# **Illustrate Tuple? Create a tuple and access eleements.Give examples.**

# **Python Tuple**

**A Tuple is a collection of immutable Python objects separated by commas.**

Tuples are just like lists, but we cannot change the elements of a tuple once it is assigned whereas in a list, elements can be changed.

**The main difference being that tuple manipulation are faster than list because tuples are immutable.**

**Since they're immutable, they can be used as keys for a dictionary**.

Also, Tuples are used whenever you want to return **multiple results** from a function.

### **Accessing tuple values**

**To access individual elements, we use [] square brackets where the index starts from 0.**

**a\_tuple = (1,2,'sunday','monday',3.14)**

**print(a\_tuple[0]) # print 1st element**

**print(a\_tuple[1]) # print 2nd element**

**print(a\_tuple[-1]) # print last element**

**print(a\_tuple[-2]) # print 2nd last element**

1. **Define file.Name two types of data files available in python**

**Types Of File in Python**

There are two types of files in Python and each of them are explained below in detail with examples for your easy understanding.

**They are:**

* Binary file
* Text file

Binary files in Python

Most of the files that we see in our computer system are called binary files.

**Example:**

1. **Document files:** .pdf, .doc, .xls etc.
2. **Image files:** .png, .jpg, .gif, .bmp etc.
3. **Video files:** .mp4, .3gp, .mkv, .avi etc.
4. **Audio files:** .mp3, .wav, .mka, .aac etc.
5. **Database files:** .mdb, .accde, .frm, .sqlite etc.
6. **Archive files:** .zip, .rar, .iso, .7z etc.
7. **Executable files:** .exe, .dll, .class etc.

All binary files follow a specific format. We can open some binary files in the normal text editor but we can’t read the content present inside the file. That’s because all the binary files will be encoded in the binary format, which can be understood only by a computer or machine.

For handling such binary files we need a specific type of software to open it.

**For Example,** You need Microsoft word software to open .doc binary files. Likewise, you need a pdf reader software to open .pdf binary files and you need a photo editor software to read the image files and so on.

Text files in Python

Text files don’t have any specific encoding and it can be opened in normal text editor itself.

**Example:**

* **Web standards:** html, XML, CSS, JSON etc.
* **Source code:** c, app, js, py, java etc.
* **Documents:** txt, tex, RTF etc.
* **Tabular data:** csv, tsv etc.
* **Configuration:** ini, cfg, reg etc.

1. **What are the two ways to open a file in python.Give example for each**

f = open("file1.txt","r")

if f:

print("file is opened successfully")

**Output:**

file is opened successfully

In the above code, we have passed **filename** as a first argument and opened file in read mode as we mentioned **r** as the second argument. The **fileptr** holds the file object and if the file is opened successfully, it will execute the print statement

f = open("test.txt") # equivalent to 'r' or 'rt'

f = open("test.txt",'w') # write in text mode

f = open("img.bmp",'r+b') # read and write in binary mode

The with statement

The **with** statement was introduced in python 2.5. The **with** statement is useful in the case of manipulating the files. It is used in the scenario where a pair of statements is to be executed with a block of code in between.

The syntax to open a file using with the statement is given below.

with open(<file name>, <access mode>) as <file-pointer>:

    #statement suite

The advantage of using with statement is that it provides the guarantee to close the file regardless of how the nested block exits.

It is always suggestible to use the **with** statement in the case of files because, if the break, return, or exception occurs in the nested block of code then it automatically closes the file, we don't need to write the **close()** function. It doesn't let the file to corrupt.

Consider the following example.

Example: with open("file.txt",'r') as f:

   content = f.read()

**print**(content)

#### Define a Dictionary. Write a program that creates a dictionary which contains 3 dictionaries(nested dictionary)

# **Python Dictionary**

A **dictionary** is a data type similar to arrays, but works with **keys and values** instead of indexes.

Dictionaries, in Python, are also known as "mappings", because they "map" or "associate" key objects to value objects.

