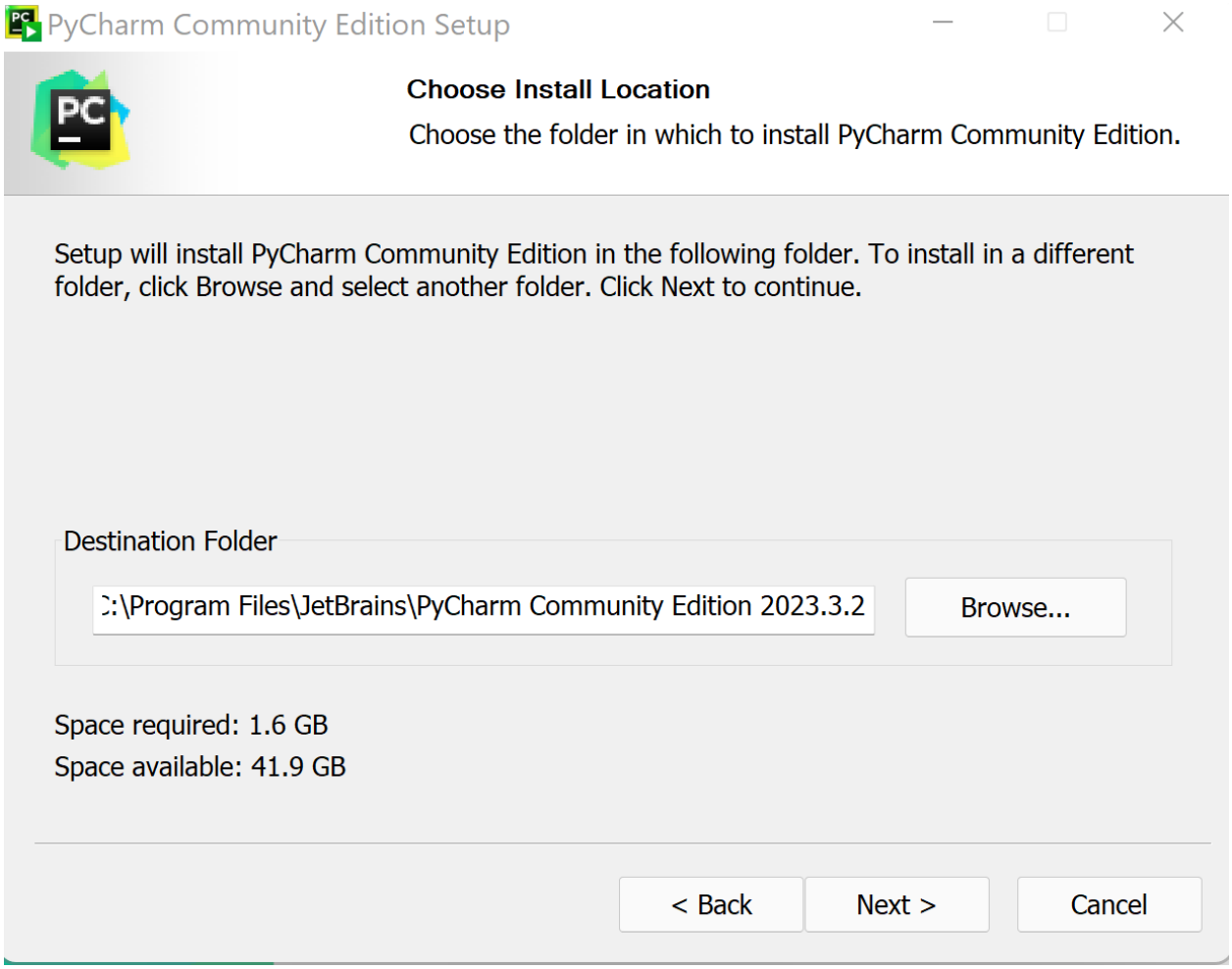
Sublime Text

Jupyter Notebooks

Download and install your preferred editor/IDE.(PyCharm)

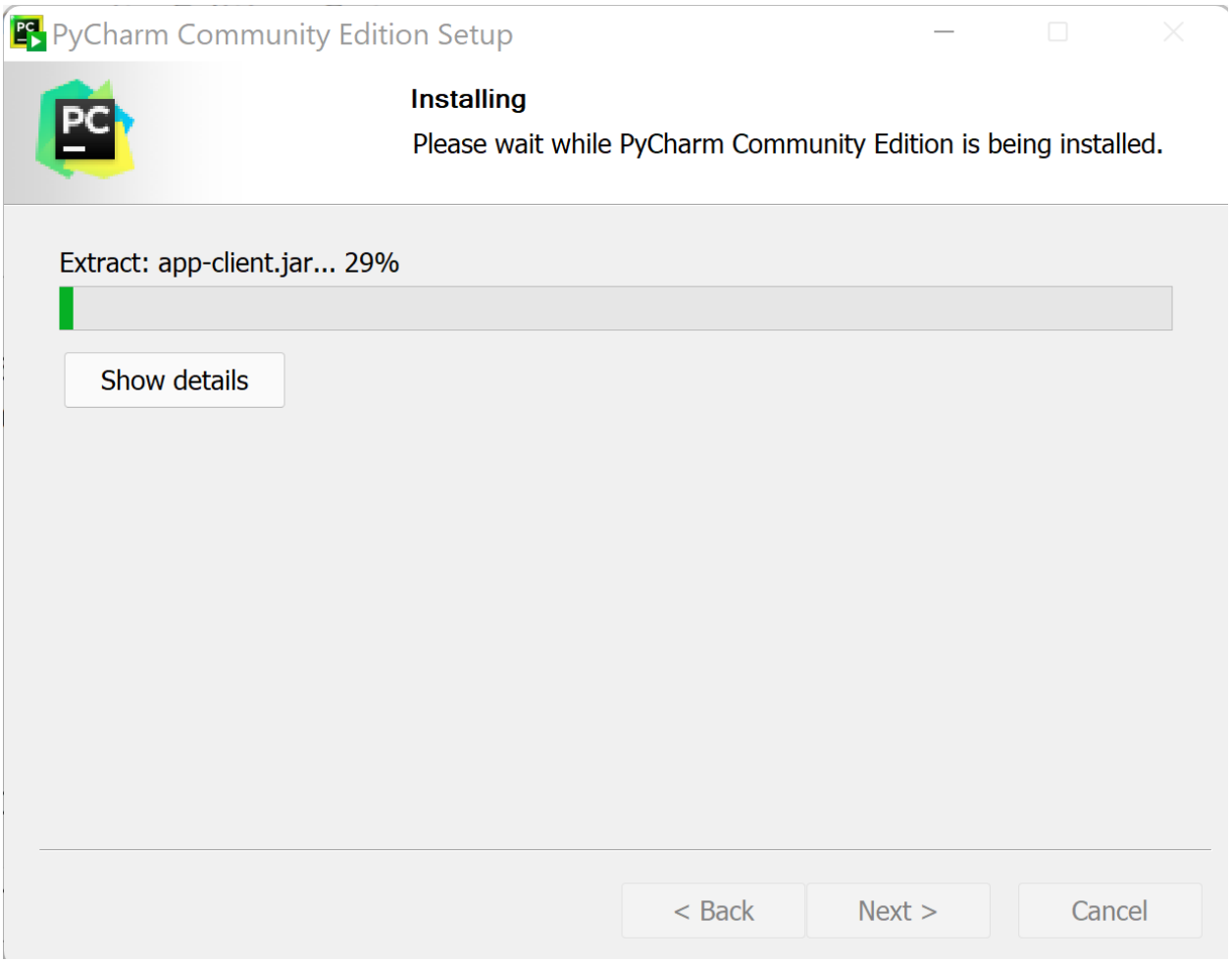
Step1: download pycharm from



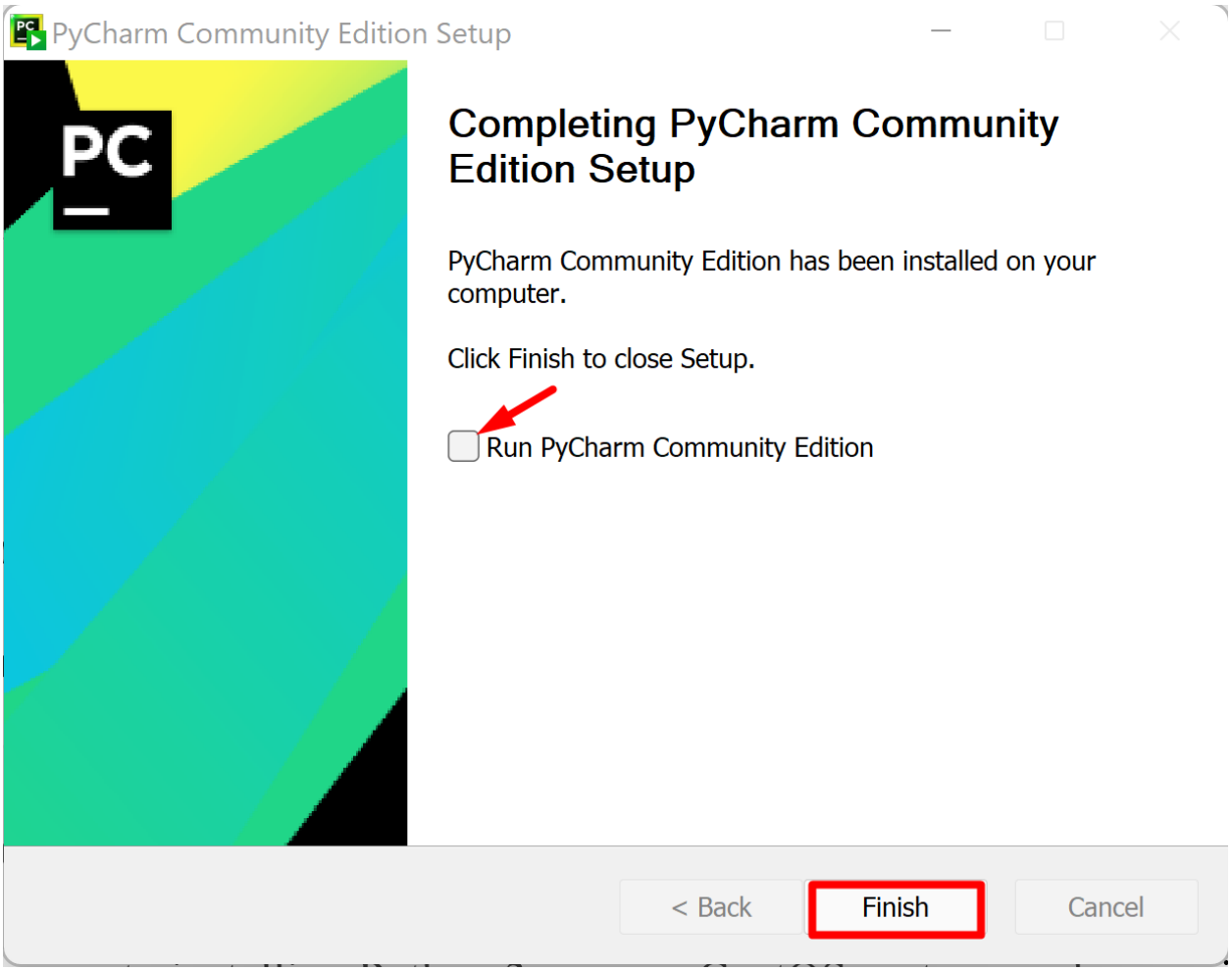


Step 2:

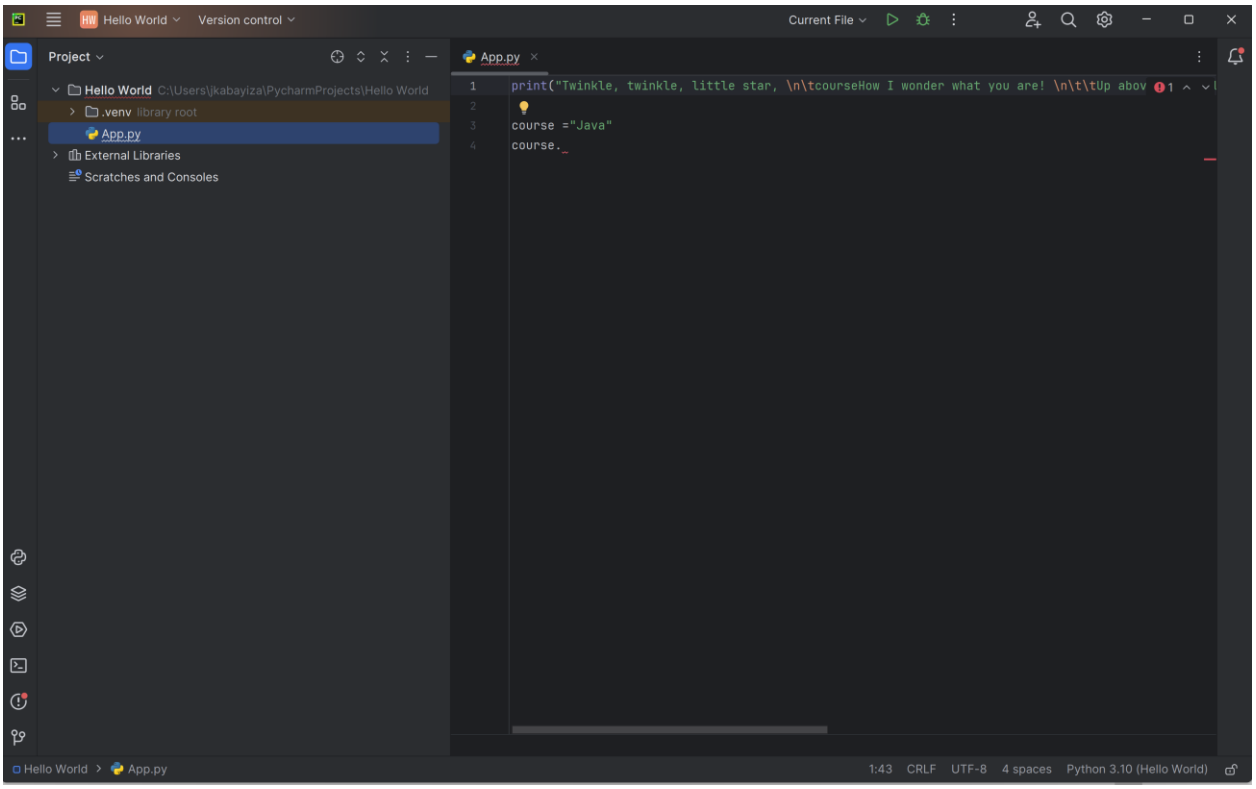
Step 3: Select Next until you finish



Step 4: Check the button and click Finish



Pycharm interface



1.2.1.2 Linux/Unix (CentOS 7/Redhat/SUSE/UBUNTU)

Installing Python on CentOS 7

1. Start by updating the repository:

```
sudo yum update -y
```

2. Install Python 3 by running the following command in the terminal window:

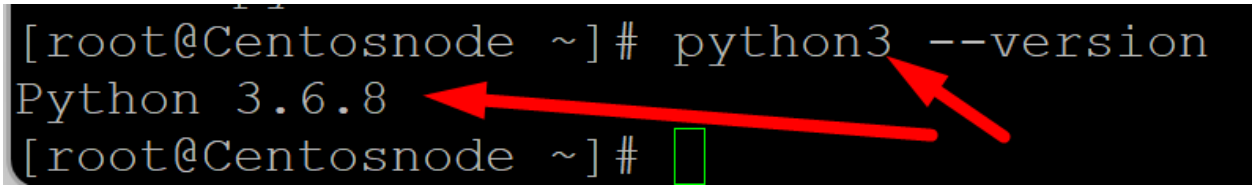
```
sudo yum install -y python3
```

Wait for the installation to complete. The output should display it has installed python3 and the required dependencies.

4. Verify you have successfully installed Python 3 with:

```
python3 --version
```

You should see the version of Python 3 now available on your CentOS system.



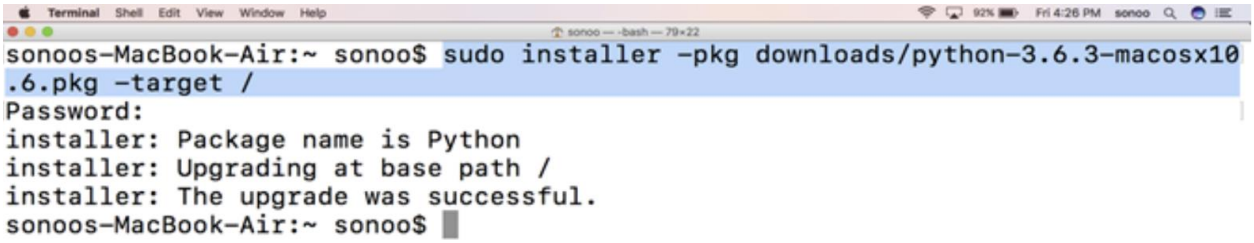
1.2.1.3. Installation on Mac OS

Step1: Download Python 3.12.1 In order to install Python 3.12.1, we must download the latest version from its official website https://www.python.org/downloads/. The file is downloaded in .pkg format which can be directly installed by using Installer command.

There are following steps which are used while installing Python3 on MacOS.

Python Releases for macOS | Python.org

Step 2. Install Python 3.12.1 Since the downloaded file already is in .pkg format hence no mounting is required and We can use installer command to install Python 3.12.1. Let's see how we can do it



Step 3. Checking python's version on the system We can check which version of the Python is currently installed on our system.

Generally, Python 3.12.1 is installed by default. Let's see how we can do it.

$ python -version

1.2.2 Set up path

1.2.3.1 Find Python Installation Path

If you need to manually find the path where Python is installed, you can usually find it in the Scripts folder within the Python installation directory. The typical path is C:\Users\YourUsername\AppData\Local\Programs\Python\PythonXX\Scripts, where YourUsername is your Windows username, and XX represents the Python version.

1.2.3.1.1 Update PATH Environment Variable

For Windows 10:

Right-click on the Start menu and select "System."

Click on "Advanced system settings" on the left.

Click the "Environment Variables..." button.

In the "User variables" or "System variables" section, find and select the "Path" variable, then click "Edit..."

Click "New" and add the path to the "Scripts" folder within your Python installation directory. For example, C:\Users\YourUsername\AppData\Local\Programs\Python\PythonXX\Scripts.

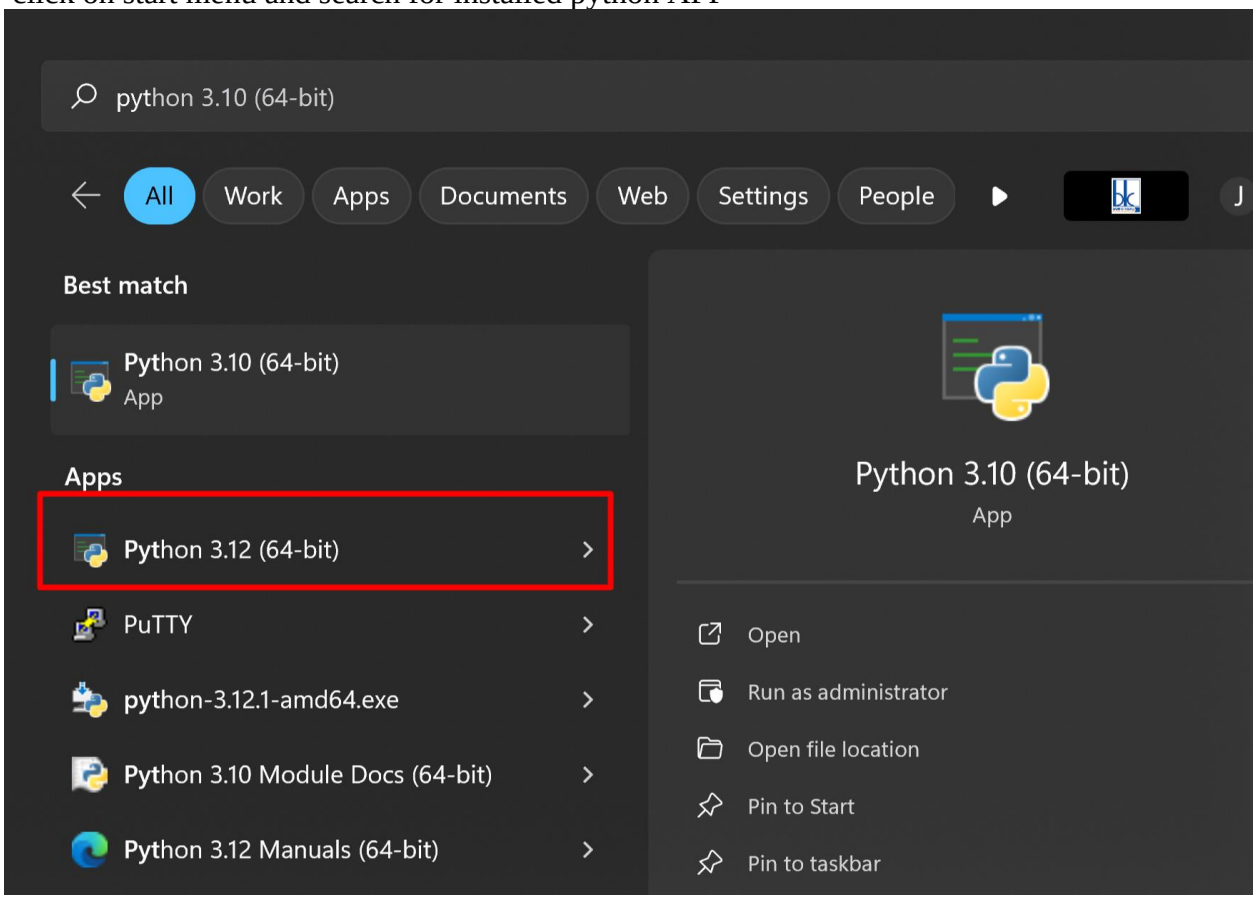
Click "OK" to close each window.

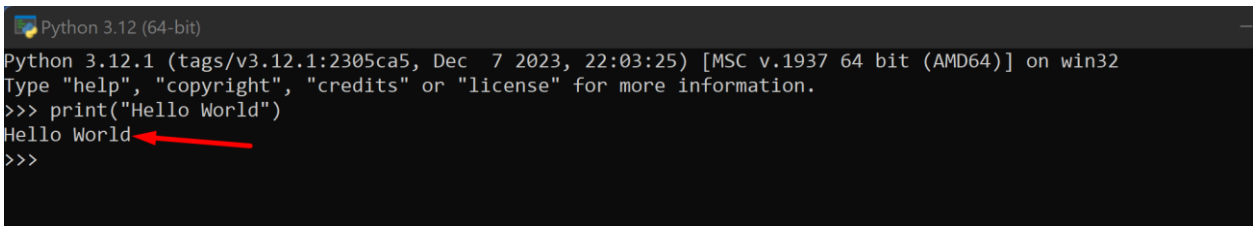
1.2.3.2 Running python

Interactive Interpreter You can start Python from Unix, DOS, or any other system that provides you a commandline interpreter or shell window. Enter python the command line.

Windows

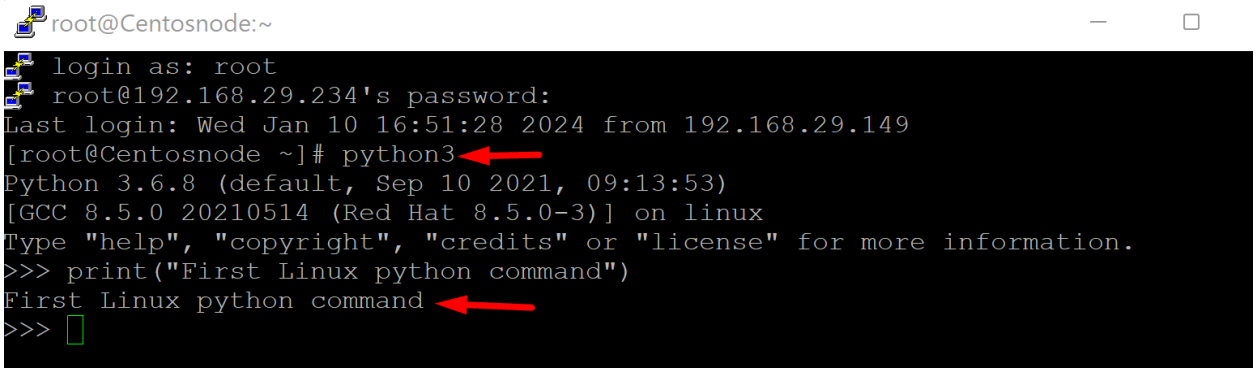
click on start menu and search for installed python APP





Linux

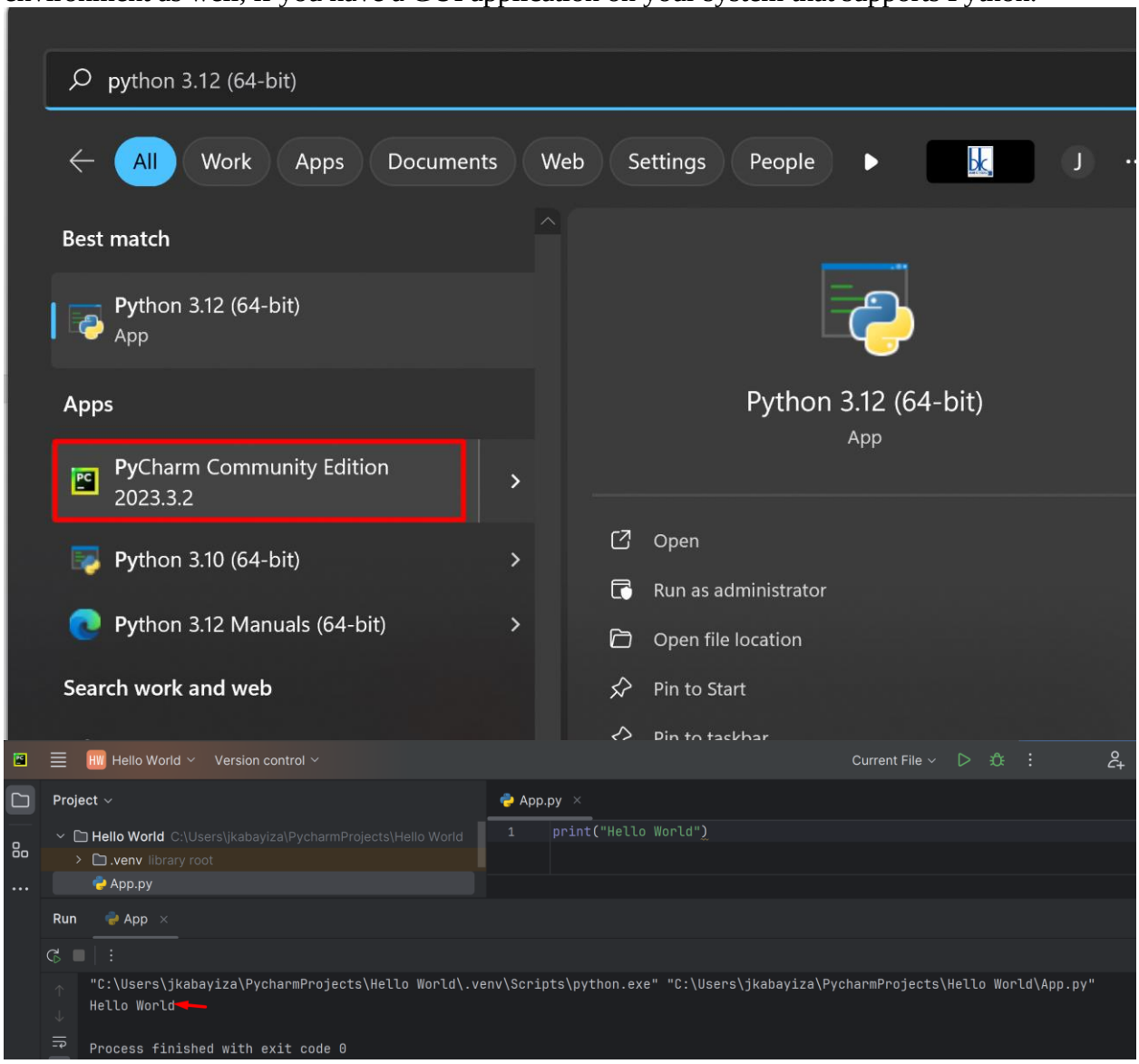
Type python3 command in linux terminal



1.2.3.3 Integrated Development Environment(IDE) or GUI

Integrated Development Environment You can run Python from a Graphical User Interface (GUI)

environment as well, if you have a GUI application on your system that supports Python.



1.3 Identify library

1.3.1 Definition of key terms

1.3.1.1 Library

A Python library is a collection of modules and functions that allow you to perform specific tasks or implement certain functionalities in your Python programs. Libraries are pre-written code that you can reuse in your own projects, saving you time and effort by not having to write everything from scratch. Python has a rich ecosystem of libraries that cover a wide range of domains, such as data science, machine learning, web development, networking, graphics, and more. These libraries are typically distributed as packages and can be easily installed using tools like pip, which is the default package installer for Python.

Example: Write a Python program that calculates the area of a circle based on the radius entered by the user.

Sample Output :

r = 1.1

Area = 3.8013271108436504

Sol/

```
# Import the 'pi' constant from the 'math' module to calculate the area of a circle
from math import pi

# Prompt the user to input the radius of the circle
r = float(input("Input the radius of the circle : "))

# Calculate the area of the circle using the formula: area = π * r^2
area = pi * r ** 2

# Display the result, including the radius and calculated area
print("The area of the circle with radius " + str(r) + " is: " + str(area))
```

Some popular Python libraries include:
NumPy: For numerical computing, providing support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.
Pandas: Used for data manipulation and analysis. It provides data structures like DataFrame for efficiently handling and analyzing structured data.
Matplotlib and Seaborn: These libraries are used for data visualization, helping you create various types of plots and charts.
Requests: Simplifies the process of making HTTP requests, allowing you to interact with web services and APIs.
Django and Flask: These are web frameworks that simplify the process of building web applications in Python.
TensorFlow and PyTorch: Popular libraries for machine learning and deep learning, providing tools for building and training neural networks.
Beautiful Soup: A library for web scraping, allowing you to extract information from web pages.

1.3.1.2 Package

When developing a large application, you may end up with many different modules that are difficult to manage. In such a case, you'll benefit from grouping and organizing your modules. That's when packages come into play.
To be considered a package (or subpackage), a directory must contain a file named `__init__.py`. This file usually includes the initialization code for the corresponding package.

1.3.1.3 Import

In Python, the "***import***" statement is used to bring in external modules or libraries into your program. This allows you to use the functions, classes, and variables defined in those modules within your own Python script or program.

There are several ways to use the import statement:

1. Basic Import

```python
import module_name
```

2. Import with alias

```python
import module_name as alias_name
```

3. Import Specific Items:

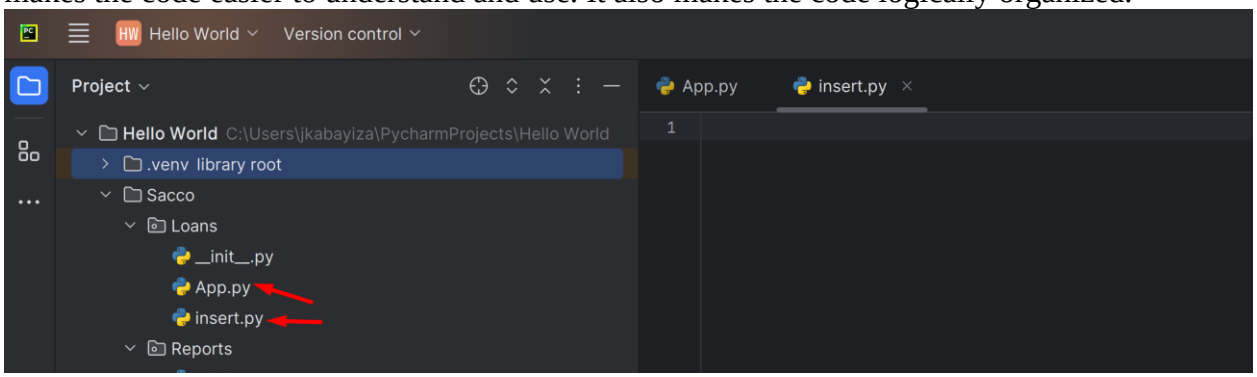
```python
from module_name import item_name
```

4. Import Everything:

```python
from module_name import *
```

1.3.1.4 Module

A Python module is a file containing Python definitions and statements. A module can define functions, classes, and variables. A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use. It also makes the code logically organized.



1.3.1.5 PIP

Pip is a package manager that is used to install and manage software packages and modules found in the Python Package index or any other indexes.

1.3.2 Getting a library

1.3.2.1 Install PIP

Install PIP on Windows Using get-pip.py

The first method uses cURL to download the installation file and additional configuration steps post-installation. Follow the steps below to install PIP using this method.

Step 1: Download PIP get-pip.py

Run the following cURL command in the command prompt:

curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py

```
C:\Users\PC>curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 2570k  100 2570k    0     0  7138k      0 --:--:-- --:--:-- --:--:-- 7180k
```

Wait for the download to complete before proceeding to the next step.

Step 2: Installing PIP on Windows

To install PIP, run the following Python command:

python get-pip.py

```
C:\Users\PC>python get-pip.py
Defaulting to user installation because normal site-packages is not writeable
Collecting pip
  Using cached pip-23.3.1-py3-none-any.whl.metadata (3.5 kB)
Using cached pip-23.3.1-py3-none-any.whl (2.1 MB)
Installing collected packages: pip
  WARNING: The scripts pip.exe, pip3.11.exe and pip3.exe are installed in 'C:\Users\PC\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed pip-23.3.1
```

Note the location from the output warning, which will be required to add PIP to the **Path** environment variable.

Step 3: Configuration

In Windows, the PIP configuration file can be found in several locations. To view the current configuration and list all possible file locations, use the following command:

pip config -v list

```
C:\Users\PC>pip config -v list
For variant 'global', will try loading 'C:\ProgramData\pip\pip.ini'
For variant 'user', will try loading 'C:\Users\PC\pip\pip.ini'
For variant 'user', will try loading 'C:\Users\PC\AppData\Roaming\pip\pip.ini'
For variant 'site', will try loading 'C:\Program Files\WindowsApps\PythonSoftwa
```

1.3.2.2 Download and Install Library

To download a Python library, you typically use a package manager, and pip is the most common one for Python. To download and install a Python library using pip, you can use the following command in your terminal or command prompt:

pip install library_name

bash Copy code

```
pip install library_name
```

Example: Install Django Library

pip install Django

In pycharm

```
(.venv) PS C:\Users\jkabayiza\PycharmProjects\Hello World>
(.venv) PS C:\Users\jkabayiza\PycharmProjects\Hello World> pip install django
Collecting django
  Obtaining dependency information for django from https://files.pythonhosted.org/packages/97/67/6804ff6fc4fa6df188924412601cc418ddc2d0a500963b080
any.whl.metadata
  Downloading Django-5.0.1-py3-none-any.whl.metadata (4.2 kB)
Collecting asgiref<4,>=3.7.0 (from django)
  Obtaining dependency information for asgiref<4,>=3.7.0 from https://files.pythonhosted.org/packages/9b/80/b9051a4a07ad231558fcd8ffc89232711b4e61
```

1.3.2.3 Import library

To import the Django library in Python, you use the import statement. Django is a web framework for building web applications using Python. If you have Django installed, you can import it into your Python script or project like this:

Importing Django

import django

Now you can use Django functionality in your script

print(django.get_version())

```python
# Importing Django
import django

# Now you can use Django functionality in your script
print(django.get_version())
```

In the above example, the django library is imported, and the get_version function is used to print the Django version.

If you are working with specific components of Django, such as models, views, or forms, you would typically import those components as needed. For example:

```python
# Importing Django model
from django.db import models

# Defining a simple model
class MyModel(models.Model):
    name = models.CharField(max_length=50)
    age = models.IntegerField()

# Importing Django views
from django.shortcuts import render

# Example view
def my_view(request):
```

LEARNING UNIT 2–DEVELOP PYTHON CONCEPT

Learning Outcome 2.1: Writing python syntax

The Python syntax defines a set of rules that are used to create Python statements while writing a Python Program. The Python Programming Language Syntax has many similarities to Perl, C, and Java Programming Languages. However, there are some definite differences between the languages.

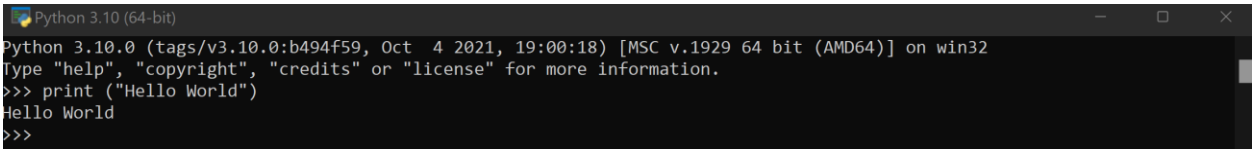
Python extension is .py

2.1.2. Use command line

Let us execute a Python "Hello, World!" Programs in different modes of programming

2.1.2.1. Python modes

1) Interactive Mode Programming



2) **Script Mode Programming**

We can invoke the Python interpreter with a script parameter which begins the execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.
Let us write a simple Python program in a script which is simple text file. Python files have extension .py. Type the following source code in a test.py file:

```
print ("Hello, World!")
```

This produces the following result:

```
Hello, World!
```

Example 2:

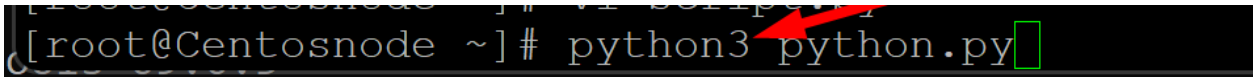
```python
# Simple Python Script

# Get user input for name
name = input("Enter your name: ")

# Get user input for age
age = input("Enter your age: ")

# Convert age to integer
age = int(age)

# Calculate age after ten years
future_age = age + 10

# Print a greeting with the user's name and age after ten years
print(f"Hello, {name}! In ten years, you'll be {future_age} yea
```

2.1.2.2 Compilation and execution

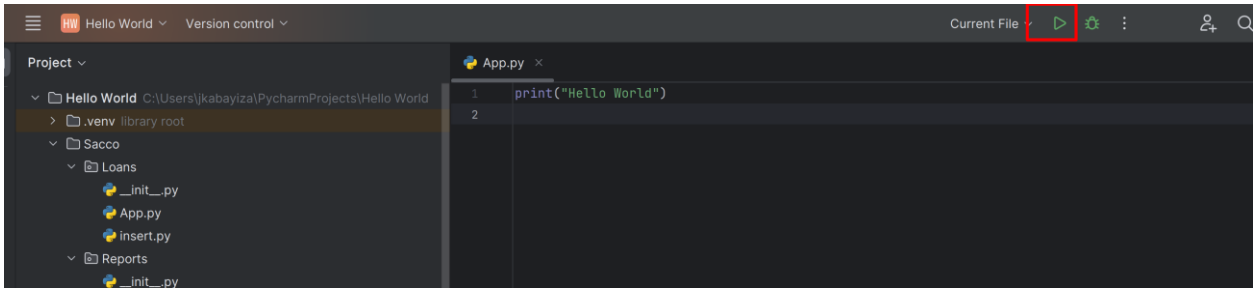
In python compilation and execution are done whenever you run python program

In terminal or command prompt:

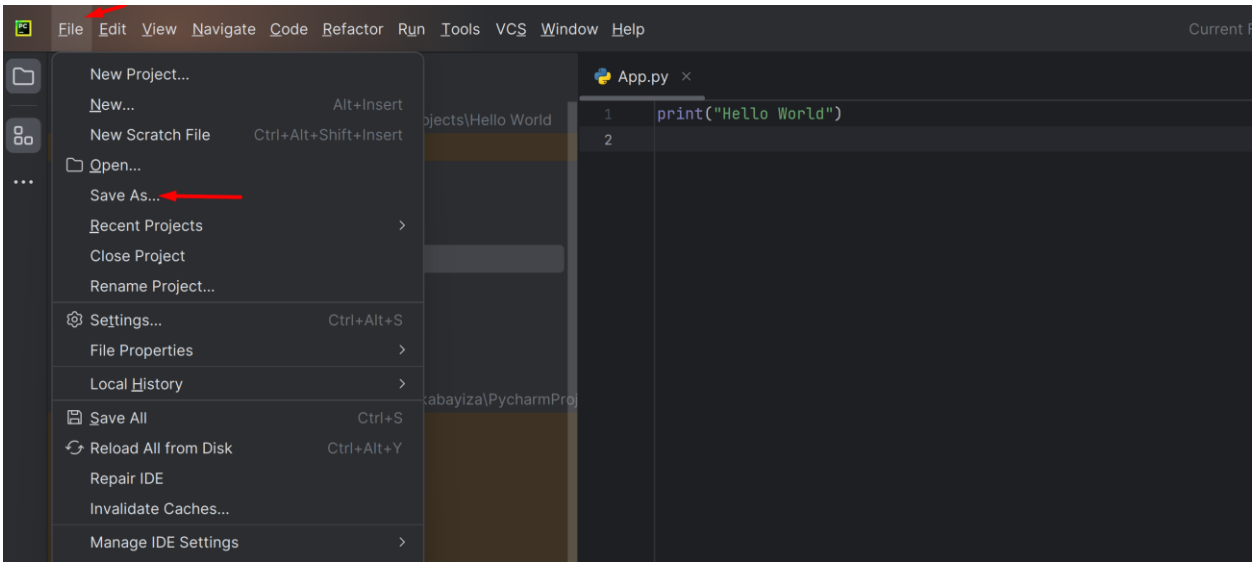
Python3 simple_script.py



In GUI (pycharm)

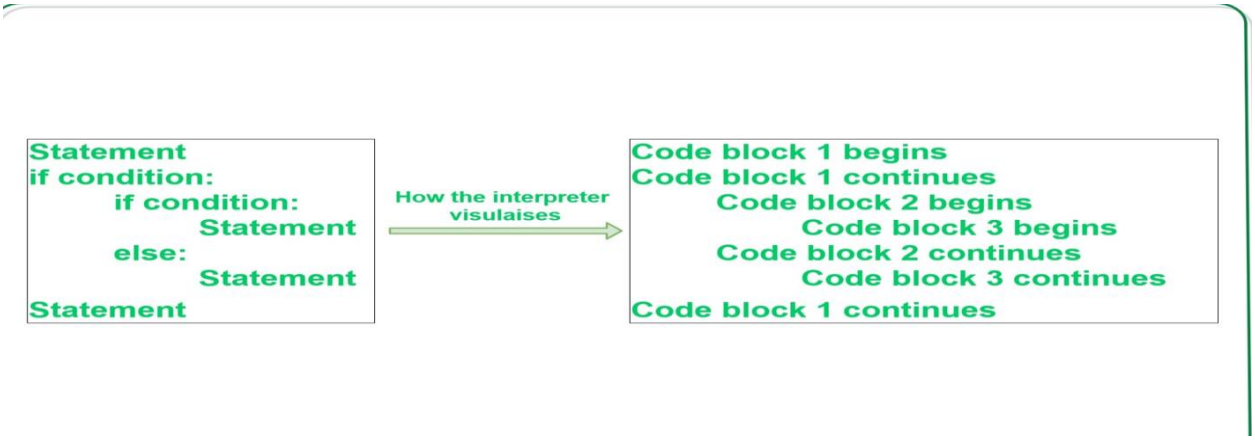


2.1.2.3 Save python project in pycharm



2.1.2.4 Identation

Indentation refers to the spaces at the beginning of a code line. Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important. Python uses indentation to indicate a block of code.



Python programming provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by **line indentation**, which is rigidly enforced.
The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example:

```
if True:
  print ("True")
else:
  print ("False")
```

However, the following block generates an error:

```
if True:
print ("Answer")
print ("True")
else:
print ("Answer")
print ("False")
```

Thus, in Python all the continuous lines indented with same number of spaces would form a block.

2.1.2.5 Python Reserved Words

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

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

2.1.3. Comments in Python

Python comments are programmer-readable explanation or annotations in the Python source code. They are added with the purpose of making the source code easier for humans to understand, and are ignored by Python interpreter. Comments enhance the readability of the code and help the programmers to understand the code very carefully. Just like most modern languages, Python supports single-line (or end-of-line) and multi-line (block) comments. Python comments are very much similar to the comments available in PHP, BASH and Perl Programming languages.

There are 2 types of comments available in Python

- Single line Comments
- Multiline Comments

2.1.3.1. Single Line Comments

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them. Example

Following is an example of a single line comment in Python: #

This is a single line comment in python

print ("Hello, World!")

This produces the following result: Hello, World!

You can type a comment on the same line after a statement or expression.

name = "Madisetti" # This is again comment

2.1.3.2. Multi-Line Comments

Python does not provide a direct way to comment multiple line. You can comment multiple lines as follows:

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

Following triple-quoted string is also ignored by Python interpreter and can be used as a multiline comments:

```
'''
This is a multiline
comment.
'''
```

Learning Outcome 2.2: Perform declaration

2.2.1 Definition of Key terms

2.2.1.1 Variable declaration and Assignment

Python variables are the reserved memory locations used to store values with in a Python Program. This means that when you create a variable you reserve some space in the memory.

Based on the data type of a variable, Python interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to Python variables, you can store integers, decimals or characters in these variables.

2.2.1.2. Variable declaration

Python variables do not need explicit declaration to reserve memory space or you can say to create a variable. A Python variable is created automatically when you assign a value to it. The equal sign (=) is used to assign values to variables.

2.2.2. Assigning values

2.2.2.1. Single value assignment

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable. For example:

```
counter = 100          # Creates an integer variable
miles   = 1000.0       # Creates a floating point variable
name    = "Zara Ali"   # Creates a string variable
```

2.2.2.2. Multiple value assignment

Python allows you to assign a single value to several variables simultaneously which means you can create multiple variables at a time. For example:

```
a = b = c = 100
print (a)
print (b)
print (c)
```

This produces the following result:

```
100
100
100
```

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example −

```
a,b,c = 1,2,"Zara Ali"
print (a)
print (b)
print (c)
```

This produces the following result:

```
1
2
Zara Ali
```

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "Zara Ali" is assigned to the variable c.

In Python, you don't explicitly declare variables with a specific type as you do in some statically typed languages. Instead, you simply assign a value to a variable, and Python dynamically determines the variable's type at runtime. Here's how you assign values to variables in Python:

```python
# Variable assignment
x = 42       # integer
name = "John"  # string
pi = 3.14     # float
is_student = True  # boolean
```

In the example above:

'x 'is assigned an integer value (**42**).

'**name**' is assigned a string value **("John").**

'**pi**' is assigned a floating-point value (**3.14**).

'**is_student'** is assigned a boolean value (**True**).

2.2.3. Python Variable Names

Every Python variable should have a unique name like a, b, c. A variable name can be meaningful like color, age, name etc. There are certain rules which should be taken care while naming a Python variable:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number or any special character like $, (, * % etc.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Python variable names are case-sensitive which means Name and NAME are two different variables in Python.
- Python reserved keywords cannot be used naming the variable.

Example
Following are valid Python variable names:

```
counter = 100
_count  = 100
name1 = "Zara"
name2 = "Nuha"
Age  = 20
zara_salary = 100000
print (counter)
```

```
print (_count)
print (name1)
print (name2)
print (Age)
print (zara_salary)
```

This will produce the following result:

```
100
100
Zara
Nuha
20
100000
```

Example
Following are invalid Python variable names:

```
1counter = 100
$_count  = 100
zara-salary = 100000


print (1counter)
print ($count)
print (zara-salary)
```

This will produce the following result:

```
File "main.py", line 3
  1counter = 100
        ^
SyntaxError: invalid syntax
```

2.2.4 Types of variables

2.2.4.1. Python Local Variable

Python Local Variables are defined inside a function. We can not access variable outside the function. Following is an example to show the usage of local variables:

```
def sum(x,y):
```

```
    sum = x + y
    return sum
print(sum(5, 10))
```

This will produce the following result:

```
15
```

Python Global Variable

Any variable created outside a function can be accessed within any function and so they have global scope. Following is an example of global variables:

```
x = 5
y = 10
def sum():
    sum = x + y
    return sum
print(sum())
```

This will produce the following result:

```
15
```

Learning Outcome 2.3: Differentiate data type

Python Data Types are used to define the type of a variable. It defines what type of data we are going to store in a variable. The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.

2.3.1. Built-in data type

Python has various built-in data types:

- Numeric - int, float, complex
- String - str
- Sequence - list, tuple, range
- Binary - bytes, bytearray, memoryview
- Mapping - dict
- Boolean - bool
- Set - set, frozenset

2.3.1.1. Python Numeric Data Type

Python numeric data types store numeric values. Number objects are created when you assign a value to them. For example:

```
var1 = 1
var2 = 10
var3 = 10.023
```

Python supports four different numerical types:

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)

- complex (complex numbers)

Examples
Here are some examples of numbers:

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

- Python allows you to use a lowercase l with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.
- A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x and y are the real numbers and j is the imaginary unit.

Example
Following is an example to show the usage of Integer, Float and Complex numbers:

```
# integer variable.
a=100
print("The type of variable having value", a, " is ", type(a))

# float variable.
b=20.345
print("The type of variable having value", b, " is ", type(b))

# complex variable.
c=10+3j
print("The type of variable having value", c, " is ", type(c))
```

2.3.1.2. Python String Data Type

A String is a data structure in Python that represents a sequence of characters. It is an immutable data type, meaning that once you have created a string, you cannot change it. Strings are used widely in many different applications, such as storing and manipulating text data, representing names, addresses, and other types of data that can be represented as text. Python Strings are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes.

Creating a String in Python

Strings in Python can be created using single quotes or double quotes or even triple quotes. Let us see how we can define a string in Python.

Example:

In this example, we will demonstrate different ways to create a Python String. We will create a string using single quotes (‘ ‘), double quotes (” “), and triple double quotes (“”” “”””). The triple quotes can be used to declare multiline strings in Python.

