1. What is Python and why is it popular?

Python is a general purpose, dynamic, high-level, and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.

Python is one of the top three most popular programming languages in 2019 and everybody is learning Python either to make their life easier or to expand their job opportunities.

Python and Django are also very popular choices for building the backend of web applications. Small or big companies use Python to re-write their existing applications or build new applications. Instagram, DropBox, and YouTube are examples of websites built with Python. Well, more accurately, the backend of these applications is built with Python and Django.

------------------------------------------------------------------------------------------------------------

1. What are the differences between Python 2 and Python 3?

|  |  |
| --- | --- |
| Python 2 | Python 3 |
| * Print function brackets optional | * Print() function brackets compulsory |
| * Prefix string with u to make Unicode string u”apple” | * String is Unicode by default |
| * Division of integers always returns integer. * 5/2 =2 | * Division of integers may result in float. * 5/2 = 2.5 |
| * raw\_input() reads string | * raw\_input() not available |
| * input() evaluates data read | * input() always reads string |
| * as keyword is not introduced | * as keyword is introduced |
| * generator.next() | * next(generator) |
|  | * py2to3 utility is introduced |

1. What is the difference between a tuple and a list in Python?

List and Tuple Syntax Differences:

The syntax of a list differs from that of a tuple. Items of a tuple are enclosed by parentheses or curved brackets (), whereas items of a list are enclosed by square brackets [].  
  
**Example code:**

# Python code to show the difference between creating a list and a tuple  
  
list\_ = [4, 5, 7, 1, 7]  
tuple\_ = (4, 1, 8, 3, 9)  
  
print("List is: ", list\_)  
print("Tuple is: ", tuple\_)

**Output:**

List is: [4, 5, 7, 1, 7]

Tuple is: (4, 1, 8, 3, 9)  
  
**Mutable List vs. Immutable Tuple:**

An important difference between a list and a tuple is that lists are mutable, whereas tuples are immutable. What exactly does this imply? It means a list's items can be changed or modified, whereas a tuple's items cannot be changed or modified.

We can't employ a list as a key of a dictionary because it is mutable. This is because a key of a Python dictionary is an immutable object. As a result, tuples can be used as keys to a dictionary if required.

Let's consider the example highlighting the difference between lists and tuples in immutability and mutability.  
**Example Code:**

# Updating the element of list and tuple at a particular index  
  
# creating a list and a tuple  
list\_ = ["Python", "Lists", "Tuples", "Differences"]  
tuple\_ = ("Python", "Lists", "Tuples", "Differences")  
  
# modifying the last string in both data structures  
list\_[3] = "Mutable"  
print(list\_)  
try:  
 tuple\_[3] = "Immutable"  
 print(tuple\_)  
except TypeError:  
 print("Tuples cannot be modified because they are immutable.")

**Output:**

['Python', 'Lists', 'Tuples', 'Mutable']

Tuples cannot be modified because they are immutable.

We altered the string of list\_ at index 3 in the above code, which the Python interpreter updated at index 3 in the output. Also, we tried to modify the last index of the tuple in a try block, but since it raised an error, we got output from the except block. This is because tuples are immutable, and the Python interpreter raised TypeError on modifying the tuple.

Size Difference:

Since tuples are immutable, Python allocates bigger chunks of memory with minimal overhead. Python, on the contrary, allots smaller memory chunks for lists. The tuple would therefore have less memory than the list. If we have a huge number of items, this makes tuples a little more memory-efficient than lists.

For example, consider creating a list and a tuple with the identical items and comparing their sizes:  
  
**Example code:**

# Code to show the difference in the size of a list and a tuple  
  
# creating a list and a tuple  
list\_ = ["Python", "Lists", "Tuples", "Differences"]  
tuple\_ = ("Python", "Lists", "Tuples", "Differences")  
# printing sizes  
print("Size of tuple: ", tuple\_.\_\_sizeof\_\_())  
print("Size of list: ", list\_.\_\_sizeof\_\_())

**output:**  
Size of tuple: 56

Size of list: 72

-------------------------------------------------------------------------------------------------------------

1. How do you create a dictionary in Python?

Creating a Dictionary

We can create a dictionary in Python by wrapping a collection of objects in curly brackets and segregating them with a comma.