Dictionaries are indexed by keys and they can be seen as associative arrays.

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.

Python - Nested Dictionaries

Nested Dictionaries

A dictionary can contain dictionaries, this is called nested dictionaries.

Example

Create a dictionary that contain three dictionaries:

myfamily={"child1":{"name" : "Emil","year" : 2004},

"child2":{"name":"Tobias","year": 2007},

"child3":{"name": "Linus","year" : 2011}}

print(myfamily)

print(myfamily['child2']['year'])

#### Write the difference between text files and binary files with examples

### Text Files:

Human-Readable: Text files are human-readable files that contain plain text. They store data in the form of characters, typically using a specific encoding like UTF-8.

End-of-Line Characters: Text files use newline characters (\n on Unix-like systems, \r\n on Windows) to represent the end of each line.

Encoding: Text files use character encoding to interpret the data. Common encodings include UTF-8, ASCII, and others. When you read from a text file, Python automatically decodes the bytes into strings using the specified encoding.

Operations: Text files are usually processed line by line. You can read and write text files using functions like open(), read(), readline(), write(), and close().

Example of reading from a text file:

with open('file.txt', 'r', encoding='utf-8') as file:

content = file.read()

print(content)

### **Binary Files:**

**Not Human-Readable:** Binary files contain data in a format that is not human-readable. They can store any kind of data, including images, audio, or other non-text formats.

**No End-of-Line Characters:** Binary files do not have a concept of end-of-line characters. They store data as a sequence of bytes without any specific line breaks.

**No Automatic Encoding:** Binary files don't use character encoding because they can stortype of data, not just text. When you read from a binary file, you get raw bytes, and it's up to you to interpret them correctly.

**Operations:** Binary files are read and written using functions like **open()**, **read()**, **write()**, and **close()**, but without specifying an encoding. When you read from a binary file, you get a sequence of bytes that you can process as needed.

Example of reading from a binary file:

with open('file.bin', 'rb') as file:

binary\_data = file.read()

# Process binary data here

#### Write a program to sort a dictionary in descending order according to key(Program should display key value pair)

import operator

x = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}

sorted\_x = sorted(x.items(), key=operator.itemgetter(1))

print(sorted\_x)

#### Explain various modes of opening a file

|  |  |  |
| --- | --- | --- |
| **SN** | **Access mode** | **Description** |
| 1 | r | It opens the file to read-only mode. The file pointer exists at the beginning. The file is by default open in this mode if no access mode is passed. |
| 2 | rb | It opens the file to read-only in binary format. The file pointer exists at the beginning of the file. |
| 3 | r+ | It opens the file to read and write both. The file pointer exists at the beginning of the file. |
| 4 | rb+ | It opens the file to read and write both in binary format. The file pointer exists at the beginning of the file. |
| 5 | w | It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exists with the same name. The file pointer exists at the beginning of the file. |
| 6 | wb | It opens the file to write only in binary format. It overwrites the file if it exists previously or creates a new one if no file exists. The file pointer exists at the beginning of the file. |
| 7 | w+ | It opens the file to write and read both. It is different from r+ in the sense that it overwrites the previous file if one exists whereas r+ doesn't overwrite the previously written file. It creates a new file if no file exists. The file pointer exists at the beginning of the file. |
| 8 | wb+ | It opens the file to write and read both in binary format. The file pointer exists at the beginning of the file. |
| 9 | a | It opens the file in the append mode. The file pointer exists at the end of the previously written file if exists any. It creates a new file if no file exists with the same name. |
| 10 | ab | It opens the file in the append mode in binary format. The pointer exists at the end of the previously written file. It creates a new file in binary format if no file exists with the same name. |
| 11 | a+ | It opens a file to append and read both. The file pointer remains at the end of the file if a file exists. It creates a new file if no file exists with the same name. |
| 12 | ab+ | It opens a file to append and read both in binary format. The file pointer remains at the end of the file. |