Python3

```
# Python Program for
# Creation of String

# Creating a String
# with single Quotes
String1 = 'Welcome to the Geeks World'
print("String with the use of Single Quotes: ")
print(String1)

# Creating a String
# with double Quotes
String1 = "I'm a Geek"
print("\nString with the use of Double Quotes: ")
print(String1)

# Creating a String
# with triple Quotes
String1 = '''I'm a Geek and I live in a world of "Geeks"'''
print("\nString with the use of Triple Quotes: ")
print(String1)

# Creating String with triple
# Quotes allows multiple lines
String1 = '''Geeks
        For
        Life'''
print("\nCreating a multiline String: ")
print(String1)
```

Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator in

Python. For example:
str = 'Hello World!'

```
print (str)          # Prints complete string
print (str[0])       # Prints first character of the string
print (str[2:5])     # Prints characters starting from 3rd to 5th
print (str[2:])      # Prints string starting from 3rd character
print (str * 2)      # Prints string two times
print (str [:5])    # prints Hello
print (str + "TEST") # Prints concatenated string
print( str [-1])      #prints the last character of the string
print (str [1:-1]     # prints the first character of the string up …and remove the last character of the string
```

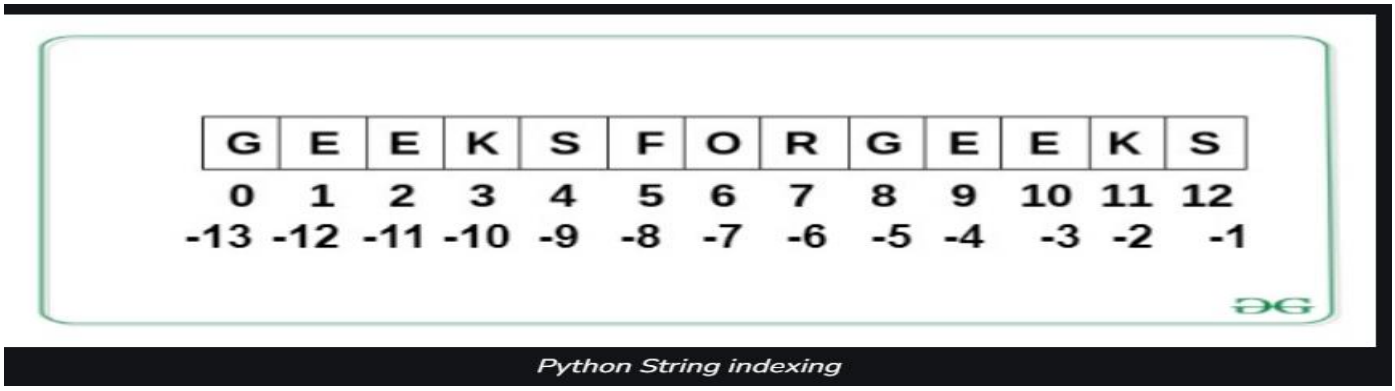
This will produce the following result:

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
d
ello Worl
```

Accessing characters in Python String

In Python, individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character, and so on.

While accessing an index out of the range will cause an IndexError. Only Integers are allowed to be passed as an index, float or other types that will cause a TypeError.



Python String indexing

Example:

In this example, we will define a string in Python and access its characters using positive and negative indexing. The 0th element will be the first character of the string whereas the -1th element is the last character of the string.

```
# Python Program to Access
# characters of String

String1 = "GeeksForGeeks"
print("Initial String: ")
print(String1)

# Printing First character
print("\nFirst character of String is: ")
print(String1[0])

# Printing Last character
print("\nLast character of String is: ")
print(String1[-1])
```

Output:

```
GeeksForGeeks
First character of String is:
G
Last cha racter of String is:
s
```

String Slicing

In Python, the String Slicing method is used to access a range of characters in the String. Slicing in a String is done by using a Slicing operator, i.e., a colon (:). One thing to keep in mind while using this method is that the string returned after slicing includes the character at the start index but not the character at the last index.

Example:

In this example, we will use the string-slicing method to extract a substring of the original string. The [3:12] indicates that the string slicing will start from the 3rd index of the string to the 12th index, (12th character not including). We can also use negative indexing in string slicing.

```
# Python Program to
# demonstrate String slicing

# Creating a String
String1 = "GeeksForGeeks"
print("Initial String: ")
print(String1)

# Printing 3rd to 12th character
print("\nSlicing characters from 3-12: ")
print(String1[3:12])

# Printing characters between
# 3rd and 2nd last character
print("\nSlicing characters between " +
        "3rd and 2nd last character: ")
print(String1[3:-2])
```

Output:

GeeksForGeeks
Slicing characters from 3-12:
ksForGeek
Slicing characters between 3rd and 2nd last character:
ksForGee

Reversing a Python String

By accessing characters from a string, we can also reverse strings in Python. We can Reverse a string by using String slicing method.
Example1:
In this example, we will reverse a string by accessing the index. We did not specify the first two parts of the slice indicating that we are considering the whole string, from the start index to the last index.

```
#Program to reverse a string
gfg = "geeksforgeeks"
print(gfg[::-1])
```

Output:
Skeegrofskeeg

Example2:
We can also reverse a string by using built-in join and reversed functions, and passing the string as the parameter to the reversed() function.

```
# Program to reverse a string

gfg = "geeksforgeeks"

# Reverse the string using reversed and join function
gfg = "".join(reversed(gfg))
print(gfg)
```

Output:
skeegrofskeeg

Deleting/Updating from a String

In Python, the Updation or deletion of characters from a String is not allowed. This will cause an error because item assignment or item deletion from a String is not supported. Although deletion of the entire String is possible with the use of a built-in del keyword. This is because Strings are immutable, hence elements of a String cannot be changed once assigned. Only new strings can be reassigned to the same name.
Updating a character

A character of a string can be updated in Python by first converting the string into a Python List and then updating the element in the list. As lists are mutable in nature, we can update the character and then convert the list back into the String.
Another method is using the string slicing method. Slice the string before the character you want to update,

then add the new character and finally add the other part of the string again by string slicing.

Example:
In this example, we are using both the list and the string slicing method to update a character. We converted the String1 to a list, changes its value at a particular element, and then converted it back to a string using the Python string join() method.
In the string-slicing method, we sliced the string up to the character we want to update, concatenated the new character, and finally concatenate the remaining part of the string.

```
# Python Program to Update
# character of a String

String1 = "Hello, I'm a Geek"
print("Initial String: ")
print(String1)

# Updating a character of the String
## As python strings are immutable, they don't support item updation directly
### there are following two ways
#1
list1 = list(String1)
list1[2] = 'p'
String2 = ''.join(list1)
print("\nUpdating character at 2nd Index: ")
print(String2)

#2
String3 = String1[0:2] + 'p' + String1[3:]
print(String3)
```

Output:

```
Initial String:

Hello, I'm a Geek
Updating character at 2nd Index:
Heplo, I'm a Geek
Heplo, I'm a Geek
```

Updating Entire String

```
# Python Program to Update
# entire String

String1 = "Hello, I'm a Geek"
print("Initial String: ")
print(String1)

# Updating a String
String1 = "Welcome to the Geek World"
print("\nUpdated String: ")
print(String1)
```

OutPut:

Initial String:
Hello, I'm a Geek
Updated String:
Welcome to the Geek World

2.3.1.3. Python List Data Type

Python Lists are the most versatile compound data types. A Python list contains items separated by commas and enclosed within square brackets ([]). To some extent, Python lists are similar to arrays in C. One difference between them is that all the items belonging to a Python list can be of different data type whereas C array can store elements related to a particular data type.
List Represents an ordered collection of elements.

Example1: numbers = [1, 2, 3]

Example2:

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print (list)           # Prints complete list
print (list[0])        # Prints first element of the list
print (list[1:3])      # Prints elements starting from 2nd till 3rd
print (list[2:])       # Prints elements starting from 3rd element
print (tinylist * 2)   # Prints list two times
print (list + tinylist) # Prints concatenated lists
```

This produces the following result:

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

2.3.1.4. Python Tuple Data Type

Python tuple is another sequence data type that is similar to a list. A Python tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as read-only lists.

For example:

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
tinytuple = (123, 'john')
print (tuple)             # Prints the complete tuple
print (tuple[0])          # Prints first element of the tuple
print (tuple[1:3])        # Prints elements of the tuple starting from 2nd till 3rd
print (tuple[2:])         # Prints elements of the tuple starting from 3rd element
print (tinytuple * 2)     # Prints the contents of the tuple twice
print (tuple + tinytuple)  # Prints concatenated tuples
```

This produces the following result :

```
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
```

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists:

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
list = [ 'abcd', 786 , 2.23, 'john', 70.2  ]
tuple[2] = 1000    # Invalid syntax with tuple
list[2] = 1000     # Valid syntax with list
```

2.3.1.5. Python Dictionary

Python dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]). For example:

```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}


print (dict['one'])       # Prints value for 'one' key
print (dict[2])           # Prints value for 2 key
print (tinydict)          # Prints complete dictionary
print (tinydict.keys())   # Prints all the keys
print (tinydict.values()) # Prints all the values
```

This produces the following result:

```
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
['sales', 6734, 'john']
```

Python dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unordered.

2.3.1.7. Python Boolean Data Types

Python boolean type is one of built-in data types which represents one of the two values either True or False. Python bool() function allows you to evaluate the value of any expression and returns either True or False based on the expression.

Examples

Following is a program which prints the value of boolean variables a and b −

```
a = True
# display the value of a
print(a)

# display the data type of a
print(type(a))
```

This produces the following result:

```
true
<class 'bool'>
```

Following is another program which evaluates the expressions and prints the return values:

```
# Returns false as a is not equal to b
a = 2
b = 4
print(bool(a==b))

# Following also prints the same
print(a==b)

# Returns False as a is None
a = None
print(bool(a))

# Returns false as a is an empty sequence
a = ()
print(bool(a))

# Returns false as a is 0
a = 0.0
print(bool(a))

# Returns false as a is 10
a = 10
print(bool(a))
```

This produces the following result:

```
False
```

```
False
False
False
False
True
```

In Python, variables can dynamically change their type based on the assigned values. Here are some common variable types in Python:

Integer (int):

Represents whole numbers.

Example: x = 42

Floating-Point (float):

Represents numbers with a decimal point.

Example: pi = 3.14

String (str):

Represents sequences of characters.

Example: name = "John"

Boolean (bool):

Represents truth values True or False.

Example: is_student = True

List (list):

Represents an ordered collection of elements.

Example: numbers = [1, 2, 3]

Tuple (tuple):

Similar to a list but immutable.

Example: coordinates = (3, 4)

Dictionary (dict):

Represents a collection of key-value pairs.

Example: person = {'name': 'Alice', 'age': 25}

Set (set):

Represents an unordered collection of unique elements.

Example: unique_numbers = {1, 2, 3}

None Type (NoneType):

Represents the absence of a value.

Example: result = None

To check the type of a variable, you can use the **type()** function:

```python
x = 42
print(type(x))  # <class 'int'>

pi = 3.14
print(type(pi))  # <class 'float'>

name = "John"
print(type(name))  # <class 'str'>

is_student = True
print(type(is_student))  # <class 'bool'>
```

Learning Outcome 2.3: Differentiate data type

In Python, data types represent the kind of values that variables can hold. Python is a dynamically typed language, meaning that you don't need to explicitly declare the data type of a variable. The interpreter infers the data type dynamically based on the value assigned to the variable. Here are some common data types in Python:

Numeric Types:

int: Integer type represents whole numbers (**e.g., 42**).

float: Floating-point type represents numbers with a decimal point (e.g., 3.14).

Text Type:

str: String type represents sequences of characters. Strings can be enclosed in single (') or double (") quotes (**e.g., 'Hello', "World").**

Boolean Type:

bool: Boolean type represents truth values. It can be either **True** or **False**.

Sequence Types:

list: List type represents ordered collections of elements. Lists are mutable and can contain elements of different data types (e.g., [**1, 'apple', True**]).

tuple: Tuple type is similar to a list but immutable (e.g., (**1, 'apple', True**)).

Mapping Type:

dict: Dictionary type represents a collection of key-value pairs. Keys must be unique, and they are associated with values (e.g., **{'name': 'John', 'age': 25}).**

Set Types:

set: Set type represents an unordered collection of unique elements. Sets do not allow duplicate elements **(e.g., {1, 2, 3}).**

None Type:

NoneType: The None type represents the absence of a value or a null value.

2.3.2. Python Data Type Conversion

Sometimes, you may need to perform conversions between the built-in data types. To convert data between different Python data types, you simply use the type name as a function.

2.3.2.1. Conversion to int

Following is an example to convert number, float and string into integer data type:

```
a = int(1)    # a will be 1
b = int(2.2)   # b will be 2
c = int("3")   # c will be 3

print (a)
print (b)
print (c)
```

This produces the following result:

```
1
2
3
```

2.3.2.2. Conversion to float

Following is an example to convert number, float and string into float data type:

```
a = float(1)    # a will be 1.0
b = float(2.2)   # b will be 2.2
c = float("3.3") # c will be 3.3

print (a)
print (b)
print (c)
```

This produces the following result:

```
1.0
2.2
```

```
3.3
```

2.3.2.3. Conversion to string

Following is an example to convert number, float and string into string data type:

```
a = str(1)     # a will be "1"
b = str(2.2)   # b will be "2.2"
c = str("3.3") # c will be "3.3"

print (a)
print (b)
print (c)
```

This produce the following result:

```
1
2.2
3.3
```

2.3.2.4. Data Type Conversion Functions

There are several built-in functions to perform conversion from one data type to another. These functions return a new object representing the converted value.

S/N	Function & Description
1	int(x [,base]) Converts x to an integer. base specifies the base if x is a string.
2	long(x [,base]) Converts x to a long integer. base specifies the base if x is a string.
3	float(x) Converts x to a floating-point number.
4	complex(real [,imag]) Creates a complex number.
5	str(x) Converts object x to a string representation.
6	repr(x) Converts object x to an expression string.
7	eval(str) Evaluates a string and returns an object.
8	tuple(s) Converts s to a tuple.
9	list(s) Converts s to a list.
10	set(s) Converts s to a set.
11	dict(d) Creates a dictionary. d must be a sequence of (key,value) tuples.

12	frozenset(s) Converts s to a frozen set.
13	chr(x) Converts an integer to a character.
14	unichr(x) Converts an integer to a Unicode character.
15	ord(x) Converts a single character to its integer value.
16	hex(x) Converts an integer to a hexadecimal string.
17	oct(x) Converts an integer to an octal string.

LEARNING UNIT 3 – DEVELOP PYTHON SCRIPT

Learning Outcome 3.1: Use of operators

3.1.1. Arithmetic operators

Arithmetic operators in Python are used to perform mathematical operations on numeric values. Here are the common arithmetic operators:

Operator	Name	Example
+	Addition	10 + 20 = 30
-	Subtraction	20 – 10 = 10
*	Multiplication	10 * 20 = 200
/	Division	20 / 10 = 2
%	Modulus	22 % 10 = 2
**	Exponent	4**2 = 16
//	Floor Division	9//2 = 4

Example1:

```
a = 10
b = 3
addition_result = a + b       # 13
subtraction_result = a - b    # 7
multiplication_result = a * b # 30
division_result = a / b       # 3.3333333333333335
floor_division_result = a // b# 3
modulus_result = a % b        # 1
```

exponentiation_result = a ** b# 1000

Example 2:

```
a = 21
b = 10

# Addition
print ("a + b : ", a + b)

# Subtraction
print ("a - b : ", a - b)

# Multiplication
print ("a * b : ", a * b)

# Division
print ("a / b : ", a / b)

# Modulus
print ("a % b : ", a % b)

# Exponent
print ("a ** b : ", a ** b)

# Floor Division
print ("a // b : ", a // b)
```

This produces the following result:

```
a + b :  31
a - b :  11
a * b :  210
a / b :  2.1
a % b :  1
a ** b :  16679880978201
a // b :  2
```

3.1.2. Comparison operators

Python comparison operators compare the values on either sides of them and decide the relation among them. They are also called relational operators. These operators are equal, not equal, greater than, less than, greater than or equal to and less than or equal to.

Operator	Name	Example
==	Equal	4 == 5 is not true.
!=	Not Equal	4 != 5 is true.
>	Greater Than	4 > 5 is not true.
<	Less Than	4 < 5 is true.
>=	Greater than or Equal to	4 >= 5 is not true.
<=	Less than or Equal to	4 <= 5 is true.

Example

Following is an example which shows all the above comparison operations:

```
a = 4
b = 5

# Equal
print ("a == b : ", a == b)

# Not Equal
print ("a != b : ", a != b)

# Greater Than
print ("a > b : ", a > b)

# Less Than
print ("a < b : ", a < b)

# Greater Than or Equal to
print ("a >= b : ", a >= b)

# Less Than or Equal to
print ("a <= b : ", a <= b)
```

This produces the following result:

```
a == b :  False
a != b :  True
a > b :  False
a < b :  True
a >= b :  False
a <= b :  True
```

3.1.3. Python Assignment Operators

Python assignment operators are used to assign values to variables. These operators include simple assignment operator, addition assign, subtraction assign, multiplication assign, division and assign operators etc.

Operator	Name	Example
=	Assignment Operator	a = 10
+=	Addition Assignment	a += 5 (Same as a = a + 5)
-=	Subtraction Assignment	a -= 5 (Same as a = a - 5)
*=	Multiplication Assignment	a *= 5 (Same as a = a * 5)
/=	Division Assignment	a /= 5 (Same as a = a / 5)
%=	Remainder Assignment	a %= 5 (Same as a = a % 5)
**=	Exponent Assignment	a **= 2 (Same as a = a ** 2)
//=	Floor Division Assignment	a //= 3 (Same as a = a // 3)

Example

Following is an example which shows all the above assignment operations:

```
# Assignment Operator
a = 10

# Addition Assignment
a += 5
print ("a += 5 : ", a)

# Subtraction Assignment
a -= 5
print ("a -= 5 : ", a)

# Multiplication Assignment
```

```
a *= 5
print ("a *= 5 : ", a)

# Division Assignment
a /= 5
print ("a /= 5 : ",a)

# Remainder Assignment
a %= 3
print ("a %= 3 : ", a)

# Exponent Assignment
a **= 2
print ("a **= 2 : ", a)

# Floor Division Assignment
a //= 3
print ("a //= 3 : ", a)
```

This produces the following result:

```
a += 5 :  105
a -= 5 :  100
a *= 5 :  500
a /= 5 :  100.0
a %= 3 :  1.0
a **= 2 :  1.0
a //= 3 :  0.0
```

3.1.4. Python Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if a = 60; and b = 13; Now in the binary format their values will be 0011 1100 and 0000 1101 respectively. Following table lists out the bitwise operators supported by Python language with an example each in those, we use the above two variables (a and b) as operands:

a = 0011 1100
b = 0000 1101
---
a&b = 12 (0000 1100)
a|b = 61 (0011 1101)
a^b = 49 (0011 0001)
~a = -61 (1100 0011)

a << 2 = 240 (1111 0000)
a>>2 = 15 (0000 1111)
There are following Bitwise operators supported by Python language

Operator	Name	Example
&	Binary AND	Sets each bit to 1 if both bits are 1
|	Binary OR	Sets each bit to 1 if one of two bits is 1
^	Binary XOR	Sets each bit to 1 if only one of two bits is 1
~	Binary Ones Complement	Inverts all the bits
<<	Binary Left Shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Binary Right Shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Example
Following is an example which shows all the above bitwise operations:

```
a = 60            # 60 = 0011 1100
b = 13            # 13 = 0000 1101

# Binary AND
c = a & b       # 12 = 0000 1100
print ("a & b : ", c)

# Binary OR
c = a | b       # 61 = 0011 1101
print ("a | b : ", c)

# Binary XOR
c = a ^ b       # 49 = 0011 0001
print ("a ^ b : ", c)

# Binary Ones Complement
c = ~a;          # -61 = 1100 0011
print ("~a : ", c)

# Binary Left Shift
c = a << 2;      # 240 = 1111 0000
print ("a << 2 : ", c)
```

```
# Binary Right Shift
c = a >> 2;       # 15 = 0000 1111
print ("a >> 2 : ", c)
```

This produces the following result:

```
a & b :  12
a | b :  61
a ^ b :  49
~a :  -61
a >> 2 :  240
a >> 2 :  15
```

3.1.5. Python Logical Operators

There are following logical operators supported by Python language. Assume variable a holds 10 and variable b holds 20 then.

Operator	Description	Example
and Logical AND	If both the operands are true, then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero, then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

3.1.6. Python Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below:

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not find a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

3.1.7. Python Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators explained below:

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

3.1.8. Python Operators Precedence

The following table lists all operators from highest precedence to lowest.

S/N	Operator & Description
1	** Exponentiation (raise to the power)
2	~ + - Complement, unary plus and minus (method names for the last two are +@ and -@)
3	* / % // Multiply, divide, modulo and floor division
4	+ - Addition and subtraction
5	>> << Right and left bitwise shift
6	& Bitwise 'AND'
7	^ | Bitwise exclusive `OR' and regular `OR'
8	<= < > >= Comparison operators
9	<> == != Equality operators
10	= %= /= //= -= += *= **= Assignment operators
11	is is not Identity operators
12	in not in Membership operators
13	not or and Logical operators

Learning Outcome 3.2: Determine collection of data.

3.2.1. List

A list in Python is used to store the sequence of various types of data. A list can be defined as a collection of values or items of different types. Python lists are mutable type which implies that we may modify its element after it has been formed. The items in the list are separated with the comma (,) and enclosed with the square brackets [].

Although Python has six data types that may hold sequences, the list is the most popular and dependable form. The collection of data is stored in a list, a sequence data type. Similar sequence data formats are Tuples and String.

Python lists are identical to dynamically scaled arrays that are specified in other languages, such as Java's ArrayList and C++'s vector. A list is a group of items that are denoted by the symbol [] and subdivided by commas.

3.2.1.1. List Declaration

Code

```
# a simple list
list1 = [1, 2, "Python", "Program", 15.9]
```

```
list2 = ["Amy", "Ryan", "Henry", "Emma"]

# printing the list
print(list1)
print(list2)

# printing the type of list
print(type(list1))
print(type(list2))
```

Output:

```
[1, 2, 'Python', 'Program', 15.9]
['Amy', 'Ryan', 'Henry', 'Emma']
< class ' list ' >
< class ' list ' >
```

3.2.1.2. Characteristics of Lists

The list has the following characteristics:

- The lists are ordered.
- The element of the list can access by index.
- The lists are the mutable type.
- The lists are mutable types.
- A list can store the number of various elements.

3.2.1.3. Ordered List Checking

Code

```
# example
a = [ 1, 2, "Ram", 3.50, "Rahul", 5, 6 ]
b = [ 1, 2, 5, "Ram", 3.50, "Rahul", 6 ]
a == b
```

Output:

```
False
```

The identical elements were included in both lists, but the second list modified the index position of the fifth element, which is against the lists' intended order. When the two lists are compared, false is returned.

Code

```
# example
a = [ 1, 2, "Ram", 3.50, "Rahul", 5, 6]
b = [ 1, 2, "Ram", 3.50, "Rahul", 5, 6]
a == b
```

Output:

```
True
```

Lists permanently preserve the element's order. It is the arranged gathering of things because of this. Let's have a look at the list example in detail.

Code

```
# list example in detail
emp = [ "John", 102, "USA"]
Dep1 = [ "CS",10]
Dep2 = [ "IT",11]
HOD_CS = [ 10,"Mr. Holding"]
HOD_IT = [11, "Mr. Bewon"]
print("printing employee data ...")
print(" Name : %s, ID: %d, Country: %s" %(emp[0], emp[1], emp[2]))
print("printing departments ...")
print("Department 1:\nName: %s, ID: %d\n Department 2:\n Name: %s, ID: %s"%( Dep1[0], Dep2[1], Dep2[0], Dep2[1]))
print("HOD Details ....")
print("CS HOD Name: %s, Id: %d" %(HOD_CS[1], HOD_CS[0]))
print("IT HOD Name: %s, Id: %d" %(HOD_IT[1], HOD_IT[0]))
print(type(emp), type(Dep1), type(Dep2), type(HOD_CS), type(HOD_IT))
```

Output:

```
printing employee data...
Name : John, ID: 102, Country: USA
printing departments...
Department 1:
Name: CS, ID: 11
Department 2:
Name: IT, ID: 11
HOD Details ....
CS HOD Name: Mr. Holding, Id: 10
IT HOD Name: Mr. Bewon, Id: 11
<class ' list '> <class ' list '> <class ' list '> <class ' list '> <class ' list '>
```

In the above example, we have created the lists which consist of the employee and department details and printed the corresponding details. Observe the above code to understand the concept of the list better.

3.2.1.4. List Indexing and Splitting

The indexing is processed in the same way as it happens with the strings. The elements of the list can be accessed by using the slice operator [].

The index starts from 0 and goes to length - 1. The first element of the list is stored at the 0th index, the second element of the list is stored at the 1st index, and so on.

List = [0, 1, 2, 3, 4, 5]

0	1	2	3	4	5

List[0] = 0

List[1] = 1

List[2] = 2

List[3] = 3

List[4] = 4

List[5] = 5

List[0:] = [0,1,2,3,4,5]

List[:] = [0,1,2,3,4,5]

List[2:4] = [2, 3]

List[1:3] = [1, 2]

List[:4] = [0, 1, 2, 3]

We can get the sub-list of the list using the following syntax.

```
list_varible(start:stop:step)
```

- The **start** denotes the starting index position of the list.
- The **stop** denotes the last index position of the list.
- The **step** is used to skip the nth element within a **start:stop**

The initial index is represented by the start parameter, the ending index is determined by the step, and also the number of elements which are "stepped" through is the value of the end parameter. In the absence of a specific value for step, the default value equals 1. Inside the resultant SubList, the item also with index start would be present, but the one with the index finish will not. A list's initial element seems to have the index of 0.

Consider the following example:

Code

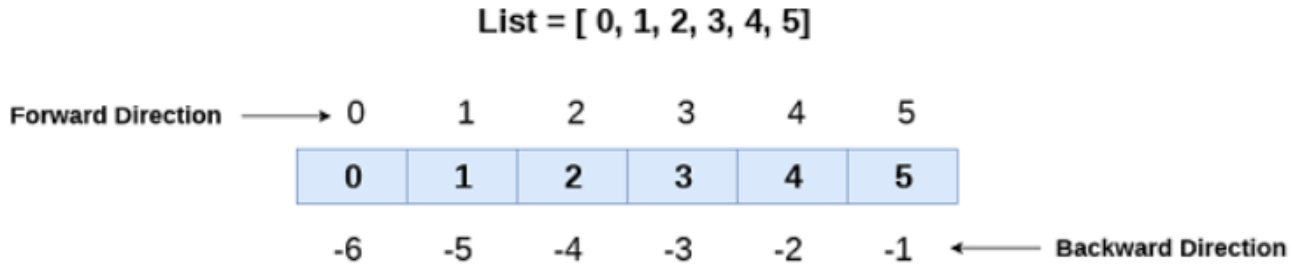
```
list = [1,2,3,4,5,6,7]
print(list[0])
print(list[1])
print(list[2])
print(list[3])
# Slicing the elements
print(list[0:6])
# By default, the index value is 0 so its starts from the 0th element and go for index -1.
print(list[:])
print(list[2:5])
```

```
print(list[1:6:2])
```

Output:

```
1
2
3
4
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6, 7]
[3, 4, 5]
[2, 4, 6]
```

In contrast to other languages, Python gives you the option to employ negative indexing as well. From the right, the negative indices are counted. The final element on the right-hand side of the list is represented by the index -1, followed by the next member on the left at the index -2, and so on until the last element on the left is reached.



Let's have a look at the following example where we will use negative indexing to access the elements of the list.

Code

```
# negative indexing example
list = [1,2,3,4,5]
print(list[-1])
print(list[-3:])
print(list[:-1])
print(list[-3:-1])
```

Output:

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

As we discussed above, we can get an element by using negative indexing. In the above code, the first print statement returned the rightmost element of the list. The second print statement returned the sub-list, and so on.

3.2.1.5. Updating List Values

Lists are the most versatile data structures in Python since they are mutable, and their values can be updated

by using the slice and assignment operator. Python also provides append() and insert() methods, which can be used to add values to the list.
Consider the following example to update the values inside the list.
Code

```
# updating list values
list = [1, 2, 3, 4, 5, 6]
print(list)
# It will assign value to the value to the second index
list[2] = 10
print(list)
# Adding multiple-element
list[1:3] = [89, 78]
print(list)
# It will add value at the end of the list
list[-1] = 25
print(list)
```

Output:

```
[1, 2, 3, 4, 5, 6]
[1, 2, 10, 4, 5, 6]
[1, 89, 78, 4, 5, 6]
[1, 89, 78, 4, 5, 25]
```

The list elements can also be deleted by using the **del** keyword. Python also provides us the **remove()** method if we do not know which element is to be deleted from the list.
Consider the following example to delete the list elements.
Code

```
list = [1, 2, 3, 4, 5, 6]
print(list)
# It will assign value to the value to second index
list[2] = 10
print(list)
# Adding multiple element
list[1:3] = [89, 78]
print(list)
# It will add value at the end of the list
list[-1] = 25
print(list)
```

Output:

```
[1, 2, 3, 4, 5, 6]
[1, 2, 10, 4, 5, 6]
[1, 89, 78, 4, 5, 6]
[1, 89, 78, 4, 5, 25]
```

3.2.1.6. Python List Operations

The concatenation (+) and repetition (*) operators work in the same way as they were working with the strings. The different operations of list are:

- Repetition
- Concatenation
- Length
- Iteration
- Membership

Let's see how the list responds to various operators.

1. Repetition

The repetition operator enables the list elements to be repeated multiple times.

Code

```
# repetition of list
# declaring the list
list1 = [12, 14, 16, 18, 20]
# repetition operator *
l = list1 * 2
print(l)
```

Output:

```
[12, 14, 16, 18, 20, 12, 14, 16, 18, 20]
```

2. Concatenation

It concatenates the list mentioned on either side of the operator.

Code

```
# concatenation of two lists
# declaring the lists
list1 = [12, 14, 16, 18, 20]
list2 = [9, 10, 32, 54, 86]
# concatenation operator +
l = list1 + list2
print(l)
```

Output:

```
[12, 14, 16, 18, 20, 9, 10, 32, 54, 86]
```

3. Length

It is used to get the length of the list

Code

```
# size of the list
# declaring the list
list1 = [12, 14, 16, 18, 20, 23, 27, 39, 40]
# finding length of the list
len(list1)
```

Output:

```
9
```

4. Iteration
The for loop is used to iterate over the list elements.
Code

```
# iteration of the list
# declaring the list
list1 = [12, 14, 16, 39, 40]
# iterating
for i in list1:
  print(i)
```

Output:

```
12
14
16
39
40
```

5. Membership
It returns true if a particular item exists in a particular list otherwise false.
Code

```
# membership of the list
# declaring the list
list1 = [100, 200, 300, 400, 500]
# true will be printed if value exists
# and false if not
_
print(600 in list1)
print(700 in list1)
print(1040 in list1)
_
print(300 in list1)
```

```
print(100 in list1)
print(500 in list1)
```

Output:

```
False
False
False
True
True
True
```

3.2.1.7. Iterating a List

A list can be iterated by using a for - in loop. A simple list containing four strings, which can be iterated as follows.

Code

```
# iterating a list
list = ["John", "David", "James", "Jonathan"]
for i in list:
    # The i variable will iterate over the elements of the List and contains each element in each iteration.
    print(i)
```

Output:

```
John
David
James
Jonathan
```

3.2.1.8. Adding Elements to the List

Method 1: Using append() method

Python provides append() function which is used to add an element to the list. However, the append() function can only add value to the end of the list.

```
# Python program to demonstrate
# Addition of elements in a List

# Creating a List
List = []
print("Initial blank List: ")
print(List)

# Addition of Elements
# in the List
```

```
List.append(1)
List.append(2)
List.append(4)
print("\nList after Addition of Three elements: ")
print(List)

# Adding elements to the List
# using Iterator
for i in range(1, 4):
        List.append(i)
print("\nList after Addition of elements from 1-3: ")
print(List)

# Adding Tuples to the List
List.append((5, 6))
print("\nList after Addition of a Tuple: ")
print(List)

# Addition of List to a List
List2 = ['For', 'Geeks']
List.append(List2)
print("\nList after Addition of a List: ")
print(List)
```

Output:

```
InInitial blank List:
[]

List after Addition of Three elements:
[1, 2, 4]

List after Addition of elements from 1-3:
[1, 2, 4, 1, 2, 3]

List after Addition of a Tuple:
[1, 2, 4, 1, 2, 3, (5, 6)]

List after Addition of a List:
[1, 2, 4, 1, 2, 3, (5, 6), ['For', 'Geeks']]
```

Method 2: Using insert() method

append() method only works for the addition of elements at the end of the List, for the addition of elements at the desired position, insert() method is used. Unlike append() which takes only one argument, the insert() method requires two arguments(position, value).

Example:

```
# Python program to demonstrate
# Addition of elements in a List
```

```
# Creating a List
List = [1,2,3,4]
print("Initial List: ")
print(List)

# Addition of Element at
# specific Position
# (using Insert Method)
List.insert(3, 12)
List.insert(0, 'Geeks')
print("\nList after performing Insert Operation: ")
print(List)
```

Output
Initial List:
[1, 2, 3, 4]

List after performing Insert Operation:
['Geeks', 1, 2, 3, 12, 4]

Method 3: Using extend() method

Other than append() and insert() methods, there's one more method for the Addition of elements, **extend()**, this method is used to add multiple elements at the same time at the end of the list.
Example:

```
# Python program to demonstrate
# Addition of elements in a List

# Creating a List
List = [1, 2, 3, 4]
print("Initial List: ")
print(List)

# Addition of multiple elements
# to the List at the end
# (using Extend Method)
List.extend([8, 'Geeks', 'Always'])
print("\nList after performing Extend Operation: ")
print(List)
```

Output
Initial List:
[1, 2, 3, 4]

List after performing Extend Operation:
[1, 2, 3, 4, 8, 'Geeks', 'Always']

3.2.1.9. Removing Elements from the List

Method 1: Using remove() method

Python provides the remove() function which is used to remove the element from the list. Consider the following example to understand this concept.

Example1:

Code

```
# Python program to demonstrate
# Removal of elements in a List

# Creating a List
List = [1, 2, 3, 4, 5, 6,
        7, 8, 9, 10, 11, 12]
print("Initial List: ")
print(List)
# Removing elements from List
# using Remove() method
List.remove(5)
List.remove(6)
print("\nList after Removal of two elements: ")
print(List)
```

Output

```
Initial List:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

List after Removal of two elements:
[1, 2, 3, 4, 7, 8, 9, 10, 11, 12]
```

Example 2:

```
# Creating a List
List = [1, 2, 3, 4, 5, 6,
        7, 8, 9, 10, 11, 12]
# Removing elements from List
# using iterator method
for i in range(1, 5):
    List.remove(i)
print("\nList after Removing a range of elements: ")
```

```
print(List)
```

Output

```
List after Removing a range of elements:
[5, 6, 7, 8, 9, 10, 11, 12]
```

Method 2: Using pop() method

Pop() function can also be used to remove and return an element from the list, but by default it removes only the last element of the list, to remove an element from a specific position of the List, the index of the element is passed as an argument to the pop() method.

Example:

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


# Removing element from the
# Set using the pop() method
List.pop()
print("\nList after popping an element: ")
print(List)


# Removing element at a
# specific location from the
# Set using the pop() method
List.pop(2)
print("\nList after popping a specific element: ")
print(List)
```

Output

```
List after popping an element:
[1, 2, 3, 4]
List after popping a specific element:
[1, 2, 4]
```

3.2.1.10. Reversing a List

Method 1: A list can be reversed by using the **reverse() method in Python**.

Example:

```
# Reversing a list
mylist = [1, 2, 3, 4, 5, 'Geek', 'Python']
mylist.reverse()
print(mylist)
```

Output

[Python, Geek, 5, 4, 3, 2, 1]

3.2.1.11. Python List Built-in Functions

Python provides the following built-in functions, which can be used with the lists.

1. len()
2. max()
3. min()

1. len( )

It is used to calculate the length of the list.

Code

```
# size of the list
# declaring the list
list1 = [12, 16, 18, 20, 39, 40]
# finding length of the list
len(list1)
```

Output:

```
6
```

2. Max( )

It returns the maximum element of the list

Code

```
# maximum of the list
list1 = [103, 675, 321, 782, 200]
# large element in the list
print(max(list1))
```

Output:

```
782
```

3. Min( )

It returns the minimum element of the list

Code

```
# minimum of the list
list1 = [103, 675, 321, 782, 200]
# smallest element in the list
print(min(list1))
```

Output:

```
103
```

Let's have a look at the few list examples.

Example: 1- Write the program to remove the duplicate element of the list.

Code

```
list1 = [1,2,2,3,55,98,65,65,13,29]
# Declare an empty list that will store unique values
list2 = []
for i in list1:
    if i not in list2:
        list2.append(i)
print(list2)
```

Output:

```
[1, 2, 3, 55, 98, 65, 13, 29]
```

Example:2- Write a program to find the sum of the element in the list.
Code

```
list1 = [3,4,5,9,10,12,24]
sum = 0
for i in list1:
    sum = sum+i
print("The sum is:",sum)
```

Output:

```
The sum is: 67
In [8]:
```

Example: 3- Write the program to find the lists consist of at least one common element.
Code

```
list1 = [1,2,3,4,5,6]
list2 = [7,8,9,2,10]
for x in list1:
    for y in list2:
        if x == y:
            print("The common element is:",x)
```

Output:
The common element is: 2

Exercises

1. Python program to interchange first and last elements in a list
   Example:
   Input : [12, 35, 9, 56, 24]
   Output : [24, 35, 9, 56, 12]
2. Python | Ways to find length of list
   Example:
   Input: lst = [10,20,30,40]
   Output: 4

3. Maximum of two numbers in Python

Example:
Input: a = 2, b = 4
Output: 4

4. Minimum of two numbers in Python

   Example: Input: a = 2, b = 4
   Output: 2

3.2.2. Tuple

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

3.2.2.1. Creating a Tuple

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also.
For example

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containing nothing −

```
tup1 = ();
```

To write a tuple containing a single value you have to include a comma, even though there is only one value:

```
tup1 = (50,);
```

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

3.2.2.2. Accessing Values in Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index.
Accessing tuple items:

I. Positive Indexing:
As we have seen that tuple items have index, as such we can access items using these indexes.
Example:

```
country = ("Spain", "Italy", "India", "England", "Germany")
#          [0]     [1]     [2]      [3]       [4]
print(country[1])
print(country[3])
print(country[0])
```

Output:
Italy

England
Spain

II. Negative Indexing:
Similar to positive indexing, negative indexing is also used to access items, but from the end of the tuple. The last item has index [-1], second last item has index [-2], third last item has index [-3] and so on.

Example1:

```
tup = (22, 3, 45, 4, 2.4, 2, 56, 890, 1)
print(tup[-4])
```

Output:

prints the fourth element from the last

2

Example2:

```
tup = (22, 3, 45, 4, 2.4, 2, 56, 890, 1)
#Now, we can use negative indexes in slicing also.
print(tup[-4:-1])
```

Output:

prints elements from last fourth to last first

(2, 56, 890)

Example3:

```
country = ("Spain", "Italy", "India", "England", "Germany")
#          [0]     [1]     [2]      [3]       [4]
print(country[-1])
print(country[-3])
print(country[-4])
```

Output:

Germany
India
Italy

III. Check for item:
We can check if a given item is present in the tuple. This is done using the ***IN*** keyword.

Example 1:

```
country = ("Spain", "Italy", "India", "England", "Germany")
if "Germany" in country:
   print("Germany is present.")
else:
   print("Germany is absent.")
```

Output:

Germany is present.
Copy

Example 2:

```
country = ("Spain", "Italy", "India", "England", "Germany")
```

```
if "Russia" in country:
   print("Russia is present.")
else:
   print("Russia is absent.")
```
Output:
Russia is absent.

IV. Range of Index:
You can print a range of tuple items by specifying where do you want to start, where do you want to end and if you want to skip elements in between the range.

Syntax:
Tuple[start : end : jumpIndex]
Note: jump Index is optional. We will see this in given examples.

Example: printing elements within a particular range:
```
animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey", "goat", "cow")
print(animals[3:7])     #using positive indexes
print(animals[-7:-2])   #using negative indexes
```

Output:
('mouse', 'pig', 'horse', 'donkey')
('bat', 'mouse', 'pig', 'horse', 'donkey')

Here, we provide index of the element from where we want to start and the index of the element till which we want to print the values.
Note: The element of the end index provided will not be included.

Example: printing all element from a given index till the end
```
animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey", "goat", "cow")
print(animals[4:])      #using positive indexes
print(animals[-4:])     #using negative indexes
```

Output:
('pig', 'horse', 'donkey', 'goat', 'cow')
('horse', 'donkey', 'goat', 'cow')

When no end index is provided, the interpreter prints all the values till the end.

Example: printing all elements from start to a given index
```
animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey", "goat", "cow")
print(animals[:6])      #using positive indexes
print(animals[:-3])     #using negative indexes
```

Output:
('cat', 'dog', 'bat', 'mouse', 'pig', 'horse')
('cat', 'dog', 'bat', 'mouse', 'pig', 'horse')

When no start index is provided, the interpreter prints all the values from start up to the end index provided.

Example: print alternate values
```
animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey", "goat", "cow")
print(animals[::2])     #using positive indexes
print(animals[-8:-1:2]) #using negative indexes
```

Output:
('cat', 'bat', 'pig', 'donkey', 'cow')
('dog', 'mouse', 'horse', 'goat')

Here, we have not provided start and index, which means all the values will be considered. But as we have provided a jump index of 2 only alternate values will be printed.

Example: printing every 3rd consecutive withing given range
```
animals = ("cat", "dog", "bat", "mouse", "pig", "horse", "donkey", "goat", "cow")
print(animals[1:8:3])
```

Output:
('dog', 'pig', 'goat')
Here, jump index is 3. Hence it prints every 3rd element within given index.

3.2.2.3. Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.
But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.
Example:
```
#Convert the tuple into a list to be able to change it:
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

Output:
('apple', 'kiwi', 'cherry')

3.2.2.4. Adding/Inserting items in a Tuple

Since tuples are immutable, they do not have a built-in append() method, but there are other ways to add items to a tuple.

You can concatenate a tuple by adding new items to the beginning or end as previously mentioned; but, if you wish to insert a new item at any location, you must convert the tuple to a list.

1. **Convert into a list**: Just like the workaround for *changing* a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.
Example
```
s= (2,5,8)
```

```
# Python conversion for list and tuple to one another
x = list(s)
print(x)
print(type(x))

# Add items by using insert ()
x.insert(5, 90)
print(x)

# Use tuple to convert a list to a tuple ().
s_insert = tuple(x)
print(s_insert)
print(type(s_insert)))
```

Output:

we get the following output of the above code.

```
[2, 5, 8]
<class 'list'>
[2, 5, 8, 90]
(2, 5, 8, 90)
<class 'tuple'>
```

2. Using append() method

A new element is added to the end of the list using the **append() method**.

Example:

```
# converting tuple to list
t=(45,67,36,85,32)
l = list(t)
print(l)
print(type(l))
# appending the element in a list
l.append(787)
print(l)

# Converting the list to tuple using tuple()
t=tuple(l)
print(t)
```

Output:

```
[45, 67, 36, 85, 32]
<class 'list'>
[45, 67, 36, 85, 32, 787]
(45, 67, 36, 85, 32, 787)
```

EXERCISE

Add an item in a tuple in the following list:

L1= (4, 6, 2, 8, 3, 1)

Merge 9 at the end of the tuple L1

L1(4, 6, 2, 8, 3, 1, 9)

Sol/ L1 = (4, 6, 2, 8, 3, 1)
l = list(L1)
print(l)
print(type(l))
# appending the element in a list
l.append(9)
print(l)

3.2.2.5. Remove Tuple Items

Note: You cannot remove items in a tuple.

Tuples are **unchangeable**, so you cannot remove items from it, but you can delete the tuple completely:

The del keyword can delete the tuple completely:

thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) **#this will raise an error because the tuple no longer exists**

Exercise check the following link:

https://www.w3resource.com/python-exercises/tuple/python-tuple-exercise-12.php

3.2.2.6. Loop through a Tuple

You can loop through the tuple items by using a for loop.

Example:

Iterate through the items and print the values:

thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)

Output:

apple

banana

cherry

3.2.2.7. Join Two Tuples

To join two or more tuples you can use the + operator:

Example:

Join two tuples:

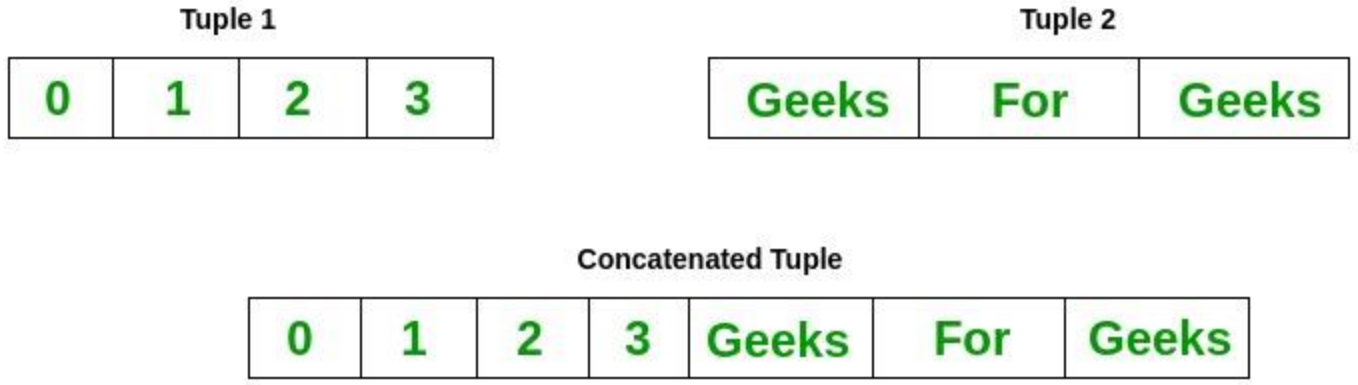
```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

Output:

('a', 'b', 'c', 1, 2, 3)

3.2.2.8. Concatenation of Tuples

Concatenation of tuple is the process of joining two or more Tuples. Concatenation is done by the use of '+' operator. Concatenation of tuples is done always from the end of the original tuple. Other arithmetic operations do not apply on Tuples.
Note- Only the same datatypes can be combined with concatenation, an error arises if a list and a tuple are combined.



Example1:

```
# Concatenation of tuples
Tuple1 = (0, 1, 2, 3)
Tuple2 = ('Geeks', 'For', 'Geeks')

Tuple3 = Tuple1 + Tuple2

# Printing first Tuple
print("Tuple 1: ")
print(Tuple1)

# Printing Second Tuple
print("\nTuple2: ")
print(Tuple2)

# Printing Final Tuple
print("\nTuples after Concatenation: ")
print(Tuple3)
```

Example2:

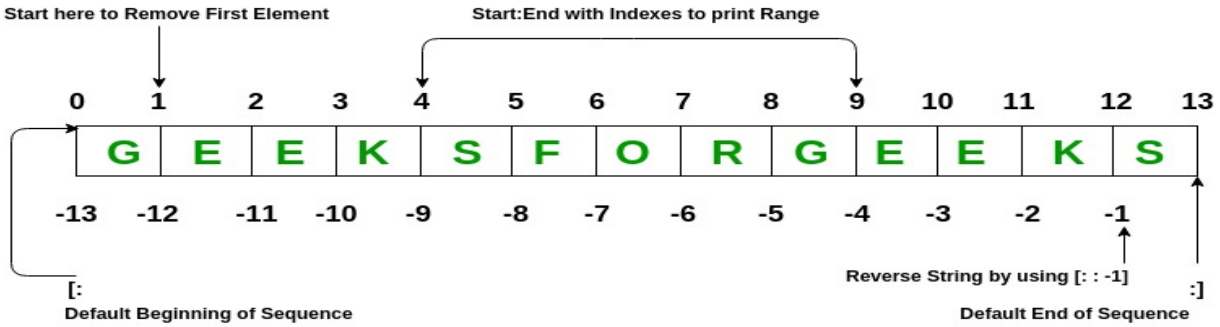
```
#!/usr/bin/python

tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');

# Following action is not valid for tuples
# tup1[0] = 100;

# So let's create a new tuple as follows
tup3 = tup1 + tup2;
print tup3;
```
When the above code is executed, it produces the following result:
(12, 34.56, 'abc', 'xyz')

Output:
Tuple 1:
(0, 1, 2, 3)

Tuple2:
('Geeks', 'For', 'Geeks')

Tuples after Concatenation:
(0, 1, 2, 3, 'Geeks', '

Slicing of Tuple
Slicing of a Tuple is done to fetch a specific range or slice of sub-elements from a Tuple. Slicing can also be done to lists and arrays. Indexing in a list results to fetching a single element whereas Slicing allows to fetch a set of elements.
Note- Negative Increment values can also be used to reverse the sequence of Tuples.



Example:

```
# Slicing of a Tuple
```

```
# Slicing of a Tuple
# with Numbers
Tuple1 = tuple('GEEKSFORGEEKS')
```

```
# Removing First element
print("Removal of First Element: ")
print(Tuple1[1:])

# Reversing the Tuple
print("\nTuple after sequence of Element is reversed: ")
print(Tuple1[::-1])

# Printing elements of a Range
print("\nPrinting elements between Range 4-9: ")
print(Tuple1[4:9])
```

Output:

```
Removal of First Element:
('E', 'E', 'K', 'S', 'F', 'O', 'R', 'G', 'E', 'E', 'K', 'S')

Tuple after sequence of Element is reversed:
('S', 'K', 'E', 'E', 'G', 'R', 'O', 'F', 'S', 'K', 'E', 'E', 'G')

Printing elements between Range 4-9:
('S', 'F', 'O', 'R', 'G')
```

3.2.2.9. Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.
To explicitly remove an entire tuple, just use the **del** statement.
For example

```
#!/usr/bin/python

tup = ('physics', 'chemistry', 1997, 2000);
print tup;
del tup;
print "After deleting tup : ";
print tup;
```

Note − an exception raised, this is because after **del tup** tuple does not exist anymore.
This produces the following result:

```
('physics', 'chemistry', 1997, 2000)
After deleting tup :
Traceback (most recent call last):
  File "test.py", line 9, in <module>
    print tup;
NameError: name 'tup' is not defined
```

3.2.2.10. Tuple length

Python tuple method **len()** returns the number of elements in the tuple.

Syntax

Following is the syntax for len() method −

len(tuple)

Example:

tuple1, tuple2 = (123, 'xyz', 'zara'), (456, 'abc')

print "First tuple length : ", len(tuple1)

print "Second tuple length : ", len(tuple2)

Output:

First tuple length : 3

Second tuple length : 2

Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string.
In fact, tuples respond to all of the general sequence operations we used on strings in the prior chapter.

Python Expression	Results	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

3.2.3. Set

Python Set

A Python set is the collection of the unordered items. Each element in the set must be unique, immutable, and the sets remove the duplicate elements. Sets are mutable which means we can modify it after its creation.

Unlike other collections in Python, there is no index attached to the elements of the set, i.e., we cannot directly access any element of the set by the index. However, we can print them all together, or we can get the list of elements by looping through the set.

Creating a set

The set can be created by enclosing the comma-separated immutable items with the curly braces {}. Python also provides the set() method, which can be used to create the set by the passed sequence.

Example 1: Using curly braces

```
Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"}
print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
    print(i)
```

Output:

```
{'Friday', 'Tuesday', 'Monday', 'Saturday', 'Thursday', 'Sunday', 'Wednesday'}
<class 'set'>
looping through the set elements ...
Friday
Tuesday
Monday
Saturday
Thursday
Sunday
Wednesday
```

Example 2: Using set() method

```
Days = set(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"])
print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
    print(i)
```

Output:

```
{'Friday', 'Wednesday', 'Thursday', 'Saturday', 'Monday', 'Tuesday', 'Sunday'}
<class 'set'>
looping through the set elements ...
Friday
Wednesday
Thursday
```

Saturday

Monday

Tuesday

Sunday

It can contain any type of element such as integer, float, tuple etc. But mutable(we can modify it) elements (list, dictionary, set) can't be a member of set. Consider the following example.

```
# Creating a set which have immutable elements
set1 = {1,2,3, "JavaTpoint", 20.5, 14}
print(type(set1))
#Creating a set which have mutable element
set2 = {1,2,3,["Javatpoint",4]}
print(type(set2))
```

Output:

```
<class 'set'>


Traceback (most recent call last)
<ipython-input-5-9605bb6fbc68> in <module>
    4
    5 #Creating a set which holds mutable elements
----> 6 set2 = {1,2,3,["Javatpoint",4]}
    7 print(type(set2))


TypeError: unhashable type: 'list'
```

In the above code, we have created two sets, the set **set1** have immutable elements and set2 have one mutable element as a list. While checking the type of set2, it raised an error, which means set can contain only immutable elements.

Creating an empty set is a bit different because empty curly {} braces are also used to create a dictionary as well. So Python provides the set() method used without an argument to create an empty set.