The Dictionary keeps track of a couple of entries, one part of this is called a key, and the other is called the value of the key and is referred to as the key: value pair.

We can store any data type as a value for the key and give the same value to two keys, but keys should be immutable and unique.

**Example code:**

# Initializing a dictionary with some elements  
Dictionary = {1: 'Java', 2: 'Python', 3: 'Dictionary'}  
print("\nDictionary created using curly braces: ")  
print(Dictionary)  
# Creating a Dictionary with keys of different data types  
Dictionary = {'Website': 'Java', 3: [2, 3, 5, 'Dictionary']}  
print("\nDictionary with keys of multiple data type: ")  
print(Dictionary)

**Output:**

Dictionary created using curly braces:

{1: 'Java', 2: 'Python', 3: 'Dictionary'}

Dictionary with keys of multiple data type:

{'Website': 'Java', 3: [2, 3, 5, 'Dictionary']}

**Dict():**

The built-in method dict() could also be used to generate a dictionary. By simply placing two curly brackets {}, a blank dictionary can be built.

Example Code:

# Initializing an empty Dictionary  
Dictionary = {}  
print("An empty Dictionary: ")  
print(Dictionary)  
  
# Creating a Dictionary using in-built dict() method  
Dictionary = dict({1: 'Python', 2: 'Java', 3: 'Dictionary'})  
print("\nDictionary created by using dict() method: ")  
print(Dictionary)  
  
# Creating dictionary by key:value pair format  
Dictionary = dict([(1, 'Java'), (2, 'Python'), (3, 'Dictionary')])  
print("\nDictionary with key:value pair format: ")  
print(Dictionary)

Output:  
  
An empty Dictionary:

{}

Dictionary created by using dict() method:

{1: 'Python', 2: 'Java', 3: 'Dictionary'}

Dictionary with key:value pair format:

{1: 'Java', 2: 'Python', 3: 'Dictionary'}

-------------------------------------------------------------------------------------------------------------

1. What is a function in Python and how do you define one?

A function is a reusable block of programming statements designed to perform a certain task. To define a function, Python provides the def keyword. The following is the syntax of defining a function.

Syntax of Python Function

# An example Python Function

def function\_name( parameters ):

# code block

The following elements make up to define a function, as seen above.

* The beginning of a function header is indicated by a keyword called def.
* function\_name is the function's name that we can use to separate it from others. We will use this name to call the function later in the program. In Python, name functions must follow the same rules as naming variables.
* We pass arguments to the defined function using parameters. However, they are optional.
* The function header is terminated by a colon (:).
* We can use a documentation string called docstring in the short form to explain the purpose of the function.
* The body of the function is made up of several valid Python statements. The indentation depth of the whole code block must be the same (usually 4 spaces).
* We can use a return expression to return a value from a defined function.

**Example code:**

# Example Python Code for User-Defined function  
def square( num ):  
 *"""  
 This function computes the square of the number.  
 """* return num\*\*2  
object\_ = square(6)  
print( "The square of the given number is: ", object\_ )

**Output:**

The square of the given number is: 36

-------------------------------------------------------------------------------------------------------

1. What is object-oriented programming (OOP) and how does it relate to Python?

Like other general-purpose programming languages, Python is also an object-oriented language since its beginning.

It allows us to develop applications using an Object-Oriented approach. In Python, we can easily create and use classes and objects.

The oops concept focuses on writing the reusable code. It is a widespread technique to solve the problem by creating objects.

Major principles of object-oriented programming system are given below.

