

✓ Introduction To Python

1) What are the types of Applications?

- i) Web Development: Python is popular for web development due to frameworks like Django, Flask, and Pyramid. These frameworks simplify the process of building robust and scalable web applications.
- ii) Data Science: Python is a go-to language for data science, including data analysis, machine learning, and data visualization. Libraries like Pandas, NumPy, SciPy, and TensorFlow make it easier to handle data and build complex models.
- iii) Machine Learning and Artificial Intelligence: Python's simplicity and readability make it ideal for developing machine learning and AI algorithms. Libraries such as Keras, Scikit-learn, and PyTorch are widely used in this field.
- iv) Desktop GUI Applications: Python can be used to create desktop applications with graphical user interfaces (GUIs). Libraries like Tkinter, PyQt, and Kivy are popular choices for this purpose.
- v) Web Scraping: Python is often used for web scraping to extract data from websites. Libraries like BeautifulSoup and Scrapy are commonly used for this.
- vi) Automation and Scripting: Python is excellent for writing scripts to automate repetitive tasks. This can range from simple file operations to complex system administration tasks.
- vii) Software Development: Python is used in various aspects of software development, including build control, management, and testing.

2) What is programming?

Programming is the process of creating a set of instructions that a computer can follow to perform specific tasks. These instructions are written in a programming language, which is a formal language designed to communicate with computers.

3) What is Python?

Python is a high-level, general purpose, interpreted programming language known for its simplicity and readability.

- i) Easy to Learn and Use: Python syntax is clear and straightforward, making it an excellent choice for beginners.
- ii) Interpreted Language: Python code is executed line by line, which makes debugging easier and allows for interactive testing.
- iii) Versatile: Python is used in web development, data science, machine learning, artificial intelligence, automation, game development, and more.
- iv) Extensive Libraries and Frameworks: Python has a rich ecosystem of libraries and frameworks, such as Pandas, NumPy, TensorFlow, Django, and Flask, which simplify complex tasks.
- v) Cross-Platform: Python runs on various operating systems, including Windows, macOS, and Linux.
- vi) Open Source: Python is free to use and distribute, making it accessible to everyone.

4) Write a Python program to check if a number is positive, negative or zero.

```
num = int(input("Enter A Number:"))
if num>0:
    print(num,"is Positive")
elif num<0:
    print(num,"is Negative")
else:
    print(num,"is Zero")
```

```
Enter A Number:8
8 is Positive
```

5) Write a Python program to get the Factorial number of given numbers.

```

num = int(input("Enter a Number:"))
fact = 1
if num<0:
    print("Not Exist")
if num==0:
    print(1)
if num>0:
    for i in range(1,num+1):
        fact*=i
    print(fact)

```

```

Enter a Number:5
120

```

```

def factorial(n):
    fact=1
    for i in range(1,n+1):
        fact=fact*i
    print(fact)
factorial(5)

```

```

120

```

6)Write a Python program to get the Fibonacci series of given range.

```

a = 0
b = 1
num = int(input("Enter a Number:"))
if num==1:
    print(a)
else:
    print(a)
    print(b)
    for i in range(2,num):
        c = a+b
        a = b
        b = c
        print(c)

```

```

Enter a Number:7
0
1
1
2
3
5
8

```

7)How memory is managed in Python?

Memory management in Python is handled automatically by the Python interpreter, which uses a combination of techniques to efficiently allocate and deallocate memory. Here are the key components of Python's memory management:

- i)Private Heap Space: All Python objects and data structures are stored in a private heap. This heap is managed by the Python memory manager, which ensures that the memory is allocated and deallocated as needed.
- ii)Garbage Collection: Python uses a garbage collector to automatically free up memory that is no longer in use. The garbage collector identifies objects that are no longer referenced by any part of the program and deallocates their memory.
- iii)Dynamic Memory Allocation: Memory is allocated dynamically as needed. This means that the interpreter allocates memory for objects when they are created and deallocates it when they are no longer needed.
- iv)Stack and Heap Memory: Python uses both stack and heap memory. Stack memory is used for function calls and local variables, while heap memory is used for dynamic memory allocation.

8)What is the purpose continuing statement in python?

The continue statement in Python is used within loops to skip the rest of the code inside the loop for the current iteration and move to the next iteration. This is particularly useful when you want to bypass certain conditions or values while iterating over a sequence.