```
# Empty curly braces will create dictionary
set3 = {}
print(type(set3))

# Empty set using set() function
set4 = set()
print(type(set4))
```

Output:

```
<class 'dict'>
<class 'set'>
```

Let's see what happened if we provide the duplicate element to the set.

```
set5 = {1,2,4,4,5,8,9,9,10}
print("Return set with unique elements:",set5)
```

Output:

Return set with unique elements: {1, 2, 4, 5, 8, 9, 10}

In the above code, we can see that **set5** consisted of multiple duplicate elements when we printed it remove the duplicity from the set.

Adding items to the set

Python provides the **add()** method and **update()** method which can be used to add some particular item to the set. The add() method is used to add a single element whereas the update() method is used to add multiple elements to the set. Consider the following example.

Example: 1 - Using add() method

```
months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(months)
print("\nAdding other months to the set...");
Months.add("July");
Months.add ("August");
print("\nPrinting the modified set...");
print(Months)
print("\nlooping through the set elements ... ")
for i in months:
   print(i)
```

Output:

printing the original set ...

{'February', 'May', 'April', 'March', 'June', 'January'}

Adding other months to the set...

Printing the modified set...

{'February', 'July', 'May', 'April', 'March', 'August', 'June', 'January'}

looping through the set elements ...

February

July

May

April

March

August

June

January

To add more than one item in the set, Python provides the **update()** method. It accepts iterable as an argument.

Consider the following example.

Example - 2 Using update() function

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nupdating the original set ... ")
Months.update(["July","August","September","October"]);
print("\nprinting the modified set ... ")
print(Months)
```

Output:

```
printing the original set ...
{'January', 'February', 'April', 'May', 'June', 'March'}


updating the original set ...
printing the modified set ...
{'January', 'February', 'April', 'August', 'October', 'May', 'June', 'July', 'September', 'March'}
```

Removing items from the set

Python provides the **discard()** method and **remove()** method which can be used to remove the items from the set. The difference between these function, using discard() function if the item does not exist in the set then the set remain unchanged whereas remove() method will through an error.

Consider the following example.

Example-1 Using discard() method

```
months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(months)
print("\nRemoving some months from the set...");
months.discard("January");
months.discard("May");
print("\nPrinting the modified set...");
print(months)
print("\nlooping through the set elements ... ")
for i in months:
   print(i)
```

Output:

printing the original set ...

{'February', 'January', 'March', 'April', 'June', 'May'}

Removing some months from the set...

Printing the modified set...

{'February', 'March', 'April', 'June'}

looping through the set elements ...

February

March

April

June

Python provides also the **remove()** method to remove the item from the set. Consider the following example to remove the items using **remove()** method.

Example-2 Using remove() function

```
months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(months)
print("\nRemoving some months from the set...");
months.remove("January");
months.remove("May");
print("\nPrinting the modified set...");
print(months)
```

Output:

printing the original set ...

{'February', 'June', 'April', 'May', 'January', 'March'}

Removing some months from the set...

Printing the modified set...

{'February', 'June', 'April', 'March'}

We can also use the pop() method to remove the item. Generally, the pop() method will always remove the last item but the set is unordered, we can't determine which element will be popped from set.

Consider the following example to remove the item from the set using pop() method.

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
```

```
print(Months)
print("\nRemoving some months from the set...");
Months.pop();
Months.pop();
print("\nPrinting the modified set...");
print(Months)
```

Output:

```
printing the original set ...
{'June', 'January', 'May', 'April', 'February', 'March'}

Removing some months from the set...

Printing the modified set...
{'May', 'April', 'February', 'March'}
```

In the above code, the last element of the **Month** set is **March** but the pop() method removed the **June and January** because the set is unordered and the pop() method could not determine the last element of the set.

Python provides the clear() method to remove all the items from the set.

Consider the following example.

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nRemoving all the items from the set...");
Months.clear()
print("\nPrinting the modified set...")
print(Months)
```

Output:

```
printing the original set ...
{'January', 'May', 'June', 'April', 'March', 'February'}

Removing all the items from the set...

Printing the modified set...
set()
```

Difference between discard() and remove()

Despite the fact that **discard()** and **remove()** method both perform the same task, There is one main difference between discard() and remove().

If the key to be deleted from the set using discard() doesn't exist in the set, the Python will not give the error. The program maintains its control flow.

On the other hand, if the item to be deleted from the set using remove() doesn't exist in the set, the Python will raise an error.

Consider the following example.

Example-

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nRemoving items through discard() method...");
Months.discard("Feb"); #will not give an error although the key feb is not available in the set
print("\nprinting the modified set...")
print(Months)
print("\nRemoving items through remove() method...");
Months.remove("Jan") #will give an error as the key jan is not available in the set.
print("\nPrinting the modified set...")
print(Months)
```

Output:

```
printing the original set ...
{'March', 'January', 'April', 'June', 'February', 'May'}

Removing items through discard() method...

printing the modified set...
{'March', 'January', 'April', 'June', 'February', 'May'}

Removing items through remove() method...
Traceback (most recent call last):
  File "set.py", line 9, in
    Months.remove("Jan")
KeyError: 'Jan'
```

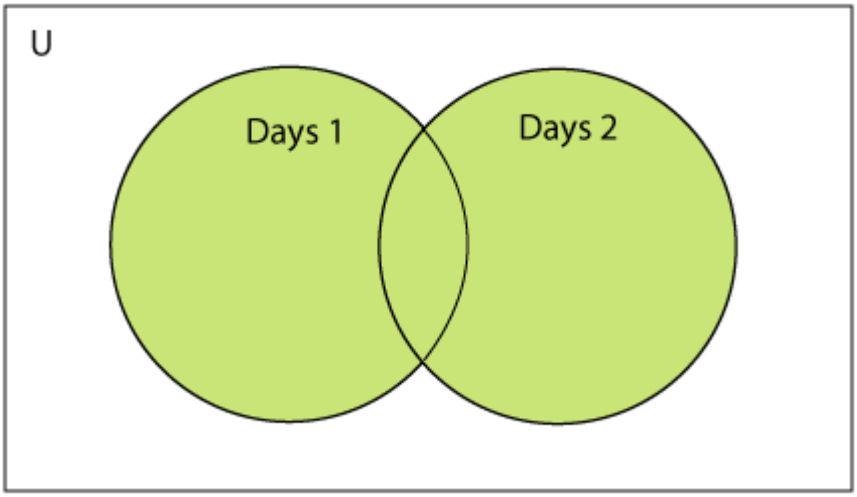
Python Set Operations

Set can be performed mathematical operation such as union, intersection, difference, and symmetric difference. Python provides the facility to carry out these operations with operators or methods. We describe these operations as follows.

Union of two Sets

To combine two or more sets into one set in Python, use the union() function. All of the distinctive characteristics from each combined set are present in the final set. As parameters, one or more sets may be passed to the union() function. The function returns a copy of the set supplied as the lone parameter if

there is just one set. The method returns a new set containing all the different items from all the arguments if more than one set is supplied as an argument.



Consider the following example to calculate the union of two sets.

Example 1: using union | operator

```
Days1 = {"Monday","Tuesday","Wednesday","Thursday", "Sunday"}
Days2 = {"Friday","Saturday","Sunday"}
print(Days1|Days2) #printing the union of the sets
```

Output:

```
{'Friday', 'Sunday', 'Saturday', 'Tuesday', 'Wednesday', 'Monday', 'Thursday'}
```

Python also provides the **union()** method which can also be used to calculate the union of two sets. Consider the following example.

Example 2: using union() method

```
Days1 = {"Monday","Tuesday","Wednesday","Thursday"}
Days2 = {"Friday","Saturday","Sunday"}
print(Days1.union(Days2)) #printing the union of the sets
```

Output:

```
{'Friday', 'Monday', 'Tuesday', 'Thursday', 'Wednesday', 'Sunday', 'Saturday'}
```

Now, we can also make the union of more than two sets using the union() function, for example:

Program:

```
# Create three sets
set1 = {1, 2, 3}
set2 = {2, 3, 4}
set3 = {3, 4, 5}


# Find the common elements between the three sets
common_elements = set1.union(set2, set3)
```

```
# Print the common elements
print(common_elements)
```

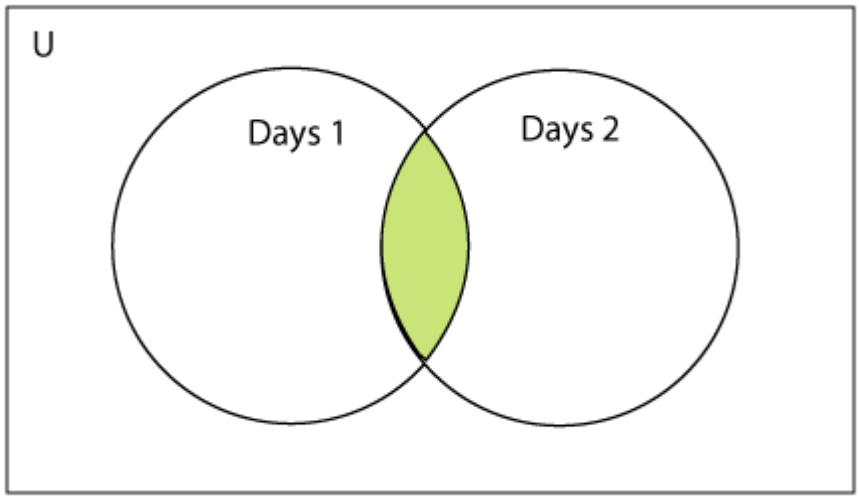
Output:

```
{1, 2, 3, 4, 5}
```

The intersection of two sets

To discover what is common between two or more sets in Python, apply the intersection() function. Only the items in all sets being compared are included in the final set. One or more sets can also be used as the intersection() function parameters. The function returns a copy of the set supplied as the lone parameter if there is just one set. The method returns a new set that only contains the elements in all the compared sets if multiple sets are supplied as arguments.

The intersection of two sets can be performed by the **and &** operator or the **intersection() function**. The intersection of the two sets is given as the set of the elements that common in both sets.



Consider the following example.

Example 1: Using & operator

```
Days1 = {"Monday","Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday","Tuesday","Sunday", "Friday"}
print(Days1&Days2) #prints the intersection of the two sets
```

Output:

```
{'Monday', 'Tuesday'}
```

Example 2: Using intersection() method

```
set1 = {"Devansh","John", "David", "Martin"}
set2 = {"Steve", "Milan", "David", "Martin"}
print(set1.intersection(set2)) #prints the intersection of the two sets
```

Output:

```
{'Martin', 'David'}
```

Example 3:

```
set1 = {1,2,3,4,5,6,7}
set2 = {1,2,20,32,5,9}
set3 = set1.intersection(set2)
print(set3)
```

Output:

{1,2,5}

Similarly, as the same as union function, we can perform the intersection of more than two sets at a time,

For Example:

Program

```
# Create three sets
set1 = {1, 2, 3}
set2 = {2, 3, 4}
set3 = {3, 4, 5}

# Find the common elements between the three sets
common_elements = set1.intersection(set2, set3)
  # Print the common elements
print(common_elements)
```

Output:

{3}

The intersection_update() method

The **intersection_update()** method removes the items from the original set that are not present in both the sets (all the sets if more than one are specified).

The **intersection_update()** method is different from the intersection() method since it modifies the original set by removing the unwanted items, on the other hand, the intersection() method returns a new set.

Consider the following example.

```
a = {"Devansh", "bob", "castle"}
b = {"castle", "dude", "emyway"}
c = {"fuson", "gaurav", "castle"}

a.intersection_update(b, c)

print(a)
```

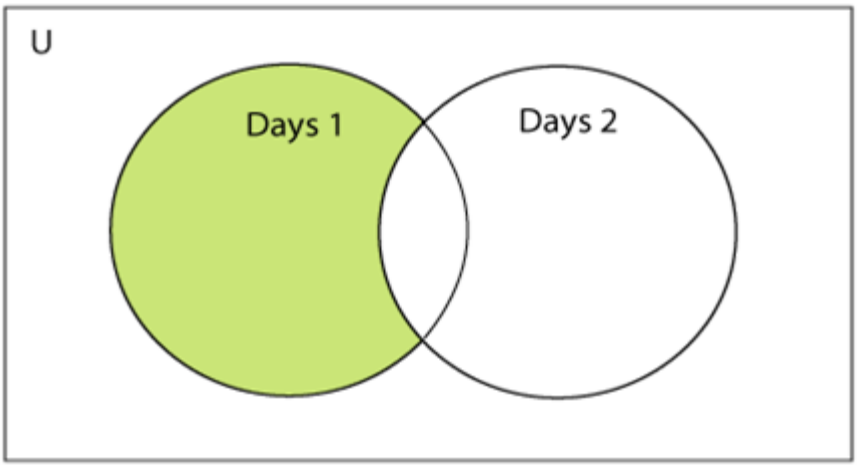
Output:

{'castle'}

Difference between the two sets

The difference of two sets can be calculated by using the subtraction (-) operator

or **intersection()** method. Suppose there are two sets A and B, and the difference is A-B that denotes the resulting set will be obtained that element of A, which is not present in the set B.



Consider the following example.

Example 1 : Using subtraction (-) operator

```
Days1 = {"Monday",  "Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday", "Tuesday", "Sunday"}
print(Days1-Days2) #{"Wednesday", "Thursday" will be printed}
```

Output:

```
{'Thursday', 'Wednesday'}
```

Example 2 : Using difference() method

```
Days1 = {"Monday",  "Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday", "Tuesday", "Sunday"}
print(Days1.difference(Days2)) # prints the difference of the two sets Days1 and Days2
```

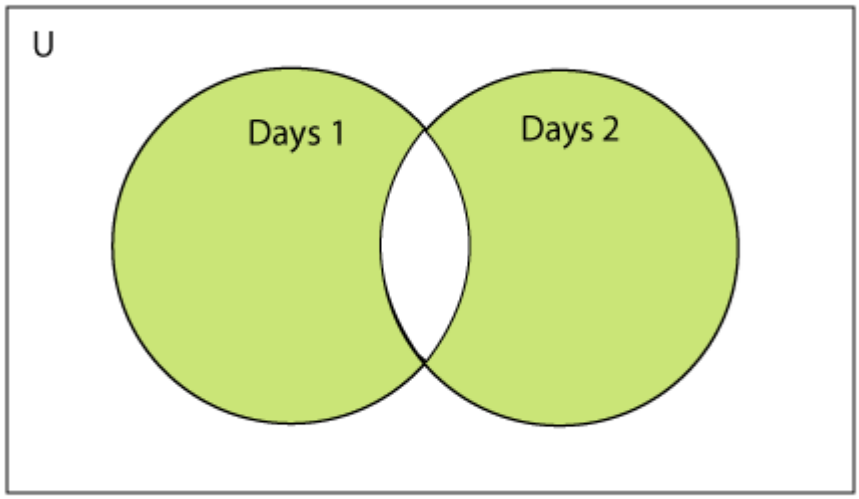
Output:

```
{'Thursday', 'Wednesday'}
```

Symmetric Difference of two sets

In Python, the symmetric Difference between set1 and set2 is the set of elements present in one set or the other but not in both sets. In other words, the set of elements is in set1 or set2 but not in their intersection.

The Symmetric Difference of two sets can be computed using Python's symmetric_difference() method. This method returns a new set containing all the elements in either but not in both. Consider the following example:



Example - 1: Using ^ operator

```
a = {1,2,3,4,5,6}
b = {1,2,9,8,10}
c = a^b
print(c)
```

Output:

{3, 4, 5, 6, 8, 9, 10}

Example - 2: Using symmetric_difference() method

```
a = {1,2,3,4,5,6}
b = {1,2,9,8,10}
c = a.symmetric_difference(b)
print(c)
```

Output:

{3, 4, 5, 6, 8, 9, 10}

Set comparisons

In Python, you can compare sets to check if they are equal, if one set is a subset or superset of another, or if two sets have elements in common.

Here are the set comparison operators available in Python:

==: checks if two sets have the same elements, regardless of their order.

!=: checks if two sets are not equal.

<: checks if the left set is a proper subset of the right set (i.e., all elements in the left set are also in the right set, but the right set has additional elements).

<=: checks if the left set is a subset of the right set (i.e., all elements in the left set are also in the right set).

>: checks if the left set is a proper superset of the right set (i.e., all elements in the right set are also in the left set, but the left set has additional elements).

>=: checks if the left set is a superset of the right set (i.e., all elements in the right set are also in the left).

Consider the following example.

```
Days1 = {"Monday", "Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday", "Tuesday"}
Days3 = {"Monday", "Tuesday", "Friday"}

#Days1 is the superset of Days2 hence it will print true.
print (Days1>Days2)

#prints false since Days1 is not the subset of Days2
print (Days1<Days2)

#prints false since Days2 and Days3 are not equivalent
print (Days2 == Days3)
```

Output:

```
True
False
False
```

Python Built-in set methods

Python contains the following methods to be used with the sets.

SN	Method	Description
1	add(item)	It adds an item to the set. It has no effect if the item is already present in the set.
2	clear()	It deletes all the items from the set.
3	copy()	It returns a shallow copy of the set.
4	difference_update(....)	It modifies this set by removing all the items that are also present in the specified sets.
5	discard(item)	It removes the specified item from the set.
6	intersection()	It returns a new set that contains only the common elements of both the sets. (all the sets if more than two are specified).
7	intersection_update(....)	It removes the items from the original set that are not present in both the sets (all the sets if more than one

		are specified).
8	Isdisjoint(....)	Return True if two sets have a null intersection.
9	Issubset(....)	Report whether another set contains this set.
10	Issuperset(....)	Report whether this set contains another set.
11	pop()	Remove and return an arbitrary set element that is the last element of the set. Raises KeyError if the set is empty.
12	remove(item)	Remove an element from a set; it must be a member. If the element is not a member, raise a KeyError.
13	symmetric_difference(....)	Remove an element from a set; it must be a member. If the element is not a member, raise a KeyError.
14	symmetric_difference_update(....)	Update a set with the symmetric difference of itself and another.
15	union(....)	Return the union of sets as a new set. (i.e. all elements that are in either set.)
16	update()	Update a set with the union of itself and others.

3.2.4. Dictionary

Dictionaries are a useful data structure for storing data in Python because they are capable of imitating real-world data arrangements where a certain value exists for a given key. In Dictionary each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this − {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Creating the Dictionary

Curly brackets are the simplest way to generate a Python dictionary, although there are other approaches as well. With many key-value pairs surrounded in curly brackets and a colon separating each key from its value, the dictionary can be built. (:). The following provides the syntax for defining the dictionary.

Syntax:

Dict = {"Name": "Gayle", "Age": 25}

In the above dictionary Dict, The keys Name and Age are the strings which comes under the category of an immutable object.

Let's see an example to create a dictionary and print its content.

Code

```
Employee = {"Name": "Johnny", "Age": 32, "salary":26000,"Company":"^TCS"}
print(type(Employee))
print("printing Employee data .... ")
print(Employee)
```

Output:

```
<class 'dict'>
printing Employee data ....
{'Name': 'Johnny', 'Age': 32, 'salary': 26000, 'Company': TCS}
```

3.2.3.1. Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value.

Example1:

A simple example is as follows:

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print ("dict['Name']: ", dict['Name'])
print ("dict['Age']: ", dict['Age'])
```

Output

When the above code is executed, it produces the following result:

```
dict['Name']:  Zara
dict['Age']:  7
```

If we attempt to access a data item with a key, which is not part of the dictionary, we get an error as follows:

Example

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print ("dict['Alice']: ", dict['Alice'])
```

Output

When the above code is executed, it produces the following result:

```
dict['Alice']:
```

```
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    print "dict['Alice']: ", dict['Alice'];
KeyError: 'Alice'
```

Example2:

```
Employee = {"Name": "Dev", "Age": 20, "salary":45000,"Company":"WIPRO"}
print(type(Employee))
print("printing Employee data .... ")
print("Name : %s" %Employee["Name"])
print("Age : %d" %Employee["Age"])
print("Salary : %d" %Employee["salary"])
print("Company : %s" %Employee["Company"])
```

Output

<class 'dict'>

printing Employee data ....

Name : Dev

Age : 20

Salary : 45000

Company : WIPRO

3.2.3.2. Updating Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example:

Example1:

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry

print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

Output

When the above code is executed, it produces the following result:

dict['Age']: 8

dict['School']: DPS School

Example2: Creating an empty Dictionary

```
Dict = {}
print("Empty Dictionary: ")
print(Dict)
# Adding elements to dictionary one at a time
Dict[0] = 'Peter'
Dict[2] = 'Joseph'
Dict[3] = 'Ricky'
print("\nDictionary after adding 3 elements: ")
print(Dict)
# Adding set of values
# with a single Key
# The Emp_ages doesn't exist to dictionary
Dict['Emp_ages'] = 20, 33, 24
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Updating existing Key's Value
Dict[3] = 'JavaTpoint'
print("\nUpdated key value: ")
print(Dict)
```

Output

```
Empty Dictionary:
{}
Dictionary after adding 3 elements:
{0: 'Peter', 2: 'Joseph', 3: 'Ricky'}

Dictionary after adding 3 elements:
{0: 'Peter', 2: 'Joseph', 3: 'Ricky', 'Emp_ages': (20, 33, 24)}

Updated key value:
{0: 'Peter', 2: 'Joseph', 3: 'JavaTpoint', 'Emp_ages': (20, 33, 24)}
```

3.2.3.3. Delete Dictionary Elements

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

Example

To explicitly remove an entire dictionary, just use the **del** statement. A simple example is as mentioned below:

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
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']
```

Note −that an exception is raised because after **del dict** dictionary does not exist any more

Output

This produces the following result:

```
dict['Age']:
Traceback (most recent call last):
  File "test.py", line 8, in <module>
    print "dict['Age']: ", dict['Age'];
TypeError: 'type' object is unsubscriptable
```

Note − del() method is discussed in subsequent section.

Example2:

The items of the dictionary can be deleted by using the **del** keyword as given below.

Code

```
Employee = {"Name": "David", "Age": 30, "salary":55000,"Company":"WIPRO"}
print(type(Employee))
print("printing Employee data .... ")
print(Employee)
print("Deleting some of the employee data")
del Employee["Name"]
del Employee["Company"]
print("printing the modified information ")
print(Employee)
print("Deleting the dictionary: Employee");
del Employee
```

```
print("Lets try to print it again ");
print(Employee)
```

Output

```
<class 'dict'>
printing Employee data ....
{'Name': 'David', 'Age': 30, 'salary': 55000, 'Company': 'WIPRO'}
Deleting some of the employee data
printing the modified information
{'Age': 30, 'salary': 55000}
Deleting the dictionary: Employee
Lets try to print it again
NameError: name 'Employee' is not defined.
```

The last print statement in the above code, it raised an error because we tried to print the Employee dictionary that already deleted.

Deleting Elements using pop() Method

A dictionary is a group of key-value pairs in Python. You can retrieve, insert, and remove items using this unordered, mutable data type by using their keys. The pop() method is one of the ways to get rid of elements from a dictionary. In this post, we'll talk about how to remove items from a Python dictionary using the pop() method.

The value connected to a specific key in a dictionary is removed using the pop() method, which then returns the value. The key of the element to be removed is the only argument needed. The pop() method can be used in the following ways:

Code

```
# Creating a Dictionary

Dict1 = {1: 'JavaTpoint', 2: 'Educational', 3: 'Website'}

# Deleting a key

# using pop() method

pop_key = Dict1.pop(2)

print(Dict1)
```

Output

```
{1: 'JavaTpoint', 3: 'Website'}
```

Additionally, Python offers built-in functions popitem() and clear() for removing dictionary items. In contrast to the clear() method, which removes all of the elements from the entire dictionary, popitem() removes any element from a dictionary.

Iterating Dictionary

A dictionary can be iterated using for loop as given below.

Example 1

Code

```
# for loop to print all the keys of a dictionary

Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"WIPRO"}

for x in Employee:

    print(x)
```

Output

```
Name

Age

salary

Company
```

Example 2

Code

```
#for loop to print all the values of the dictionary

Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"WIPRO"} for x in Employee:

 print(Employee[x])
```

Output

```
John

29

25000

WIPRO
```

Example - 3

Code

```
#for loop to print the values of the dictionary by using values() method.

Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"WIPRO"}

for x in Employee.values():

    print(x)
```

Output

```
John

29

25000

WIPRO
```

Example 4

Code

```
#for loop to print the items of the dictionary by using items() method

Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"WIPRO"}

for x in Employee.items():

    print(x)
```

Output

```
 ('Name', 'John')

('Age', 29)

('salary', 25000)

('Company', 'WIPRO')
```

3.2.3.4. Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys:

More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins.

For example

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}

print "dict['Name']: ", dict['Name']
```

Output

When the above code is executed, it produces the following result −

```
dict['Name']:  Manni
```

Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.

Example

An example is as follows:

```
#!/usr/bin/python

dict = {['Name']: 'Zara', 'Age': 7}

print "dict['Name']: ", dict['Name']
```

Output

When the above code is executed, it produces the following result:

```
Traceback (most recent call last):

  File "test.py", line 3, in <module>

    dict = {['Name']: 'Zara', 'Age': 7};

TypeError: list objects are unhashable
```

Learning Outcome 3.3: Understand condition statements

3.3.1. Explanation of Logical condition and IF statement

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.

Indentation

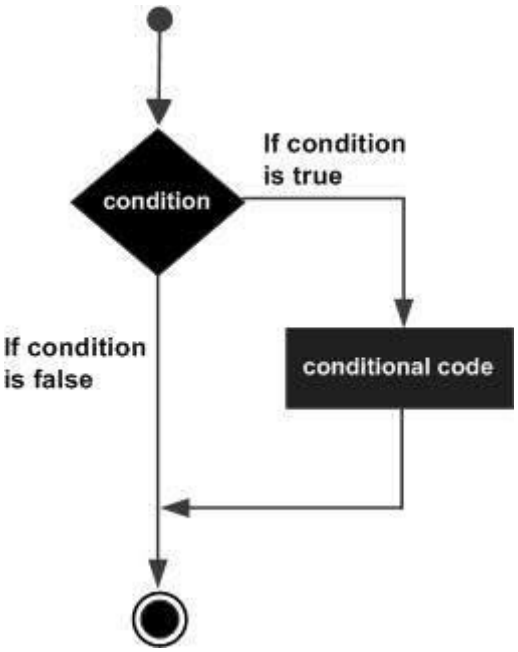
Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

Example

If statement, without indentation (will raise an error):

```
a = 33
b = 200
if b > a:
print("b is greater than a") # you will get an error
```

The if statement
The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.



An "if statement" is written by using the if keyword.

Python programming language provides the following types of decision-making statements.

S/N	Statement & Description
1	if statements An if statement consists of a boolean expression followed by one or more statements.
2	if...else statements An if statement can be followed by an optional else statement, which executes when the boolean expression is FALSE.
3	nested if statements You can use one if or else if statement inside another if or else if statement(s).

The syntax of the if-statement is given below.

```
if expression:
    statement
```

Example1:

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

Example2:

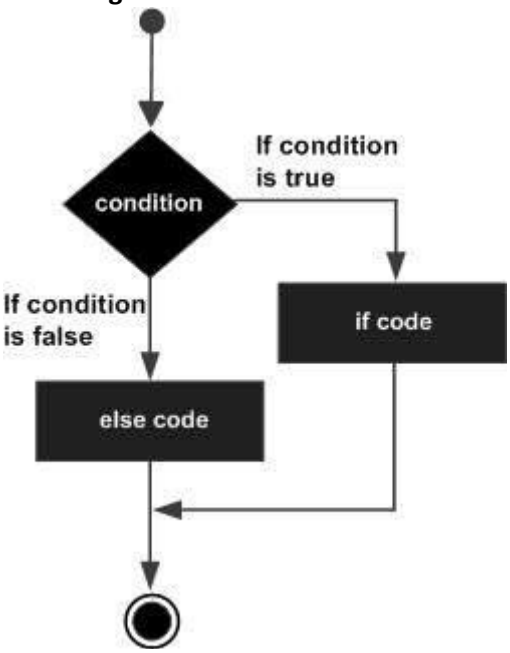
```
# Simple Python program to understand the if statement
num = int(input("enter the number:"))
# Here, we are taking an integer num and taking input dynamically
if num%2 == 0:
# Here, we are checking the condition. If the condition is true, we will enter the block
    print("The Given number is an even number")
```

Output:
enter the number: 10
The Given number is an even number

The if-else statement

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.
If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.

Flow Diagram



Example

The syntax of the if-else statement is given below.

```
if condition:
    #block of statements
else:
    #another block of statements (else-block)
```

Example 1 : Program to check whether a person is eligible to vote or not.

```
# Simple Python Program to check whether a person is eligible to vote or not.
age = int (input("Enter your age: "))
# Here, we are taking an integer num and taking input dynamically
if age>=18:
# Here, we are checking the condition. If the condition is true, we will enter the block
    print("You are eligible to vote !!");
else:
    print("Sorry! you have to wait !!");
```

Output:
Enter your age: 90
You are eligible to vote !!

Example 2: Program to check whether a number is even or not.

```
# Simple Python Program to check whether a number is even or not.
num = int(input("enter the number:"))
# Here, we are taking an integer num and taking input dynamically
if num%2 == 0:
# Here, we are checking the condition. If the condition is true, we will enter the block
    print("The Given number is an even number")
else:
    print("The Given Number is an odd number")
```

Output:

```
enter the number: 10
The Given number is even number
```

Example3:

```
amount = int(input("Enter amount: "))

if amount<1000:
   discount = amount*0.05
   print ("Discount",discount)
else:
   discount = amount*0.10
   print ("Discount",discount)

print ("Net payable:",amount-discount)
```

Output

In the above example, discount is calculated on the input amount. Rate of discount is 5%, if the amount is less than 1000, and 10% if it is above 10000. When the above code is executed, it produces the following result:

```
Enter amount: 600
Discount 30.0
Net payable: 570.0

Enter amount: 1200
Discount 120.0
Net payable: 1080.0
```

The elif statement

The elif keyword is Python's way of saying "if the previous conditions were not true, then try this condition". The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.

The syntax of the elif statement is given below.

```
if expression 1:
    # block of statements

elif expression 2:
    # block of statements
```

```
elif expression 3:
   # block of statements

else:
   # block of statements
```

Example 1

```
# Simple Python program to understand elif statement
number = int(input("Enter the number?"))
# Here, we are taking an integer number and taking input dynamically
if number==10:
# Here, we are checking the condition. If the condition is true, we will enter the block
   print("The given number is equals to 10")
elif number==50:
# Here, we are checking the condition. If the condition is true, we will enter the block
   print("The given number is equal to 50");
elif number==100:
# Here, we are checking the condition. If the condition is true, we will enter the block
   print("The given number is equal to 100");
else:
   print("The given number is not equal to 10, 50 or 100");
```

Output:

```
Enter the number?15
The given number is not equal to 10, 50 or 100
```

Example 2

```
# Simple Python program to understand elif statement
marks = int(input("Enter the marks? "))
# Here, we are taking an integer marks and taking input dynamically
if marks > 85 and marks <= 100:
# Here, we are checking the condition. If the condition is true, we will enter the block
  print("Congrats ! you scored grade A ...")
elif marks > 60 and marks <= 85:
# Here, we are checking the condition. If the condition is true, we will enter the block
  print("You scored grade B + ...")
elif marks > 40 and marks <= 60:
# Here, we are checking the condition. If the condition is true, we will enter the block
  print("You scored grade B ...")
elif (marks > 30 and marks <= 40):
# Here, we are checking the condition. If the condition is true, we will enter the block
  print("You scored grade C ...")
else:
  print("Sorry you are fail ?")
```

Output:

```
Enter the marks? 89
Congrats ! you scored grade A ...
```

ELSE

The else keyword catches anything which isn't caught by the preceding conditions.

Example

```
a = 200
```

```
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".

Output: a is greater than b
You can also have an else without the elif:
Example

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

Short Hand If

If you have only one statement to execute, you can put it on the same line as the if statement.
Example:
One line if statement:

```
if a > b: print("a is greater than b")
```

Short Hand If ... Else
If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:
Example
One line if else statement:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

Output: B

This technique is known as **Ternary Operators**, or **Conditional Expressions**.
You can also have multiple else statements on the same line:
Example
One line if else statement, with 3 conditions:

```
a = 330
b = 330
print("A") if a > b else print("=") if a == b else print("B")
```

Output: =

And
The and keyword is a logical operator, and is used to combine conditional statements:
Example
Test if a is greater than b, AND if c is greater than a:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

Or

The or keyword is a logical operator, and is used to combine conditional statements:

Example

Test if a is greater than b, OR if a is greater than c:

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

Not

The not keyword is a logical operator, and is used to reverse the result of the conditional statement:

Example

Test if a is NOT greater than b:

```
a = 33
b = 200
if not a > b:
  print("a is NOT greater than b")
```

Nested If

You can have if statements inside if statements, this is called *nested* if statements.

Example

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

Exercises

1. Write a python Program to print the largest of the three numbers.
2. Write a python program to compute and display the tax and the net salary from the gross salary entered by a user. Use the following conditions:

   The salary below 60,000 is not taxable

   The salary between 60,000 and 100,000 pays a 20% tax.

   The salary above 100,000 pays 30%.

   Ask the user to enter his social security contribution amount and the gross salary too.

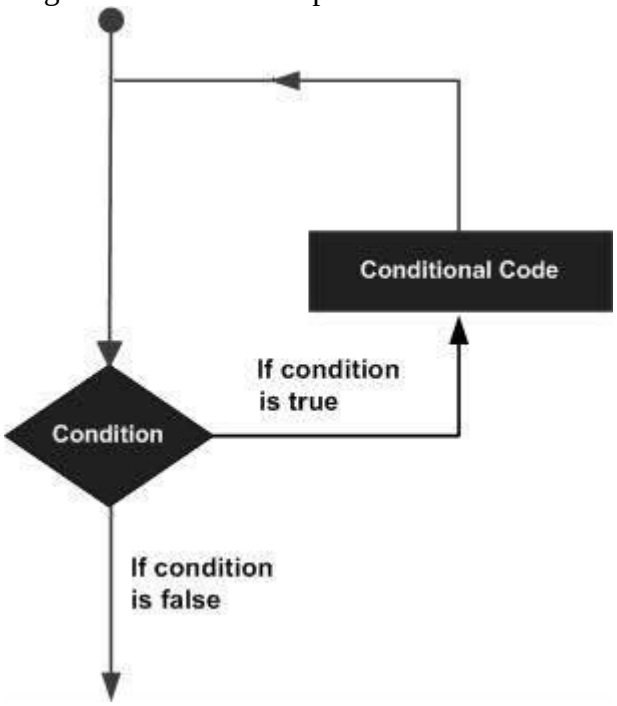
Learning Outcome 3.4: Identify other functions and classes

3.4.1. Loops

In general, statements are executed sequentially − The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

Programming languages provide various control structures that allow more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement:



Python programming language provides the following types of loops to handle looping requirements.

S/N	Loop Type & Description
1	while loop Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
2	for loop Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	nested loops You can use one or more loop inside any another while, or for loop.

3.4.1.1. WHILE LOOP

In coding, loops are designed to execute a specified code block repeatedly. We'll learn how to construct a while loop, for loop, in Python, the syntax of a while loop, loop controls like break and continue, and other exercises.

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax

The syntax of a **while** loop in Python programming language is:

Statement
```
while expression:
   statement(s)
```

Example1:
```
#Print i as long as i is less than 6:
i = 1
while i < 6:
  print(i)
  i += 1
```

Note: remember to increment i, or else the loop will continue forever.

OutPut:
1
2
3
4
5

Example2:
Now we give code examples of while loops in Python for Printing those numbers divisible by either 5 or 7 within 1 to 50 using a while loop. The code is given below -
```
i=1
while i<51:
    if i%5 == 0 or i%7==0 :
        print(i, end=' ')
    i+=1
```
Output:
5 7 10 14 15 20 21 25 28 30 35 40 42 45 49 50

Python supports having an **else** statement associated with a loop statement.

- If the **else** statement is used with a **for** loop, the **else** statement is executed when the loop has exhausted iterating the list.
- If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

The following example illustrates the combination of an else statement with a while statement that prints a number as long as it is less than 5, otherwise the else statement gets executed.
Example
```
#!/usr/bin/python3

count = 0
while count < 5:
   print (count, " is  less than 5")
   count = count + 1
else:
   print (count, " is not less than 5")
```
Output
When the above code is executed, it produces the following result:
0 is less than 5

1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than 5

3.4.1.2. for loop

Python's for loop is designed to repeatedly execute a code block while iterating through a list, tuple, dictionary, set, string or other iterable objects of Python. The process of traversing a sequence is known as iteration.

Syntax

```
for value in sequence:
    { code block }
```

In this case, the variable value is used to hold the value of every item present in the sequence before the iteration begins until this particular iteration is completed.

Loop iterates until the final item of the sequence are reached.

Example1: Print each fruit in a fruit list:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Output:

apple

banana

cherry

Example 2:

```
# Python program to show how the for loop works
# Creating a sequence which is a tuple of numbers
numbers = [4, 2, 6, 7, 3, 5, 8, 10, 6, 1, 9, 2]
# Creating an empty list
squares = []
# Creating a for loop
for value in numbers:
  square = value ** 2
  squares.append(square)
print("The list of squares is", squares)
```

Output:

The list of squares is [16, 4, 36, 49, 9, 25, 64, 100, 36, 1, 81, 4]

Using else Statement with Loops

Python supports having an else statement associated with a loop statement.

- If the **else** statement is used with a **for** loop, the **else** block is executed only if for loops terminates normally (and not by encountering break statement).
- If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

Example1:

The following example illustrates the combination of an else statement with a **for** statement that searches for even number in given list.

```
#!/usr/bin/python3

numbers = [11,33,55,39,55,75,37,21,23,41,13]

for num in numbers:
  if num%2 == 0:
     print ('the list contains an even number')
     break
else:
  print ('the list does not contain even number')
```

Output

When the above code is executed, it produces the following result:

```
the list does not contain even number
```

Example2:

```
# Python program to show how to use else statement with for loop
    # Creating a sequence
tuple_ = (3, 4, 6, 8, 9, 2, 3, 8, 9, 7)
# Initiating the loop
for value in tuple_:
   if value % 2 != 0:
      print(value)
# giving an else statement
else:
   print("These are the odd numbers present in the tuple")
```

Output:

```
3
9
```

3

9

7

These are the odd numbers present in the tuple

The range() Function

To loop through a set of code a specified number of times, we can use the range() function,
The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example

Using the range() function:

```
for x in range(6):
  print(x)
```

Output:

0

1

2

3

4

5

With the help of the range() function, we may produce a series of numbers. range(10) will produce values between 0 and 9. (10 numbers).We can give specific start, stop, and step size values in the manner range(start, stop, step size).

Example2:

```
# Python program to show the working of range() function
  print(range(15))
print(list(range(15)))
print(list(range(4, 9)))
print(list(range(5, 25, 4)))
```

Output:

```
range(0, 15)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
[4, 5, 6, 7, 8]
[5, 9, 13, 17, 21]
```

To iterate through a sequence of items, we can apply the range() method in for loops. We can use indexing to iterate through the given sequence by combining it with an iterable's len() function. Here's an illustration.

Code

```
# Python program to iterate over a sequence with the help of indexing

tuple_ = ("Python", "Loops", "Sequence", "Condition", "Range")

# iterating over tuple_ using range() function
for i in range(len(tuple_)):
    print(tuple_[i].upper())
```

Output:

PYTHON
LOOPS
SEQUENCE
CONDITION
RANGE

3.4.1.3. continue statement

The **continue** statement in Python returns the control to the beginning of the current loop. When encountered, the loop starts next iteration without executing the remaining statements in the current iteration.

The **continue** statement can be used in both *while* and *for* loops.

Example:

Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

Ouput:

1
2
4
5
6

3.4.1.4. break statement

The **break** statement is used for premature termination of the current loop. After abandoning the loop, execution at the next statement is resumed, just like the traditional break statement in C.

With the break statement we can stop the loop even if the while condition is true:

Example

```
#Exit the loop when i is 3:
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

Output:

1

2
3
While loop stops the execution of the loop when the break statement is reached.

Example:

```
# Python program to show how to use the break statement

# Initiating the loop
for string in "Python Loops":
    if string == 'n':
        break
    print('Current Letter: ', string)
```

Output:
Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

```
Current Letter:  P
Current Letter:  y
Current Letter:  t
Current Letter:  h
Current Letter:  o
```

3.4.2. Definition of Functions

A function is a block of code which only runs when it is called you can pass data, known as parameters, into a function. A function can return data as a result.
Python functions are necessary for intermediate-level programming and are easy to define. Function names meet the same standards as variable names do. The objective is to define a function and group-specific frequently performed actions. Instead of repeatedly creating the same code block for various input variables, we can call the function and reuse the code it contains with different variables.

As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.

3.4.2.1. Creating a function

In Python a function is defined using the def keyword.
Example:

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

3.4.2.2. Calling function

To call a function, use the function name followed by parenthesis.
Example:

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

Output: Hello from a function

3.4.2.3. Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.
The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name.
Example:

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

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Output:

```
Emil Refsnes
Tobias Refsnes
Linus Refsnes
```

Number of Arguments
By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.
This function expects 2 arguments, and gets 2 arguments:

```
def my_function(fname, lname):
  print(fname + " " + lname)
my_function("Emil", "Refsnes")
```

If you try to call the function with 1 or 3 arguments, you will get an error:

```
def my_function(fname, lname):
  print(fname + " " + lname)
my_function("Emil")
```

Arbitrary Arguments, *args
If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
This way the function will receive a tuple of arguments, and can access the items accordingly:
If the number of arguments is unknown, add a * before the parameter name:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

Keyword Arguments
You can also send arguments with the key = value syntax.
This way the order of the arguments does not matter.

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

Arbitrary Keyword Arguments, **kwargs
If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition. This way the function will receive a dictionary of arguments, and can access the items accordingly.
If the number of keyword arguments is unknown, add a double ** before the parameter name:

```
def my_function(**kid):
  print("His last name is " + kid["lname"])
```

```
my_function(fname = "Tobias", lname = "Refsnes")
```

3.4.2.4. Default parameter value

The following example shows how to use a default parameter value. If we call the function without argument, it uses the default value:

```
def my_function(country = "Norway"):
  print("I am from " + country)


my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

3.4.2.5. Passing a list as an argument

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.
E.g., if you send a List as an argument, it will still be a List when it reaches the function:

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

3.4.2.7. Lambda

A lambda function is a small anonymous function. A lambda function can take any number of arguments, but can only have one expression.
Syntax

```
lambda arguments : expression
```

The expression is executed and the result is returned:

```
x = lambda a : a + 10
print(x(5))
```

Lambda functions can take any number of arguments.
Example:

- Multiply argument **a** with argument **b** and return the result:

```
x = lambda a, b : a * b
print(x(5, 6))
```

- Summarize argument a, b, and c and return the result:

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

Why Use Lambda Functions?
The power of lambda is better shown when you use them as an anonymous function inside another function.
Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

```
def myfunc(n):
  return lambda a : a * n
```

Use that function definition to make a function that always doubles the number you send in:
Example:

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

Use that function definition to make a function that always doubles the number you send in:

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

Or, use the same function definition to make a function that always triples the number you send in:

```
def myfunc(n):
  return lambda a : a * n

mytripler = myfunc(3)

print(mytripler(11))
```

Or, use the same function definition to make both functions, in the same program:

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)
```

```
print(mydoubler(11))
print(mytripler(11))
```

3.4.2.8. Arrays

An array are used to store multiple values in one single variable, is also defined as a collection of items that are stored at contiguous memory locations. It is a container which can hold a fixed number of items, and these items should be of the same type. An array is popular in most programming languages like C/C++, JavaScript, etc.
Array is an idea of storing multiple items of the same type together and it makes easier to calculate the position of each element by simply adding an offset to the base value.

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

car1 = "Lamborghini"
car2 = "Bugatti"

car3 = "Koenigsegg"

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

The array can be handled in Python by a module named array. It is useful when we have to manipulate only specific data values. Following are the terms to understand the concept of an array:

Element: Each item stored in an array is called an element.
Index: The location of an element in an array has a numerical index, which is used to identify the position of the element.

Array Representation

An array can be declared in various ways and different languages. The important points that should be considered are as follows:

- Index starts with 0.
- We can access each element via its index.
- The length of the array defines the capacity to store the elements.

Array operations

Some of the basic operations supported by an array are as follows:

- **Traverse:** It prints all the elements one by one.
- **Insertion:** It adds an element at the given index.
- **Deletion:** It deletes an element at the given index.
- **Search:** It searches an element using the given index or by the value.
- **Update:** It updates an element at the given index.

The Array can be created in Python by importing the **array module** to the python program.

```
from array import *
arrayName = array(typecode, [initializers])
```

Creating Python Arrays

To create an array of numeric values, we need to import the array module.

For example:

```
import array as arr
a = arr.array('d', [1.1, 3.5, 4.5])
print(a)
```

Output

```
array('d', [1.1, 3.5, 4.5])
```

Here, we created an array of float type. The letter d is a type code. This determines the type of the array during creation.

Accessing array elements

We can access the array elements using the respective indices of those elements.

Example1:

```
import array as arr
a = arr.array('i', [2, 4, 6, 8])
print("First element:", a[0])
print("Second element:", a[1])
print("Second last element:", a[-1])
```

Output:

```
First element: 2
Second element: 4
Second last element: 8
```

Explanation: In the above example, we have imported an array, defined a variable named as "a" that holds the elements of an array and print the elements by accessing elements through indices of an array.

Example2: Here, we created an array of integer type. The letter i is a type code. This determines the type of the array during creation.

```
import array as arr
```

```
a = arr.array('i', [2, 4, 5, 6])
print("First element is:", a[0])
print("Second element is:", a[1])
print("Third element is:", a[2])
print("Forth element is:", a[3])
print("last element is:", a[-1])
print("Second last element is:", a[-2])
print("Third last element is:", a[-3])
print("Forth last element is:", a[-4])
print(a[0], a[1], a[2], a[3], a[-1],a[-2],a[-3],a[-4])
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

```
First element is: 2
Second element is: 4
Third element is: 5
Forth element is: 6
last element is: 6
Second last element is: 5
Third last element is: 4
Forth last element is: 2
2 4 5 6 6 5 4 2
```

Explanation: In the above example, we have imported an array, defined a variable named as "a" that holds the elements of an array and print the elements by accessing elements through indices of an array.

Commonly used type codes are listed as follows:

Code	C Type	Python Type	Min bytes
`b`	signed char	int	1
`B`	unsigned char	int	1
`u`	Py_UNICODE	Unicode	2
`h`	signed short	int	2
`H`	unsigned short	int	2

i	signed int	int	2
I	unsigned int	int	2
l	signed long	int	4
L	unsigned long	int	4
f	float	float	4
d	double	float	8

How to change or add elements

Arrays are mutable, and their elements can be changed in a similar way like lists.

```
import array as arr
numbers = arr.array('i', [1, 2, 3, 5, 7, 10])

# changing first element
numbers[0] = 0
print(numbers)  # Output: array('i', [0, 2, 3, 5, 7, 10])

# changing 3rd to 5th element
numbers[2:5] = arr.array('i', [4, 6, 8])
print(numbers)  # Output: array('i', [0, 2, 4, 6, 8, 10])
```

Output:

```
array('i', [0, 2, 3, 5, 7, 10])
array('i' ,[0, 2, 4, 6, 8, 10])
```

Explanation: In the above example, we have imported an array and defined a variable named as "numbers" which holds the value of an array. If we want to change or add the elements in an array, we can do it by defining the particular index of an array where you want to change or add the elements.

How to delete elements from an array?

The elements can be deleted from an array using Python's del statement. If we want to delete any value from the array, we can do that by using the indices of a particular element.

```
import array as arr
number = arr.array('i', [1, 2, 3, 3, 4])
del number[2]                    # removing third element
print(number)                    # Output: array('i', [1, 2, 3, 4])
Output:
```

```
array('i', [10, 20, 40, 60])
```

Explanation: In the above example, we have imported an array and defined a variable named as "number" which stores the values of an array. Here, by using del statement, we are removing the third element [3] of the given array.

Finding the length of an array

The length of an array is defined as the number of elements present in an array. It returns an integer value that is equal to the total number of the elements present in that array.

Syntax

```
len(array_name)
```

Example:

```
arr = [0, 1, 2, 3, 4]
print("The given array is: ")
print (arr)
# Finding length of the given array
size = len(arr)
print ("The length of array is: ")
print (size)
```

Output:

```
The given array is:
[0, 1, 2, 3, 4]
The length of array is: 5
```

Array Concatenation

We can easily concatenate any two arrays using the + symbol.

Example

```
a=arr.array('d',[1.1 , 2.1 ,3.1,2.6,7.8])
b=arr.array('d',[3.7,8.6])
c=arr.array('d')
c=a+b
print("Array c = ",c)
```

Output:

```
Array c= array('d', [1.1, 2.1, 3.1, 2.6, 7.8, 3.7, 8.6])
```

Explanation

In the above example, we have defined variables named as "a, b, c" that hold the values of an array.

Example

```
import array as arr
x = arr.array('i', [4, 7, 19, 22])
print("First element:", x[0])
print("Second element:", x[1])
print("Second last element:", x[-1])
```

Output:

First element: 4

Second element: 7

Second last element: 22

Explanation: In the above example, first, we have imported an array and defined a variable named as "x" which holds the value of an array and then, we have printed the elements using the indices of an array.

Slicing Python Arrays

We can access a range of items in an array by using the slicing operator :.

```
import array as arr

numbers_list = [2, 5, 62, 5, 42, 52, 48, 5]
numbers_array = arr.array('i', numbers_list)

print(numbers_array[2:5]) # 3rd to 5th
print(numbers_array[:-5]) # beginning to 4th
print(numbers_array[5:])  # 6th to end
print(numbers_array[:])   # beginning to end
```

Output

```
array('i', [62, 5, 42])
array('i', [2, 5, 62])
array('i', [52, 48, 5])
array('i', [2, 5, 62, 5, 42, 52, 48, 5])
```

We can use the remove() method to remove the given item, and pop() method to remove an item at the given index.

```
import array as arr

numbers = arr.array('i', [10, 11, 12, 12, 13])

numbers.remove(12)
print(numbers)   # Output: array('i', [10, 11, 12, 13])

print(numbers.pop(2))   # Output: 12
print(numbers)   # Output: array('i', [10, 11, 13])
```

Output

```
array('i', [10, 11, 12, 13])
12
array('i', [10, 11, 13])
```

Python Lists Vs Arrays

In Python, we can treat lists as arrays. However, we cannot constrain the type of elements stored in a list.

For example:

```
# elements of different types
a = [1, 3.5, "Hello"]
```

If you create arrays using the array module, all elements of the array must be of the same numeric type.

```
import array as arr
# Error
a = arr.array('d', [1, 3.5, "Hello"])
```

Output

```
Traceback (most recent call last):
  File "<string>", line 3, in <module>
    a = arr.array('d', [1, 3.5, "Hello"])
TypeError: must be real number, not str
```

3.4.3. Definition of Classes/Objects

Python is a versatile programming language that supports various programming styles, including object-oriented programming (OOP) through the use of **objects** and **classes**.

Python Classes

A class is considered a blueprint of objects. Classes provide a means of bundling data and functionality together.

We can think of the class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows, etc.

Based on these descriptions, we build the house; the house is the object.

Since many houses can be made from the same description, we can create many objects from a class.

Another example to understand the need for creating a class and object in Python let's consider an example, let's say you wanted to track the number of dogs that may have different attributes like breed and age. If a list is used, the first element could be the dog's breed while the second element could represent its age. Let's suppose there are 100 different dogs, then how would you know which element is supposed to be which? What if you wanted to add other properties to these dogs? This lacks organization and it's the exact need for classes.