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

-------------------------------------------------------------------------------------------------------------

1. How do you handle exceptions in Python?

Python uses try and except keywords to handle exceptions. Both keywords are followed by indented blocks.

try:

#statements in try block

except:

#executed when error in try block

**Example Code:**

try:  
 a=5  
 b='0'  
 print(a/b)  
except:  
 print('Some error occurred.')  
print("Out of try except blocks.")

**Output:**

Some error occurred.

Out of try except blocks.

A single try block may have multiple except blocks. The following example uses two except blocks to process two different exception types:  
  
**Example Code:**

try:  
 a=5  
 b=0  
 print (a/b)  
except TypeError:  
 print('Unsupported operation')  
except ZeroDivisionError:  
 print ('Division by zero not allowed')  
print ('Out of try except blocks')

**Output:**

Division by zero not allowed

Out of try except blocks

**else and finally:**

In Python, keywords else and finally can also be used along with the try and except clauses.

While the except block is executed if the exception occurs inside the try block, the else block gets processed if the try block is found to be exception free.

try:

#statements in try block

except:

#executed when error in try block

else:

#executed if try block is error-free

finally:

#executed irrespective of exception occured or not

**Example code:**

try:  
 print('try block')  
 x=int(input('Enter a number: '))  
 y=int(input('Enter another number: '))  
 z=x/y  
except ZeroDivisionError:  
 print("except ZeroDivisionError block")  
 print("Division by 0 not accepted")  
else:  
 print("else block")  
 print("Division = ", z)  
finally:  
 print("finally block")  
 x=0  
 y=0  
print ("Out of try, except, else and finally blocks." )

**Output:**

try block

Enter a number: 10

Enter another number: 2

else block

Division = 5.0

finally block

Out of try, except, else and finally blocks.

1. How do you read and write files in Python?
2. Open the file:

**Example code:**

#directory: /home/imtiaz/code.py  
text\_file = open('file.txt','r')  
  
#Another method using full location  
text\_file2 = open('/home/imtiaz/file.txt','r')  
print('First Method')  
print(text\_file)  
  
print('Second Method')  
print(text\_file2)

**Output:**

First Method

Second Method

2. Read and write to files in Python:

* read() : This function reads the entire file and returns a string
* readline() : This function reads lines from that file and returns as a string. It fetch the line n, if it is been called nth time.
* readlines() : This function returns a list where each element is single line of that file.
* readlines() : This function returns a list where each element is single line of that file.
* write() : This function writes a fixed sequence of characters to a file.
* writelines() : This function writes a list of string.
* append() : This function append string to the file instead of overwriting the file.

**Example code:**

* #open the file  
  text\_file = open('/Users/pankaj/abc.txt','r')  
    
  #get the list of line  
  line\_list = text\_file.readlines();  
    
  #for each line from the list, print the line  
  for line in line\_list:  
   print(line)  
    
  text\_file.close() #don't forget to close the file

**Output:**

Hi Pankaj

I am Here

3. Copy files in Python using the shutil() method:

import shutil  
  
shutil.copy2('/Users/pankaj/abc.txt', '/Users/pankaj/abc\_copy2.txt')  
  
#another way to copy file  
  
shutil.copyfile('/Users/pankaj/abc.txt', '/Users/pankaj/abc\_copyfile.txt')  
  
print("File Copy Done")

4. Delete files in Python with the shutil.os.remove() method:

import shutil  
import os  
  
#two ways to delete file  
shutil.os.remove('/Users/pankaj/abc\_copy2.txt')  
  
os.remove('/Users/pankaj/abc\_copy2.txt')

5. Close the file:

text\_file = open('/Users/pankaj/abc.txt','r')  
# some file operations here  
  
text\_file.close()

1. How do you install and use external packages in Python?  
   Using pip we can install external packages and using imports we can use the external packages.

-----------------------------------------------------------------------------------------------------------

1. How do you use the "if" statement in Python to perform conditional execution?  
     
   Using the (AND, OR, NOT) condition to evaluate the Boolean expression followed by colon(:)