9) Write python program that swap two number with temp variable and without temp variable.

i) Using temp variable

```
num1 = int(input("Enter a Number:"))
num2 = int(input("Enter a Number:"))
temp = num1
num1 = num2
num2 = temp
print("Print swap of num1 is",num1)
print("Print swap of num2 is",num2)
```

```
↩ Enter a Number:10
Enter a Number:20
Print swap of num1 is 20
Print swap of num2 is 10
```

ii) Without temp variable

```
num1 = int(input("Enter a Number:"))
num2 = int(input("Enter a Number:"))
num1 = num1 + num2
num2 = num1 - num2
num1 = num1 - num2
print("Print swap of num1 is",num1)
print("Print swap of num2 is",num2)
```

```
↩ Enter a Number:10
Enter a Number:20
Print swap of num1 is 20
Print swap of num2 is 10
```

10) Write a Python program to find whether a given number is even or odd, print out an appropriate message to the user.

```
num = int(input("Enter a Number:"))
if num%2==0:
    print(num,"is Even")
else:
    print(num,"is Odd")
```

```
↩ Enter a Number:10
10 is Even
```

11) Write a Python program to test whether a passed letter is a vowel or not.

```
my_str = input("Enter Name :")
if 'a' in my_str or 'e' in my_str or 'i' in my_str or 'o' in my_str or 'u' in my_str:
    print("Vowel")
else:
    print("Consonant")
```

```
↩ Enter Name :e
Vowel
```

12) Write a Python program to sum of three given integers. However, if two values are equal sum will be zero.

```
a = int(input("Enter a Number a:"))
b = int(input("Enter a Number b:"))
c = int(input("Enter a Number c:"))
if a==b or b==c or c==a:
    print(0)
else:
    print(a+b+c)
```

```
↩ Enter a Number a:10
Enter a Number b:20
Enter a Number c:10
```

13) Write a Python program that will return true if the two given integer values are equal or their sum or difference is 5.

```
a = int(input("Enter a Number a:"))
b = int(input("Enter a Number b:"))
if a==b or (a+b)==5 or (a-b)==5:
    print(True)
else:
    print(False)
```

```
Enter a Number a:7
Enter a Number b:6
False
```

14)Write a python program to sum of the first n positive integers.

```
num = int(input("Enter a Number:"))
sum = 0
for i in range(1,num+1):
    sum=sum+i
print(sum)
```

```
Enter a Number:4
10
```

15)Write a Python program to calculate the length of a string.

```
my_str = input("Enter a String:")
print(len(my_str))
```

```
Enter a String:Neha
4
```

16) Write a Python program to count the number of characters (character frequency) in a string.

```
def count_frequency_string(mystr):
    frequency_dict={}
    for i in mystr:
        if i in frequency_dict:
            frequency_dict[i]+=1
        else:
            frequency_dict[i]=1
    print(frequency_dict)
```

```
count_frequency_string("Hello Python")
```

```
{'H': 1, 'e': 1, 'l': 2, 'o': 2, ' ': 1, 'P': 1, 'y': 1, 't': 1, 'h': 1, 'n': 1}
```

17)What are negative indexes and why are they used?

Negative indexes in Python are used to access elements from the end of a sequence, such as a list or a string. Instead of starting from the beginning (index 0), negative indexing starts from the end, with the last element being indexed as -1, the second last as -2, and so on.

why negative indexes are useful:

- i)Convenience: They provide a convenient way to access elements from the end of a sequence without needing to calculate the length of the sequence.
- ii)Readability: They make the code more readable and concise, especially when you need to access elements near the end of a sequence.
- iii)Flexibility: They allow for more flexible slicing and indexing operations.


18) Write a Python program to count occurrences of a substring in a string.

```
def count_occurrences(main_string, sub_string):
    count = main_string.count(sub_string)
    print(count)
count_occurrences("Hello world , welcom to world python programming", "world")
```

 2


19) Write a Python program to count the occurrences of each word in a given sentence.

```
def count_word_occurrences(my_str):
    words = my_str.split()
    word_counts = {}
    for word in words:
        if word in word_counts:
            word_counts[word] += 1
        else:
            word_counts[word] = 1
    print(word_counts)
count_word_occurrences("Hello world , welcom to world python programming")
```

 {'Hello': 1, 'world': 2, ',': 1, 'welcom': 1, 'to': 1, 'python': 1, 'programming': 1}