3.4.3.1. Create class

To create a class, use the keyword class:

```
class ClassName:
    # class definition
```

Example:

Create a class named MyClass, with a property named x:

```
class MyClass:
  x = 5
```

3.4.3.2. Create Object

An object is any entity that has **attributes** and **behaviors**. For example, a parrot is an object. It has
attributes - name, age, color, etc.
behavior - dancing, singing, etc.
Similarly, a class is a blueprint for that object.

Syntax:

```
# Declare an object of a class
object_name = Class_Name(arguments)
```

Example:

Now we can use the class named MyClass to create objects:
Create an object named p1, and print the value of x:

```
p1 = MyClass()
print(p1.x)
```

The __init__() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in __init__() function.

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

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Example

Create a class named Person, use the __init__() function to assign values for name and age:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Output: John
36

Note: The __init__() function is called automatically every time the class is being used to create a new object.

3.4.3.3. Object methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

Let us create a method in the Person class:

Example

Insert a function that prints a greeting, and execute it on the p1 object:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

The self Parameter

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

Modify Object Properties

You can modify properties on objects like this:

Example:

Set the age of p1 to 40:

p1.age = 40

Delete Object Properties

You can delete properties on objects by using the del keyword:
Example
Delete the age property from the p1 object:
del p1.age

Delete Objects
You can delete objects by using the del keyword:
Example
Delete the p1 object:
del p1

The pass Statement

class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error.
Example

```
class Person:
  pass
```

3.4.3.9 Inheritance

Inheritance is a mechanism that allows you to create a hierarchy of classes that share a set of properties and methods by deriving a class from another class. Inheritance is the capability of one class to derive or inherit the properties from another class.

Creating a Parent Class

A parent class is a class whose properties are inherited by the child class. Any class can be a parent class, so the syntax is the same as creating any other class:

Example:

Create a class named Person, with firstname and lastname properties, and a printname method:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)
```

#Use the Person class to create an object, and then execute the printname method:

```
x = Person("John", "Doe")
x.printname()
```

Output: John Doe

Create a Child Class

To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class:

Example

Create a class named Student, which will inherit the properties and methods from the Person class:

```
class Student(Person):
  pass
```

Note: Use the pass keyword when you do not want to add any other properties or methods to the class.

Now the Student class has the same properties and methods as the Person class.

Example

Use the Student class to create an object, and then execute the printname method:

```
x = Student("Mike", "Olsen")
x.printname()
```

Output: John Doe

Mike Olsen

3.4.3.10. Iterators

An iterator is an object that contains a countable number of values.

An iterator is an object that can be iterated upon, meaning that you can traverse through all the values.

Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods __iter__() and __next__().

Iterator vs Iterable

Lists, tuples, dictionaries, and sets are all iterable objects. They are iterable containers which you can get an iterator from.

All these objects have a iter() method which is used to get an iterator:

Example:

Return an iterator from a tuple, and print each value:

```
mytuple = ("apple", "banana", "cherry")

myit = iter(mytuple)


print(next(myit))

print(next(myit))

print(next(myit))
```

Output:

```
apple

banana

cherry
```

Even strings are iterable objects, and can return an iterator:

Example:

Strings are also iterable objects, containing a sequence of characters:

```
mystr = "banana"

myit = iter(mystr)

print(next(myit))

print(next(myit))

print(next(myit))

print(next(myit))

print(next(myit))

print(next(myit))
```

Output:

b

a

n

a

n

a

Looping Through an Iterator

We can also use a **for** loop to iterate through an iterable object:

Example: Iterate the values of a tuple:

```
mytuple = ("apple", "banana", "cherry")
for x in mytuple:
  print(x)
output:
apple
banana
cherry
```

Example: Iterate the characters of a string:

```
mystr = "banana"

for x in mystr:
  print(x)
```

The **for** loop actually creates an iterator object and executes the **next()** method for each loop.

Create an Iterator

To create an object/class as an iterator you have to implement the methods **__iter__()** and **__next__()** to your object.

As you have learned in the Python Classes/Objects chapter, all classes have a function called __init__(), which allows you to do some initializing when the object is being created.

The **__iter__()** method acts similar, you can do operations (initializing etc.), but must always return the iterator object itself.

The **__next__()** method also allows you to do operations, and must return the next item in the sequence.

define a list

https://www.programiz.com/python-programming/iterator

my_list = [4, 7, 0]

```
# create an iterator from the list
iterator = iter(my_list)

# get the first element of the iterator
print(next(iterator))  # prints 4

# get the second element of the iterator
print(next(iterator))  # prints 7

# get the third element of the iterator
print(next(iterator))  # prints 0Stop Iteration
```

The example above would continue forever if you had enough next() statements, or if it was used in a for loop.
To prevent the iteration to go on forever, we can use the StopIteration statement.
In the __next__() method, we can add a terminating condition to raise an error if the iteration is done a specified number of times:
Example: Stop after 20 iterations.

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    if self.a <= 20:
      x = self.a
      self.a += 1
      return x
    else:
      raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

for x in myiter:
  print(x)
```

3.4.4. Definition of other tools

3.4.4.1. Python scope

A variable is only available from inside the region it is created. This is called **scope**.
Local Scope
A variable created inside a function belongs to the local scope of that function, and can only be used inside that function.
Example: A variable created inside a function is available inside that function.

```
def myfunc():
  x = 300
  print(x)

myfunc()
```

Function Inside Function

As explained in the example above, the variable x is not available outside the function, but it is available for any function inside the function.

Example: The local variable can be accessed from a function within the function.

```
def myfunc():
  x = 300
  def myinnerfunc():
    print(x)
  myinnerfunc()

myfunc()
```

Example: A variable created outside of a function is global and can be used by anyone.

```
x = 300

def myfunc():
  print(x)

myfunc()

print(x)
```

Naming Variables

If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function).

Example: The function will print the local x, and then the code will print the global x.

```
x = 300

def myfunc():
  x = 200
  print(x)

myfunc()
```

```
print(x)
```

Global Keyword

If you need to create a global variable, but are stuck in the local scope, you can use the global keyword. The global keyword makes the variable global.

Example: If you use the global keyword, the variable belongs to the global scope:

```
def myfunc():
  global x
  x = 300

myfunc()

print(x)
```

Also, use the global keyword if you want to make a change to a global variable inside a function.

Example: To change the value of a global variable inside a function, refer to the variable by using the global keyword.

```
x = 300

def myfunc():
  global x
  x = 200

myfunc()

print(x)
```

3.4.4.2. Python modules

Consider a module to be the same as a code library. A file containing a set of functions you want to include in your application.

Create a Module

To create a module just save the code you want in a file with the file extension .py:

Example: Save this code in a file named mymodule.py

```
def greeting(name):
  print("Hello, " + name)
```

Use a Module

Now we can use the module we just created, by using the import statement:

Example: Import the module named mymodule, and call the greeting function.

```
import mymodule
mymodule.greeting("Jonathan")
```

Variables in Module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):
Example: Save this code in the file mymodule.py

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example: Import the module named mymodule, and access the person1 dictionary:

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

Naming a Module
You can name the module file whatever you like, but it must have the file extension .py
Re-naming a Module
You can create an alias when you import a module, by using the as keyword:
Example: Create an alias for mymodule called mx.

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

Built-in Modules
There are several built-in modules in Python, which you can import whenever you like.
Example: Import and use the platform module.

```
import platform
x = platform.system()
print(x)
```

Using the dir() Function
There is a built-in function to list all the function names (or variable names) in a module. The dir() function:
Example: List all the defined names belonging to the platform module.

```
import platform
x = dir(platform)
print(x)
```

Import From Module
You can choose to import only parts from a module, by using the from keyword.
Example: The module named mymodule has one function and one dictionary.

```
def greeting(name):
  print("Hello, " + name)
```

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Example: Import only the person1 dictionary from the module.

```
from mymodule import person1
print (person1["age"])
```

3.4.4.3. Python dates

A date in Python is not a data type of its own, but we can import a module named datetime to work with dates as date objects.

Example: Import the datetime module and display the current date.

```
import datetime
x = datetime.datetime.now()
print(x)
```

Date Output

When we execute the code from the example above the result will be:

2023-02-28 10:04:14.243841

The date contains year, month, day, hour, minute, second, and microsecond. The datetime module has many methods to return information about the date object.

Here are a few examples, you will learn more about them later in this chapter:

Example: Return the year and name of weekday.

```
import datetime
x = datetime.datetime.now()
print(x.year)
print(x.strftime("%A"))
```

Creating Date Objects

To create a date, we can use the datetime() class (constructor) of the datetime module.

The datetime() class requires three parameters to create a date: year, month, day.

Example: Create a date object.

```
import datetime
x = datetime.datetime(2020, 5, 17)
print(x)
```

The datetime() class also takes parameters for time and timezone (hour, minute, second, microsecond, tzone), but they are optional, and has a default value of 0, (None for timezone).

The strftime() Method

The datetime object has a method for formatting date objects into readable strings.

The method is called strftime(), and takes one parameter, format, to specify the format of the returned string:

Example: Display the name of the month.

```
import datetime
```

```
x = datetime.datetime(2018, 6, 1)
print(x.strftime("%B"))
```

A reference of all the legal format codes:

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%C	Century	20
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%

%G	ISO 8601 year	2018
%u	ISO 8601 weekday (1-7)	1
%V	ISO 8601 weeknumber (01-53)	01

3.4.4.4. Python JSON

JSON is a syntax for storing and exchanging data. JSON is text, written with JavaScript object notation. Python has a built-in package called **json**, which can be used to work with JSON data.
Example: Import the json module.

```
import json
```

Parse JSON - Convert from JSON to Python
If you have a JSON string, you can parse it by using the json.loads() method. The result will be a Python dictionary.
Example: Convert from JSON to Python.

```
import json
# some JSON:
x =  '{ "name":"John", "age":30, "city":"New York"}'
# parse x:
y = json.loads(x)
# the result is a Python dictionary:
print(y["age"])
```

Convert from Python to JSON
If you have a Python object, you can convert it into a JSON string by using the json.dumps() method.
Example: Convert from Python to JSON.

```
import json
# a Python object (dict):
x = {
  "name": "John",
  "age": 30,
  "city": "New York"
}
# convert into JSON:
y = json.dumps(x)

# the result is a JSON string:
print(y)
```

You can convert Python objects of the following types, into JSON strings:

- dict
- list

- tuple
- string
- int
- float
- True
- False
- None

Example: Convert Python objects into JSON strings, and print the values.

```
import json
print(json.dumps({"name": "John", "age": 30}))
print(json.dumps(["apple", "bananas"]))
print(json.dumps(("apple", "bananas")))
print(json.dumps("hello"))
print(json.dumps(42))
print(json.dumps(31.76))
print(json.dumps(True))
print(json.dumps(False))
print(json.dumps(None))
```

When you convert from Python to JSON, Python objects are converted into the JSON (JavaScript) equivalent:

Python	JSON
dict	Object
list	Array
tuple	Array
str	String
int	Number
float	Number
True	true
False	false
None	null

Example: Convert a Python object containing all the legal data types.

```
import json
x = {
  "name": "John",
  "age": 30,
  "married": True,
  "divorced": False,
  "children": ("Ann","Billy"),
  "pets": None,
  "cars": [
```

```
  {"model": "BMW 230", "mpg": 27.5},
  {"model": "Ford Edge", "mpg": 24.1}
]
}
print(json.dumps(x))
```

Format the Result
The example above prints a JSON string, but it is not very easy to read, with no indentations and line breaks.
The json.dumps() method has parameters to make it easier to read the result:
Example: Use the indent parameter to define the numbers of indents.

```
json.dumps(x, indent=4)
```

You can also define the separators, default value is (", ", ": "), which means using a comma and a space to separate each object, and a colon and a space to separate keys from values:
Example: Use the separators parameter to change the default separator.

```
json.dumps(x, indent=4, separators=(". ", " = "))
```

Order the Result
The json.dumps() method has parameters to order the keys in the result.
Example: Use the sort_keys parameter to specify if the result should be sorted or not:

```
json.dumps(x, indent=4, sort_keys=True)
```

3.4.4.5. Python PIP

What is PIP?
PIP is a package manager for Python packages, or modules if you like.
Note: If you have Python version 3.4 or later, PIP is included by default.
What is a Package?
A package contains all the files you need for a module. Modules are Python code libraries you can include in your project.
Check if PIP is Installed
Navigate your command line to the location of Python's script directory, and type the following.
Example: Check PIP version.

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip --version
```

3.4.4.6. Python try…. Except

The **try** block lets you test a block of code for errors.
The **except** block lets you handle the error.
The **else** block lets you execute code when there is no error.
The **finally** block lets you execute code, regardless of the result of the try- and except blocks.
Exception Handling
When an error occurs, or exception as we call it, Python will normally stop and generate an error message. These exceptions can be handled using the try statement.
Example: The try block will generate an exception, because x is not defined:

```
try:
  print(x)
except:
  print("An exception occurred")
```

Since the try block raises an error, the except block will be executed. Without the try block, the program will crash and raise an error.
Example: This statement will raise an error, because x is not defined.

```
print(x)
```

Many Exceptions
You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error.
Example: Print one message if the try block raises a **NameError** and another for other errors

```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

Else
You can use the else keyword to define a block of code to be executed if no errors were raised.
Example: In this example, the try block does not generate any error.

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

Finally
The finally block, if specified, will be executed regardless if the try block raises an error or not.
Example:

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

This can be useful to close objects and clean up resources.
Example: Try to open and write to a file that is not writable:

```
try:
  f = open("demofile.txt")
  try:
    f.write("Lorum Ipsum")
  except:
```

```
  print("Something went wrong when writing to the file")
 finally:
  f.close()
except:
 print("Something went wrong when opening the file")
```

The program can continue, without leaving the file object open.
Raise an exception
As a Python developer you can choose to throw an exception if a condition occurs. To throw (or raise) an exception, use the raise keyword.
Example: Raise an error and stop the program if x is lower than 0.

```
x = -1
if x < 0:
 raise Exception("Sorry, no numbers below zero")
```

The **raise** keyword is used to raise an exception. You can define what kind of error to raise, and the text to print to the user.
Example: Raise a TypeError if x is not an integer.

```
x = "hello"
if not type(x) is int:
 raise TypeError("Only integers are allowed")
```

3.4.4.7. User input

- Python allows for user input.
- That means we are able to ask the user for input.
- The method is a bit different in Python 3.6 than Python 2.7.
- Python 3.6 uses the **input()** method.
- Python 2.7 uses the **raw_input()** method.

The following example asks for the username, and when you entered the username, it gets printed on the screen:

- Python 3.6

```
username = input("Enter username:")
print("Username is: " + username)
```

- Python 2.7

```
username = raw_input("Enter username:")
print("Username is: " + username)
```

3.4.4.8. Python String Formatting

To make sure a string will display as expected, we can format the result with the format() method.

```
String format()
```

The **format()** method allows you to format selected parts of a string. Sometimes there are parts of a text that you do not control, maybe they come from a database, or user input?
To control such values, add placeholders (curly brackets {}) in the text, and run the values through the format()

method.
Example:
Add a placeholder where you want to display the price:

```
price = 49
txt = "The price is {} dollars"
print(txt.format(price))
```

You can add parameters inside the curly brackets to specify how to convert the value.

Example:
Format the price to be displayed as a number with two decimals:

```
txt = "The price is {:.2f} dollars"
```

Multiple Values
If you want to use more values, just add more values to the format() method:

```
print(txt.format(price, itemno, count))
```

And add more placeholders:

```
quantity = 3
itemno = 567
price = 49
myorder = "I want {} pieces of item number {} for {:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

Index Numbers
You can use index numbers (a number inside the curly brackets {0}) to be sure the values are placed in the correct placeholders.

```
Example:
quantity = 3
itemno = 567
price = 49
myorder = "I want {0} pieces of item number {1} for {2:.2f} dollars."
print(myorder.format(quantity, itemno, price))
```

Also, if you want to refer to the same value more than once, use the index number.
Example:

```
age = 36
name = "John"
txt = "His name is {1}. {1} is {0} years old."
print(txt.format(age, name))
```

Named Indexes
You can also use named indexes by entering a name inside the curly brackets {carname}, but then you must use names when you pass the parameter values txt.format(carname = "Ford").
Example:

```
myorder = "I have a {carname}, it is a {model}."
print(myorder.format(carname = "Ford", model = "Mustang"))
```

LEARNING UNIT 4–DEVELOP PYTHON SCRIPT

Learning Outcome 4.1: Perform File handling

4.1.1. Practice to read file

4.1.1.1. Open file

File handling is an important part of any web application.
A file contains a collection of data and information. Data can be updated and shared through files.
File handling in python has several functions for creating, reading, updating, and deleting files.

File Handling

The key function for working with files in Python is the open() function. The open() function takes two parameters; filename, and mode. There are four different methods (modes) for opening a file:

- "r" - Read - Default value. Opens a file for reading, error if the file does not exist.
- "a" - Append - Opens a file for appending, creates the file if it does not exist.
- "w" - Write - Opens a file for writing, creates the file if it does not exist.
- "x" - Create - Creates the specified file, returns an error if the file exists.

In addition you can specify if the file should be handled as binary or text mode

- "t" - Text - Default value. Text mode
- "b" - Binary - Binary mode (e.g. images)

Syntax
To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

The code above is the same as:

```
f = open("demofile.txt", "rt")
```

Because "r" for read, and "t" for text are the default values, you do not need to specify them.

4.1.1.2. Read file permission

Open a File on the Server
Assume we have the following file, located in the same folder as Python:

```
demofile.txt:
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

To open the file, use the built-in open() function. The open() function returns a file object, which has a read() method for reading the content of the file.
Example:

```
f = open("demofile.txt", "r")
print(f.read())
```

If the file is located in a different location, you will have to specify the file path, like this.
Example: Open a file on a different location.

```
f = open("D:\\myfiles\welcome.txt", "r")
```

```
print(f.read())
```

Read Only Parts of the File

By default the read() method returns the whole text, but you can also specify how many characters you want to return.
Example: Return the 5 first characters of the file.

```
f = open("demofile.txt", "r")
print(f.read(5))
```

Read Lines

You can return one line by using the readline() method:
Example
Read one line of the file:

```
f = open("demofile.txt", "r")
print(f.readline())
```

Run Example »
By calling readline() two times, you can read the two first lines:
Example
Read two lines of the file:

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

4.1.2. Practice to write/create file

4.1.2.1. Create a new file

Create a New File
To create a new file in Python, use the open() method, with one of the following parameters:

- "x" - Create - will create a file, returns an error if the file exist.
- "a" - Append - will create a file if the specified file does not exist.
- "w" - Write - will create a file if the specified file does not exist.

Example: Create a file called "myfile.txt".

```
f = open("myfile.txt", "x")
```

Result: a new empty file is created!
Example:
Create a new file if it does not exist:

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

4.1.2.2. Write to existing file

To write to an existing file, you must add a parameter to the open() function:

- "a" - Append - will append to the end of the file
- "w" - Write - will overwrite any existing content

Example: Open the file "demofile2.txt" and append content to the file.

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
```

```
f.close()
#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Example: Open the file "demofile3.txt" and overwrite the content.

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the overwriting:
f = open("demofile3.txt", "r")
print(f.read())
```

4.1.3. Practice to delete file

4.1.3.1. Remove file

To delete a file, you must import the OS module, and run its os.remove() function.
Example: Remove the file "demofile.txt".

```
import os
os.remove("demofile.txt")
```

4.1.3.2. Check if file exists

To avoid getting an error, you might want to check if the file exists before you try to delete it.
Example: Check if file exists, then delete it:

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

4.1.3.3. Delete folder

To delete an entire folder, use the os.rmdir() method.
Example: Remove the folder "myfolder".

```
import os
os.rmdir("myfolder")
```

Learning Outcome 4.2: Determine Python library

4.2.1. Identification of Libraries

4.2.1.1. Using Numpy

What is NumPy?
NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open-source project and you can use it freely. NumPy stands for Numerical Python.

Why Use NumPy?

In Python we have lists that serve the purpose of arrays, but they are slow to process.
NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
The array object in NumPy is called **ndarray**, it provides a lot of supporting functions that make working with **ndarray** very easy. Arrays are very frequently used in data science, where speed and resources are very important.
Data Science: is a branch of computer science where we study how to store, use and analyze data for deriving information from it.
Why is NumPy Faster Than Lists?
NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently. This behavior is called locality of reference in computer science. This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.
Which Language is NumPy written in?
NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.
Installation of NumPy
If you have Python and PIP already installed on a system, then installation of NumPy is very easy.
Install it using this command:

```
C:\Users\Your Name>pip install numpy
```

If this command fails, then use a python distribution that already has NumPy installed like, Anaconda, Spyder etc.
Import NumPy
Once NumPy is installed, import it in your applications by adding the import keyword:

```
import numpy
```

Now NumPy is imported and ready to use.
Example:

```
import numpy
arr = numpy.array([1, 2, 3, 4, 5])
print(arr)
```

NumPy as np
NumPy is usually imported under the np alias.
alias: In Python alias are an alternate name for referring to the same thing.
Create an alias with the as keyword while importing:

```
import numpy as np
```

Now the NumPy package can be referred to as np instead of numpy.
Example:

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

Checking NumPy Version
The version string is stored under __version__ attribute.
Example:

```
import numpy as np
print(np.__version__)
```

4.2.1.2. Pandas

What is Pandas?
Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and

manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas?
Pandas allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science.
Data Science: is a branch of computer science where we study how to store, use and analyze data for deriving information from it.
What Can Pandas Do?
Pandas gives you answers about the data. Like:

- Is there a correlation between two or more columns?
- What is average value?
- Max value?
- Min value?

Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

4.2.1.3. MultplotLib

- Matplotlib is a low level graph plotting library in python that serves as a visualization utility.
- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.
- Create publication quality plots.
- Make interactive figures that can zoom, pan, update.
- Customize visual style and layout.
-
- Matplotlib was created by John D. Hunter.
- Matplotlib is open source and we can use it freely.
- Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

Installation of Matplotlib
If you have Python and PIP already installed on a system, then installation of Matplotlib is very easy.
Install it using this command:

```
C:\Users\Your Name>pip install matplotlib
```

If this command fails, then use a python distribution that already has Matplotlib installed, like Anaconda, Spyder etc.
Import Matplotlib
Once Matplotlib is installed, import it in your applications by adding the import module statement:

```
import matplotlib
```

Now Matplotlib is imported and ready to use:
Checking Matplotlib Version
The version string is stored under __version__ attribute.
Example:

```
import matplotlib
print(matplotlib.__version__)
```

Note: two underscore characters are used in __version__.

4.2.1.4. SciPy

What is SciPy?

SciPy is an open-source Python library which is used to solve scientific and mathematical problems. is also a Python library useful for solving many mathematical equations and algorithms It is built on the NumPy extension and allows the user to manipulate and visualize data with a wide range of high-level commands. As mentioned earlier, SciPy builds on NumPy and therefore if you import SciPy, there is no need to import NumPy.

NumPy vs SciPy

Both NumPy and SciPy are Python libraries used for used mathematical and numerical analysis. NumPy contains array data and basic operations such as sorting, indexing, etc whereas, SciPy consists of all the numerical code. Though NumPy provides a number of functions that can help resolve linear algebra, Fourier transforms, etc . SciPy was created by NumPy's creator Travis Olliphant.

Why Use SciPy?

If SciPy uses NumPy underneath, why can we not just use NumPy? SciPy has optimized and added functions that are frequently used in NumPy and Data Science.

Which Language is SciPy Written in?

SciPy is predominantly written in Python, but a few segments are written in C.

Installation of SciPy

If you have Python and PIP already installed on a system, then installation of SciPy is very easy.

Install it using this command:

```
C:\Users\Your Name>pip install scipy
```

If this command fails, then use a Python distribution that already has SciPy installed like, Anaconda, Spyder etc.

Import SciPy

Once SciPy is installed, import the SciPy module(s) you want to use in your applications by adding the from scipy import module statement:

```
from scipy import constants
```

Now we have imported the constants module from SciPy, and the application is ready to use it:

Example:

How many cubic meters are in one liter:

```
from scipy import constants

print(constants.liter)
```

constants: SciPy offers a set of mathematical constants, one of them is liter which returns 1 liter as cubic meters. You will learn more about constants in the next chapter.

Checking SciPy Version

The version string is stored under the __version__ attribute.

Example:

```
import scipy

print(scipy.__version__)
```

Note: two underscore characters are used in __version__.

Learning Outcome 4.3: Interact with database

4.3.1. Python MySQL commands

4.3.1.1. Install driver

Python needs a MySQL driver to access the MySQL database.

In this section, we will use the driver "MySQL Connector". We recommend that you use PIP to install "MySQL Connector". PIP is most likely already installed in your Python environment. Navigate your command line to the

location of PIP, and type the following:
Download and install "MySQL Connector":

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>python -m pip install mysql-connector-python
```

Now you have downloaded and installed a MySQL driver.

4.3.1.2. Test MySQL connector

To test if the installation was successful, or if you already have "MySQL Connector" installed, create a Python page with the following content:
demo_mysql_test.py:

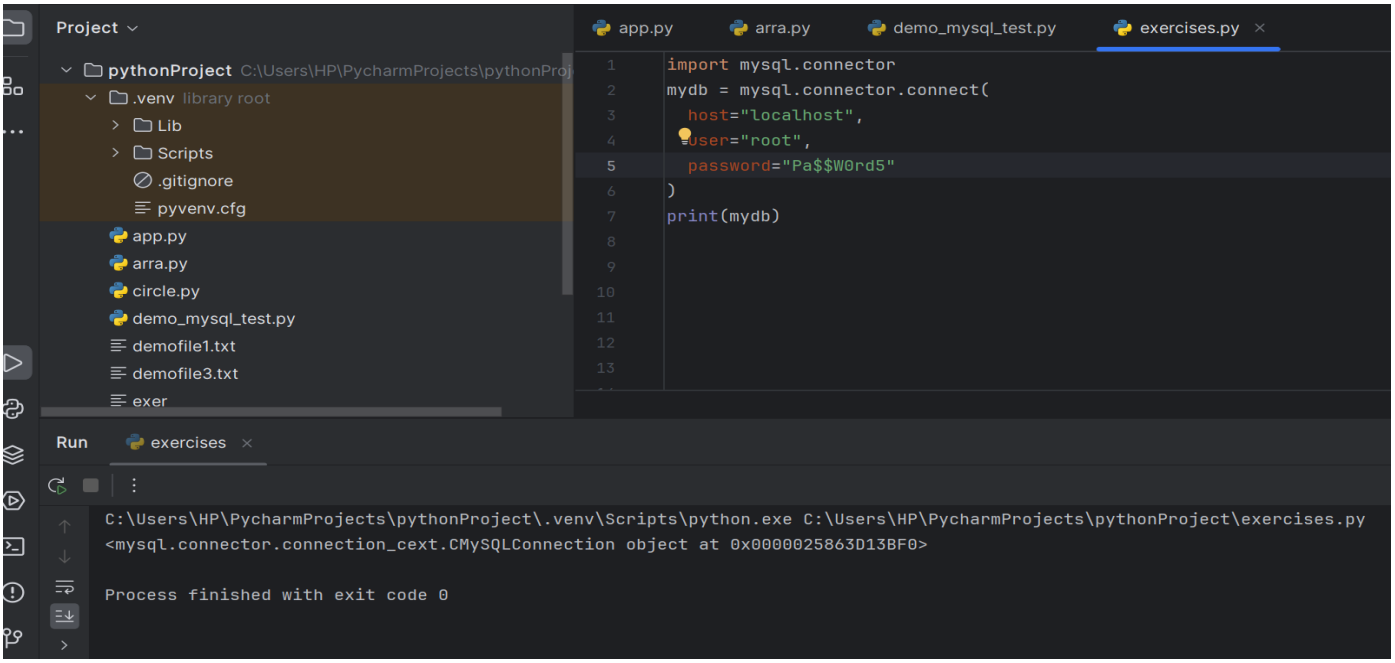
```
import mysql.connector
```

If the above code was executed with no errors, "MySQL Connector" is installed and ready to be used.

4.3.1.3. Create connection

Start by creating a connection to the database. Use the username and password from your MySQL database:
demo_mysql_connection.py:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)
print(mydb)
```



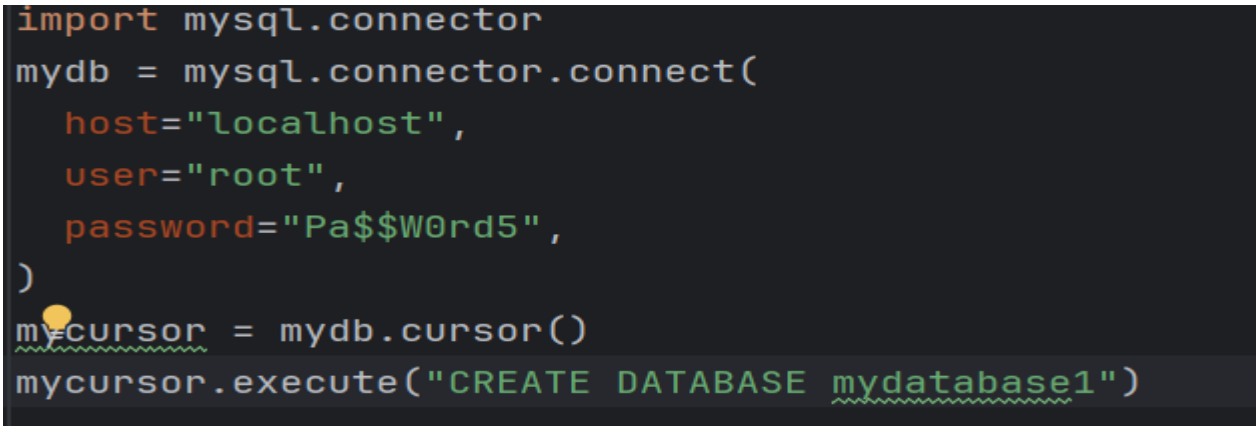
Now you can start querying the database using SQL statements.

4.3.1.4. Create database

To create a database in MySQL, use the "CREATE DATABASE" statement.
Example: create a database named "mydatabase1".

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)
mycursor = mydb.cursor()
mycursor.execute("CREATE DATABASE mydatabase1")
```

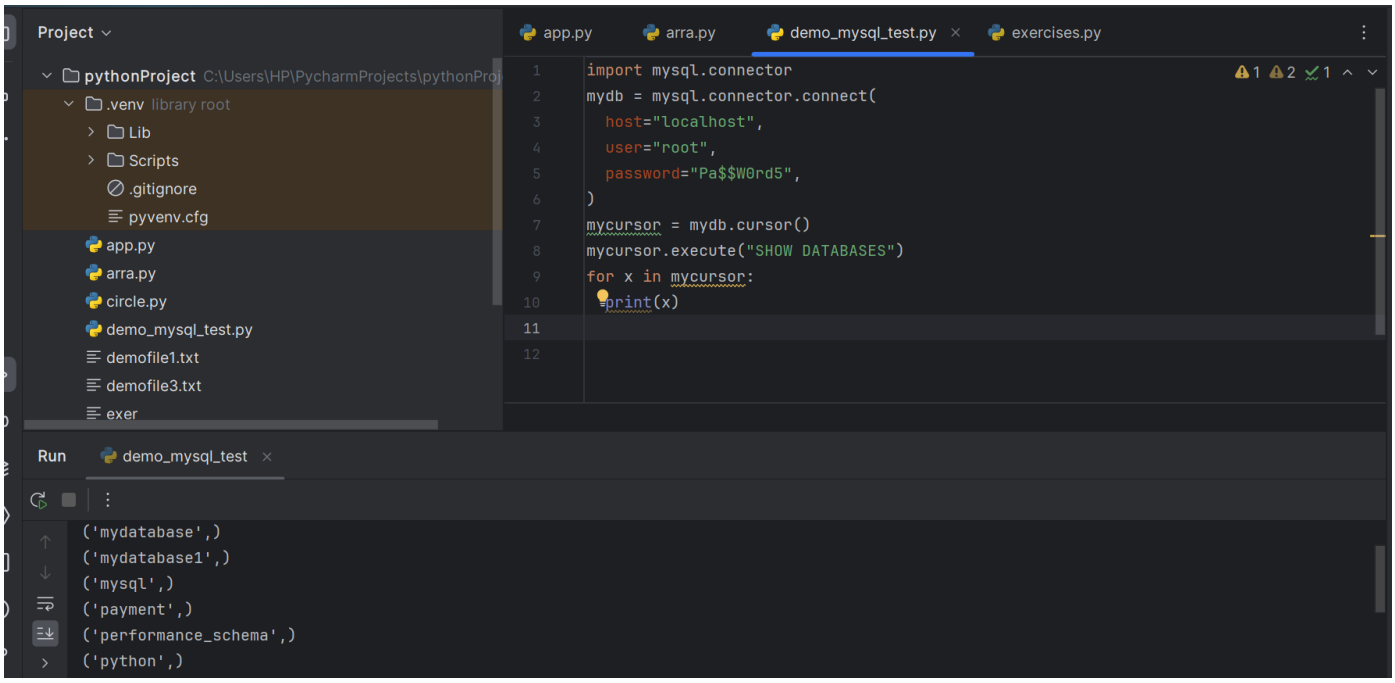


If the above code was executed with no errors, you have successfully created a database.

Check if Database Exists

You can check if a database exist by listing all databases in your system by using the "SHOW DATABASES" statement:
Example: Return a list of your system's databases.

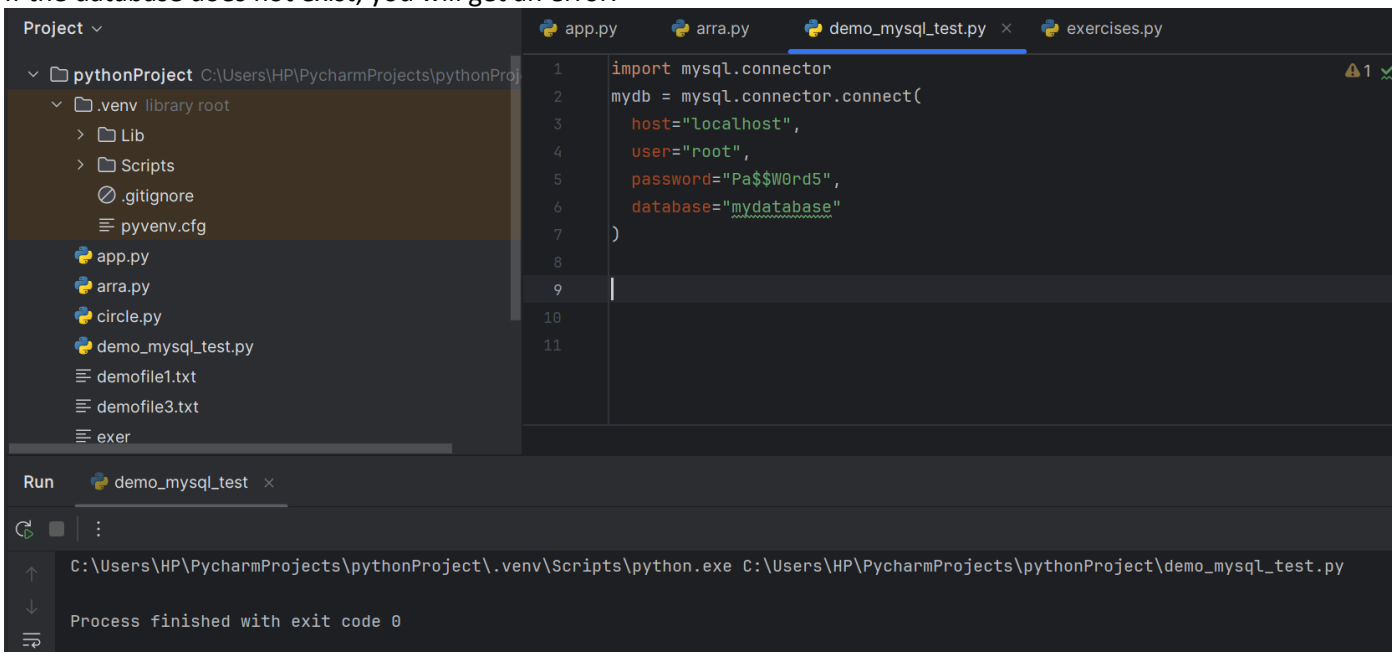
```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)
mycursor = mydb.cursor()
mycursor.execute("SHOW DATABASES")
for x in mycursor:
  print(x)
```



Or you can try to access the database when making the connection:
Example: Try connecting to the database "mydatabase".

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
```

If the database does not exist, you will get an error.



4.3.1.5. Create table

For creating tables we will follow the similar approach of writing the SQL commands as strings and then passing it to the execute() method of the cursor object. SQL command for creating a table is –

CREATE TABLE

(

column_name_1 column_Data_type,

column_name_2 column_Data_type,

:

:

column_name_n column_Data_type

);

Example: Creating MySQL table using Python

Example1:

Example: Create a table named "customers".

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))")
```

If the above code was executed with no errors, you have now successfully created a table.

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Pa$$W0rd5",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))")
```

Output:

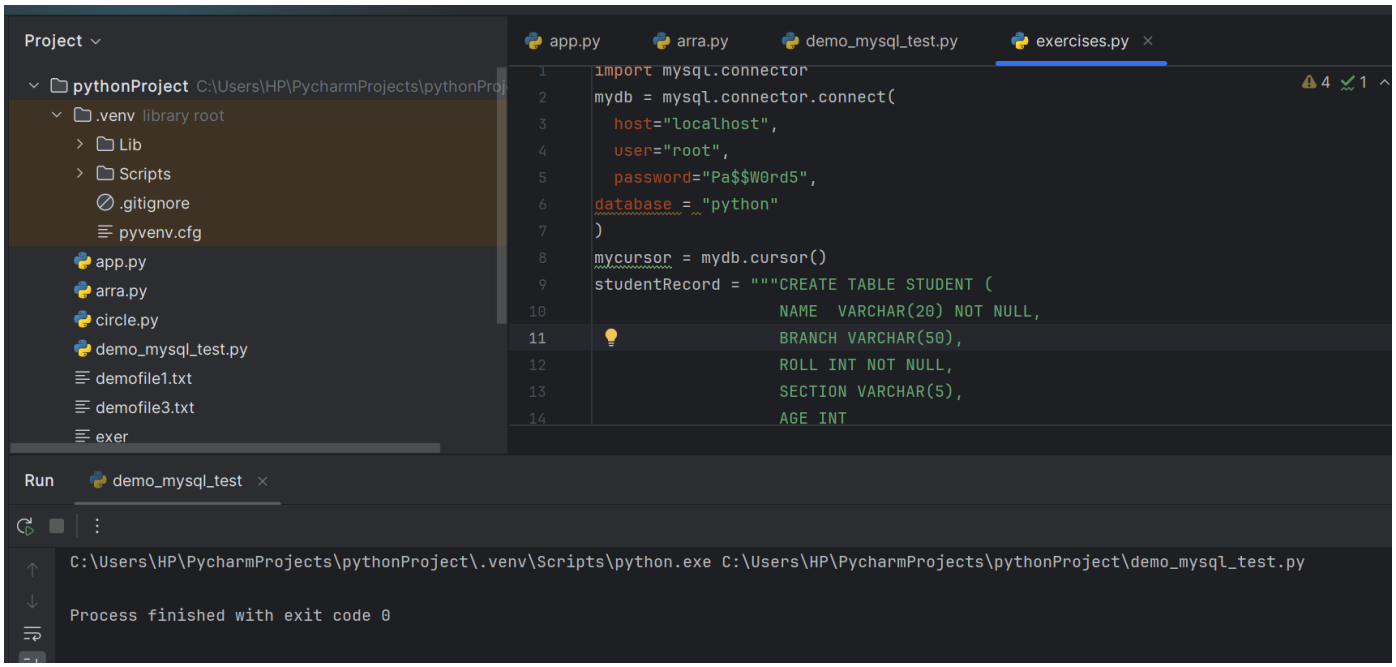
```
mysql> use mydatabase
Database changed
mysql> show tables;
+----------------------+
| Tables_in_mydatabase |
+----------------------+
| customers            |
+----------------------+
1 row in set (0.00 sec)
```

```
mysql> desc customers;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| name    | varchar(255) | YES  |     | NULL    |       |
| address | varchar(255) | YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Example2: Create a table named "STUDENT".

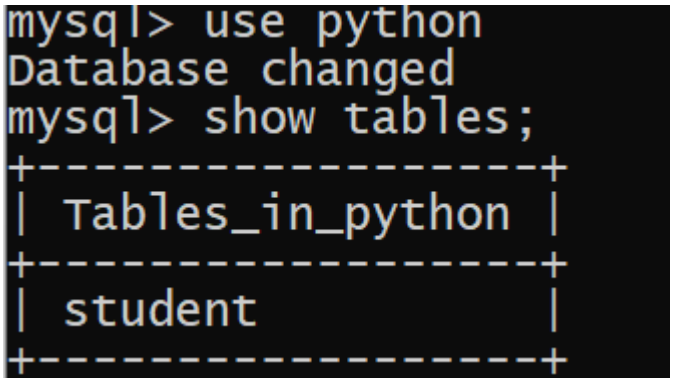
```
# importing required libraries
import mysql.connector

dataBase = mysql.connector.connect(
host ="localhost",
user ="user",
passwd ="password",
database = "student"
)

# preparing a cursor object
cursorObject = dataBase.cursor()

# creating table
studentRecord = """CREATE TABLE STUDENT (
                                    NAME VARCHAR(20) NOT NULL,
                                    BRANCH VARCHAR(50),
                                    ROLL INT NOT NULL,
                                    SECTION VARCHAR(5),
                                    AGE INT
                                    )"""

# table created
cursorObject.execute(studentRecord)

# disconnecting from server
```

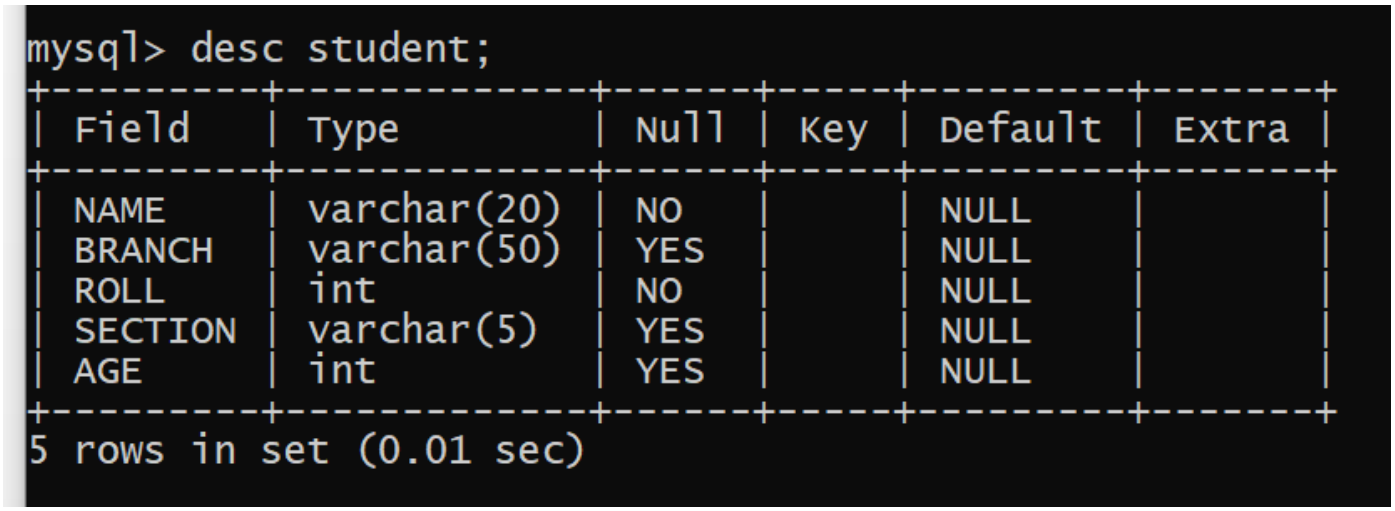
dataBase.close()



If the above code was executed with no errors, you have now successfully created a table.

Output:





Check if Table Exists

You can check if a table exist by listing all tables in your database with the "SHOW TABLES" statement.

Example: Return a list of your system's databases.

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("SHOW TABLES")
for x in mycursor:
  print(x)
```

Primary Key

When creating a table, you should also create a column with a unique key for each record. This can be done by defining a PRIMARY KEY.

We use the statement "INT AUTO_INCREMENT PRIMARY KEY" which will insert a unique number for each record. Starting at 1 and increased by one for each record.

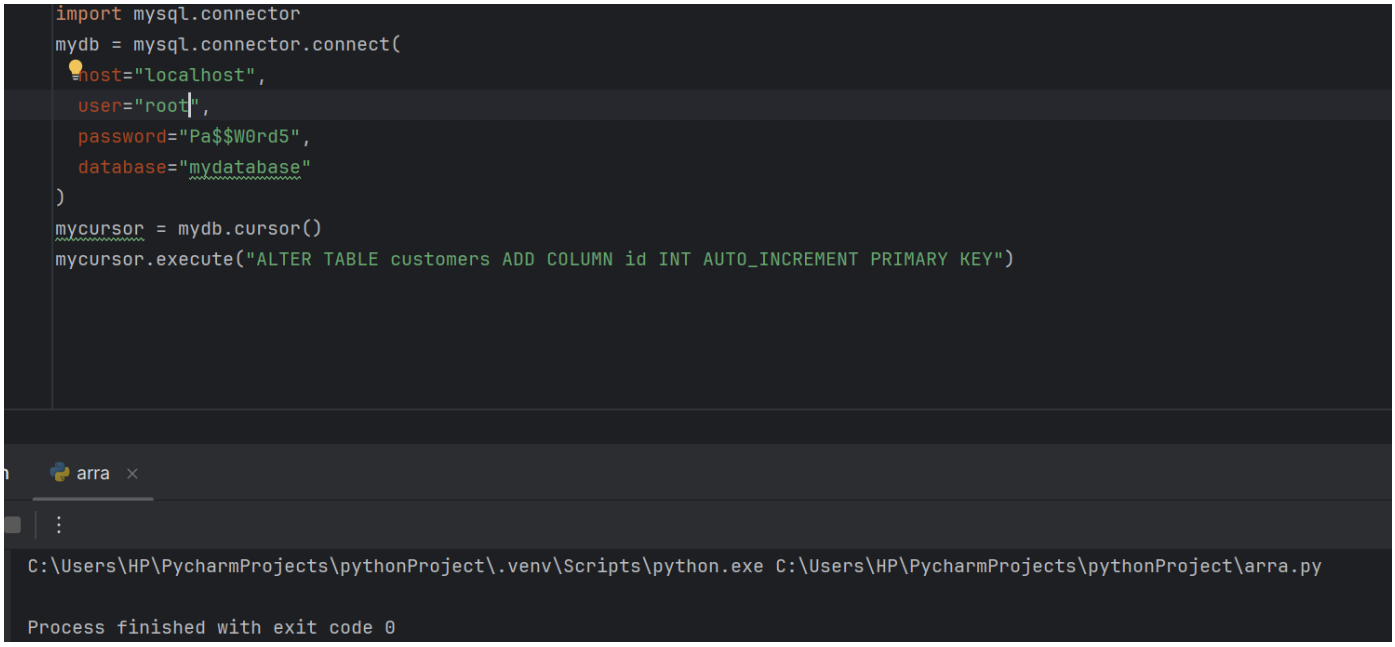
Example: Create primary key when creating the table:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
```

```
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("CREATE TABLE customers (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), address VARCHAR(255))")
```

If the table already exists, use the ALTER TABLE keyword:
Example: Create primary key on an existing table:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("ALTER TABLE customers ADD COLUMN id INT AUTO_INCREMENT PRIMARY KEY")
```



4.3.1.5. Insert

To fill a table in MySQL, use the "INSERT INTO" statement.
Example: Insert a record in the "customers" table:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
```

```
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("John", "Highway 21")
mycursor.execute(sql, val)
mydb.commit()
print(mycursor.rowcount, "record inserted.")
```

Important! Notice the statement: mydb.commit(). It is required to make the changes, otherwise no changes are made to the table.

Output:

```
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="Pa$$W0rd5",
  database="mydatabase"
)
mycursor = mydb.cursor()
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("John", "Highway 21")
mycursor.execute(sql, val)
mydb.commit()
print(mycursor.rowcount, "record inserted.")
```

```
Run    arra
C:\Users\HP\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:
1 record inserted.

Process finished with exit code 0
```

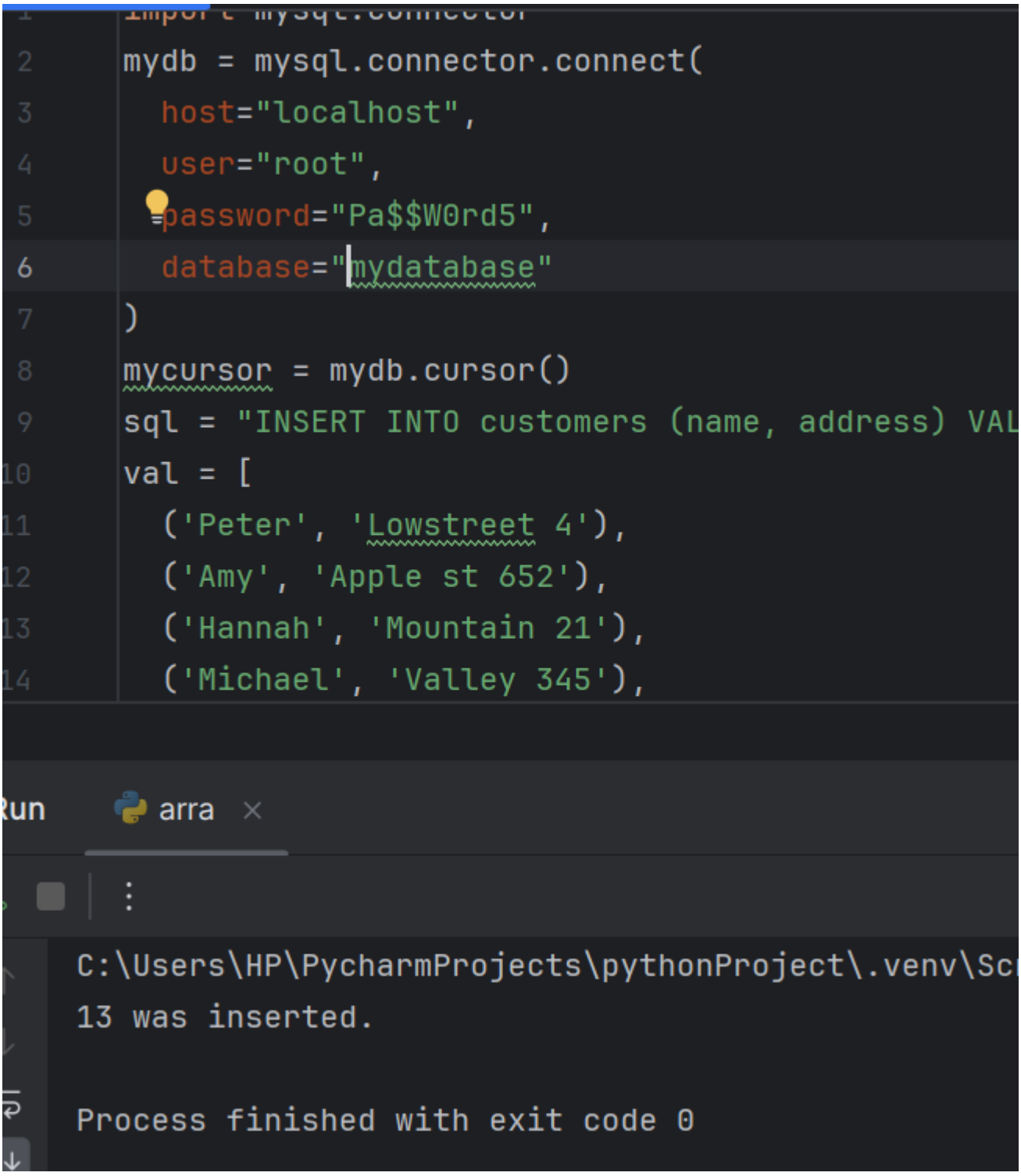
```
mysql> select *from customers;
+------+-------------+----+
| name | address     | id |
+------+-------------+----+
| John | Highway 21 |  1 |
```

Insert Multiple Rows

To insert multiple rows into a table, use the executemany() method. The second parameter of the executemany() method is a list of tuples, containing the data you want to insert:

Example: Fill the "customers" table with data.

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = [
  ('Peter', 'Lowstreet 4'),
  ('Amy', 'Apple st 652'),
  ('Hannah', 'Mountain 21'),
  ('Michael', 'Valley 345'),
  ('Sandy', 'Ocean blvd 2'),
  ('Betty', 'Green Grass 1'),
  ('Richard', 'Sky st 331'),
  ('Susan', 'One way 98'),
  ('Vicky', 'Yellow Garden 2'),
  ('Ben', 'Park Lane 38'),
  ('William', 'Central st 954'),
  ('Chuck', 'Main Road 989'),
  ('Viola', 'Sideway 1633')
]
mycursor.executemany(sql, val)
mydb.commit()
print(mycursor.rowcount, "was inserted.")
```



Get Inserted ID

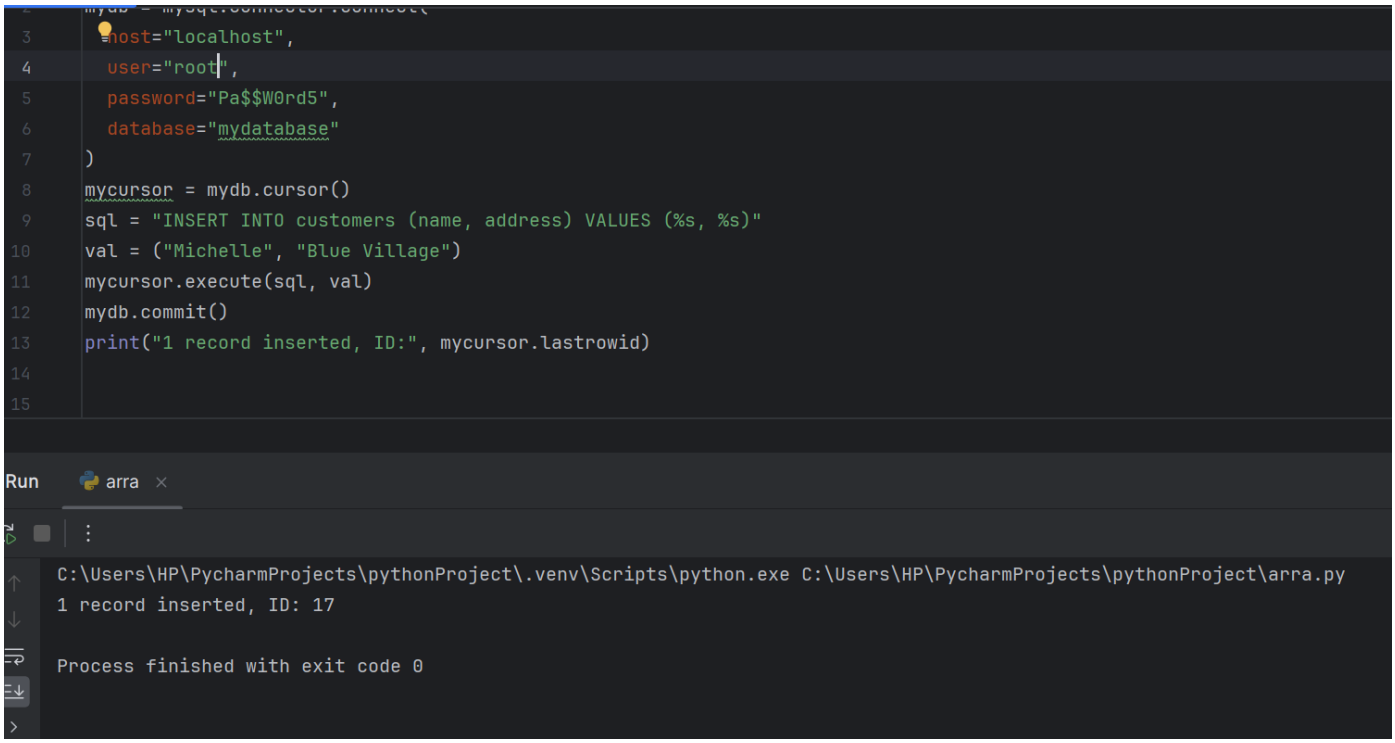
You can get the id of the row you just inserted by asking the cursor object.

Note: If you insert more than one row, the id of the last inserted row is returned.

Example: Insert one row, and return the ID.

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
```

```
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("Michelle", "Blue Village")
mycursor.execute(sql, val)
mydb.commit()
print("1 record inserted, ID:", mycursor.lastrowid)
```

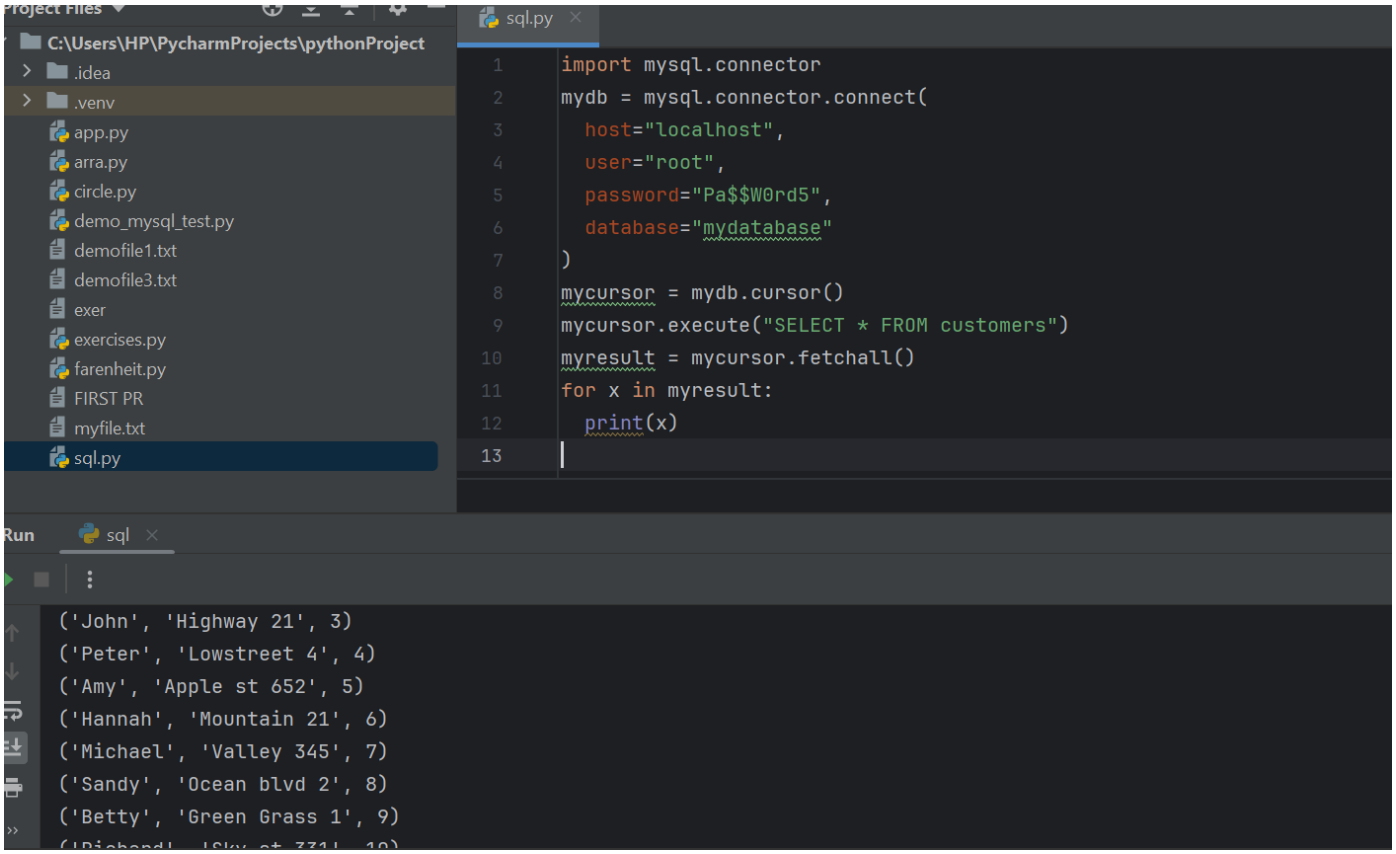


4.3.1.6. Select

To select from a table in MySQL, use the "SELECT" statement.
Example: Select all records from the "customers" table, and display the result.

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM customers")
```

```
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```



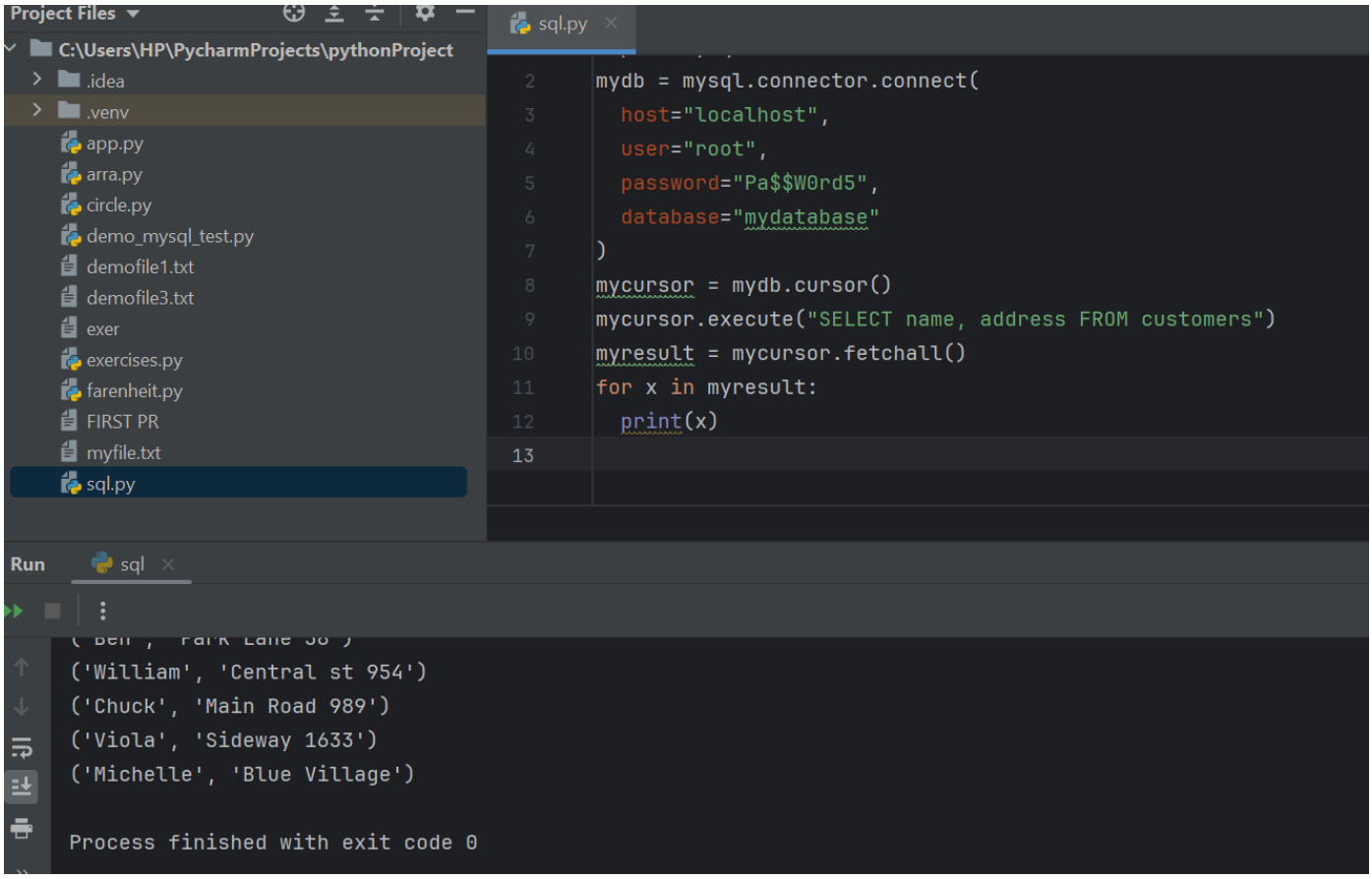
Note: We use the fetchall() method, which fetches all rows from the last executed statement.

Selecting Columns

To select only some of the columns in a table, use the "SELECT" statement followed by the column name(s):

Example: Select only the name and address columns.

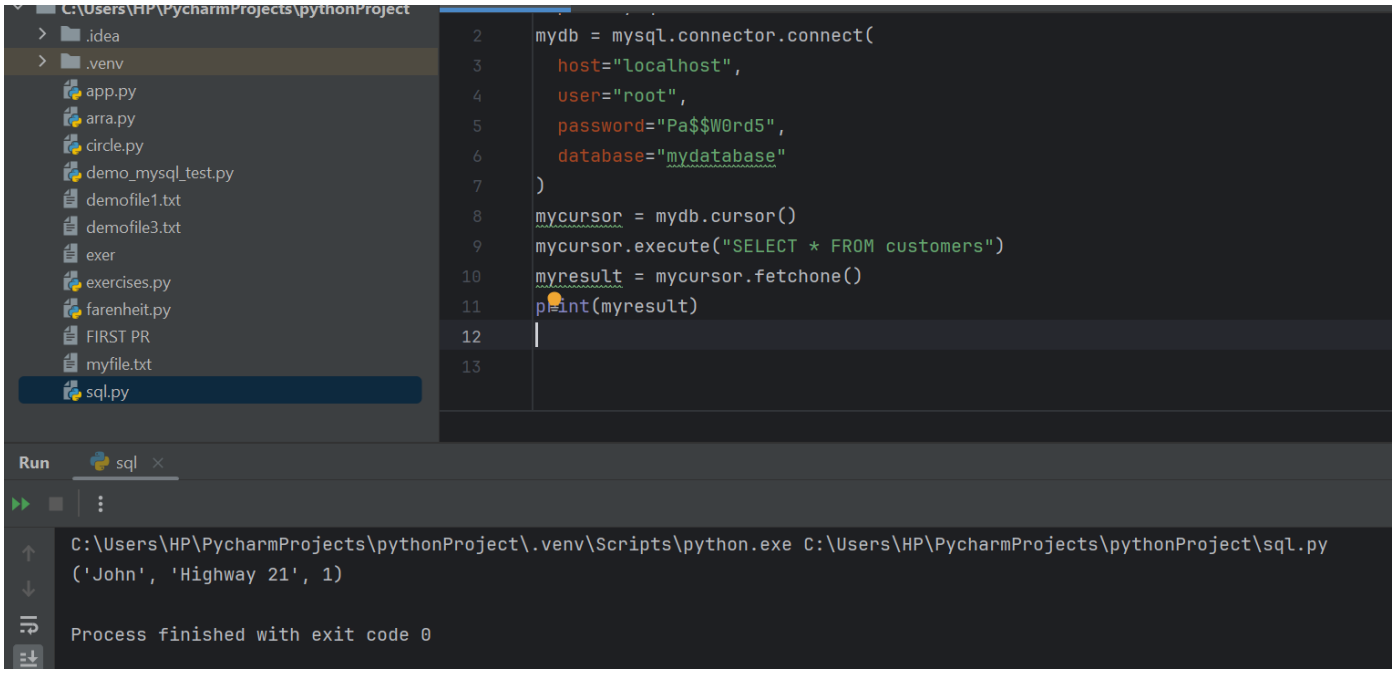
```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("SELECT name, address FROM customers")
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```



Using the fetchone() Method

If you are only interested in one row, you can use the fetchone() method. The fetchone() method will return the first row of the result.
Example: Fetch only one row.

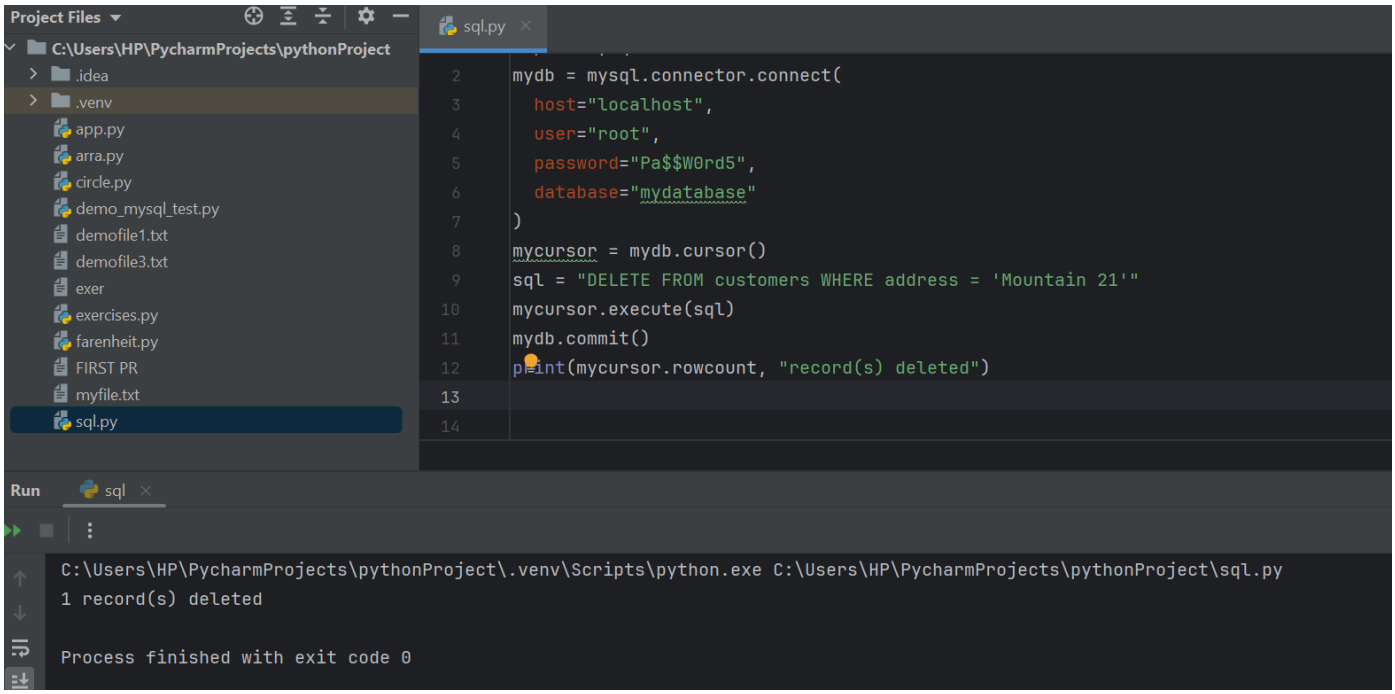
```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM customers")
myresult = mycursor.fetchone()
print(myresult)
```



4.3.1.7. Delete

You can delete records from an existing table by using the "DELETE FROM" statement.
Example: Delete any record where the address is "Mountain 21".

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
sql = "DELETE FROM customers WHERE address = 'Mountain 21'"
mycursor.execute(sql)
mydb.commit()
print(mycursor.rowcount, "record(s) deleted")
```


Project Files
C:\Users\HP\PycharmProjects\pythonProject
.idea
.venv
app.py
arra.py
circle.py
demo_mysql_test.py
demofile1.txt
demofile3.txt
exer
exercises.py
farenheit.py
FIRST PR
myfile.txt
sql.py
sql.py
mydb = mysql.connector.connect(
host="localhost",
user="root",
password="Pa$$W0rd5",
database="mydatabase"
)
mycursor = mydb.cursor()
sql = "DELETE FROM customers WHERE address = 'Mountain 21'"
mycursor.execute(sql)
mydb.commit()
print(mycursor.rowcount, "record(s) deleted")
Run
sql
C:\Users\HP\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\HP\PycharmProjects\pythonProject\sql.py
1 record(s) deleted
Process finished with exit code 0

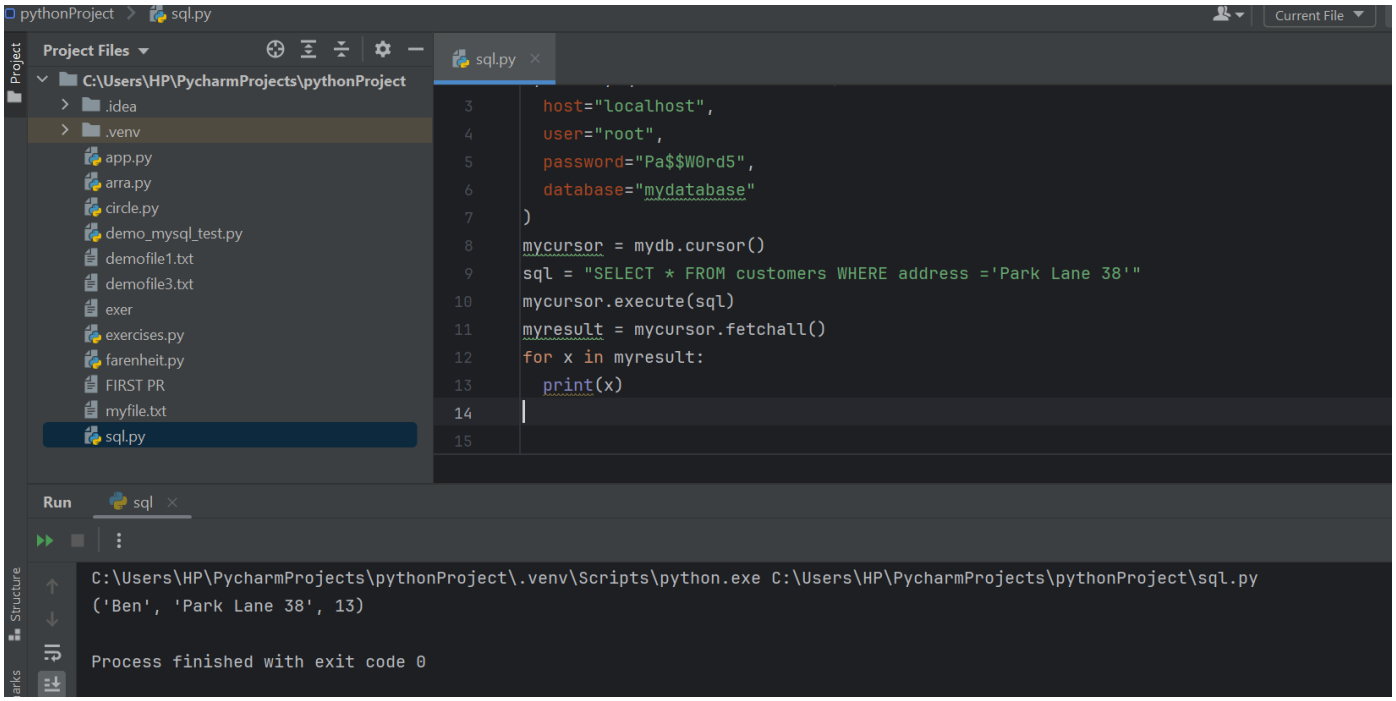
```
mysql> use mydatabase
Database changed
mysql> show tables;
+----------------------+
| Tables_in_mydatabase |
+----------------------+
| customers            |
+----------------------+
1 row in set (0.21 sec)

mysql> select *from customers;
+----------+------------------+----+
| name     | address          | id |
+----------+------------------+----+
| John     | Highway 21       |  1 |
| John     | Highway 21       |  2 |
| John     | Highway 21       |  3 |
| Peter    | Lowstreet 4      |  4 |
| Amy      | Apple st 652     |  5 |
| Michael  | Valley 345       |  7 |
| Sandy    | Ocean blvd 2     |  8 |
| Betty    | Green Grass 1    |  9 |
| Richard  | Sky st 331       | 10 |
| Susan    | One way 98       | 11 |
| Vicky    | Yellow Garden 2  | 12 |
| Ben      | Park Lane 38     | 13 |
| William  | Central st 954   | 14 |
| Chuck    | Main Road 989    | 15 |
| Viola    | Sideway 1633     | 16 |
| Michelle | Blue Village     | 17 |
+----------+------------------+----+
16 rows in set (0.00 sec)
```

4.3.1.8. Where condition

When selecting records from a table, you can filter the selection by using the "WHERE" statement.
Example: Select record(s) where the address is "Park Lane 38": result.

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
```

```
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
sql = "SELECT * FROM customers WHERE address ='Park Lane 38'"
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

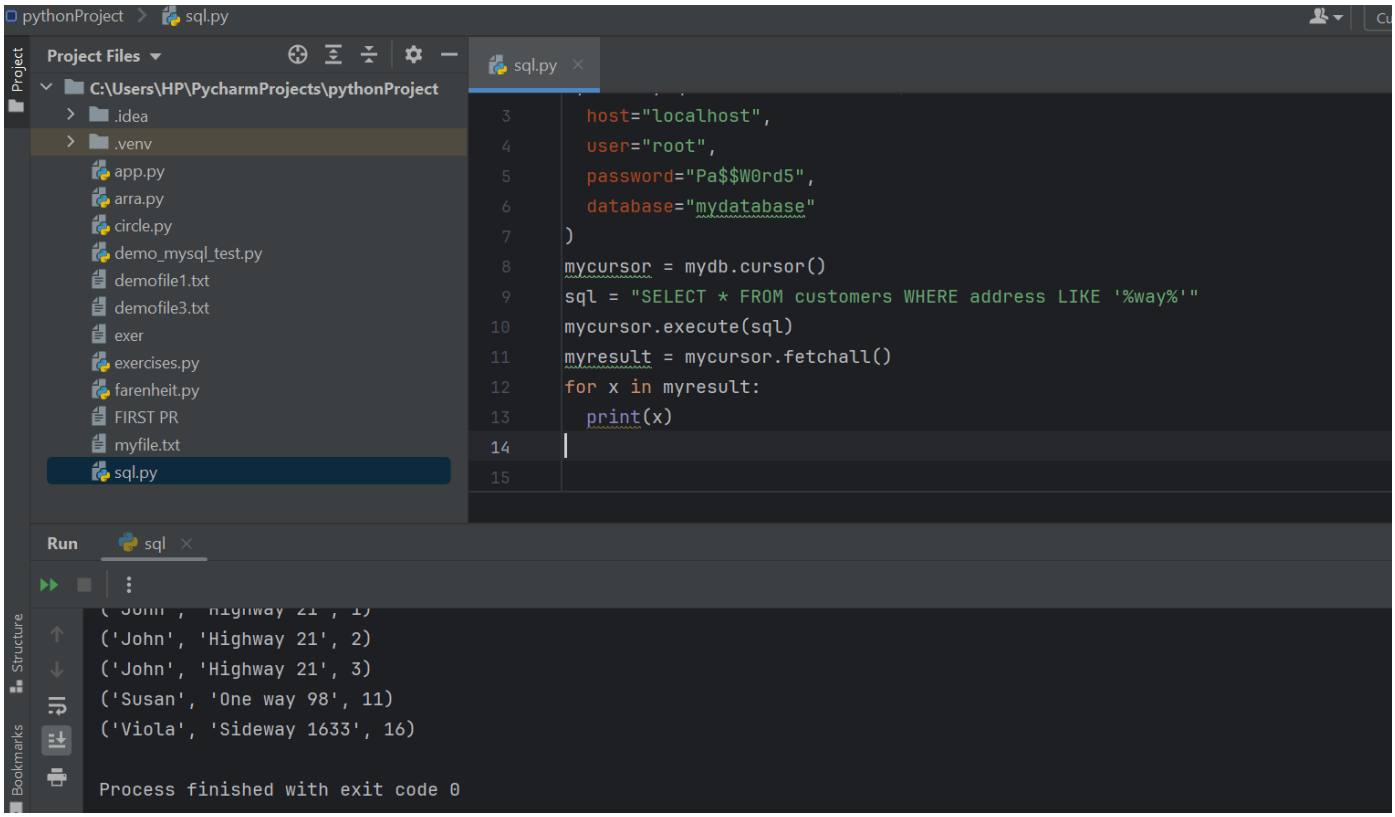


Wildcard Characters

You can also select the records that starts, includes, or ends with a given letter or phrase. Use the % to represent wildcard characters:
Example: Select records where the address contains the word "way":

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
```

```
sql = "SELECT * FROM customers WHERE address LIKE '%way%'"
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

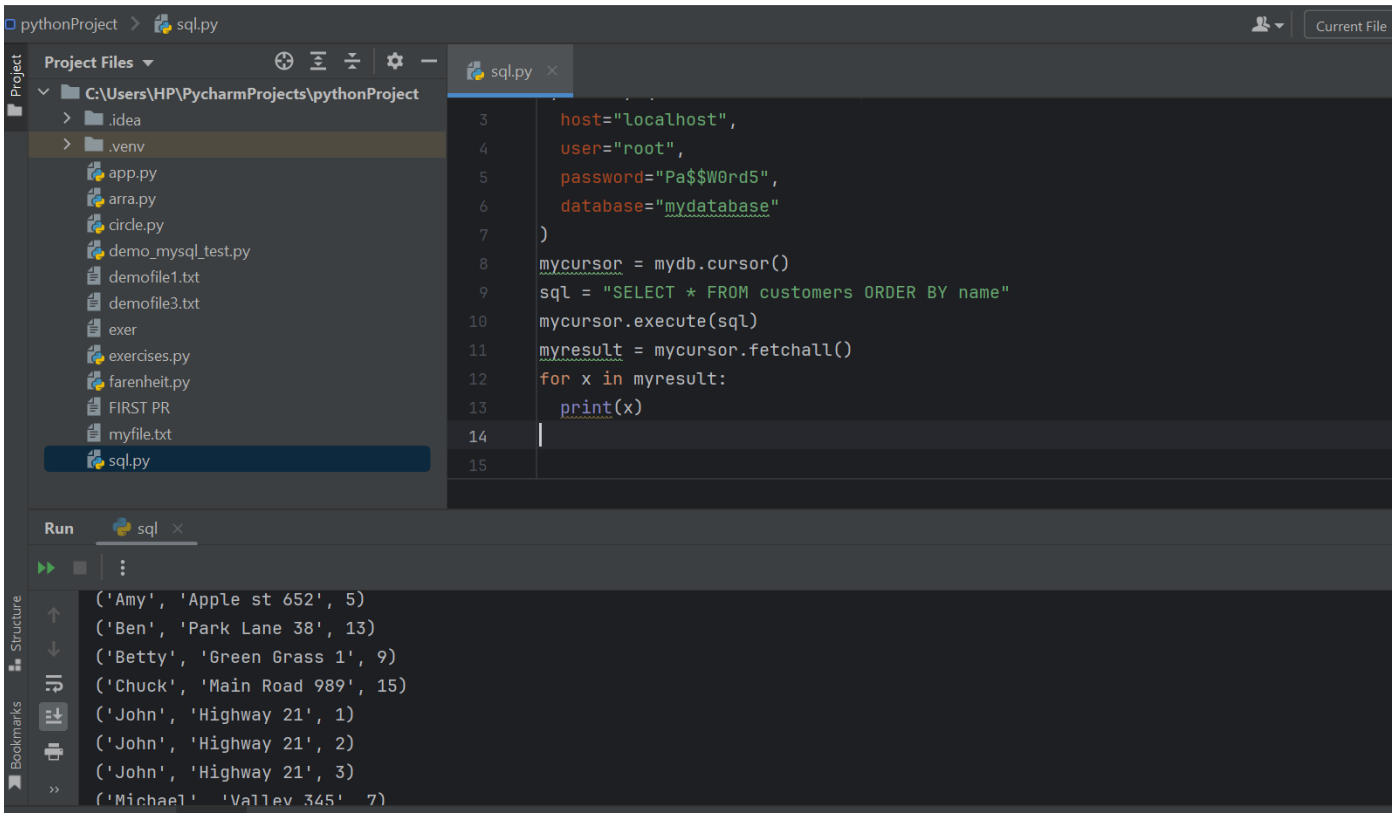


4.3.1.9. Order by

Use the ORDER BY statement to sort the result in ascending or descending order. The ORDER BY keyword sorts the result ascending by default. To sort the result in descending (DESC) order, use the DESC keyword.
Example: Sort the result alphabetically by name: result.

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
sql = "SELECT * FROM customers ORDER BY name"
mycursor.execute(sql)
myresult = mycursor.fetchall()
```

```
for x in myresult:
  print(x)
```

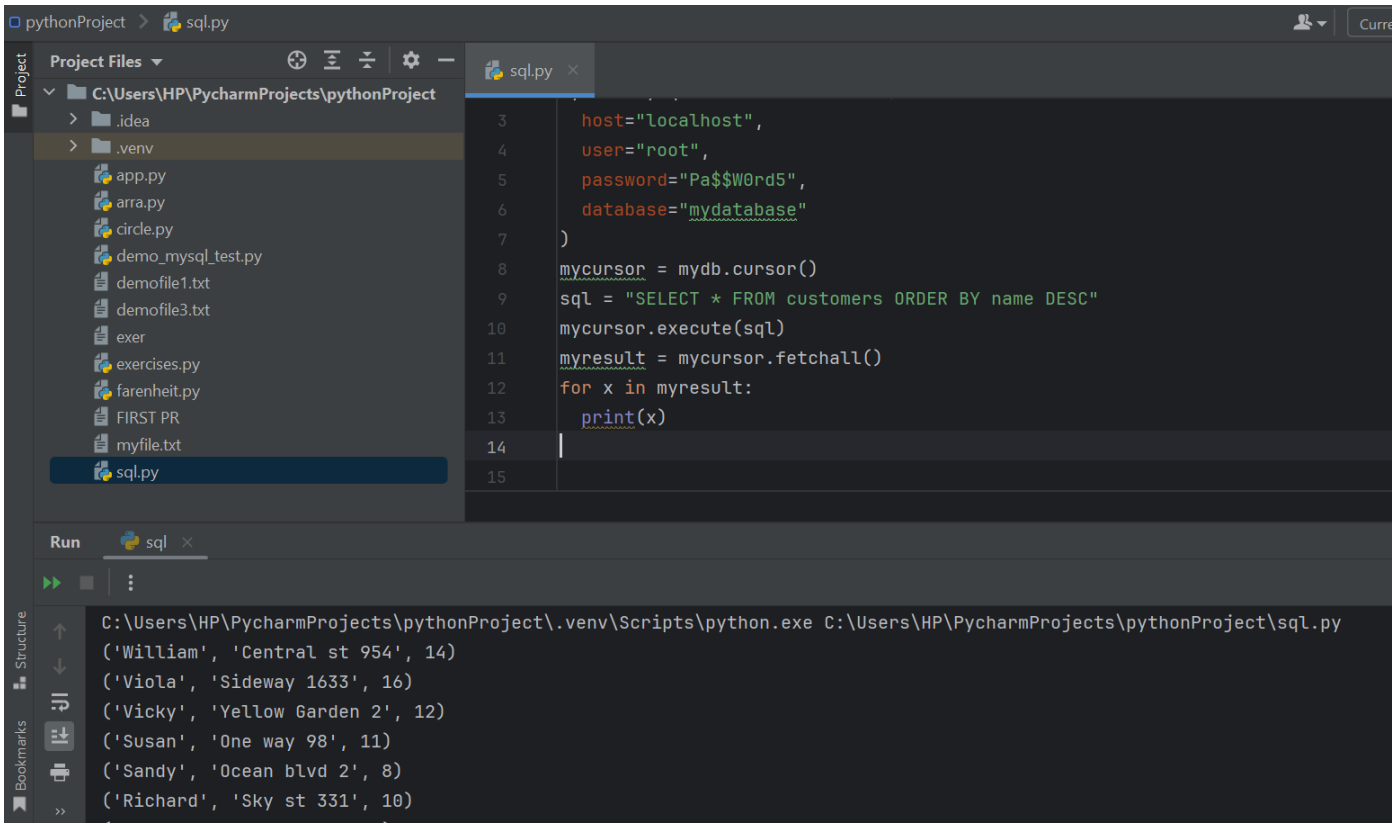


ORDER BY DESC

Use the DESC keyword to sort the result in a descending order.

Example: Sort the result reverse alphabetically by name.

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
sql = "SELECT * FROM customers ORDER BY name DESC"
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```



4.3.1.10. Drop tables

Drop command affects the structure of the table and not data. It is used to delete an already existing table. For cases where you are not sure if the table to be dropped exists or not DROP TABLE IF EXISTS command is used. Both cases will be dealt with in the following examples.You can delete an existing table by using the "DROP TABLE" statement:

Example: Delete the table "customers".

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
sql = "DROP TABLE customers"
mycursor.execute(sql)
```

```
mysql> use mydatabase
Database changed
mysql> show tables;
Empty set (0.00 sec)
```

Drop Only if Exist

If the table you want to delete is already deleted, or for any other reason does not exist, you can use the IF EXISTS keyword to avoid getting an error.

Example: Delete the table "customers" if it exists.

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
sql = "DROP TABLE IF EXISTS customers"
mycursor.execute(sql)
```

4.3.1.11. Update

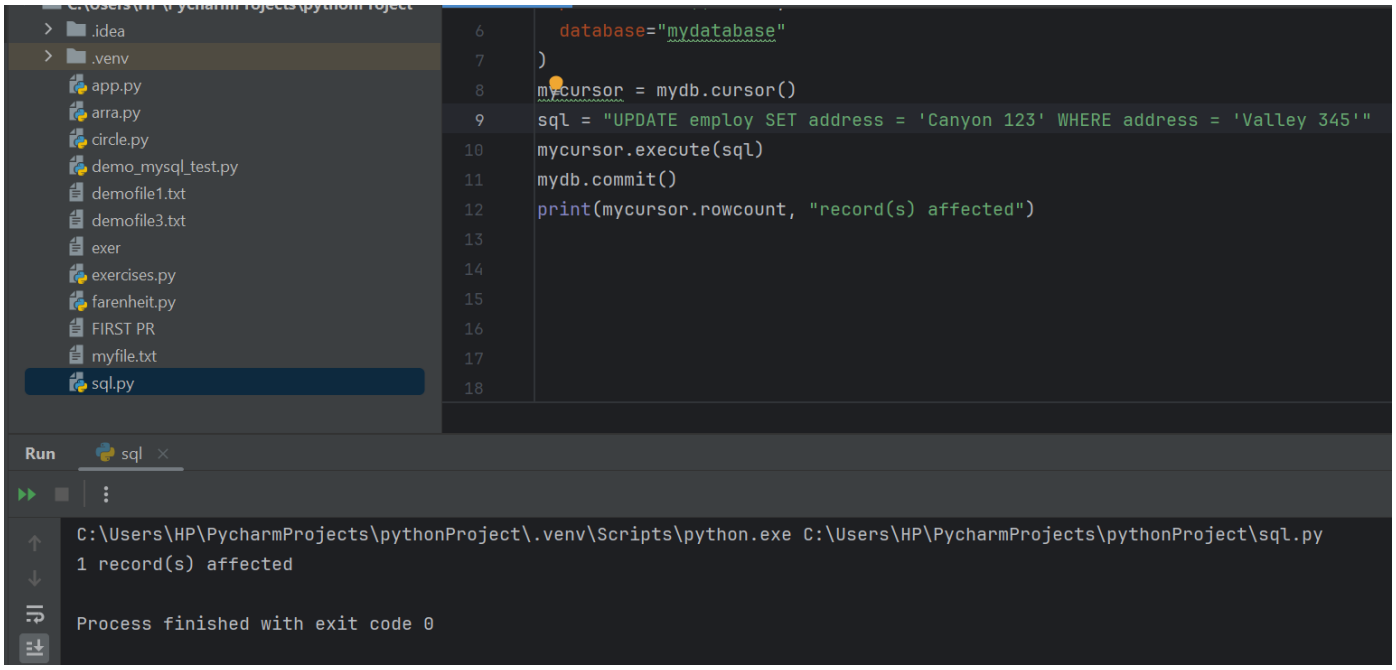
You can update existing records in a table by using the "UPDATE" statement:

Example: Overwrite the address column from "Valley 345" to "Canyon 123".

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
sql = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'"
mycursor.execute(sql)
mydb.commit()
print(mycursor.rowcount, "record(s) affected")
```

```
mysql> select *from employ;
+---------+------------------+
| name    | address          |
+---------+------------------+
| John    | Highway 21       |
| Peter   | Lowstreet 4      |
| Amy     | Apple st 652     |
| Hannah  | Mountain 21      |
| Michael | Valley 345       |
| Sandy   | Ocean blvd 2     |
| Betty   | Green Grass 1    |
| Richard | Sky st 331       |
| Susan   | One way 98       |
| Vicky   | Yellow Garden 2  |
| Ben     | Park Lane 38     |
| William | Central st 954   |
| Chuck   | Main Road 989    |
| Viola   | Sideway 1633     |
+---------+------------------+
14 rows in set (0.00 sec)
```

After update



```
mysql> select *from employ;
+---------+------------------+
| name    | address          |
+---------+------------------+
| John    | Highway 21       |
| Peter   | Lowstreet 4      |
| Amy     | Apple st 652     |
| Hannah  | Mountain 21      |
| Michael | Canyon 123       |
| Sandy   | Ocean blvd 2     |
| Betty   | Green Grass 1    |
| Richard | Sky st 331       |
| Susan   | One way 98       |
| Vicky   | Yellow Garden 2  |
| Ben     | Park Lane 38     |
| William | Central st 954   |
| Chuck   | Main Road 989    |
| Viola   | Sideway 1633     |
+---------+------------------+
14 rows in set (0.00 sec)
```

Prevent SQL Injection

It is considered a good practice to escape the values of any query, also in update statements. This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database. The mysql.connector module uses the placeholder %s to escape values in the delete statement:

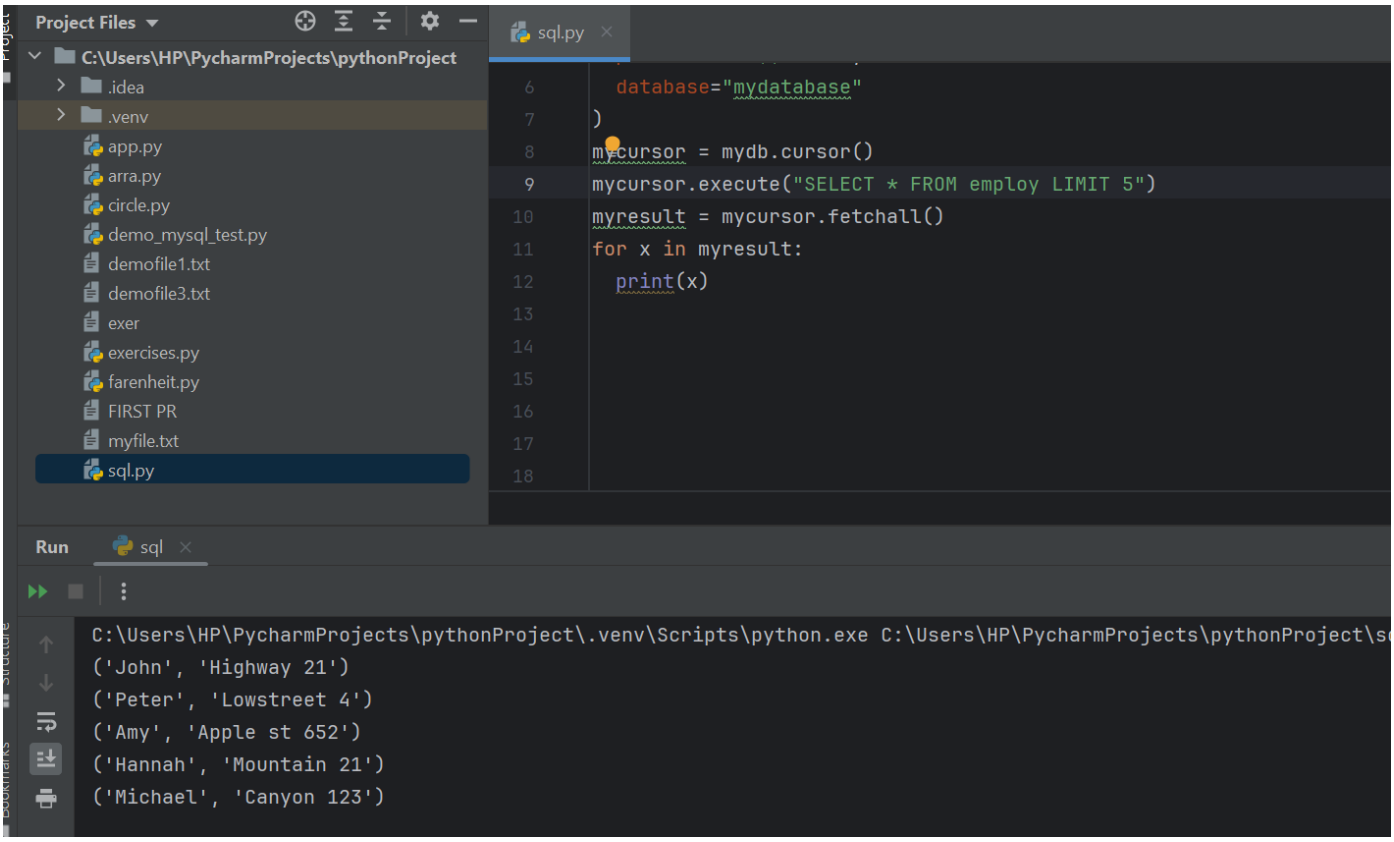
Example: Escape values by using the placeholder %s method.

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
sql = "UPDATE customers SET address = %s WHERE address = %s"
val = ("Valley 345", "Canyon 123")
mycursor.execute(sql, val)
mydb.commit()
print(mycursor.rowcount, "record(s) affected")
```

4.3.1.12. Limit

You can limit the number of records returned from the query, by using the "LIMIT" statement:
Example: Select the 5 first records in the "customers" table:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM customers LIMIT 5")
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

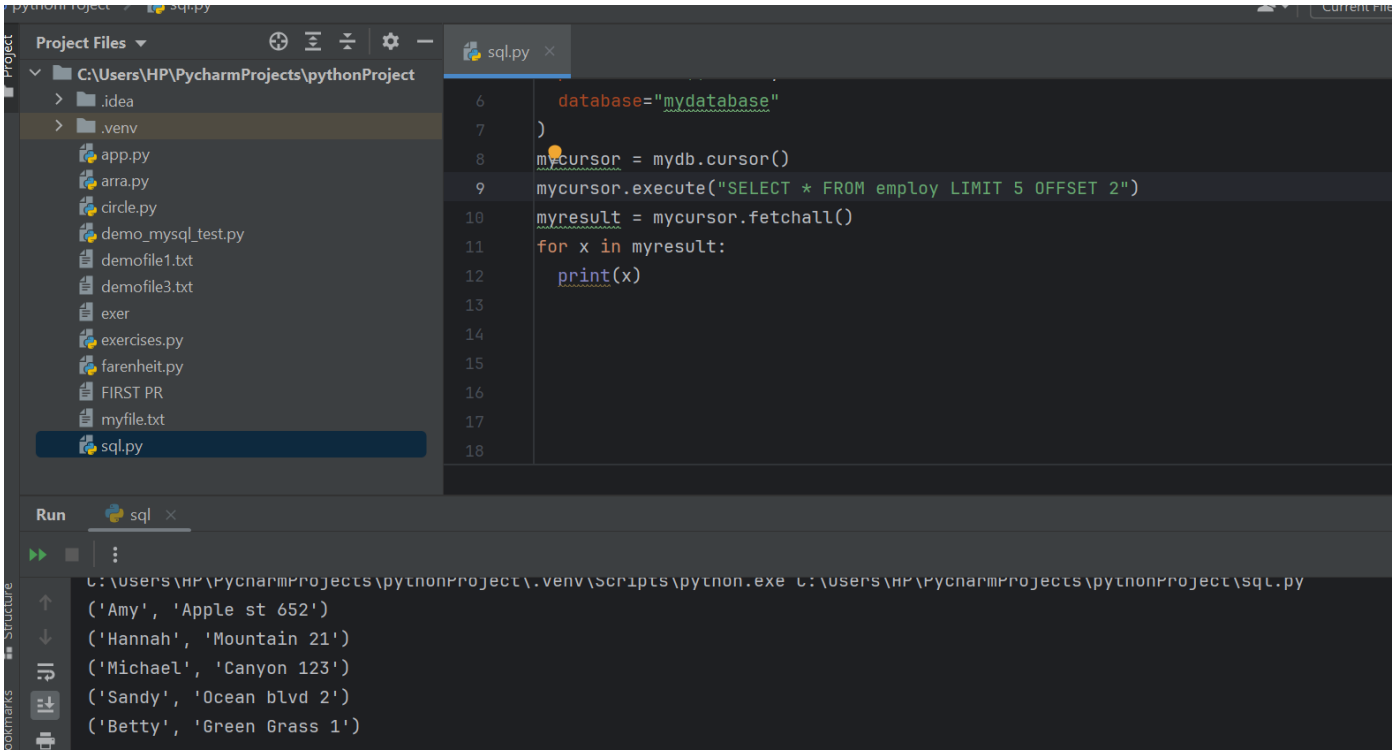


Start From Another Position

If you want to return five records, starting from the third record, you can use the "OFFSET" keyword:
Example: Start from position 3, and return 5 records:

```
import mysql.connector
mydb = mysql.connector.connect(
```

```
 host="localhost",
 user="yourusername",
 password="yourpassword",
 database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM customers LIMIT 5 OFFSET 2")
myresult = mycursor.fetchall()
for x in myresult:
 print(x)
```



4.3.1.13. Join

You can combine rows from two or more tables, based on a related column between them, by using a JOIN statement.Consider you have a "users" table and a "products" table:

```
users:
{ id: 1, name: 'John', fav: 154},
{ id: 2, name: 'Peter', fav: 154},
{ id: 3, name: 'Amy', fav: 155},
{ id: 4, name: 'Hannah', fav:},
{ id: 5, name: 'Michael', fav:}
products:
{ id: 154, name: 'Chocolate Heaven' },
```

```
{ id: 155, name: 'Tasty Lemons' },
{ id: 156, name: 'Vanilla Dreams' }
```

These two tables can be combined by using users' fav field and products' id field.
Example: Join users and products to see the name of the users favorite product.

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
sql = "SELECT \
  users.name AS user, \
  products.name AS favorite \
  FROM users \
  INNER JOIN products ON users.fav = products.id"
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

Note: You can use JOIN instead of INNER JOIN. They will both give you the same result.
LEFT JOIN
In the example above, Hannah, and Michael were excluded from the result, that is because INNER JOIN only shows the records where there is a match.If you want to show all users, even if they do not have a favorite product, use the LEFT JOIN statement.
Example: Select all users and their favorite product.