20) Write a Python program to get a single string from two given strings, separated by a space and swap the first two characters of each string.

```
def swap_two_string(str1, str2):
    new_str1 = str2[:2] + str1[2:] #wollo
    new_str2 = str1[:2] + str2[2:] #Herld
    result = new_str1 + " " + new_str2
    print(result)
swap_two_string("Hello", "World")
```

 Wollo Herld


21) Write a Python program to add 'in' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add 'ly' instead if the string length of the given string is less than 3, leave it unchanged.

```
a = input("Enter a string")
if len(a) < 3:
    print(a)
elif a.endswith('ing'):
    print(a + 'ly')
else:
    print(a + 'ing')
```

 Enter a string
borring
borringly

22) Write a Python function to reverse a string if its length is a multiple of 4.

```
def reverse_if_multiple_four(s):
    if len(s) % 4 == 0:
        return s[::-1]
    else:
        return s
reverse_if_multiple_four("Neha")
```

 'aheN'

23) Write a Python program to get a string made of the first 2 and the last 2 chars from a given a string. If the string length is less than 2, return instead of the empty string.

```
def len_string(s):
    if len(s)<2:
        return " "
    else:
        return s[:2]+s[-2:]
len_string("Hello")
```

➦ 'Helo'

24)Write a Python function to insert a string in the middle of a string.

```
def insert_in_middle(original,new_insert):
    middle_index = len(original)//2
    result = original[:middle_index]+" "+new_insert+original[middle_index:]
    print(result)
insert_in_middle("Hello World","Python")
```

➦ Hello Python World

25)What is List? How will you reverse a list?

A list in Python is a collection of items that are ordered, changeable, and allow duplicate values. Lists are created using square brackets, and items within a list are separated by commas. For example:my_list = ["apple", "banana", "cherry"]

To reverse a list in Python, you can use several methods.

i)**reverse()**:This method reverses the list in place, meaning it modifies the original list.

```
my_list = [1, 2, 3, 4, 5]
my_list.reverse()
print(my_list)
```

➦ [5, 4, 3, 2, 1]

ii)**slicing**:This method creates a new list that is the reverse of the original list.

```
my_list = [1, 2, 3, 4, 5]
reversed_list = my_list[::-1]
print(reversed_list)
```

➦ [5, 4, 3, 2, 1]

iii)**reversed()**:This function returns an iterator that accesses the given sequence in reverse order. You can convert it to a list using the list() function.

```
my_list = [1, 2, 3, 4, 5]
reversed_list = list(reversed(my_list))
print(reversed_list)
```

➦ [5, 4, 3, 2, 1]

26)How will you remove last object from a list?

You can remove the last object from a list in Python using the pop() method without any arguments, or by using the del statement.

i)**pop()**:

```
my_list = [1, 2, 3, 4, 5]
my_list.pop()
print(my_list)
```

➦ [1, 2, 3, 4]

ii)**del statement**:

```
my_list = [1, 2, 3, 4, 5]
del my_list[-1]
print(my_list)
```

```
➦ [1, 2, 3, 4]
```

27) Suppose list1 is [2, 33, 222, 14, and 25], what is list1 [-1]?

```
list1 = [2, 33, 222, 14, 25]
print(list1[-1])
```

```
➦ 25
```

28) Differentiate between append () and extend () methods?

In Python, append() and extend() are both list methods used to add elements to a list, but they work differently:

i) append(): Adds a single element to the end of the list.

```
my_list = [1,2,3]
my_list.append(4)
print(my_list)
my_list.append([4,5,6])
print(my_list)
```

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

ii) extend(): Adds multiple elements to the end of the list. The argument passed to extend() must be an iterable (like a list, tuple, set).

```
my_list = [1,2,3]
my_list.extend([4,5,6])
print(my_list)
my_list.extend("abc")
print(my_list)
```

```
➦ [1, 2, 3, 4, 5, 6]
  [1, 2, 3, 4, 5, 6, 'a', 'b', 'c']
```

29) Write a Python function to get the largest number, smallest num and sum of all from a list.

```
a = [24,67,87,45,98,23,93]
print(max(a))
print(min(a))
sum = 0
for i in a:
    sum+=i
print(sum)
```

```
➦ 98
  23
  437
```

30) How will you compare two lists?

```
list1 = [1, 2, 3]
list2 = [1, 2, 3]
print(list1 == list2)
```

```
➦ True
```