```
sql = "SELECT \
  users.name AS user, \
  products.name AS favorite \
  FROM users \
  LEFT JOIN products ON users.fav = products.id"
```

RIGHT JOIN
If you want to return all products, and the users who have them as their favorite, even if no user have them as their favorite, use the RIGHT JOIN statement:
Example: Select all products, and the user(s) who have them as their favorite:

```
sql = "SELECT \
  users.name AS user, \
  products.name AS favorite \
```

```
 FROM users \
 RIGHT JOIN products ON users.fav = products.id"
```

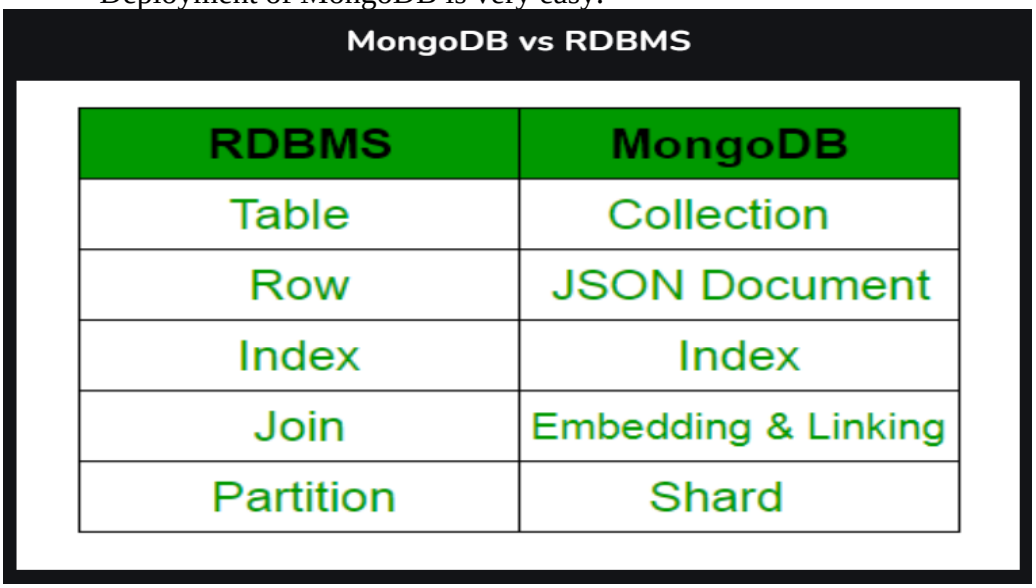
4.3.2. MongoDB

Introduction

MongoDB is a cross-platform, document-oriented database that works on the concept of collections and documents. MongoDB offers high speed, high availability, and high scalability.
The next question which arises in the mind of the people is “Why MongoDB”?

Reasons to opt for MongoDB :

- It supports hierarchical data structure (Please refer docs for details)
- It supports associate arrays like Dictionaries in Python.
- Built-in Python drivers to connect python-application with Database. Example- PyMongo
- It is designed for Big Data.
- Deployment of MongoDB is very easy.



4.3.2.1. Install driver

Python needs a MongoDB driver to access the MongoDB database. We will use the MongoDB driver "PyMongo".
We recommend that you use PIP to install "PyMongo". PIP is most likely already installed in your Python environment. Navigate your command line to the location of PIP, and type the following:
Download and install "PyMongo":

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>python -m pip install pymongo
```

Now you have downloaded and installed a mongoDB driver.

Test PyMongo
To test if the installation was successful, or if you already have "pymongo" installed, create a Python page with the following content:
demo_mongodb_test.py:

```
import pymongo
```

If the above code was executed with no errors, "pymongo" is installed and ready to be used.

4.3.2.2. Create database

To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database you want to create. MongoDB will create the database if it does not exist, and make a connection to it.
Example: Create a database called "mydatabase".

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
```

Important: In MongoDB, a database is not created until it gets content!

4.3.2.3. Create collection

To create a collection in MongoDB, use database object and specify the name of the collection you want to create. MongoDB will create the collection if it does not exist.
Example: Create a collection called "customers".

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
```

Important: In MongoDB, a collection is not created until it gets content!
MongoDB waits until you have inserted a document before it actually creates the collection.
Check if Collection Exists
Remember: In MongoDB, a collection is not created until it gets content, so if this is your first time creating a collection, you should complete the next chapter (create document) before you check if the collection exists!
You can check if a collection exists in a database by listing all collections:
Example: Return a list of all collections in your database.

```
print(mydb.list_collection_names())
```

Or you can check a specific collection by name.
Example: Check if the "customers" collection exists.
collist = mydb.list_collection_names()

```
if "customers" in collist:
  print("The collection exists.")
```

4.3.2.4. Insert

Note: A document in MongoDB is the same as a record in SQL databases.
To insert a record, or document as it is called in MongoDB, into a collection, we use the insert_one() method.

The first parameter of the insert_one() method is a dictionary containing the name(s) and value(s) of each field in the document you want to insert.
Example: Insert a record in the "customers" collection.

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
```

```
mycol = mydb["customers"]
mydict = { "name": "John", "address": "Highway 37" }
x = mycol.insert_one(mydict)
```

Return the _id Field

The insert_one() method returns a InsertOneResult object, which has a property, inserted_id, that holds the id of the inserted document.

Example: Insert another record in the "customers" collection, and return the value of the _id field.

```
mydict = { "name": "Peter", "address": "Lowstreet 27" }
x = mycol.insert_one(mydict)
print(x.inserted_id)
```

If you do not specify an _id field, then MongoDB will add one for you and assign a unique id for each document.

In the example above no _id field was specified, so MongoDB assigned a unique _id for the record (document).

Insert Multiple Documents

To insert multiple documents into a collection in MongoDB, we use the insert_many() method.

The first parameter of the insert_many() method is a list containing dictionaries with the data you want to insert:

Example:

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
mylist = [
  { "name": "Amy", "address": "Apple st 652"},
  { "name": "Hannah", "address": "Mountain 21"},
  { "name": "Michael", "address": "Valley 345"},
  { "name": "Sandy", "address": "Ocean blvd 2"},
  { "name": "Betty", "address": "Green Grass 1"},
  { "name": "Richard", "address": "Sky st 331"},
  { "name": "Susan", "address": "One way 98"},
  { "name": "Vicky", "address": "Yellow Garden 2"},
  { "name": "Ben", "address": "Park Lane 38"},
  { "name": "William", "address": "Central st 954"},
  { "name": "Chuck", "address": "Main Road 989"},
  { "name": "Viola", "address": "Sideway 1633"}
]
x = mycol.insert_many(mylist)
#print list of the _id values of the inserted documents:
print(x.inserted_ids)
```

The insert_many() method returns a InsertManyResult object, which has a property, inserted_ids, that holds the ids of the inserted documents.

Insert Multiple Documents, with Specified IDs

If you do not want MongoDB to assign unique ids for you document, you can specify the _id field when you insert the document(s).

Remember that the values has to be unique. Two documents cannot have the same _id.

Example:

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
mylist = [
 { "_id": 1, "name": "John", "address": "Highway 37"},
 { "_id": 2, "name": "Peter", "address": "Lowstreet 27"},
 { "_id": 3, "name": "Amy", "address": "Apple st 652"},
 { "_id": 4, "name": "Hannah", "address": "Mountain 21"},
 { "_id": 5, "name": "Michael", "address": "Valley 345"},
 { "_id": 6, "name": "Sandy", "address": "Ocean blvd 2"},
 { "_id": 7, "name": "Betty", "address": "Green Grass 1"},
 { "_id": 8, "name": "Richard", "address": "Sky st 331"},
 { "_id": 9, "name": "Susan", "address": "One way 98"},
 { "_id": 10, "name": "Vicky", "address": "Yellow Garden 2"},
 { "_id": 11, "name": "Ben", "address": "Park Lane 38"},
 { "_id": 12, "name": "William", "address": "Central st 954"},
 { "_id": 13, "name": "Chuck", "address": "Main Road 989"},
 { "_id": 14, "name": "Viola", "address": "Sideway 1633"}
]
x = mycol.insert_many(mylist)
#print list of the _id values of the inserted documents:
print(x.inserted_ids)
```

4.3.2.5. Find

In MongoDB we use the **find()** and **find_one()** methods to find data in a collection. Just like the SELECT statement is used to find data in a table in a MySQL database.

Find One

To select data from a collection in MongoDB, we can use the **find_one()** method. The **find_one()** method returns the first occurrence in the selection.

Example: Find the first document in the customers collection.

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
```

```
x = mycol.find_one()
print(x)
```

Find All

To select data from a table in MongoDB, we can also use the **find()** method. The **find()** method returns all occurrences in the selection.

The first parameter of the find() method is a query object. In this example we use an empty query object, which selects all documents in the collection.

No parameters in the find() method gives you the same result as SELECT * in MySQL.

Example: Return all documents in the "customers" collection, and print each document.

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
for x in mycol.find():
  print(x)
```

Return Only Some Fields

The second parameter of the find() method is an object describing which fields to include in the result. This parameter is optional, and if omitted, all fields will be included in the result.

Example: Return only the names and addresses, not the _ids.

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
for x in mycol.find({},{ "_id": 0, "name": 1, "address": 1 }):
  print(x)
```

You are not allowed to specify both 0 and 1 values in the same object (except if one of the fields is the _id field). If you specify a field with the value 0, all other fields get the value 1, and vice versa:

Example: This example will exclude "address" from the result.

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
for x in mycol.find({},{ "address": 0 }):
  print(x)
```

Example: You get an error if you specify both 0 and 1 values in the same object (except if one of the fields is the _id field).

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
```

```
mycol = mydb["customers"]
for x in mycol.find({},{ "name": 1, "address": 0 }):
  print(x)
```

4.3.2.6. Query

When finding documents in a collection, you can filter the result by using a query object. The first argument of the find() method is a query object, and is used to limit the search.
Example: Find document(s) with the address "Park Lane 38".

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
myquery = { "address": "Park Lane 38" }
mydoc = mycol.find(myquery)
for x in mydoc:
  print(x)
```

Advanced Query
To make advanced queries you can use modifiers as values in the query object.
E.g., to find the documents where the "address" field starts with the letter "S" or higher (alphabetically), use the greater than modifier: {"$gt": "S"}:
Example: Find documents where the address starts with the letter "S" or higher.

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
myquery = { "address": { "$gt": "S" } }
mydoc = mycol.find(myquery)
for x in mydoc:
  print(x)
```

Filter With Regular Expressions
You can also use regular expressions as a modifier. Regular expressions can only be used to query strings.
To find only the documents where the "address" field starts with the letter "S", use the regular expression {"$regex": "^S"}:
Example: Find documents where the address starts with the letter "S".

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
myquery = { "address": { "$regex": "^S" } }
mydoc = mycol.find(myquery)
```

```
for x in mydoc:
  print(x)
```

4.3.2.7. Sort

Use the **sort()** method to sort the result in ascending or descending order. The **sort()** method takes one parameter for "fieldname" and one parameter for "direction" (ascending is the default direction).
Example: Sort the result alphabetically by name.

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
mydoc = mycol.find().sort("name")
for x in mydoc:
  print(x)
```

Sort Descending
Use the value -1 as the second parameter to sort descending.

- sort("name", 1) #ascending
- sort("name", -1) #descending

Example: Sort the result reverse alphabetically by name.

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
mydoc = mycol.find().sort("name", -1)
for x in mydoc:
  print(x)
```

4.3.2.8. Delete document

To delete one document, we use the **delete_one()** method. The first parameter of the **delete_one()** method is a query object defining which document to delete.
Note: If the query finds more than one document, only the first occurrence is deleted.
Example: Delete the document with the address "Mountain 21".

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
myquery = { "address": "Mountain 21" }
mycol.delete_one(myquery)
```

Delete Many Documents
To delete more than one document, use the **delete_many()** method. The first parameter of the **delete_many()**

method is a query object defining which documents to delete.

Example: Delete all documents were the address starts with the letter S.

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
myquery = { "address": {"$regex": "^S"} }
x = mycol.delete_many(myquery)
print(x.deleted_count, " documents deleted.")
```

Delete All Documents in a Collection
To delete all documents in a collection, pass an empty query object to the delete_many() method:
Example: Delete all documents in the "customers" collection.

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
x = mycol.delete_many({})
print(x.deleted_count, " documents deleted.")
```

4.3.2.9. Drop collection

You can delete a table, or collection as it is called in MongoDB, by using the drop() method.
Example: Delete the "customers" collection.

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
mycol.drop()
```

The drop() method returns true if the collection was dropped successfully, and false if the collection does not exist.

4.3.2.10. Update

You can update a record, or document as it is called in MongoDB, by using the **update_one()** method. The first parameter of the **update_one()** method is a query object defining which document to update.
Note: If the query finds more than one record, only the first occurrence is updated. The second parameter is an object defining the new values of the document.
Example: Change the address from "Valley 345" to "Canyon 123".

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
myquery = { "address": "Valley 345" }
```

```
newvalues = { "$set": { "address": "Canyon 123" } }
mycol.update_one(myquery, newvalues)
#print "customers" after the update:
for x in mycol.find():
  print(x)
```

Update Many

To update all documents that meets the criteria of the query, use the update_many() method.

Example: Update all documents where the address starts with the letter "S".

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
myquery = { "address": { "$regex": "^S" } }
newvalues = { "$set": { "name": "Minnie" } }
x = mycol.update_many(myquery, newvalues)
print(x.modified_count, "documents updated.")
```

4.3.2.11. Limit

To limit the result in MongoDB, we use the **limit()** method. The **limit()** method takes one parameter, a number defining how many documents to return.

Consider you have a "customers" collection:

```
{'_id': 1, 'name': 'John', 'address': 'Highway37'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
```

Example: Limit the result to only return 5 documents.

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
```

```
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
myresult = mycol.find().limit(5)
#print the result:
for x in myresult:
  print(x)
```

LEARNING UNIT 5–DEVELOP AI BASED APPLICATIONS

Learning Outcome 5.1: Introduce AI

5.1.1. Definitions of Key terms

5.1.1.1. What is AI?

In today's world, technology is growing very fast, and we are getting in touch with different new technologies day by day.

Here, one of the booming technologies of computer science is Artificial Intelligence which is ready to create a new revolution in the world by making intelligent machines. The Artificial Intelligence is now all around us. It is currently working with a variety of subfields, ranging from general to specific, such as self-driving cars, playing chess, proving theorems, playing music, Painting, etc.

AI is one of the fascinating and universal fields of Computer science which has a great scope in future. AI holds a tendency to cause a machine to work as a human.

Artificial Intelligence is composed of two words Artificial and Intelligence, where Artificial defines "man-made," and intelligence defines "thinking power", hence AI means "a man-made thinking power."

So, we can define AI as:

"It is a branch of computer science by which we can create intelligent machines which can behave like a human, think like humans, and able to make decisions."

Artificial Intelligence exists when a machine can have human based skills such as learning, reasoning, and solving problems.

With Artificial Intelligence you do not need to preprogram a machine to do some work, despite that you can create a machine with programmed algorithms which can work with own intelligence, and that is the awesomeness of AI.

It is believed that AI is not a new technology, and some people says that as per Greek myth, there were Mechanical men in early days which can work and behave like humans.

5.1.1.2. History of AI

The idea of 'a machine that thinks' dates back to ancient Greece. But since the advent of electronic computing (and relative to some of the topics discussed in this article) important events and milestones in the evolution of artificial intelligence include the following:

- **1950:** Alan Turing publishes Computing Machinery and Intelligence. In the paper, Turing—famous for breaking the Nazi's ENIGMA code during WWII—proposes to answer the question 'can machines think?' and introduces the Turing Test to determine if a computer can demonstrate the same intelligence (or the results of the same intelligence) as a human. The value of the Turing test has been debated ever since.
- **1956:** John McCarthy coins the term 'artificial intelligence' at the first-ever AI conference at Dartmouth College. (McCarthy would go on to invent the Lisp language.) Later that year, Allen Newell, J.C. Shaw, and Herbert Simon create the Logic Theorist, the first-ever running AI software program.
- **1967:** Frank Rosenblatt builds the Mark 1 Perceptron, the first computer based on a neural network that 'learned' though trial and error. Just a year later, Marvin Minsky and Seymour Papert publish a book titled Perceptrons, which becomes both the landmark work on neural networks and, at least for a while, an argument against future neural network research projects.

- **1980s:** Neural networks which use a backpropagation algorithm to train itself become widely used in AI applications.
- **1997:** IBM's Deep Blue beats then world chess champion Garry Kasparov, in a chess match (and rematch).
- **2011:** IBM Watson beats champions Ken Jennings and Brad Rutter at Jeopardy!
- **2015:** Baidu's Minwa supercomputer uses a special kind of deep neural network called a convolutional neural network to identify and categorize images with a higher rate of accuracy than the average human.
- **2016:** DeepMind's AlphaGo program, powered by a deep neural network, beats Lee Sodol, the world champion Go player, in a five-game match. The victory is significant given the huge number of possible moves as the game progresses (over 14.5 trillion after just four moves!). Later, Google purchased DeepMind for a reported USD 400 million.

5.1.1.3. Benefits of AI

1. Automation

Automation is one of the most commonly cited benefits of AI technology, and it has had significant impacts on the communications, transportation, consumer products, and service industries. AI enables the automation of repetitive tasks, freeing up human workers to focus on more creative and strategic endeavors. This leads to increased efficiency and productivity.

2. Smart Decision Making

Artificial Intelligence has always been used for making smarter business decisions. AI technology can coordinate data delivery, analyze trends, develop data consistency, provide forecasts, and quantify uncertainties to make the best decisions for the company. AI systems can analyze vast amounts of data quickly and accurately, providing valuable insights for decision-making processes. This leads to more informed and data-driven decisions.

3. Enhanced Customer Experience

AI-powered solutions can help businesses to respond to customer queries and grievances quickly and address the situations efficiently. The use of chatbots that couple conversational AI with Natural Language Processing technology can generate highly personalized messages for customers, which helps to find the best solution for their needs. AI tools can also help to reduce the strain from the customer service staff, which will lead to better productivity.

4. Medical Advances

The use of Artificial Intelligence solutions in the healthcare sector is becoming increasingly popular these days. Remote patient monitoring technology, for instance, allows healthcare providers to perform clinical diagnoses and suggest treatments quickly without requiring the patient to visit the hospital in-person. AI can also be beneficial in monitoring the progression of contagious diseases and even predict their future effects and outcomes.

5.Innovation: AI fosters innovation by enabling the development of new products, services, and business models. It fuels advancements in areas such as robotics, natural language processing, computer vision, and autonomous systems.

6.Accessibility: AI technologies can improve accessibility for people with disabilities by providing assistive tools and devices that enhance communication, mobility, and independent living.

7. Business Continuity

Business forecasting using AI technology not only helps companies make critical decisions but also prepares them for any emergency to ensure business continuity. As risk management heavily relies on data management and analysis today, AI-powered tools can help organizations to respond to the crisis proactively. AI and Machine Learning can also create scenarios to help businesses plan for a speedy disaster recovery strategy.

8. Managing Repetitive Tasks

Performing recurring business tasks is not just time-consuming but it can also be monotonous and reduce the productivity of the employees over time. AI-powered Robotic Process Automation tools can automate interactions between different business systems and make the tiresome work easy for the company. It can imitate the actions of humans within the digital systems in the HR, IT, marketing, or sales departments to execute any business process quickly without needing any manual effort.

9. Minimizing Errors

Another great benefit of automating regular business tasks using AI tools is that it helps to reduce the chances of

manual errors. As Robotic Process Automation tools take care of the data entry and processing jobs, it can make the digital systems more efficient and less likely to run into or create any problems due to data processing mistakes. This can be especially beneficial for businesses that cannot afford to make even the slightest of errors.

10. Increased Business Efficiency

Artificial Intelligence can help to ensure 24-hour service availability and will deliver the same performance and consistency throughout the day. Taking care of repetitive tasks will not make AI tools get tired or bored either. This can help to improve the efficiency of the business and reduce the stress on the employees, who can be re-assigned to perform more complex business tasks that require manual intervention.

5.1.2. Types of AI

5.1.2.1. Weak AI

Narrow AI, also known as weak AI, is designed to perform specific tasks or solve particular problems within a limited domain.

Examples include virtual assistants (like Siri or Alexa), recommendation systems (such as those used by Netflix or Amazon), and image recognition software

5.1.2.2. Strong A

Strong AI is made up of Artificial General Intelligence (AGI) and Artificial Super Intelligence (ASI). Artificial general intelligence (AGI), or general AI, is a theoretical form of AI where a machine would have an intelligence equaled to humans; it would have a self-aware consciousness that has the ability to solve problems, learn, and plan for the future. General AI refers to AI systems with human-like intelligence and the ability to understand, learn, and apply knowledge across a wide range of tasks and domains.

5.1.3. Real life example of AI

Artificial Intelligence has various applications in today's society. It is becoming essential for today's time because it can solve complex problems with an efficient way in multiple industries, such as Healthcare, entertainment, finance, education, etc. AI is making our daily life more comfortable and fast. Following are some **sectors, which** have the application of Artificial Intelligence:

1. Security and surveillance:

Many airports, train stations, and other transportation hubs use facial recognition systems for security and surveillance purposes. These systems can detect and identify individuals on watchlists or track the movement of people in crowded areas in real-time

2. AI in Healthcare

In the last, five to ten years, AI becoming more advantageous for the healthcare industry and going to have a significant impact on this industry.

Healthcare Industries are applying AI to make a better and faster diagnosis than humans.

Facial recognition technology is used in healthcare for patient identification, monitoring, and diagnosis. It can help medical professionals identify patients accurately, track their vital signs, and detect signs of certain medical conditions by analyzing facial expressions and symptoms AI can help doctors with diagnoses and can inform when patients are worsening so that medical help can reach to the patient before hospitalization.

3. AI in Gaming

AI can be used for gaming purpose. The AI machines can play strategic games like chess, where the machine needs to think of a large number of possible places.

4. AI in Finance

AI and finance industries are the best matches for each other. The finance industry is implementing automation, chatbot, adaptive intelligence, algorithm trading, and machine learning into financial processes.

5. Attendance tracking:

Facial recognition systems are used in educational institutions, workplaces, and events for attendance tracking purposes. Instead of manual attendance taking, these systems can automatically identify and record the presence of individuals based on their facial features.

6. AI in Social Media

Social Media sites such as Facebook, Twitter, and Snapchat contain billions of user profiles, which need to be stored and managed in a very efficient way. AI can organize and manage massive amounts of data. AI can analyze lots of

data to identify the latest trends, hashtag, and requirement of different users.

7. AI in Travel & Transport

AI is becoming highly demanding for travel industries. AI is capable of doing various travel related works such as from making travel arrangement to suggesting the hotels, flights, and best routes to the customers. Travel industries are using AI-powered chatbots which can make human-like interaction with customers for better and fast response.

8. AI in Automotive Industry

Some Automotive industries are using AI to provide virtual assistant to their user for better performance. Such as Tesla has introduced TeslaBot, an intelligent virtual assistant.

Various Industries are currently working for developing self-driven cars which can make your journey more safe and secure.

9. AI in Robotics

Artificial Intelligence has a remarkable role in Robotics. Usually, general robots are programmed such that they can perform some repetitive task, but with the help of AI, we can create intelligent robots which can perform tasks with their own experiences without pre-programmed.

Humanoid Robots are best examples for AI in robotics, recently the intelligent Humanoid robot named as Erica and Sophia has been developed which can talk and behave like humans.

10. AI in Entertainment

We are currently using some AI based applications in our daily life with some entertainment services such as Netflix or Amazon. With the help of ML/AI algorithms, these services show the recommendations for programs or shows.

11. AI in Agriculture

Agriculture is an area which requires various resources, labor, money, and time for best result. Now a day's agriculture is becoming digital, and AI is emerging in this field. Agriculture is applying AI as agriculture robotics, solid and crop monitoring, predictive analysis. AI in agriculture can be very helpful for farmers.

12. AI in E-commerce

AI is providing a competitive edge to the e-commerce industry, and it is becoming more demanding in the e-commerce business. AI is helping shoppers to discover associated products with recommended size, color, or even brand.

13. AI in education

AI can automate grading so that the tutor can have more time to teach. AI chatbot can communicate with students as a teaching assistant.

AI in the future can be work as a personal virtual tutor for students, which will be accessible easily at any time and any place.

5.1.4. Future of AI

5.1.4.1. Military Bots

Robot soldiers are autonomous robots or remote-controlled mobile robots designed for military applications, from transport to search & rescue and attack. Military robots can be built and programmed to do a certain task and will do it better than how a human would do it. Military robots can make the difference between a winning war and a losing war, Military robots can include different types of weapons and are more efficient than a human controlling these weapons.

5.1.4.2. Laws

Will AI replace lawyers?

AI tools undoubtedly present opportunities for lawyers. Automation saves lawyers time that would otherwise be spent helping clients and taking on more billable work.

However, there are also challenges AI presents that legal professionals should be aware of. It’s important for lawyers to understand their ethical obligations when using AI technologies.

While AI can save lawyers time, support automation, and even add a flare to one’s legal writing, it’s not here to steal jobs away from human beings. And having fear about technology which can save you and your firm time, might be holding you back from helping even more clients.

5.1.4.3. Music

Technology has always played a crucial role in the world of music. Digital production software, loop pedals, and multi-track recording have been rapidly transforming the music industry. Artificial intelligence, however, is the most disruptive technology in terms of its impact. Taryn Southern composed a complete album using AI in 2018. Various AI software programs such as Google NSynth, IBM Watson Beat, AIVA, and Amper can create original music from scratch or just refine the existing music.

Future of AI in music is still a mystery which will be unraveled with the passage of time. There is a possibility of AI becoming the musician rather than an assistant to the musician. Same can be said about AI-powered self-driving cars. However, one cannot deny the enormous benefits of AI in the music industry. For instance, SMEs, social media users, and video game creators can leverage AI for composing their own soundtracks.

Not just that, AI can prove to be a reliable compositional instrument for both producers and musicians. Music promotion and distribution services will witness an increasing reliance on artificial intelligence in the near future. Further, artificial intelligence is an exceptional technology for enthusiasts who want to venture into the universe of music.

5.1.4.4. Business

In the big business race of who-beats-who, having an artificially intelligent system by your side is bound to give you an edge. Not just that, it may just be the reason you're even invited to the competition! Days of good ol' salesmanship and customer service representatives seem to be dying out. Here's some ways you can benefit from integrating AI into your business:

1) Saves time and money

AI systems work best in optimizing and performing accurately repetitive tasks. The technology also enables businesses to inform customers/workers of delays or expected completion times. AI can go through previous records in the business to determine common failures and inform employees beforehand.

2) Improves customer experience

AI can offer customers better and more personalized experiences of working with your business. Machine Learning especially focuses on this, since it can cater to a person dealing with your business with recommendations based on their previous behavior. An obvious example of this are shopping websites, where if you look for a pair of jeans the website will suggest similar items alongside the original one.

3) Automates customer interactions

Although interactions with people require the human touch and technology's not quite there yet, artificial intelligence is bit by bit trying to transform that too. By reading and analyzing information collected from previous interactions, AI trains itself to deal with customers and respond to familiar questions accurately.

A great example of this are chatbots. These nifty pieces of tech interact with people that come to a site or app, can initiate conversations with them and even deal with multiple users at once. Sounds pretty neat for a non-human.

4) Predicts outcomes

Computers are much better predictors of certain outcomes than humans are, provided the right kind of data is available and no impromptu judgement is required. Based on data analysis, AI tech computer can predict patterns of data about consumer behavior. It can also predict, based on previous customer records, what items will be sold in the next big sale.

These functions make AI an essential technology for businesses today.

5.1.4.5. Healthcare

Hospitals and research institutions will be able to use AI to analyze large amounts of data from electronic health records, lab results, and other sources to identify patterns or trends that researchers can use to predict future health outcomes.

It will also be possible for scientists and doctors to use a predictive model to identify patients at high risk of developing a specific condition or to predict which patients are most likely to respond to a specific treatment. Eventually, AI will provide physicians with real-time information and recommendations to help them make more informed treatment decisions while seeing patients in their offices.

At a higher level, policymakers will be able to use AI to analyze data from disparate sources—from personal health records to sewage analysis to air quality—to identify patterns or trends that they can use to improve the health of entire populations.

Learning Outcome 5.2: Implement Machine Learning

5.2.1. Definition of Key Terms:

5.2.1.1. Machine Learning

In the real world, we are surrounded by humans who can learn everything from their experiences with their learning capability, and we have computers or machines which work on our instructions. But can a machine also learn from experiences or past data like a human does? So here comes the role of **Machine Learning** Machine Learning (ML) is a subfield of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computer systems to learn from data, improve from experience, and make predictions or decisions without explicit instructions.
Without being explicitly programmed, machine learning enables a machine to automatically learn from data, improve performance from experiences, and predict things.
Machine learning algorithms create a mathematical model that, without being explicitly programmed, aids in making predictions or decisions with the assistance of sample historical data, or training data.

In ML, a computer program is trained on a large dataset, and the learning algorithm uses that data to find patterns and relationships that can be used to make predictions or decisions. The more data the program is trained on, the more accurate its predictions and decisions become. There are several types of ML algorithms, including supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning.
Machine Learning has numerous applications in various industries, including healthcare, finance, marketing, and transportation, among others. It is used for tasks such as image and speech recognition, natural language processing, fraud detection, and recommendation systems.
The goal of ML is to automate decision-making and predictions and to create systems that can improve their performance over time.

5.2.1.2. Data splitting

Data splitting is when data is divided into two or more subsets for training. Typically, with a two-part split, one part is used to evaluate or test the data and the other to train the model.

How data splitting works

In a basic two-part data split, the training data set is used to train and develop models. Training sets are commonly used to estimate different parameters or to compare different model performance.

The testing data set is used after the training is done. The training and test data are compared to check that the final model works correctly. Here's a breakdown of the typical data splitting process:

Training Set: The training set is the largest portion of the dataset and is used to train the machine learning model. The model learns patterns and relationships in the data from this set.

Validation Set: The validation set is used to fine-tune the model's hyperparameters and assess its performance during training. It helps in selecting the best model architecture and hyperparameters while preventing overfitting on the training data.

Testing Set: The testing set is used to evaluate the final performance of the trained model after tuning hyperparameters on the validation set. It provides an unbiased estimate of the model's generalization ability on unseen data.

By splitting the data into separate training, validation, and testing sets, machine learning practitioners can assess the model's performance accurately, optimize its parameters effectively, and make informed decisions about its deployment in real-world applications.

5.2.1.3. Numerical data

Numerical data in AI refers to data that consists of numerical values or measurements. This type of data is commonly used in machine learning and statistical modeling tasks. Numerical data can be continuous or discrete.Numerical data, or quantitative data, is any form of measurable data such as your height, weight, or the cost of your phone bill. You can determine if a set of data is numerical by attempting to average out the numbers or sort them in ascending or descending order. Exact or whole numbers (ie. 26 students in a class) are considered discrete numbers, while those which fall into a given range (ie. 3.6 percent interest rate) are considered continuous numbers. While learning this type of data, keep in mind that numerical data is not tied to any specific point in time, they are simply raw numbers.

5.2.1.4. Categorical data

Categorical data is sorted by defining characteristics. This can include gender, social class, ethnicity, hometown, the industry you work in, or a variety of other labels.
Categorical data in AI refers to data that represents categories or groups rather than numerical values. Categorical data can take on a limited number of distinct values or levels and is often used to represent qualitative attributes or characteristics. There are two main types of categorical data:
Nominal Categorical Data: Nominal categorical data consists of categories with no inherent order or ranking. Each category is distinct, and there is no meaningful way to compare them. Examples include colors (e.g., red, blue, green), types of animals (e.g., cat, dog, bird), and categorical variables like gender or country.
Ordinal Categorical Data: Ordinal categorical data consists of categories with a natural order or ranking. While the categories are still discrete, they can be ordered based on some criteria. Examples include ratings (e.g., low, medium, high), educational levels (e.g., elementary, middle, high school), and satisfaction levels (e.g., very unsatisfied, unsatisfied, neutral, satisfied, very satisfied).
Categorical data is great for grouping individuals or ideas that share similar attributes, helping your machine learning model streamline its data analysis.

5.2.1.5. Ordinal data

This denotes a nominal variable with categories falling in an ordered list . Examples include clothing sizes such as small, medium , and large , or a measurement of customer satisfaction on a scale from “not at all happy” to “very happy”.

5.2.1.6. Percentile

Percentile is a measure used in statistics to calculate the percentile of data in the columns of a data table. When a set of data is ordered from the smallest to largest (ascending order) and is divided into 100 groups(25th percentile (first quartile), 50th percentile (median), 75th percentile (third quartile), 90th percentile, 95th percentile, 99th percentile)
The percentile indicates the value below which a given percentage of data falls.
Percentiles are used in statistics to give you a number that describes the value that a given percent of the values are lower than.
Example: Let's say we have an array of the ages of all the people that live in a street.

```
ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]
```

What is the 75. percentile? The answer is 43, meaning that 75% of the people are 43 or younger.

The NumPy module has a method for finding the specified percentile:
Example1: Use the NumPy percentile() method to find the percentiles:

```
import numpy
ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]
x = numpy.percentile(ages, 75)
print(x)
```

Example2:

```
import numpy as np

# 1D array
arr = [20, 2, 7, 1, 34]
print("arr : ", arr)
print("50th percentile of arr : ",
       np.percentile(arr, 50))
print("25th percentile of arr : ",
       np.percentile(arr, 25))
print("75th percentile of arr : ",
       np.percentile(arr, 75))
```

```
Output: arr :  [20, 2, 7, 1, 34]
50th percentile of arr :  7.0
25th percentile of arr :  2.0
75th percentile of arr :  20.0
```

Example: What is the age that 90% of the people are younger than?

```
import numpy
ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]
x = numpy.percentile(ages, 90)
print(x)
```

5.2.1.7. Data distribution

In the real world, the data sets are much bigger, but it can be difficult to gather real world data, at least at an early stage of a project.

How Can we Get Big Data Sets?

To create big data sets for testing, we use the Python module NumPy, which comes with a number of methods to create random data sets, of any size.

Example: Create an array containing 250 random floats between 0 and 5.

```
import numpy
x = numpy.random.uniform(0.0, 5.0, 250)
print(x)
```

Histogram

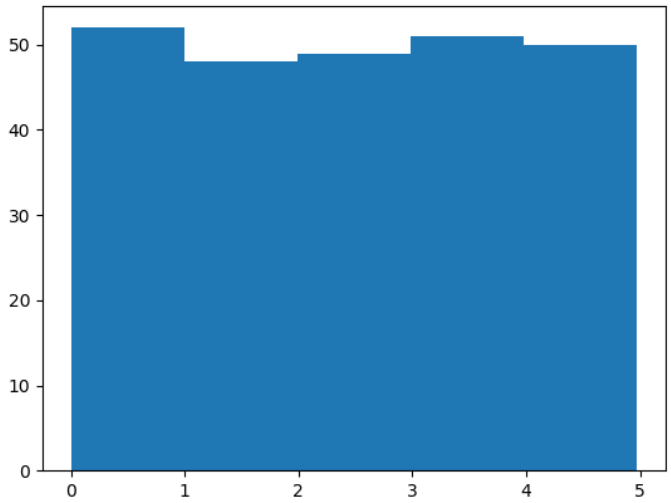
To visualize the data set we can draw a histogram with the data we collected.

We will use the Python module Matplotlib to draw a histogram.

Example: Draw a histogram.

```
import numpy
import matplotlib.pyplot as plt
x = numpy.random.uniform(0.0, 5.0, 250)
plt.hist(x, 5)
plt.show()
```

Result:



Histogram Explained

- We use the array from the example above to draw a histogram with 5 bars.
- The first bar represents how many values in the array are between 0 and 1.
- The second bar represents how many values are between 1 and 2.
- Etc.

Which gives us this result:
52 values are between 0 and 1
48 values are between 1 and 2
49 values are between 2 and 3
51 values are between 3 and 4
50 values are between 4 and 5
Note: The array values are random numbers and will not show the exact same result on your computer.
Big Data Distributions
An array containing 250 values is not considered very big, but now you know how to create a random set of values, and by changing the parameters, you can create the data set as big as you want.
Example: Create an array with 100000 random numbers, and display them using a histogram with 100 bars.

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.uniform(0.0, 5.0, 100000)

plt.hist(x, 100)
plt.show()
```

5.2.1.8. Algorithm

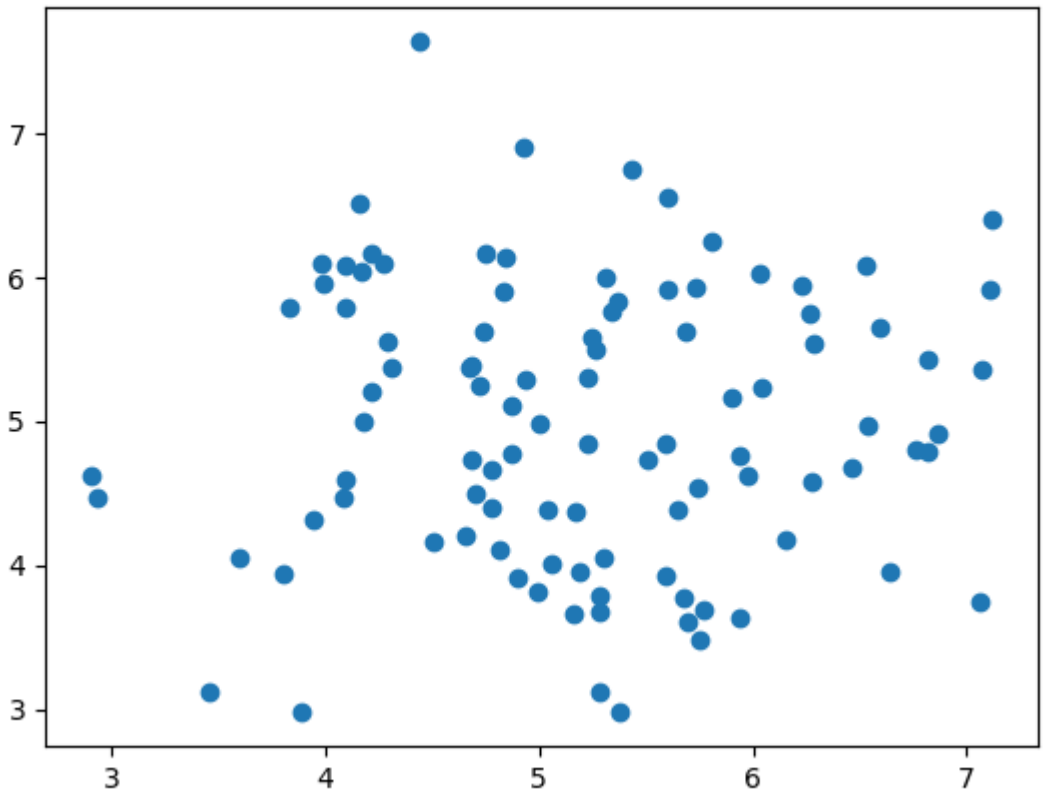
Machine Learning algorithms are the programs that can learn the hidden patterns from the data, predict the output, and improve the performance from experiences on their own. Different algorithms can be used in machine learning for different tasks, such as simple linear regression that can be used for prediction problems like stock market prediction, and the KNN algorithm can be used for classification problems.

5.2.1.9. Model

A machine learning model is a computer program that are used to recognize patterns in data or make predictions. is also a program that can find patterns or make decisions from a previously unseen dataset. For example, in natural language processing, machine learning models can parse and correctly recognize the intent behind previously unheard sentences or combinations of words. In image recognition, a machine learning model can be taught to recognize objects - such as cars or dogs. A machine learning model can perform such tasks by having it 'trained' with a large dataset. During training, the machine learning algorithm is optimized to find certain patterns or outputs from the dataset, depending on the task. The output of this process, often a computer program with specific rules and data structures, is called a machine learning model.
Machine learning models are created from machine learning algorithms, which undergo a training process using either labeled, unlabeled, or mixed data. Different machine learning algorithms are suited to different goals, such as classification or prediction modeling, so data scientists use different algorithms as the basis for different models. As data is introduced to a specific algorithm, it is modified to better manage a specific task and becomes a machine learning model.
For example, a decision tree is a common algorithm used for both classification and prediction modeling. A data scientist looking to create a machine learning model that identifies different animal species might train a decision tree algorithm with various animal images. Over time, the algorithm would become modified by the data and become increasingly better at classifying animal images.

5.2.1.10. Scatter plot

A scatter plot is a diagram where each value in the data set is represented by a dot.



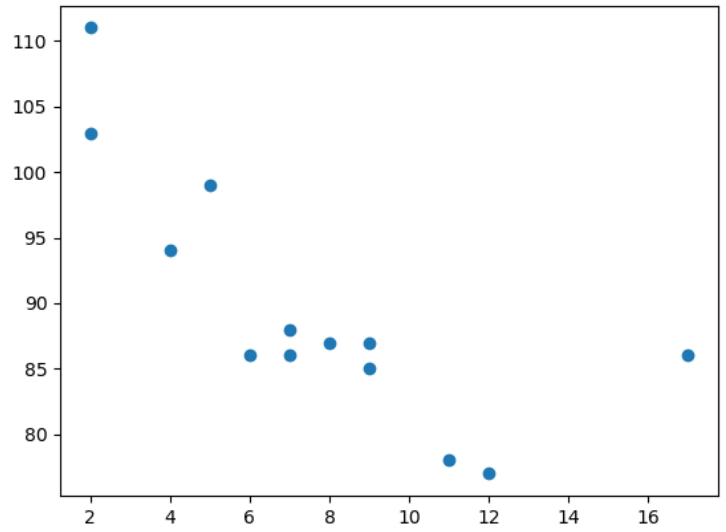
The Matplotlib module has a method for drawing scatter plots, it needs two arrays of the same length, one for the values of the x-axis, and one for the values of the y-axis:

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

The **x** array represents the age of each car.
The **y** array represents the speed of each car.
Example: Use the scatter() method to draw a scatter plot diagram.

```
import matplotlib.pyplot as plt
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
plt.scatter(x, y)
plt.show()
```

Result:

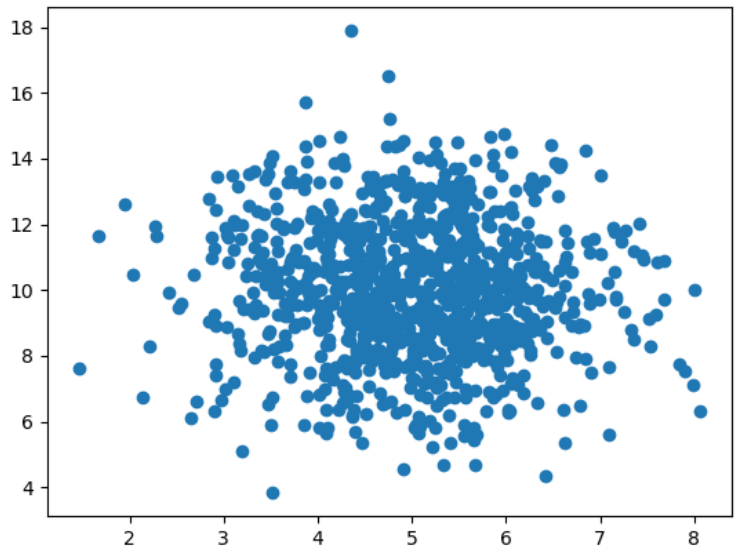


Scatter Plot Explained
The x-axis represents ages, and the y-axis represents speeds.
What we can read from the diagram is that the two fastest cars were both 2 years old, and the slowest car was 12 years old.
Note: It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.
Random Data Distributions
In Machine Learning the data sets can contain thousands-, or even millions, of values.
You might not have real world data when you are testing an algorithm, you might have to use randomly generated values.
As we have learned in the previous chapter, the NumPy module can help us with that!
Let us create two arrays that are both filled with 1000 random numbers from a normal data distribution.
The first array will have the mean set to 5.0 with a standard deviation of 1.0.
The second array will have the mean set to 10.0 with a standard deviation of 2.0:
Example: A scatter plot with 1000 dots.

```
import numpy
import matplotlib.pyplot as plt
x = numpy.random.normal(5.0, 1.0, 1000)
y = numpy.random.normal(10.0, 2.0, 1000)
plt.scatter(x, y)
```

```
plt.show()
```

Result:



Scatter Plot Explained

We can see that the dots are concentrated around the value 5 on the x-axis, and 10 on the y-axis.
We can also see that the spread is wider on the y-axis than on the x-axis.

5.2.2. Define Variables and data

5.2.2.1. Predictor variable

A predictor variable is a variable used to predict another variable's value, and these values can be utilized in both classification and regression. In most machine learning and statistics projects, there will be many predictor variables, as accuracy usually increases with more data.

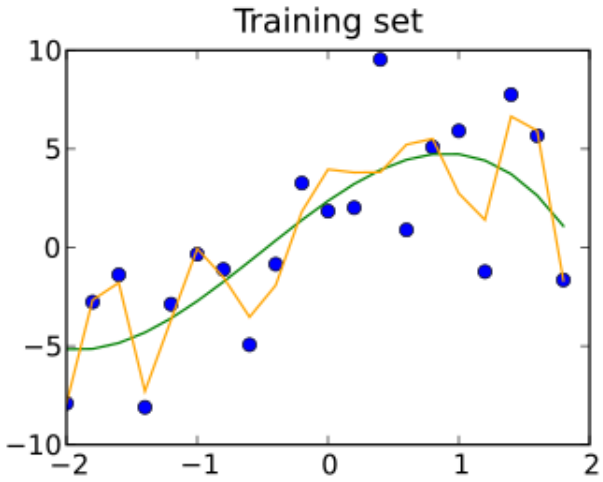
For example, if you want to predict the price of a house, the predictor variables might be the size of the house, the number of bedrooms, the number of bathrooms, and the location.

5.2.2.2. Response variable

A Response Variable (or dependent variable) is that variable whose variation depends on other variables. The response variable is often related to the independent variable, sometimes denoted as the explanatory variable. In short, the response variable is the subject of change within an experiment, often as a result of differences in the explanatory variables.

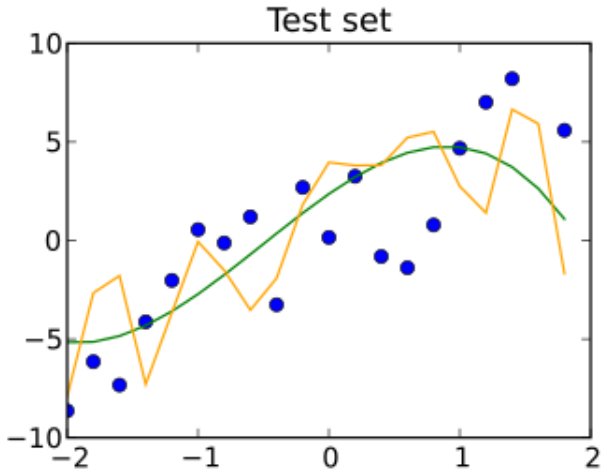
5.2.2.3. Training data

The training data is the biggest (in -size) subset of the original dataset, which is used to train or fit the machine learning model. Firstly, the training data is fed to the ML algorithms, which lets them learn how to make predictions for the given task.



5.2.2.4. Testing data

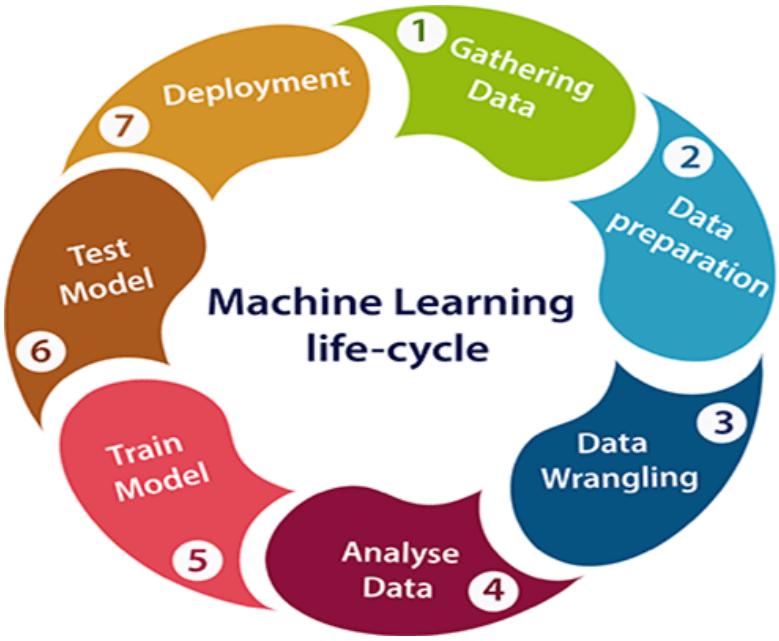
A test data set is a data set that is independent of the training data set, but that follows the same probability distribution as the training data set. If a model fit to the training data set also fits the test data set well, minimal overfitting has taken place. A better fitting of the training data set as opposed to the test data set usually points to over-fitting.



5.2.3. Machine Learning processes (Lifecycle)

Machine learning life cycle involves seven major steps, which are given below:

- Gathering Data
- Data preparation
- Data Wrangling
- Analyze Data
- Train the model
- Test the model
- Deployment



5.2.3.1. Define Objective

The above steps are preceded with objective definition. An objective what you want to find (predict) in supervised learning in machine learning.

5.2.3.2. Data gathering

Data Gathering is the first step of the machine learning life cycle. The goal of this step is to identify and obtain all data-related problems.

In this step, we need to identify the different data sources, as data can be collected from various sources such as files, database, internet, or mobile devices. It is one of the most important steps of the life cycle. The quantity and quality of the collected data will determine the efficiency of the output. The more will be the data, the more accurate will be the prediction.

This step includes the below tasks:

- Identify various data sources
- Collect data
- Integrate the data obtained from different sources

By performing the above task, we get a coherent set of data, also called as a dataset. It will be used in further steps.

5.2.3.3. Data Preparation

After collecting the data, we need to prepare it for further steps. Data preparation is a step where we put our data into a suitable place and prepare it to use in our machine learning training.

In this step, first, we put all data together, and then randomize the ordering of data.

This step can be further divided into two processes:

- **Data exploration:** It is used to understand the nature of data that we have to work with. We need to understand the characteristics, format, and quality of data.

A better understanding of data leads to an effective outcome. In this, we find Correlations, general trends, and outliers.

- **Data pre-processing:** Now the next step is preprocessing of data for its analysis.

5.2.3.4. Data Wrangling

Data wrangling is the process of cleaning and converting raw data into a useable format. It is the process of cleaning the data, selecting the variable to use, and transforming the data in a proper format to make it more suitable for analysis in the next step. It is one of the most important steps of the complete process. Cleaning of data is required to address the quality issues.

It is not necessary that data we have collected is always of our use as some of the data may not be useful. In real-world applications, collected data may have various issues, including:

- Missing Values
- Duplicate data
- Invalid data
- Noise

So, we use various filtering techniques to clean the data.

It is mandatory to detect and remove the above issues because it can negatively affect the quality of the outcome.

5.2.3.5. Data Analysis

Now the cleaned and prepared data is passed on to the analysis step. This step involves:

- Selection of analytical techniques
- Building models
- Review the result

The aim of this step is to build a machine learning model to analyze the data using various analytical techniques and review the outcome. It starts with the determination of the type of the problems, where we select the machine learning techniques such as Classification, Regression, Cluster analysis, Association, etc. then build the model using prepared data, and evaluate the model.

Hence, in this step, we take the data and use machine learning algorithms to build the model.

5.2.3.6. Train the Model

Now the next step is to train the model, in this step we train our model to improve its performance for better outcome of the problem.

We use datasets to train the model using various machine learning algorithms. Training a model is required so that it can understand the various patterns, rules, and, features.

5.2.3.7. Test the model

Once our machine learning model has been trained on a given dataset, then we test the model. In this step, we check for the accuracy of our model by providing a test dataset to it.

Testing the model determines the percentage accuracy of the model as per the requirement of project or problem.

5.2.3.8. Deployment

The last step of machine learning life cycle is deployment, where we deploy the model in the real-world system.

If the above-prepared model is producing an accurate result as per our requirement with acceptable speed, then we deploy the model in the real system. But before deploying the project, we will check whether it is improving its performance using available data or not. The deployment phase is similar to making the final report for a project.

5.2.4. Types of Machine Learning

5.2.4.1. Supervised

Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output. The labelled data means some input data is already tagged with the correct output.

In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher.

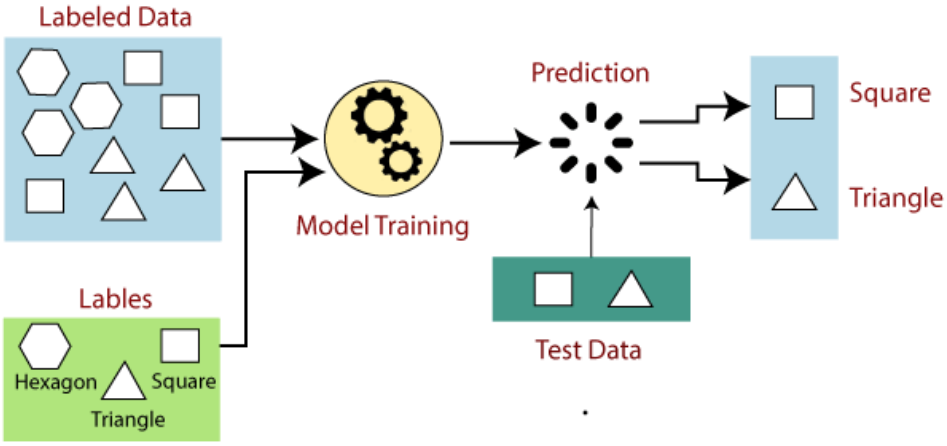
Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to find a mapping function to map the input variable(x) with

the output variable(y).
In the real-world, supervised learning can be used for **Risk Assessment, Image classification, Fraud Detection, spam filtering, etc.**

How Supervised Learning Works?

In supervised learning, models are trained using labelled dataset, where the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.

The working of Supervised learning can be easily understood by the below example and diagram:



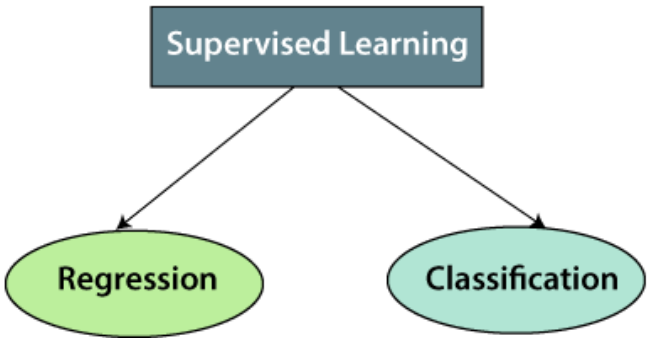
Suppose we have a dataset of different types of shapes which includes square, rectangle, triangle, and Polygon. Now the first step is that we need to train the model for each shape.

- If the given shape has four sides, and all the sides are equal, then it will be labelled as a **Square**.
- If the given shape has three sides, then it will be labelled as a **triangle.**
- If the given shape has six equal sides then it will be labelled as **hexagon.**

Now, after training, we test our model using the test set, and the task of the model is to identify the shape.
The machine is already trained on all types of shapes, and when it finds a new shape, it classifies the shape on the bases of a number of sides, and predicts the output.

Types of supervised Machine learning Algorithms:

Supervised learning can be further divided into two types of problems:



Regression algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc. Below are some popular Regression algorithms which come under supervised learning:

- Linear Regression
- Regression Trees
- Non-Linear Regression
- Bayesian Linear Regression
- Polynomial Regression

2. Classification

Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, True-false, etc.

- Spam Filtering,
- Random Forest
- Decision Trees
- Logistic Regression
- Support vector Machines

Advantages of Supervised learning:

- With the help of supervised learning, the model can predict the output on the basis of prior experiences.
- In supervised learning, we can have an exact idea about the classes of objects.
- Supervised learning model helps us to solve various real-world problems such as fraud detection, spam filtering, etc.

Disadvantages of supervised learning:

- Supervised learning models are not suitable for handling the complex tasks.
- Supervised learning cannot predict the correct output if the test data is different from the training dataset.
- Training required lots of computation times.
- In supervised learning, we need enough knowledge about the classes of object.

5.2.4.2. Unsupervised

As the name suggests, unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. It can be defined as: Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.

Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format.

Example: Suppose the unsupervised learning algorithm is given an input dataset containing images of different types of cats and dogs. The algorithm is never trained upon the given dataset, which means it does not have any idea about the features of the dataset. The task of the unsupervised learning algorithm is to identify the image features on their own. Unsupervised learning algorithm will perform this task by clustering the image dataset into the groups according to similarities between images.

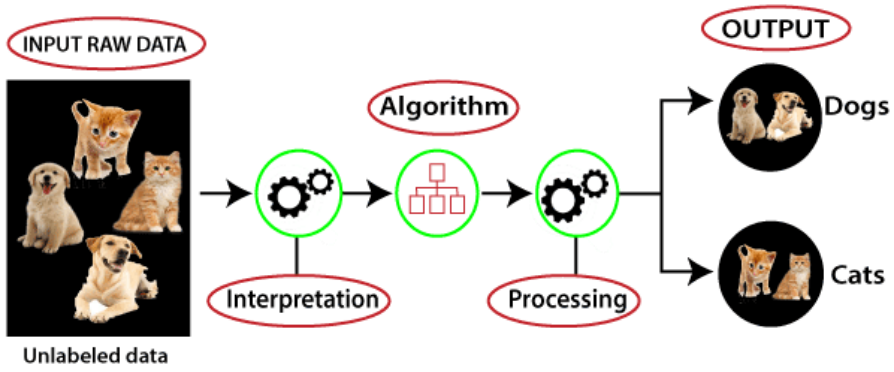
Why use Unsupervised Learning?

Below are some main reasons which describe the importance of Unsupervised Learning:

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.
- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

Working of Unsupervised Learning

Working of unsupervised learning can be understood by the below diagram:

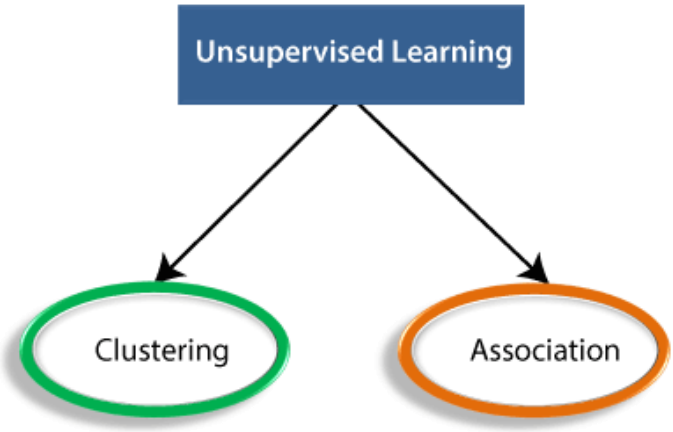


Here, we have taken an unlabeled input data, which means it is not categorized and corresponding outputs are also not given. Now, this unlabeled input data is fed to the machine learning model in order to train it. Firstly, it will interpret the raw data to find the hidden patterns from the data and then will apply suitable algorithms such as k-means clustering, Decision tree, etc.

Once it applies the suitable algorithm, the algorithm divides the data objects into groups according to the similarities and difference between the objects.

Types of Unsupervised Learning Algorithm:

The unsupervised learning algorithm can be further categorized into two types of problems:



Clustering: Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.

Association: An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item. A typical example of Association rule is Market Basket Analysis.

Unsupervised Learning algorithms:

Below is the list of some popular unsupervised learning algorithms:

- K-means clustering
- KNN (k-nearest neighbors)
- Hierarchal clustering
- Anomaly detection
- Neural Networks

- Principle Component Analysis
- Independent Component Analysis
- Apriori algorithm
- Singular value decomposition

Advantages of Unsupervised Learning

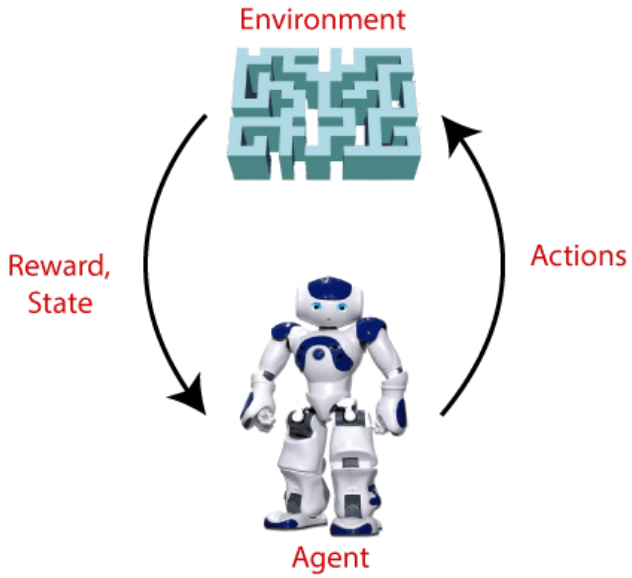
- Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
- Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to labeled data.

Disadvantages of Unsupervised Learning

- Unsupervised learning is intrinsically more difficult than supervised learning as it does not have corresponding output.
- The result of the unsupervised learning algorithm might be less accurate as input data is not labeled, and algorithms do not know the exact output in advance.

5.2.4.3. Reinforcement

- Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.
- In Reinforcement Learning, the agent learns automatically using feedbacks without any labeled data, unlike supervised learning.
- Since there is no labeled data, so the agent is bound to learn by its experience only.
- RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as game-playing, robotics, etc.
- The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.
- The agent learns with the process of hit and trial, and based on the experience, it learns to perform the task in a better way. Hence, we can say that "Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that." How a Robotic dog learns the movement of his arms is an example of Reinforcement learning.
- It is a core part of Artificial intelligence, and all AI agent works on the concept of reinforcement learning. Here we do not need to pre-program the agent, as it learns from its own experience without any human intervention.
- Example: Suppose there is an AI agent present within a maze environment, and his goal is to find the diamond. The agent interacts with the environment by performing some actions, and based on those actions, the state of the agent gets changed, and it also receives a reward or penalty as feedback.
- The agent continues doing these three things (take action, change state/remain in the same state, and get feedback), and by doing these actions, he learns and explores the environment.
- The agent learns that what actions lead to positive feedback or rewards and what actions lead to negative feedback penalty. As a positive reward, the agent gets a positive point, and as a penalty, it gets a negative point.



Terms used in Reinforcement Learning

- **Agent():** An entity that can perceive/explore the environment and act upon it.
- **Environment():** A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
- **Action():** Actions are the moves taken by an agent within the environment.
- **State():** State is a situation returned by the environment after each action taken by the agent.
- **Reward():** A feedback returned to the agent from the environment to evaluate the action of the agent.
- **Policy():** Policy is a strategy applied by the agent for the next action based on the current state.
- **Value():** It is expected long-term retuned with the discount factor and opposite to the short-term reward.
- **Q-value():** It is mostly similar to the value, but it takes one additional parameter as a current action (a).

Key Features of Reinforcement Learning

In RL, the agent is not instructed about the environment and what actions need to be taken.
It is based on the hit and trial process.
The agent takes the next action and changes states according to the feedback of the previous action.
The agent may get a delayed reward.
The environment is stochastic, and the agent needs to explore it to reach to get the maximum positive rewards.

Approaches to implement Reinforcement Learning

There are mainly three ways to implement reinforcement-learning in ML, which are:

1. **Value-based:**

The value-based approach is about to find the optimal value function, which is the maximum value at a state under any policy. Therefore, the agent expects the long-term return at any state(s) under policy π.

2. **Policy-based:**

Policy-based approach is to find the optimal policy for the maximum future rewards without using the value function. In this approach, the agent tries to apply such a policy that the action performed in each step helps to maximize the future reward.
The policy-based approach has mainly two types of policy:
Deterministic: The same action is produced by the policy (π) at any state.
Stochastic: In this policy, probability determines the produced action.

3. **Model-based:**

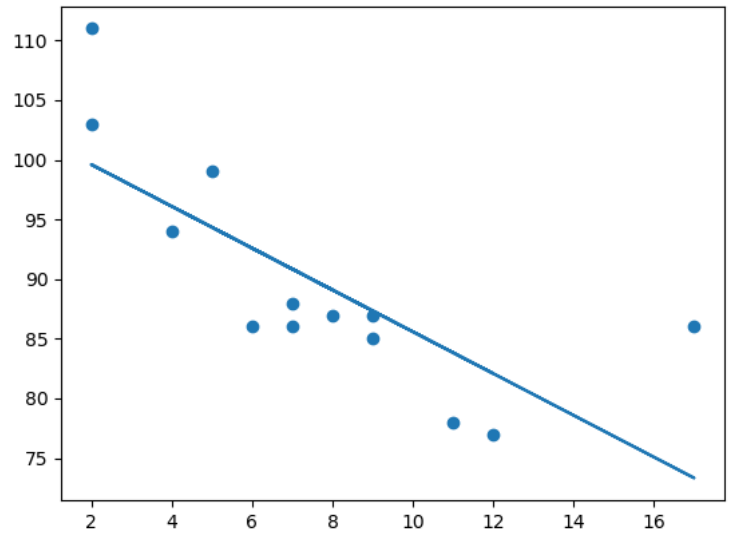
In the model-based approach, a virtual model is created for the environment, and the agent explores that environment to learn it. There is no particular solution or algorithm for this approach because the model

representation is different for each environment.

5.2.5. Machine Learning Algorithm

5.2.5.1. Linear regression

Linear regression uses the relationship between the data-points to draw a straight line through all them. This line can be used to predict future values.



In Machine Learning, predicting the future is very important.

How Does it Work?

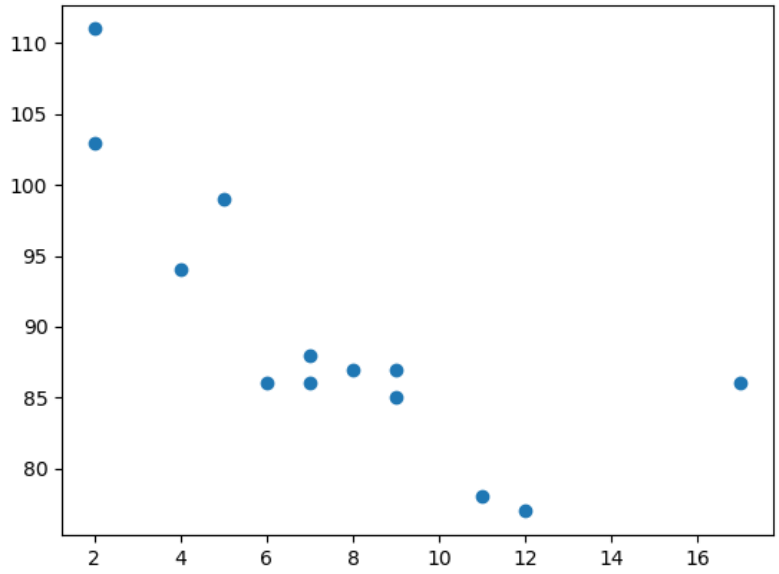
Python has methods for finding a relationship between data-points and to draw a line of linear regression. We will show you how to use these methods instead of going through the mathematic formula.

In the example below, the x-axis represents age, and the y-axis represents speed. We have registered the age and speed of 13 cars as they were passing a tollbooth. Let us see if the data we collected could be used in a linear regression:

Example: Start by drawing a scatter plot:

```
import matplotlib.pyplot as plt
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
plt.scatter(x, y)
plt.show()
```

Result:

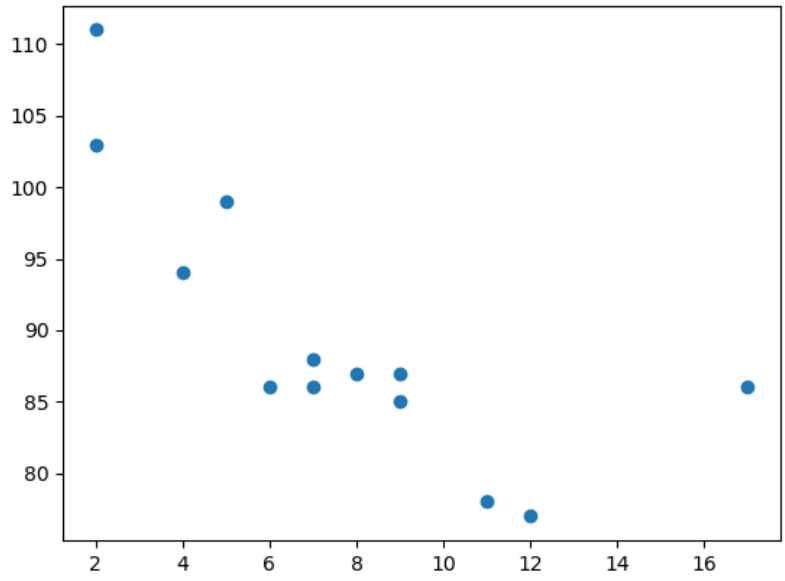


Example

Import scipy and draw the line of Linear Regression:

```
import matplotlib.pyplot as plt
from scipy import stats
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept, r, p, std_err = stats.linregress(x, y)
def myfunc(x):
  return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

Result:



Example Explained

- Import the modules you need.

```
import matplotlib.pyplot as plt
from scipy import stats
```

Create the arrays that represent the values of the x and y axis:

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

Execute a method that returns some important key values of Linear Regression:

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

Create a function that uses the slope and intercept values to return a new value. This new value represents where on the y-axis the corresponding x value will be placed:

```
def myfunc(x):
  return slope * x + intercept
```

Run each value of the x array through the function. This will result in a new array with new values for the y-axis:

```
mymodel = list(map(myfunc, x))
```

Draw the original scatter plot:

```
plt.scatter(x, y)
```

Draw the line of linear regression:

```
plt.plot(x, mymodel)
```

Display the diagram:

```
plt.show()
```

R for Relationship

It is important to know how the relationship between the values of the x-axis and the values of the y-axis is, if there are no relationship the linear regression can not be used to predict anything.

This relationship - the coefficient of correlation - is called r.

The r value ranges from -1 to 1, where 0 means no relationship, and 1 (and -1) means 100% related.

Python and the Scipy module will compute this value for you, all you have to do is feed it with the x and y values.

Example: How well does my data fit in a linear regression?

```
from scipy import stats
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept, r, p, std_err = stats.linregress(x, y)
print(r)
```

Note: The result -0.76 shows that there is a relationship, not perfect, but it indicates that we could use linear regression in future predictions.

Predict Future Values

Now we can use the information we have gathered to predict future values.

Example: Let us try to predict the speed of a 10 years old car.

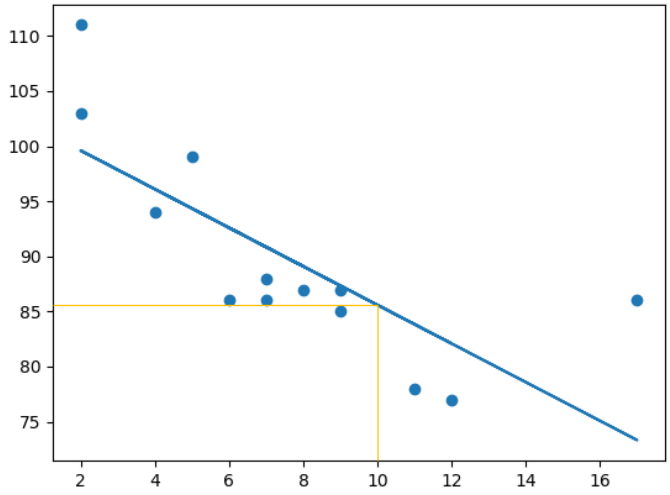
To do so, we need the same myfunc() function from the example above:

```
def myfunc(x):
  return slope * x + intercept
```

Example: Predict the speed of a 10 years old car.

```
from scipy import stats
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept, r, p, std_err = stats.linregress(x, y)
def myfunc(x):
  return slope * x + intercept
speed = myfunc(10)
print(speed)
```

The example predicted a speed at 85.6, which we also could read from the diagram:



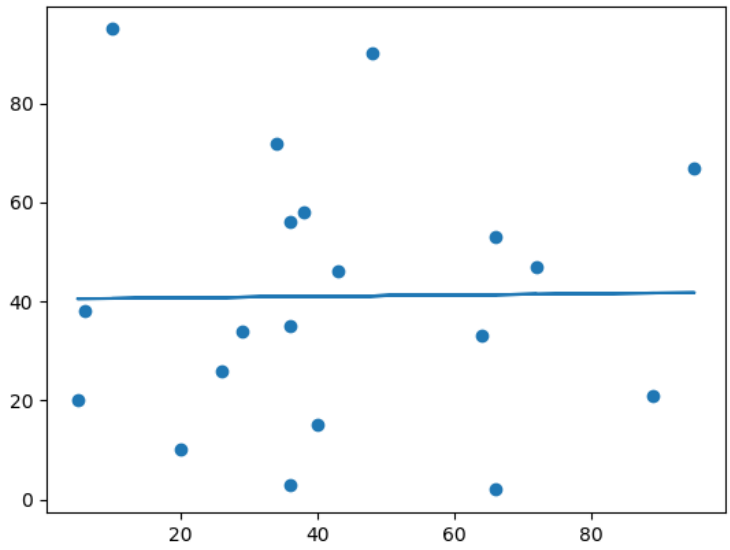
Bad Fit?

Let us create an example where linear regression would not be the best method to predict future values.

Example: These values for the x- and y-axis should result in a very bad fit for linear regression.

```
import matplotlib.pyplot as plt
from scipy import stats
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
slope, intercept, r, p, std_err = stats.linregress(x, y)
def myfunc(x):
  return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

Result:



And the r for relationship?

Example: You should get a very low r value.

```
import numpy
from scipy import stats
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
slope, intercept, r, p, std_err = stats.linregress(x, y)
print(r)
```

The result: 0.013 indicates a very bad relationship, and tells us that this data set is not suitable for linear regression.

5.2.5.2. Logistic regression

Logistic regression aims to solve classification problems. It does this by predicting categorical outcomes, unlike linear regression that predicts a continuous outcome.
In the simplest case there are two outcomes, which is called binomial, an example of which is predicting if a tumor is malignant or benign. Other cases have more than two outcomes to classify, in this case it is called multinomial. A common example for multinomial logistic regression would be predicting the class of an iris flower between 3 different species.
Here we will be using basic logistic regression to predict a binomial variable. This means it has only two possible outcomes.

How does it work?

In Python we have modules that will do the work for us. Start by importing the NumPy module.

```
import numpy
```

Store the independent variables in X.
Store the dependent variable in y.
Below is a sample dataset:

```
#X represents the size of a tumor in centimeters.
```

```
X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)

#Note: X has to be reshaped into a column from a row for the LogisticRegression() function to work.

#y represents whether or not the tumor is cancerous (0 for "No", 1 for "Yes").

y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

We will use a method from the sklearn module, so we will have to import that module as well:

```
from sklearn import linear_model
```

From the sklearn module we will use the LogisticRegression() method to create a logistic regression object.
This object has a method called fit() that takes the independent and dependent values as parameters and fills the regression object with data that describes the relationship:

```
logr = linear_model.LogisticRegression()

logr.fit(X,y)
```

Now we have a logistic regression object that is ready to whether a tumor is cancerous based on the tumor size:
#predict if tumor is cancerous where the size is 3.46mm:

```
predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))
```

Example: See the whole example in action.

```
import numpy

from sklearn import linear_model

#Reshaped for Logistic function.

X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)

y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear_model.LogisticRegression()

logr.fit(X,y)

#predict if tumor is cancerous where the size is 3.46mm:

predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))

print(predicted)
```

Result

```
 [0]
```

We have predicted that a tumor with a size of 3.46mm will not be cancerous.

Coefficient

In logistic regression the coefficient is the expected change in log-odds of having the outcome per unit change in X. This does not have the most intuitive understanding so let's use it to create something that makes more sense, odds.

Example: See the whole example in action.

```
import numpy

from sklearn import linear_model
```

```
#Reshaped for Logistic function.
X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
logr = linear_model.LogisticRegression()
logr.fit(X,y)
log_odds = logr.coef_
odds = numpy.exp(log_odds)
print(odds)
```

Result

```
[4.03541657]
```

This tells us that as the size of a tumor increases by 1mm the odds of it being a tumor increases by 4x.

Probability

The coefficient and intercept values can be used to find the probability that each tumor is cancerous.
Create a function that uses the model's coefficient and intercept values to return a new value. This new value represents probability that the given observation is a tumor:

```
def logit2prob(logr,x):
  log_odds = logr.coef_ * x + logr.intercept_
  odds = numpy.exp(log_odds)
  probability = odds / (1 + odds)
  return(probability)
```

Function Explained

To find the log-odds for each observation, we must first create a formula that looks similar to the one from linear regression, extracting the coefficient and the intercept.

```
log_odds = logr.coef_ * x + logr.intercept_
To then convert the log-odds to odds we must exponentiate the log-odds.
odds = numpy.exp(log_odds)
```

Now that we have the odds, we can convert it to probability by dividing it by 1 plus the odds.

```
probability = odds / (1 + odds)
```

Let us now use the function with what we have learned to find out the probability that each tumor is cancerous.
Example: See the whole example in action.

```
import numpy
from sklearn import linear_model
X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69, 5.88]).reshape(-1,1)
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
logr = linear_model.LogisticRegression()
logr.fit(X,y)
def logit2prob(logr, X):
```

```
    log_odds = logr.coef_ * X + logr.intercept_
    odds = numpy.exp(log_odds)
    probability = odds / (1 + odds)
    return(probability)
print(logit2prob(logr, X))
```

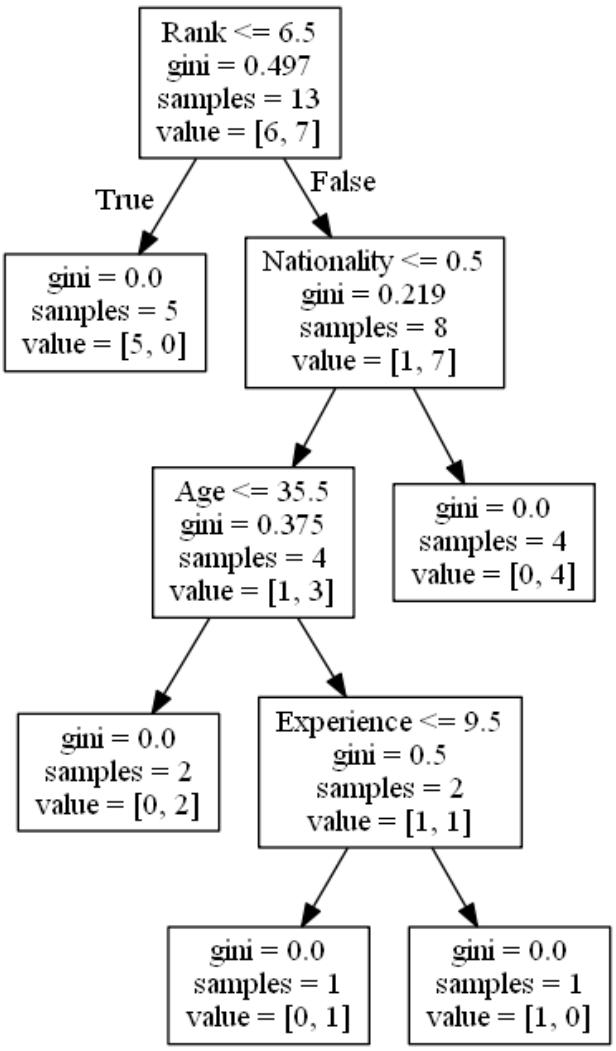
Result

```
[[0.60749955]
 [0.19268876]
 [0.12775886]
 [0.00955221]
 [0.08038616]
 [0.07345637]
 [0.88362743]
 [0.77901378]
 [0.88924409]
 [0.81293497]
 [0.57719129]
 [0.96664243]]
```

Results Explained

3.78 0.61: The probability that a tumor with the size 3.78cm is cancerous is 61%.
2.44 0.19: The probability that a tumor with the size 2.44cm is cancerous is 19%.
2.09 0.13: The probability that a tumor with the size 2.09cm is cancerous is 13%.

5.2.5.3. Decision tree



A Decision Tree is a Flow Chart, and can help you make decisions based on previous experience.
In the example, a person will try to decide if he/she should go to a comedy show or not.
Luckily our example person has registered every time there was a comedy show in town, and registered some information about the comedian, and also registered if he/she went or not.

Age	Experience	Rank	Nationality	Go
36	10	9	UK	NO
42	12	4	USA	NO
23	4	6	N	NO
52	4	4	USA	NO
43	21	8	USA	YES
44	14	5	UK	NO
66	3	7	N	YES
35	14	9	UK	YES
52	13	7	N	YES
35	5	9	N	YES
24	3	5	USA	NO
18	3	7	UK	YES
45	9	9	UK	YES

Now, based on this data set, Python can create a decision tree that can be used to decide if any new shows are worth attending to.

How Does it Work?

First, read the dataset with pandas:
Example: Read and print the data set.

```
import pandas
df = pandas.read_csv("data.csv")
print(df)
```

To make a decision tree, all data has to be numerical.
We have to convert the non numerical columns 'Nationality' and 'Go' into numerical values.
Pandas has a map() method that takes a dictionary with information on how to convert the values.

```
{'UK': 0, 'USA': 1, 'N': 2}
```

Means convert the values 'UK' to 0, 'USA' to 1, and 'N' to 2.
Example: Change string values into numerical values.

```
d = {'UK': 0, 'USA': 1, 'N': 2}
df['Nationality'] = df['Nationality'].map(d)
d = {'YES': 1, 'NO': 0}
df['Go'] = df['Go'].map(d)
print(df)
```

Then we have to separate the feature columns from the target column.
The feature columns are the columns that we try to predict from, and the target column is the column with the values we try to predict.
Example: X is the feature columns, y is the target column.

```
features = ['Age', 'Experience', 'Rank', 'Nationality']
X = df[features]
y = df['Go']
print(X)
print(y)
```

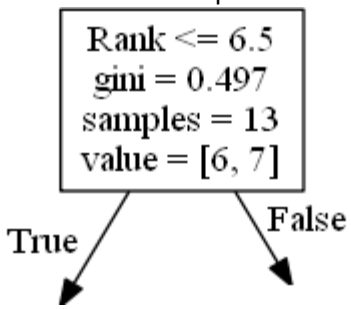
Now we can create the actual decision tree, fit it with our details. Start by importing the modules we need:
Example: Create and display a Decision Tree.

```
import pandas
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
df = pandas.read_csv("data.csv")
d = {'UK': 0, 'USA': 1, 'N': 2}
df['Nationality'] = df['Nationality'].map(d)
d = {'YES': 1, 'NO': 0}
df['Go'] = df['Go'].map(d)
features = ['Age', 'Experience', 'Rank', 'Nationality']
X = df[features]
```

```
y = df['Go']
dtree = DecisionTreeClassifier()
dtree = dtree.fit(X, y)
tree.plot_tree(dtree, feature_names=features)
```

Result Explained

The decision tree uses your earlier decisions to calculate the odds for you to wanting to go see a comedian or not.
Let us read the different aspects of the decision tree:



Rank

- Rank <= 6.5 means that every comedian with a rank of 6.5 or lower will follow the True arrow (to the left), and the rest will follow the False arrow (to the right).
- gini = 0.497 refers to the quality of the split, and is always a number between 0.0 and 0.5, where 0.0 would mean all of the samples got the same result, and 0.5 would mean that the split is done exactly in the middle.
- samples = 13 means that there are 13 comedians left at this point in the decision, which is all of them since this is the first step.
- value = [6, 7] means that of these 13 comedians, 6 will get a "NO", and 7 will get a "GO".

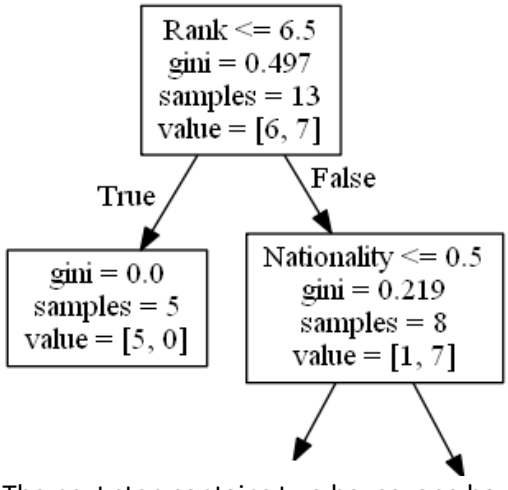
Gini

There are many ways to split the samples, we use the GINI method in this lesson.
The Gini method uses this formula:

```
Gini = 1 - (x/n)2 + (y/n)2
```

Where x is the number of positive answers("GO"), n is the number of samples, and y is the number of negative answers ("NO"), which gives us this calculation:

```
1 - (7 / 13)2 + (6 / 13)2 = 0.497
```

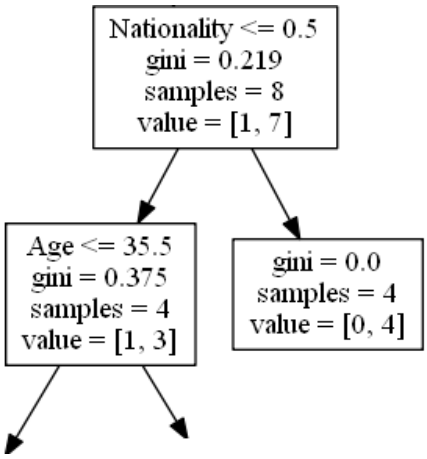


The next step contains two boxes, one box for the comedians with a 'Rank' of 6.5 or lower, and one box with the rest.

True - 5 Comedians End Here:
gini = 0.0 means all of the samples got the same result.
samples = 5 means that there are 5 comedians left in this branch (5 comedian with a Rank of 6.5 or lower).
value = [5, 0] means that 5 will get a "NO" and 0 will get a "GO".
False - 8 Comedians Continue:

Nationality

- Nationality <= 0.5 means that the comedians with a nationality value of less than 0.5 will follow the arrow to the left (which means everyone from the UK,), and the rest will follow the arrow to the right.
- gini = 0.219 means that about 22% of the samples would go in one direction.
- samples = 8 means that there are 8 comedians left in this branch (8 comedian with a Rank higher than 6.5).
- value = [1, 7] means that of these 8 comedians, 1 will get a "NO" and 7 will get a "GO".



True - 4 Comedians Continue:

Age

Age <= 35.5 means that comedians at the age of 35.5 or younger will follow the arrow to the left, and the rest will follow the arrow to the right.

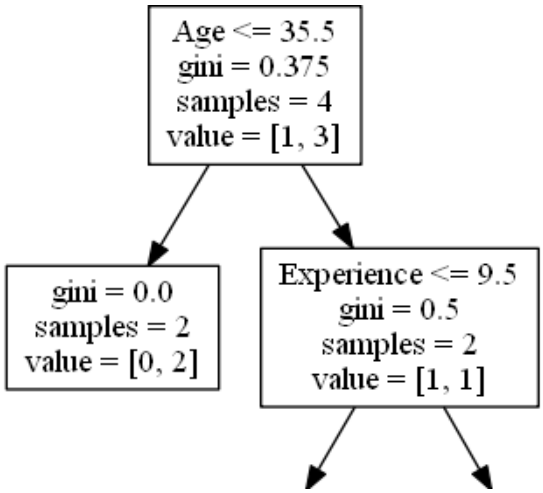
gini = 0.375 means that about 37,5% of the samples would go in one direction.

samples = 4 means that there are 4 comedians left in this branch (4 comedians from the UK).

value = [1, 3] means that of these 4 comedians, 1 will get a "NO" and 3 will get a "GO".

False - 4 Comedians End Here:

- gini = 0.0 means all of the samples got the same result.
- samples = 4 means that there are 4 comedians left in this branch (4 comedians not from the UK).
- value = [0, 4] means that of these 4 comedians, 0 will get a "NO" and 4 will get a "GO".



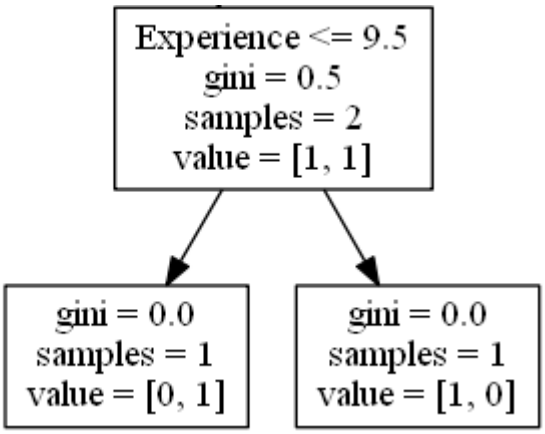
True - 2 Comedians End Here:

- gini = 0.0 means all of the samples got the same result.
- samples = 2 means that there are 2 comedians left in this branch (2 comedians at the age 35.5 or younger).
- value = [0, 2] means that of these 2 comedians, 0 will get a "NO" and 2 will get a "GO".

False - 2 Comedians Continue:

Experience

- Experience <= 9.5 means that comedians with 9.5 years of experience, or less, will follow the arrow to the left, and the rest will follow the arrow to the right.
- gini = 0.5 means that 50% of the samples would go in one direction.
- samples = 2 means that there are 2 comedians left in this branch (2 comedians older than 35.5).
- value = [1, 1] means that of these 2 comedians, 1 will get a "NO" and 1 will get a "GO".



True - 1 Comedian Ends Here:

- gini = 0.0 means all of the samples got the same result.
- samples = 1 means that there is 1 comedian left in this branch (1 comedian with 9.5 years of experience or less).
- value = [0, 1] means that 0 will get a "NO" and 1 will get a "GO".

False - 1 Comedian Ends Here:

- gini = 0.0 means all of the samples got the same result.
- samples = 1 means that there is 1 comedians left in this branch (1 comedian with more than 9.5 years of experience).
- value = [1, 0] means that 1 will get a "NO" and 0 will get a "GO".

Predict Values
We can use the Decision Tree to predict new values.
Example: Should I go see a show starring a 40 years old American comedian, with 10 years of experience, and a comedy ranking of 7?
Example: Use predict() method to predict new values.

```
print(dtree.predict([[40, 10, 7, 1]]))
```

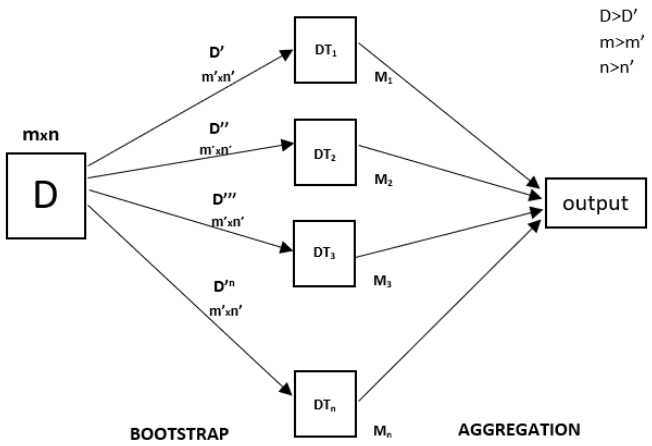
Example: What would the answer be if the comedy rank was 6?

```
print(dtree.predict([[40, 10, 6, 1]]))
```

Different Results
You will see that the Decision Tree gives you different results if you run it enough times, even if you feed it with the same data.
That is because the Decision Tree does not give us a 100% certain answer. It is based on the probability of an outcome, and the answer will vary.

5.2.5.4. Random forest

Every decision tree has high variance, but when we combine all of them together in parallel then the resultant variance is low as each decision tree gets perfectly trained on that particular sample data, and hence the output doesn't depend on one decision tree but on multiple decision trees. In the case of a classification problem, the final output is taken by using the majority voting classifier. In the case of a regression problem, the final output is the mean of all the outputs. This part is called **Aggregation.**



Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as **bagging**.
The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.
Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.
We need to approach the Random Forest regression technique like any other machine learning technique

- Design a specific question or data and get the source to determine the required data.
- Make sure the data is in an accessible format else convert it to the required format.
- Specify all noticeable anomalies and missing data points that may be required to achieve the required data.
- Create a machine-learning model.
- Set the baseline model that you want to achieve
- Train the data machine learning model.
- Provide an insight into the model with test data

- Now compare the performance metrics of both the test data and the predicted data from the model.
- If it doesn't satisfy your expectations, you can try improving your model accordingly or dating your data, or using another data modeling technique.
- At this stage, you interpret the data you have gained and report accordingly.

You will be using a similar sample technique in the below example.

Implementation:

Step 1: Import the required libraries.

```
# Importing the 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)
```

Output:

```
            Position  Level   Salary
0   Business Analyst      1    45000
1  Junior Consultant      2    50000
2  Senior Consultant      3    60000
3            Manager      4    80000
4    Country Manager      5   110000
5     Region Manager      6   150000
6            Partner      7   200000
7     Senior Partner      8   300000
8            C-level      9   500000
9                CEO     10  1000000
```

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.

```
[[ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]]
```

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)
```

Output:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
           oob_score=False, random_state=0, verbose=0, warm_start=False)
```

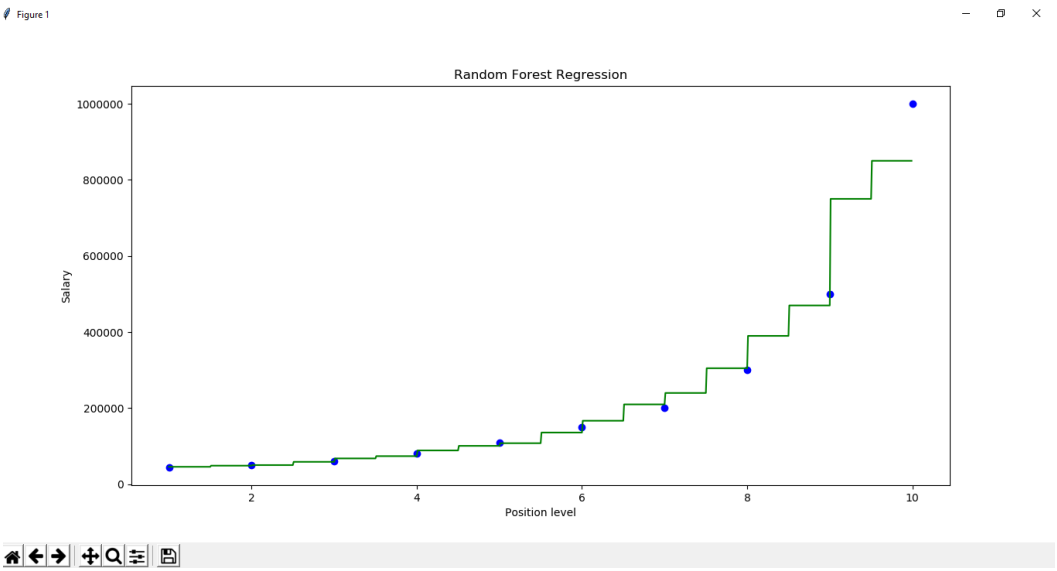
Step 5: Predicting a new result

Y_pred = regressor.predict(np.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:



5.2.5.5. K Nearest Neighbor

KNN is a simple, supervised machine learning (ML) algorithm that can be used for classification or regression tasks - and is also frequently used in missing value imputation. It is based on the idea that the observations closest to a given data point are the most "similar" observations in a data set, and we can therefore classify unforeseen points based on the values of the closest existing points. By choosing K, the user can select the number of nearby observations to use in the algorithm.

Here, we will show you how to implement the KNN algorithm for classification, and show how different values of K affect the results.

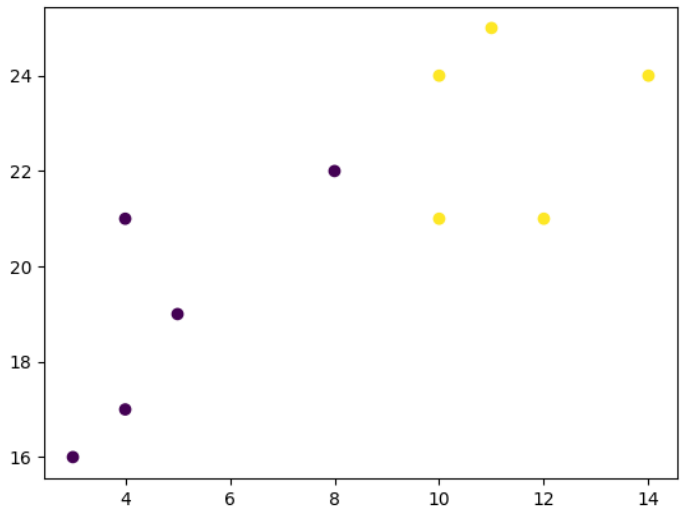
How does it work?

K is the number of nearest neighbors to use. For classification, a majority vote is used to determined which class a new observation should fall into. Larger values of K are often more robust to outliers and produce more stable decision boundaries than very small values (K=3 would be better than K=1, which might produce undesirable results.

Example: Start by visualizing some data points.

```
import matplotlib.pyplot as plt
x = [4, 5, 10, 4, 3, 11, 14 , 8, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
classes = [0, 0, 1, 0, 0, 1, 1, 0, 1, 1]
plt.scatter(x, y, c=classes)
plt.show()
```

Result



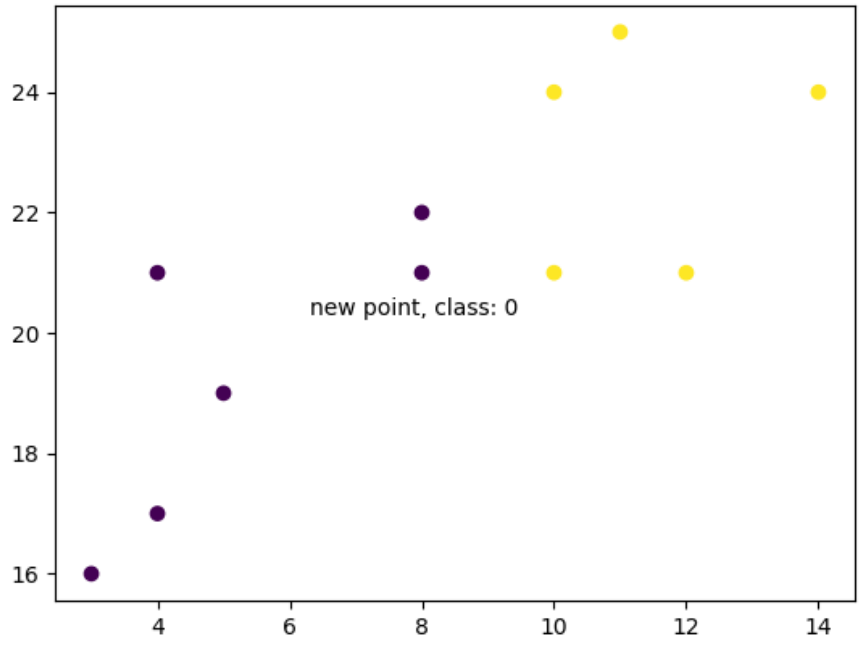
Now we fit the KNN algorithm with K=1:

```
from sklearn.neighbors import KNeighborsClassifier
data = list(zip(x, y))
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(data, classes)
```

And use it to classify a new data point:
Example

```
new_x = 8
new_y = 21
new_point = [(new_x, new_y)]
prediction = knn.predict(new_point)
plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```

Result

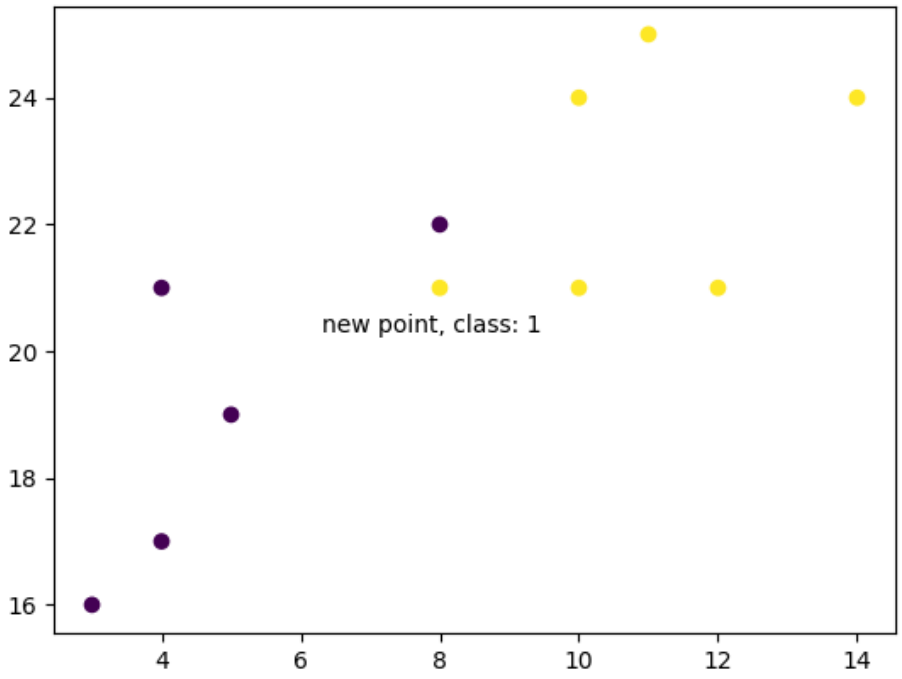


Now we do the same thing, but with a higher K value which changes the prediction:

Example

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(data, classes)
prediction = knn.predict(new_point)
plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```

Result



Example Explained

Import the modules you need.

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
```

Create arrays that resemble variables in a dataset. We have two input features (x and y) and then a target class (class). The input features that are pre-labeled with our target class will be used to predict the class of new data. Note that while we only use two input features here, this method will work with any number of variables:

```
x = [4, 5, 10, 4, 3, 11, 14 , 8, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
classes = [0, 0, 1, 0, 0, 1, 1, 0, 1, 1]
```

Turn the input features into a set of points:

```
data = list(zip(x, y))
print(data)
```

Result:

```
[(4, 21), (5, 19), (10, 24), (4, 17), (3, 16), (11, 25), (14, 24), (8, 22), (10, 21), (12, 21)]
```

Using the input features and target class, we fit a KNN model on the model using 1 nearest neighbor:

```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(data, classes)
```

Then, we can use the same KNN object to predict the class of new, unforeseen data points. First we create new x and y features, and then call knn.predict() on the new data point to get a class of 0 or 1:

```
new_x = 8
```

```
new_y = 21
new_point = [(new_x, new_y)]
prediction = knn.predict(new_point)
print(prediction)
```

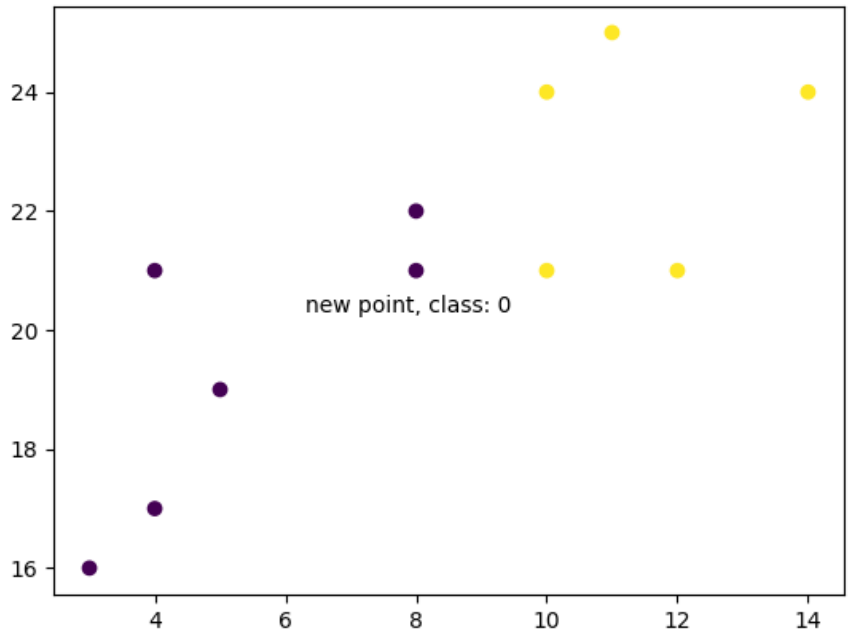
Result:

```
[0]
```

When we plot all the data along with the new point and class, we can see it's been labeled blue with the 1 class. The text annotation is just to highlight the location of the new point:

```
plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```

Result:



However, when we changes the number of neighbors to 5, the number of points used to classify our new point changes. As a result, so does the classification of the new point:

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(data, classes)
prediction = knn.predict(new_point)
print(prediction)
```

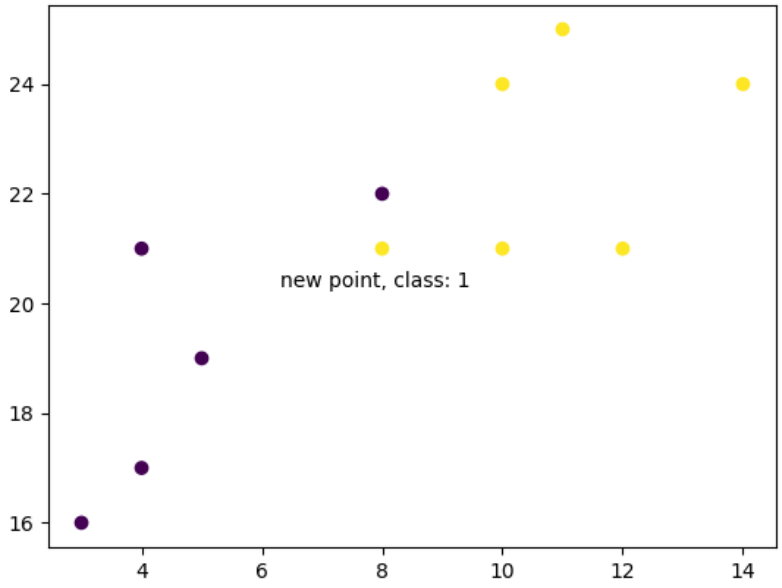
Result:

```
[1]
```

When we plot the class of the new point along with the older points, we note that the color has changed based on the associated class label:

```
plt.scatter(x + [new_x], y + [new_y], c=classes + [prediction[0]])
plt.text(x=new_x-1.7, y=new_y-0.7, s=f"new point, class: {prediction[0]}")
plt.show()
```

Result:

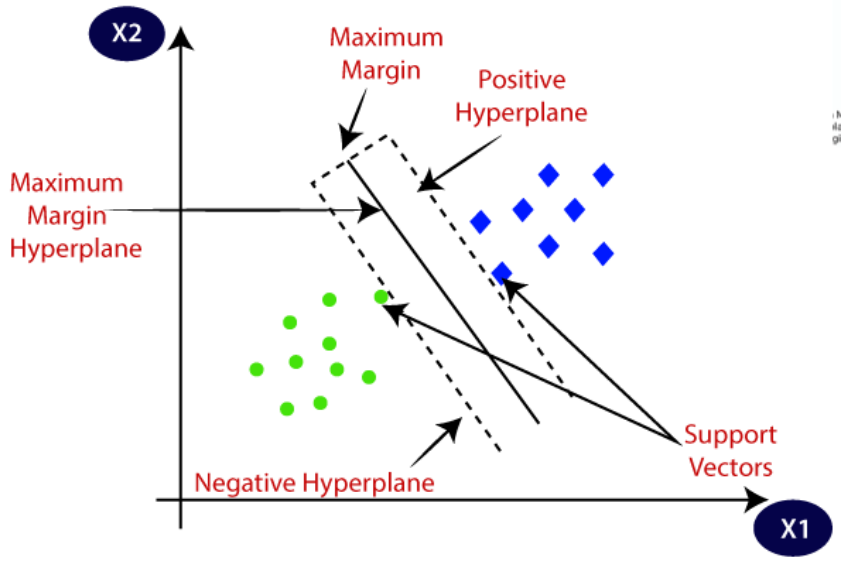


5.2.5.6. Support vector machines

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

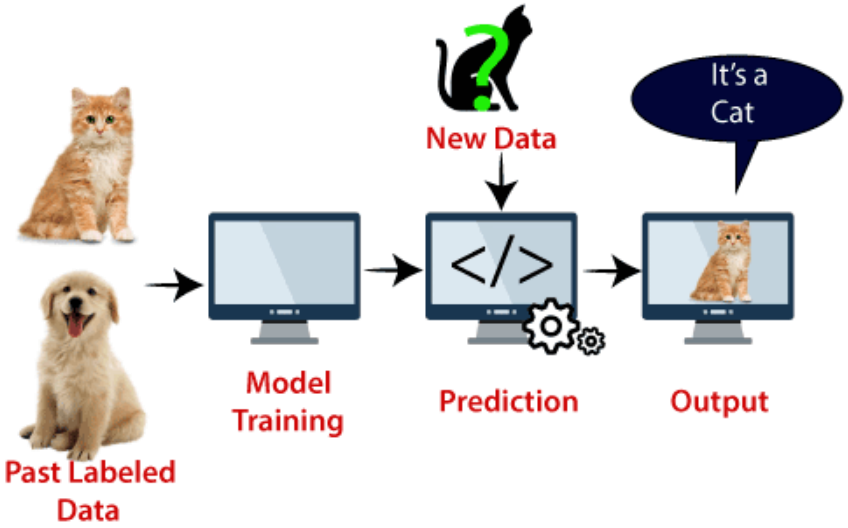
The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a

strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



SVM algorithm can be used for Face detection, image classification, text categorization, etc.