31) Write a Python program to count the number of strings where the string length is 2 or more and the first and last character are same from a given list of strings.

```
def count_string(my_str):
    count = 0
    for i in my_str:
        if len(i)>=2 and i[0]==i[-1]:
            count+=1
    return count
my_str = ['abc', 'xyz', 'aba', '1221', 'aa']
result = count_string(my_str)
print(result)
```


 3

32)Write a Python program to remove duplicates from a list.

```
def remove_duplicates_list(my_list):
    return list(set(my_list))
remove_duplicates_list([1,2,3,2,3,6,4,3,5])
```

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

```
def remove_duplicates_list(my_list):
    return list(dict.fromkeys(my_list))
remove_duplicates_list([1,2,3,2,3,6,4,3,5])
```

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

33)Write a Python program to check a list is empty or not.

```
def is_list_empty(my_list):
    return len(my_list)==0
is_list_empty([1,2,3])
```

 False

34) Write a Python function that takes two lists and returns true if they have at least one common member.

```
def common_member_list(list1,list2):
    for i in list1:
        if i in list2:
            return True
    return False
list1 = [1,2,3,4,43]
list2 = [3,565,87,678,899]
print(common_member_list(list1,list2))
```

 True

35) Write a Python program to generate and print a list of first and last 5 elements where the values are square of numbers between 1 and 30.

```
squares = [x**2 for x in range(1, 31)]
result = squares[:5]+squares[-5:]
print(result)
```

 [1, 4, 9, 16, 25, 676, 729, 784, 841, 900]

36) Write a Python function that takes a list and returns a new list with unique elements of the first list.

```
def list_unique_elements(my_list):
    unique_elements = []
    seen = set()
    for i in my_list:
        if i not in seen:
            unique_elements.append(i)
            seen.add(i)
    return unique_elements
list_unique_elements([1,45,77,87,45,89,45,35,98,34])
```

➦ [1, 45, 77, 87, 89, 35, 98, 34]

37)Write a Python program to convert a list of characters into a string.

```
def list_to_string(my_list):
    result = ''.join(my_list)
    return result
list_to_string(['H','e','l','l','o',' ',' ','W','o','r','l','d'])
```

➦ 'Hello World'

38)Write a Python program to select an item randomly from a list.

```
import random
def select_random_item_list(my_list):
    return random.choice(my_list)
select_random_item_list([1,2,3,'a','d','s','n'])
```

➦ 'd'

39)Write a Python program to find the second smallest number in a list.

```
def second_smallest_number_list(my_list):
    unique_numbers = list(set(my_list))
    unique_numbers.sort()
    return unique_numbers[1] if len(unique_numbers) > 1 else None
second_smallest_number_list([1,2,7,8,65,76,34,97])
```

➦ 2

40)Write a Python program to get unique values from a list.

```
def get_unique_values_list(my_list):
    return list(set(my_list))
get_unique_values_list([1,45,87,34,87,12,23,87,65,45])
```

➦ [1, 34, 65, 12, 45, 23, 87]

41)Write a Python program to check whether a list contains a sub list.

```
def contains_sublist(main_list, sub_list):
    len_main = len(main_list)
    len_sub = len(sub_list)
    if len_sub > len_main:
        return False
    for i in range(len_main - len_sub + 1):
        if main_list[i:i + len_sub] == sub_list:
            return True
    return False
main_list = [1, 2, 3, 4, 5]
sub_list1 = [2, 3, 4]
sub_list2 = [3, 5]
print(contains_sublist(main_list,sub_list1))
print(contains_sublist(main_list,sub_list2))
```

True
False

42)Write a Python program to split a list into different variables.

```
my_list = [1,2,3,4,5]
a,b,c,d,e = my_list
print(a)
print(b)
print(c)
print(d)
print(e)
```

1
2
3
4
5

43)What is tuple? Difference between list and tuple.

A tuple in Python is an immutable, ordered collection of elements. This means that once a tuple is created, its elements cannot be changed, added, or removed. Tuples are defined using parentheses ()

```
import pandas as pd
data = {
    "Feature": ["Syntax", "Mutability", "Methods", "Performance", "Usage", "Memory"],
    "List": ["Defined using []", "Mutable (can be changed)",
            "More built-in methods (e.g., append, remove, etc.)",
            "Generally slower due to extra overhead for mutability",
            "Suitable for collections of items that may change",
            "Consumes more memory"],
    "Tuple": ["Defined using ()", "Immutable (cannot be changed)",
            "Fewer built-in methods (e.g., count, index)",
            "Generally faster due to immutability",
            "Suitable for collections of items that should not change",
            "Consumes less memory"]
}
df = pd.DataFrame(data)
df
```

	Feature	List	Tuple
0	Syntax	Defined using []	Defined using ()
1	Mutability	Mutable (can be changed)	Immutable (cannot be changed)
2	Methods	More built-in methods (e.g., append, remove, e...	Fewer built-in methods (e.g., count, index)
3	Performance	Generally slower due to extra overhead for mut...	Generally faster due to immutability
4	Usage	Suitable for collections of items that mav change	Suitable for collections of items that should ...