Types of SVM

SVM can be of two types:

Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

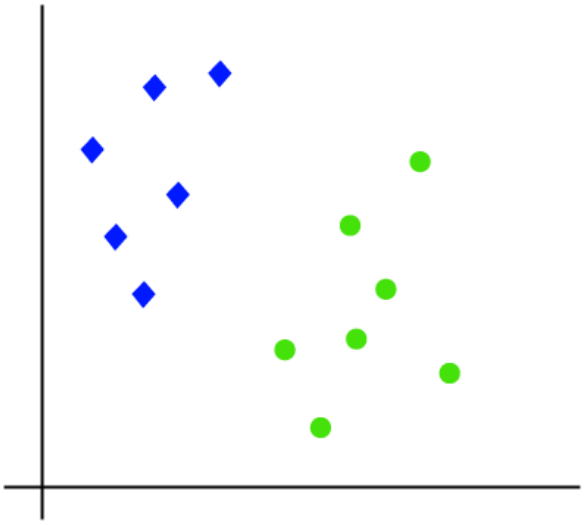
Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

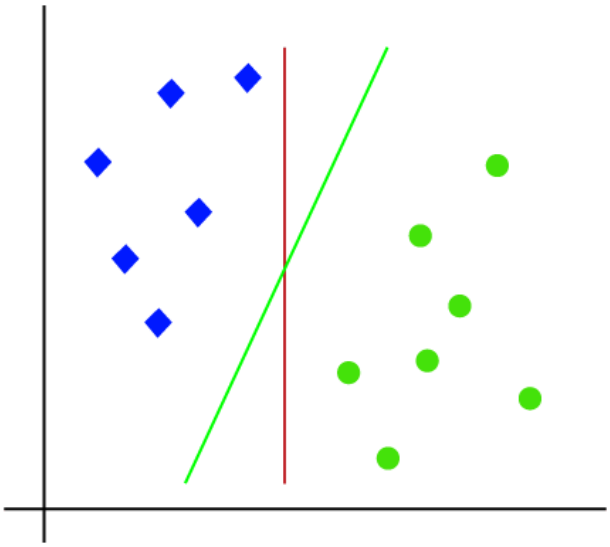
How does SVM works?

Linear SVM:

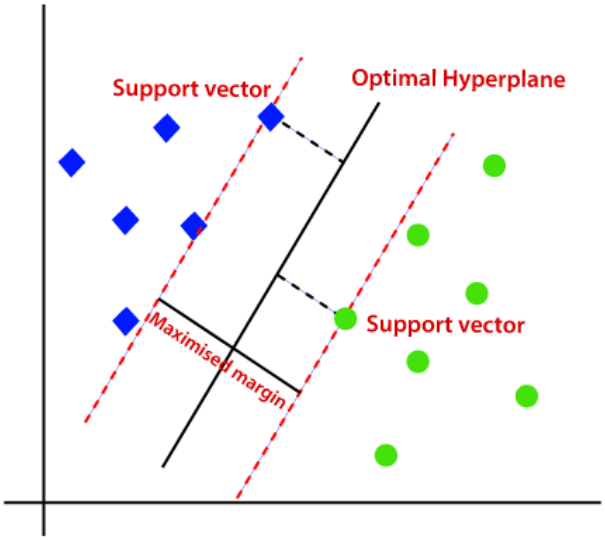
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

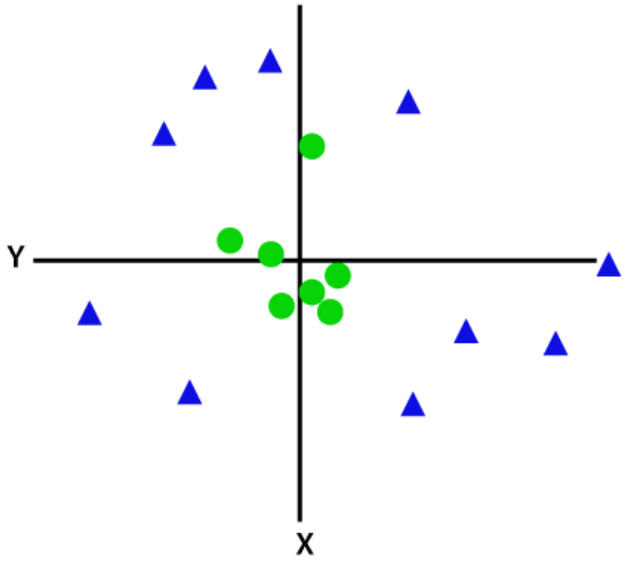


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a hyperplane. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as margin. And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane.



Non-Linear SVM:

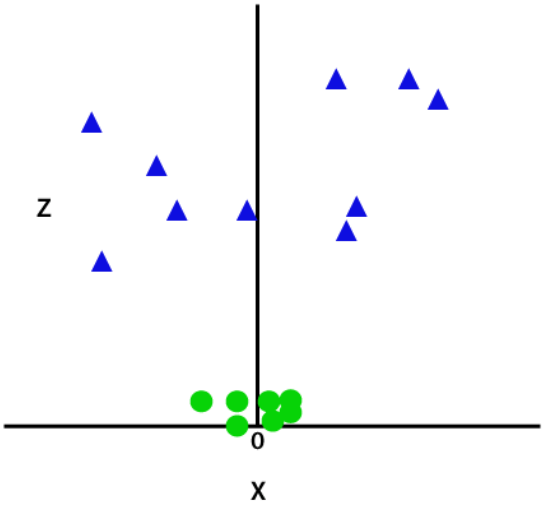
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



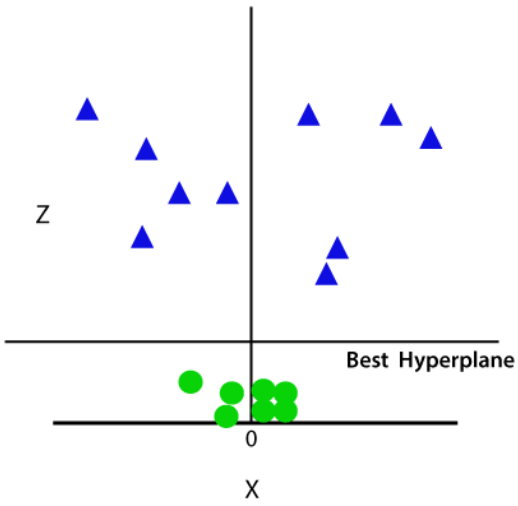
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$$z=x2 +y2$$

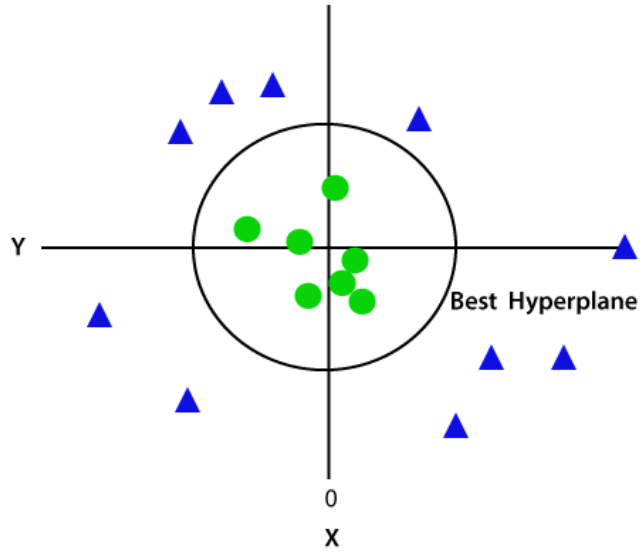
By adding the third dimension, the sample space will become as below image:



So now, SVM will divide the datasets into classes in the following way. Consider the below image:



Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:



Hence we get a circumference of radius 1 in case of non-linear data.

Python Implementation of Support Vector Machine

Now we will implement the SVM algorithm using Python. Here we will use the same dataset user_data, which we have used in Logistic regression and KNN classification.

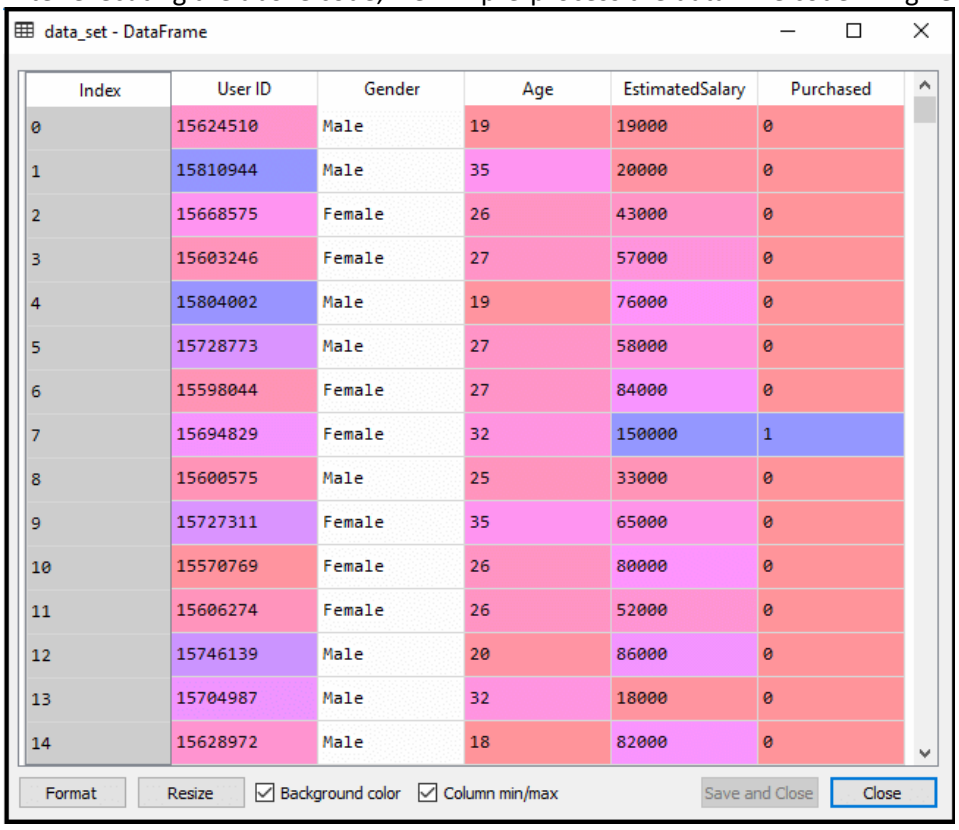
Data Pre-processing step

Till the Data pre-processing step, the code will remain the same. Below is the code:

```
#Data Pre-processing Step
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

```

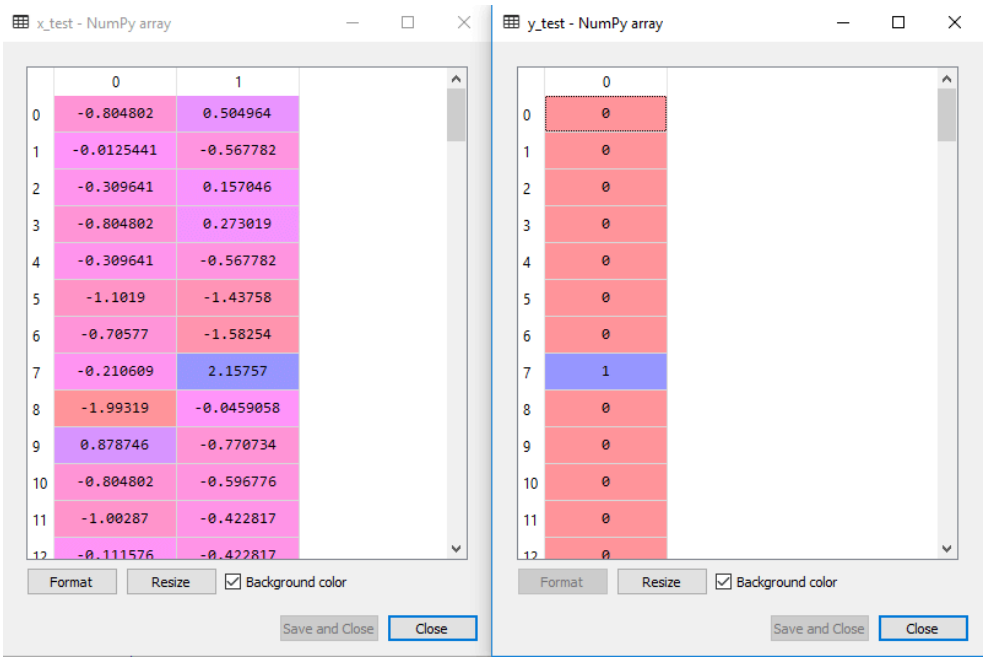
```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

After executing the above code, we will pre-process the data. The code will give the dataset as:



Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0

The scaled output for the test set will be:



Fitting the SVM classifier to the training set:

Now the training set will be fitted to the SVM classifier. To create the SVM classifier, we will import SVC class from Sklearn.svm library. Below is the code for it:

```
from sklearn.svm import SVC # "Support vector classifier"
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)
```

In the above code, we have used kernel='linear', as here we are creating SVM for linearly separable data. However, we can change it for non-linear data. And then we fitted the classifier to the training dataset(x_train, y_train)

Output:

```
Out[8]:
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=0,
  shrinking=True, tol=0.001, verbose=False)
```

The model performance can be altered by changing the value of C(Regularization factor), gamma, and kernel.

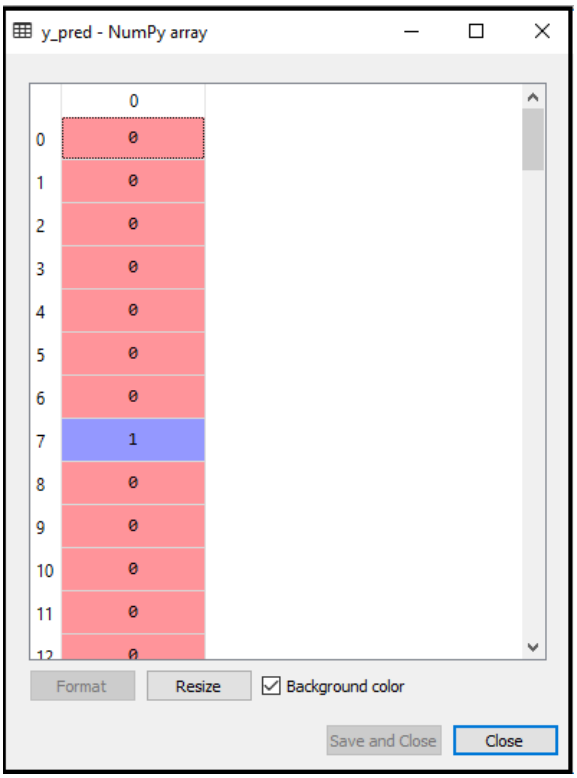
Predicting the test set result:

Now, we will predict the output for test set. For this, we will create a new vector y_pred. Below is the code for it:

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

After getting the y_pred vector, we can compare the result of y_pred and y_test to check the difference between the actual value and predicted value.

Output: Below is the output for the prediction of the test set:



Creating the confusion matrix:

Now we will see the performance of the SVM classifier that how many incorrect predictions are there as compared to the Logistic regression classifier. To create the confusion matrix, we need to import the confusion_matrix function of the sklearn library. After importing the function, we will call it using a new variable cm. The function takes two parameters, mainly y_true(the actual values) and y_pred (the targeted value return by the classifier). Below is the code for it:

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
```

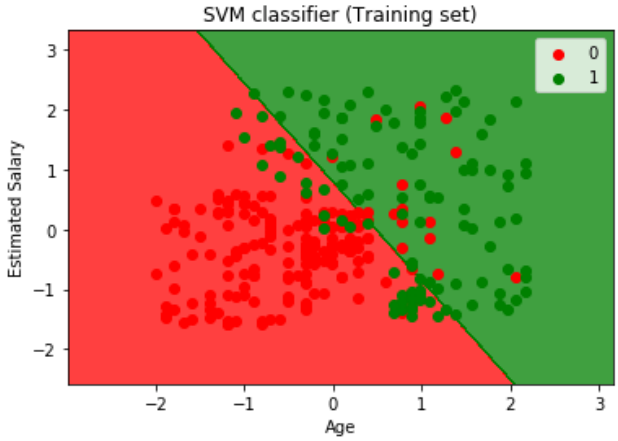
Output:

As we can see in the above output image, there are 66+24= 90 correct predictions and 8+2= 10 correct predictions. Therefore we can say that our SVM model improved as compared to the Logistic regression model.

- **Visualizing the training set result:** Now we will visualize the training set result, below is the code for it:

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output: By executing the above code, we will get the output as:



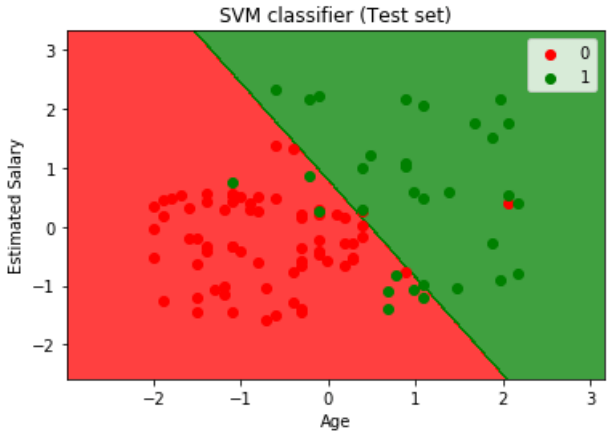
As we can see, the above output is appearing similar to the Logistic regression output. In the output, we got the straight line as hyperplane because we have used a linear kernel in the classifier. And we have also discussed above that for the 2d space, the hyperplane in SVM is a straight line.

- **Visualizing the test set result:**

```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
```

```
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output: By executing the above code, we will get the output as:



As we can see in the above output image, the SVM classifier has divided the users into two regions (Purchased or Not purchased). Users who purchased the SUV are in the red region with the red scatter points. And users who did not purchase the SUV are in the green region with green scatter points. The hyperplane has divided the two classes into Purchased and not purchased variable.

Learning Outcome 5.3: Building data model

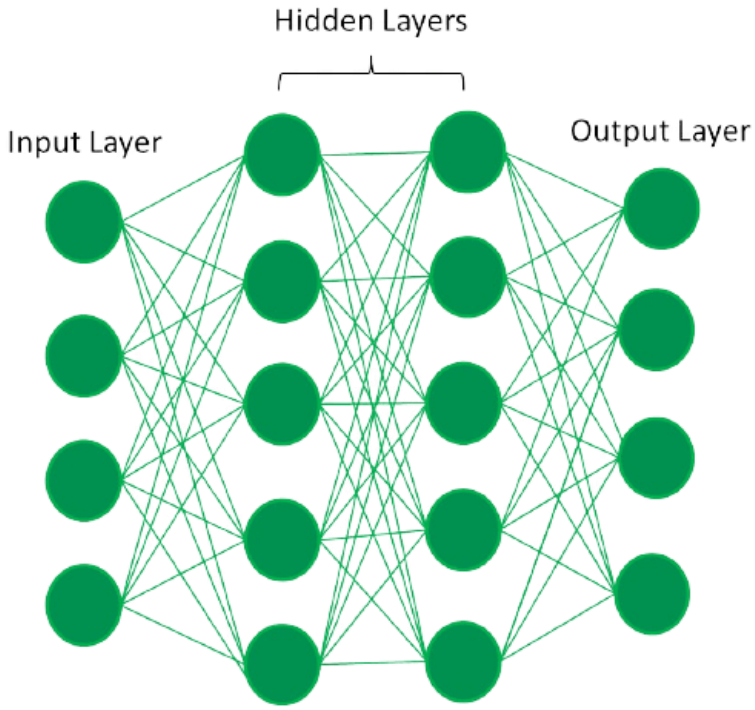
5.3.1. Artificial Neural networks

5.3.1.1. Definition

Artificial Neural Networks contain artificial neurons which are called units. These units are arranged in a series of layers that together constitute the whole Artificial Neural Network in a system. A layer can have only a dozen units or millions of units as this depends on how the complex neural networks will be required to learn the hidden patterns in the dataset. Commonly, Artificial Neural Network has an input layer, an output layer as well as hidden layers. The input layer receives data from the outside world which the neural network needs to analyze or learn about. Then this data passes through one or multiple hidden layers that transform the input into data that is valuable for the output layer. Finally, the output layer provides an output in the form of a response of the Artificial Neural Networks to input data provided.

In the majority of neural networks, units are interconnected from one layer to another. Each of these connections has weights that determine the influence of one unit on another unit. As the data transfers from one unit to

another, the neural network learns more and more about the data which eventually results in an output from the output layer.



The structures and operations of human neurons serve as the basis for artificial neural networks. It is also known as neural networks or neural nets. The input layer of an artificial neural network is the first layer, and it receives input from external sources and releases it to the hidden layer, which is the second layer. In the hidden layer, each neuron receives input from the previous layer neurons, computes the weighted sum, and sends it to the neurons in the next layer. These connections are weighted means effects of the inputs from the previous layer are optimized more or less by assigning different-different weights to each input and it is adjusted during the training process by optimizing these weights for improved model performance.

Artificial neurons vs Biological neurons

The concept of artificial neural networks comes from biological neurons found in animal brains So they share a lot of similarities in structure and function wise.

Structure: The structure of artificial neural networks is inspired by biological neurons. A biological neuron has a cell body or soma to process the impulses, dendrites to receive them, and an axon that transfers them to other neurons. The input nodes of artificial neural networks receive input signals, the hidden layer nodes compute these input signals, and the output layer nodes compute the final output by processing the hidden layer's results using activation functions.

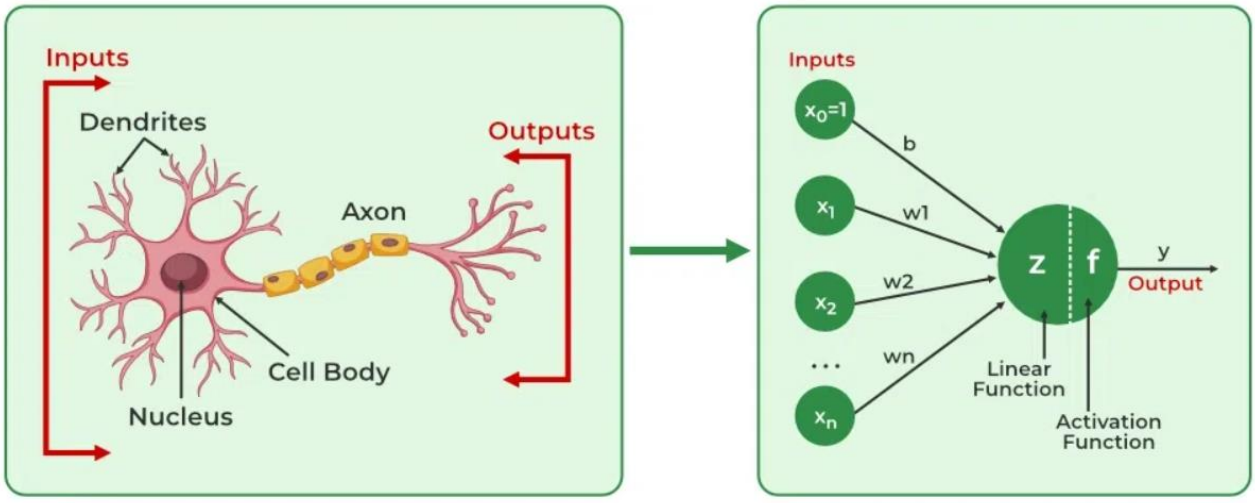
Biological Neuron	Artificial Neuron
Dendrite	Inputs
Cell nucleus or Soma	Nodes
Synapses	Weights
Axon	Output

Synapses: Synapses are the links between biological neurons that enable the transmission of impulses from dendrites to the cell body. Synapses are the weights that join the one-layer nodes to the next-layer nodes in artificial neurons. The strength of the links is determined by the weight value.

Learning: In biological neurons, learning happens in the cell body nucleus or soma, which has a nucleus that helps to process the impulses. An action potential is produced and travels through the axons if the impulses are powerful enough to reach the threshold. This becomes possible by synaptic plasticity, which represents the ability of synapses to become stronger or weaker over time in reaction to changes in their activity. In artificial neural networks, backpropagation is a technique used for learning, which adjusts the weights between nodes according to the error or differences between predicted and actual outcomes.

Biological Neuron	Artificial Neuron
Synaptic plasticity	Backpropagations

Activation: In biological neurons, activation is the firing rate of the neuron which happens when the impulses are strong enough to reach the threshold. In artificial neural networks, A mathematical function known as an activation function maps the input to the output, and executes activations.



How do Artificial Neural Networks learn?

Artificial neural networks are trained using a training set. For example, suppose you want to teach an ANN to recognize a cat. Then it is shown thousands of different images of cats so that the network can learn to identify a cat. Once the neural network has been trained enough using images of cats, then you need to check if it can identify cat images correctly. This is done by making the ANN classify the images it is provided by deciding whether they are cat images or not. The output obtained by the ANN is corroborated by a human-provided description of whether the image is a cat image or not. If the ANN identifies incorrectly then back-propagation is used to adjust whatever it has learned during training. Backpropagation is done by fine-tuning the weights of the connections in ANN units based on the error rate obtained. This process continues until the artificial neural network can correctly recognize a cat in an image with minimal possible error rates.

What are the types of Artificial Neural Networks?

Feedforward Neural Network: The feedforward neural network is one of the most basic artificial neural networks. In this ANN, the data or the input provided travels in a single direction. It enters into the ANN through the input layer and exits through the output layer while hidden layers may or may not exist. So the feedforward neural network has a front-propagated wave only and usually does not have backpropagation.

Convolutional Neural Network: A Convolutional neural network has some similarities to the feed-forward neural network, where the connections between units have weights that determine the influence of one unit on another unit. But a CNN has one or more than one convolutional layer that uses a convolution operation on the input and then passes the result obtained in the form of output to the next layer. CNN has applications in speech and image processing which is particularly useful in computer vision.

Modular Neural Network: A Modular Neural Network contains a collection of different neural networks that work independently towards obtaining the output with no interaction between them. Each of the different neural networks performs a different sub-task by obtaining unique inputs compared to other networks. The advantage of

this modular neural network is that it breaks down a large and complex computational process into smaller components, thus decreasing its complexity while still obtaining the required output.

Radial basis function Neural Network: Radial basis functions are those functions that consider the distance of a point concerning the center. RBF functions have two layers. In the first layer, the input is mapped into all the Radial basis functions in the hidden layer and then the output layer computes the output in the next step. Radial basis function nets are normally used to model the data that represents any underlying trend or function.

Recurrent Neural Network: The Recurrent Neural Network saves the output of a layer and feeds this output back to the input to better predict the outcome of the layer. The first layer in the RNN is quite similar to the feed-forward neural network and the recurrent neural network starts once the output of the first layer is computed. After this layer, each unit will remember some information from the previous step so that it can act as a memory cell in performing computations.

5.3.1.2. Use case implementation steps

1. **Define the problem:** The first step is to clearly define the problem that you want to solve using machine learning. This involves understanding the business requirements, defining the problem statement, and identifying the data sources that will be used.
2. **Data collection:** Once the problem has been defined, the next step is to collect the relevant data required for the machine learning model. This data should be representative of the problem domain and should be of good quality.
3. **Data preprocessing:** Data preprocessing involves cleaning the data, handling missing values, and removing any irrelevant features. This step is crucial in ensuring that the machine learning model is fed with accurate and relevant data.
4. **Feature engineering:** Feature engineering involves selecting the relevant features from the data and transforming them into a format that can be used by the machine learning model. This step is important in improving the performance of the model.
5. **Model selection:** The next step is to select the appropriate machine learning model that best fits the problem statement. This involves evaluating different algorithms and selecting the one that provides the best performance.
6. **Model training:** Once the model has been selected, the next step is to train the model using the data collected in step 2. The data is split into training and testing sets to evaluate the model's performance.
7. **Model evaluation:** Model evaluation involves measuring the performance of the model using various metrics such as accuracy, precision, and recall. This step helps to determine if the model is performing well enough to be used in a production environment.
8. **Model deployment:** Once the model has been trained and evaluated, it is deployed in a production environment where it can be used to make predictions on new data.
9. **Model monitoring:** After the model has been deployed, it is important to monitor its performance over time. This helps to ensure that the model continues to provide accurate predictions and is not affected by changes in the environment.
10. **Model maintenance:** Finally, the model needs to be maintained and updated regularly to ensure that it continues to perform well over time. This involves retraining the model with new data and updating the algorithms and features as necessary.

5.3.1.3. Neuro Networks examples

These are examples of neural networks, and there are many more variations and combinations that can be used to solve different types of problems.

1. **Convolutional Neural Networks (CNNs):** CNNs are widely used for image classification tasks. They consist of multiple convolutional layers that learn to recognize patterns in the images.

2. **Recurrent Neural Networks (RNNs):** RNNs are used for sequential data processing tasks, such as speech recognition and language translation. They use a feedback loop to process data in a sequence and can remember previous inputs.
3. **Generative Adversarial Networks (GANs):** GANs are used for generating new data that is similar to the training data. They consist of a generator network that creates new data and a discriminator network that distinguishes between the generated and real data.
4. **Autoencoders:** Autoencoders are used for data compression and feature extraction. They consist of an encoder network that compresses the data into a lower-dimensional representation and a decoder network that reconstructs the original data.
5. **Deep Belief Networks (DBNs):** DBNs are used for unsupervised learning tasks such as feature learning and pattern recognition. They consist of multiple layers of hidden units that learn to represent the data at increasingly abstract levels.
6. **Long Short-Term Memory Networks (LSTMs):** LSTMs are used for processing sequential data with long-term dependencies. They use a memory cell that can selectively remember or forget information over time.

5.3.2. Building model steps

5.3.2.1. Identify business problems

The first step is to identify the business problem that you want to solve using machine learning. This involves understanding the business requirements, defining the problem statement, and identifying the data sources that will be used.

5.3.2.2. Identify and understand data

Once the problem has been defined, the next step is to identify the relevant data required for the machine learning model. This involves understanding the characteristics of the data, such as its size, format, and quality.

5.3.2.3. Collect and prepare data

After the relevant data has been identified, it needs to be collected and prepared for analysis. This involves cleaning and transforming the data, handling missing values, and preparing it for modeling.

5.3.2.4. Determine models and train data

Once the data has been prepared, the next step is to determine the appropriate machine learning models to use and train them on the data. This involves selecting the right algorithms, setting parameters, and tuning the models for optimal performance.

5.3.2.5. Evaluate models

Once the models have been trained, they need to be evaluated to determine how well they perform. This involves measuring accuracy, precision, recall, and other metrics to assess their effectiveness.

5.3.2.6. Experiment and adjust model

Based on the results of the model evaluation, adjustments may need to be made to improve the performance of the models. This involves experimenting with different algorithms, adjusting parameters, and retraining the models until the desired level of performance is achieved.

Since neural networks imitate the human brain and so deep learning will do. In deep learning, nothing is programmed explicitly. Basically, it is a machine learning class that makes use of numerous nonlinear processing units so as to perform feature extraction as well as transformation. The output from each preceding layer is taken as input by each one of the successive layers.

5.3.3. Deep learning

5.3.3.1. Definitions

Deep learning is a subset of machine learning that uses artificial neural networks to learn from large amounts of data. Deep learning models are designed to automatically learn hierarchical representations of data by using multiple layers of processing. These layers allow the models to extract features from raw data in a more efficient

and effective way than traditional machine learning models.

5.3.3.2. NLP

NLP stands for Natural Language Processing, which is a field of artificial intelligence that focuses on the interaction between computers and humans using natural language. NLP involves the use of algorithms, statistical models, and machine learning techniques to understand and generate human language.
NLP has many applications, including language translation, sentiment analysis, speech recognition, text summarization, and question answering. Some popular NLP techniques include tokenization, part-of-speech tagging, named entity recognition, sentiment analysis, and machine translation.

5.3.3.3. Image and object recognition

Object Recognition is a technology that lies under the broader domain of Computer Vision. This technology is capable of identifying objects that exist in images and videos and tracking them. Object Recognition also known as **Object Detection**, has various applications like face recognition, vehicle recognition, pedestrian counting, self-driving vehicles, security systems, and a lot more.
The two significant objectives of object recognition involve:

- Identification of all objects that exist in an image
- Filtration of the object that seeks attention

In the following lesson, we will understand how to perform Object Recognition in the Python programming language using the **ImageAI** library. We will create a basic object recognition model using the **ImageAI** library in Python.
So, let's get begun.

Deep Learning for Object Recognition

Techniques of Deep learning have been shown state of the art for different problems related to Object Recognition. Some of the generally used approaches of deep learning for object recognition are as follows:

1. ImageAI
2. Single Shot Detectors
3. YOLO (You Only Look Once)
4. Region-based Convolutional Neural Networks

However, in this lesson, we will understand what ImageAI is and how we can use it in performing Object Recognition.

Understanding the ImageAI library

Python offers a library built to empower programmers and developers for building applications and systems with self-contained deep learning and Computer Vision capabilities with the help of some lines of simple coding script. **ImageAI** consists of Python implementation of nearly all state-of-the-art deep learning algorithms such as **RetinaNet**, **YOLOv3**, and **TinyYOLOv3**.
ImageAI makes use of several APIs that work offline - it has object detection, video detection, and object tracking APIs that can be called without accessing the Internet. **ImageAI** uses a pre-trained model and can easily be customized.
The **ObjectDetection** class of the **ImageAI** library consists of methods in order to perform object detection on any image or set of images with the help of pre-trained models. With ImageAI, we can detect and recognize eighty distinct types of common, everyday objects.

Setting up the Environment

We will consider working through the installation of required libraries, including ImageAI.
In order to utilize ImageAI, we have to install some dependencies. The initial step is to have Python installed on the system.
Once we have installed Python on the system successfully, we have to install the following dependencies with the help of the **pip** installer:

1. OpenCV
2. TensorFlow

3. Keras
4. ImageAI

The installation command for the same is shown below:
Syntax:

```
# installing OpenCV
$ pip install opencv-python

# installing TensorFlow
$ pip install tensorflow

# installing Keras
$ pip install keras

# installing ImageAI
$ pip install imageAI
```

Now we have to download the TinyYOLOv3 model file containing the classification method that we will use for object recognition.
The link for the same can be found below:
https://github.com/OlafenwaMoses/ImageAI/releases/download/1.0/yolo-tiny.h5

Performing Object Recognition using ImageAI

In this section, we will discuss how we can utilize the ImageAI library in Python. The procedure of performing Object Recognition is divided into several steps for better understanding and clarity.

Step 1

The initial step is to create the necessary folders. We will need the folders as shown below:

1. **Object_Recognition:** This will be the root folder.
2. **Models:** This folder will store the pre-trained model.
3. **Input:** This folder will store the image file on which we have to perform object detection.
4. **Output:** This folder will store the image file with detected objects.

Once we created the necessary folder, the Object Recognition folder should have the following sub-folders:

```
?   Input
?   Models
?   Output
3 directories, 0 files
```

Step 2

For the second step, we will open the preferred text editor, which is Visual Studio Code, in this case, to write a Python script and create a new file recognizer.py

Step 3

Now, let us begin importing ObjectDetection class from the ImageAI library. The syntax for the same is shown below:
File: recognizer.py

```
# importing the required library
from imageai.Detection import ObjectDetection
```

Step 4

Now that the required ImageAI library is imported and the ObjectDetection class, the next thing is to create an instance of the class ObjectDetection. Let us consider the following snippet of code for the same.

File: recognizer.py

```
# instantiating the class
recognizer = ObjectDetection()
```

Step 5

Let us specify the path from the model, input image, and output image using the following snippet of code.

File: recognizer.py

```
# defining the paths
path_model = "./Models/yolo-tiny.h5"
path_input = "./Input/images.jpg"
path_output = "./Output/newimage.jpg"
```

Step 6

Once, we instantiated the ObjectDetection class we can now call different functions from the class. The class consists of the following functions in order to call pre-trained models:

```
setModelTypeAsRetinaNet()
setModelTypeAsYOLOv3()
setModelTypeAsTinyYOLOv3()
```

We will utilize the pre-trained TinyYOLOv3 model, and thus, we will be using the setModelTypeAsTinyYOLOv3() function in order to load the model.

Let us consider the following snippet of code for the same:

File: recognizer.py

```
# using the setModelTypeAsTinyYOLOv3() function
recognizer.setModelTypeAsTinyYOLOv3()
```

Step 7

Now, we will be going to call the function setModelPath(). This function will accept a string that consists of the path to the pre-trained model.

Let us consider the following snippet of code for the same:

File: recognizer.py

```
# setting the path to the pre-trained Model
recognizer.setModelPath(path_model)
```

Step 8

In this step, we will call the loadModel() function from the recognizer instance. This function will load the model from the path specified above with the help of the setModelPath() class method.

Let us consider the following snippet of code for the same.

File: recognizer.py

```
# loading the model
recognizer.loadModel()
```

Step 9
We have to call the detectObjectsFromImage() function with the help of the recognizer object that we created earlier.
This function accepts two parameters: **input_image** and **output_image_path**. The **input_image** parameter is the path where the image we recognize is situated, whereas the **output_image_path** parameter is the path storing the image with detected objects. This function will return a diction containing the names and percentage probabilities of every object detected in the image.

The syntax for the same is shown below:
File: recognizer.py

```
# calling the detectObjectsFromImage() function
recognition = recognizer.detectObjectsFromImage(
    input_image = path_input,
    output_image_path = path_output
    )
```

Step 10
At last, we can access the dictionary elements by iterating through each element present in the dictionary.
The syntax for the same is shown below:
File: recognizer.py

```
# iterating through the items found in the image
for eachItem in recognition:
    print(eachItem["name"] , " : ", eachItem["percentage_probability"])
```

Complete Python script for Object Recognition model
Let us consider the following script for the Object Recognition model.
File: recognizer.py

```
# importing the required library
from imageai.Detection import ObjectDetection
_
# instantiating the class
recognizer = ObjectDetection()
_
# defining the paths
path_model = "./Models/yolo-tiny.h5"
path_input = "./Input/images.jpg"
path_output = "./Output/newimage.jpg"
_
# using the setModelTypeAsTinyYOLOv3() function
recognizer.setModelTypeAsTinyYOLOv3()
# setting the path of the Model
recognizer.setModelPath(path_model)
```

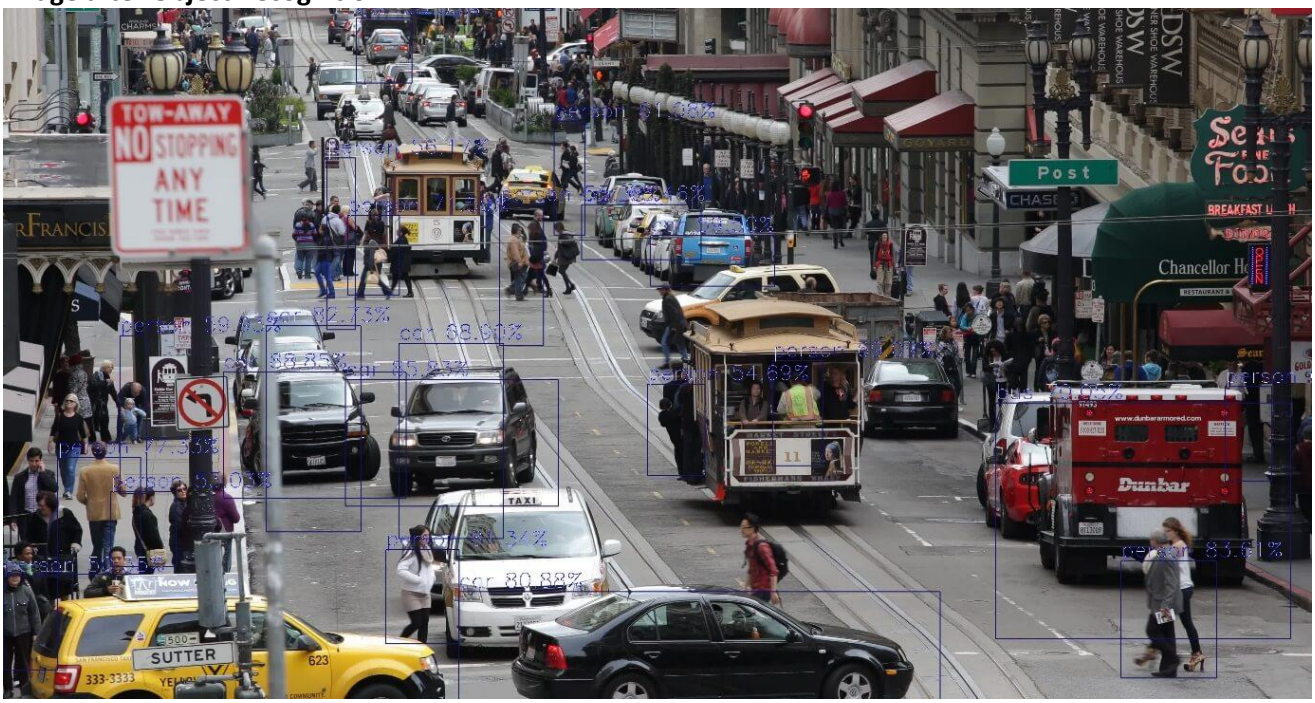
```
# loading the model
recognizer.loadModel()
# calling the detectObjectsFromImage() function
recognition = recognizer.detectObjectsFromImage(
    input_image = path_input,
    output_image_path = path_output
    )

# iterating through the items found in the image
for eachItem in recognition:
    print(eachItem["name"] , " : ", eachItem["percentage_probability"])
```

Output:

```
car  :  88.85036110877991
car  :  85.83406209945679
bus  :  70.04978060722351
car  :  80.88288903236389
car  :  55.334705114364624
person  :  61.084866523742676
car  :  68.46083402633667
person  :  56.165677309036255
person  :  71.58655524253845
car  :  59.49597954750061
person  :  55.276620388031006
person  :  69.08922791481018
person  :  59.92640256881714
car  :  82.73208141326904
person  :  54.69227433204651
person  :  67.25137233734131
car  :  68.9003050327301
person  :  77.32996344566345
person  :  53.02640199661255
person  :  81.33729696273804
person  :  83.60849618911743
person  :  50.34937262535095
```

Actual Image:

Image after Object Recognition:



At last, we can observe that ImageAI has successfully identified cars and persons in the image.

5.3.3.4. Deep Learning layers

Deep learning models consist of multiple layers of processing that allow them to automatically learn hierarchical representations of data. Each layer in a deep learning model performs a different type of processing on the data. Here are some common types of layers used in deep learning models:

1. **Input layer:** This layer receives the input data and passes it on to the next layer.
2. **Convolutional layer:** This layer is commonly used in image and video processing tasks. It applies a set of filters to the input data, allowing the model to learn features such as edges, textures, and shapes.
3. **Pooling layer:** This layer is often used after a convolutional layer to reduce the spatial size of the output data and control overfitting.

4. **Recurrent layer:** This layer is commonly used in natural language processing tasks. It processes sequences of input data, allowing the model to learn temporal patterns and dependencies.
5. **Fully connected layer:** This layer connects every neuron in the previous layer to every neuron in the current layer. It is often used as the output layer for classification tasks.
6. **Dropout layer:** This layer randomly drops out a percentage of neurons during training, helping to prevent overfitting.
7. **Batch normalization layer:** This layer normalizes the output of the previous layer, helping to prevent vanishing gradients and accelerating training.
8. **Activation layer:** This layer applies a non-linear activation function to the output of the previous layer, allowing the model to learn complex patterns in the data.

These are just a few examples of the many types of layers used in deep learning models. The choice of layers and their configuration depends on the specific task and the nature of the data.

5.3.3.5. Tensor flow

TensorFlow is a machine learning framework and developed by **Google Brain Team**. It is derived from its core framework: Tensor. In TensorFlow, all the computations involve tensors. A tensor is a vector or a matrix of n-dimensions which represents the types of data. All the values in a TensorFlow identify data type with a known shape. The shape of the data is the dimension of the matrix or array.

Representation of a Tensor

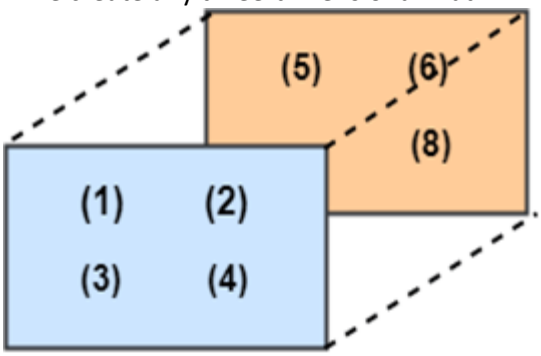
In TensorFlow, a tensor is the collection of feature vector (Like, array) of n-dimension. For instance, if we have any 2x3 matrix with values 1 to 6, we write:

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

TensorFlow represents this matrix as:

```
[[1, 3, 5],
[2, 4, 6]]
```

If we create any three-dimensional matrix with values 1 to 8, we have:



TensorFlow represents this matrix as:

```
[ [[1, 2],
[[3, 4],
[[5, 6],
[[7, 8] ]
```

Note: A tensor is represented with a scalar or can have a shape of more than three dimensions. It is just difficult to anticipate high dimension.

Types of Tensor

All computations pass through one or more Tensors in TensorFlow. A tensor is an object which has three properties

which are as follows:

- A unique label (name)
- A dimension (shape)
- A data type (dtype)

Each operation we will TensorFlow involves the manipulation of a tensor. There are four main tensors we can create:

- tf.Variable
- tf.constant
- tf.placeholder
- tf.SparseTensor

REFERENCES

What Is Python Used For? A Beginner's Guide | Coursera

Python Scripting Tutorial | HolyPython.com

What Is Python Used For? A Beginner's Guide | Coursera