44)Write a Python program to create a tuple with different data types.

```
my_tuple = (1, "Hello", 3.14, True, [1, 2, 3], {'a': 1, 'b': 2}, (5, 6), None)
for i in my_tuple:
    j = type(i)
    print(f"{i}:{j}")
```

1:<class 'int'>
Hello:<class 'str'>
3.14:<class 'float'>
True:<class 'bool'>
[1, 2, 3]:<class 'list'>
{'a': 1, 'b': 2}:<class 'dict'>
(5, 6):<class 'tuple'>
None:<class 'NoneType'>

45)Write a Python program to unzip a list of tuples into individual lists.

```
my_list = [(1,"Apple"),(2,"Banana"),(3,"Mango"),(4,"Cherry")]
list1 , list2 = zip(*my_list)
print("List1:",list(list1))
print("List2:",list(list2))
```

```
➦ List1: [1, 2, 3, 4]
List2: ['Apple', 'Banana', 'Mango', 'Cherry']
```

46)Write a Python program to convert a list of tuples into a dictionary.

```
my_list = [(1,"Apple"),(2,"Banana"),(3,"Mango"),(4,"Cherry")]
my_dict = dict(my_list)
print(my_dict)
```

```
➦ {1: 'Apple', 2: 'Banana', 3: 'Mango', 4: 'Cherry'}
```

47)How will you create a dictionary using tuples in python?

```
my_tuple = ((1,"Apple"),(2,"Banana"),(3,"Mango"),(4,"Cherry"))
my_dict = dict(my_tuple)
print(my_dict)
```

```
➦ {1: 'Apple', 2: 'Banana', 3: 'Mango', 4: 'Cherry'}
```

48)Write a Python script to sort (ascending and descending) a dictionary by value.

```
my_dict = {'Apple': 3, 'Banana': 1, 'Cherry': 2, 'Mango': 4}
sorted_asending = dict(sorted(my_dict.items(),key=lambda item: item[1]))
sorted_desending = dict(sorted(my_dict.items(),key=lambda item:item[1],reverse=True))
print("Asending:",sorted_asending)
print("Desending:",sorted_desending)
```

```
➦ Asending: {'Banana': 1, 'Cherry': 2, 'Apple': 3, 'Mango': 4}
Desending: {'Mango': 4, 'Apple': 3, 'Cherry': 2, 'Banana': 1}
```

49)Write a Python script to concatenate following dictionaries to create a new one.

```
dict1 = {'Apple':1 , 'Banana':2}
dict2 = {'Cherry':3 , 'Mango':4}
my_list = [dict1,dict2]
my_dict = {}
for i in my_list:
    my_dict.update(i)
print(my_dict)
```

```
➦ {'Apple': 1, 'Banana': 2, 'Cherry': 3, 'Mango': 4}
```

50)Write a Python script to check if a given key already exists in a dictionary.

```
my_dict = {'Apple': 3, 'Banana': 1, 'Cherry': 2, 'Mango': 4}
key_check = 'Mango'
if key_check in my_dict:
    print("Exists in a dictionary")
else:
    print("Not Exists in a dictionary")
```

```
➦ Exists in a dictionary
```

51)How Do You Traverse Through a Dictionary Object in Python?

Traversing through a Python dictionary allows you to access its keys, values, or key-value pairs.

i)Using keys():

```
my_dict = {'Apple': 3, 'Banana': 1, 'Cherry': 2, 'Mango': 4}
for key in my_dict.keys():
    print(key)
```

```
➦ Apple
    Banana
    Cherry
    Mango
```

i)Using values():

```
my_dict = {'Apple': 3, 'Banana': 1, 'Cherry': 2, 'Mango': 4}
for value in my_dict.values():
    print(value)
```

```
➦ 3
   1
   2
   4
```

iii)Using items():

```
my_dict = {'Apple': 3, 'Banana': 1, 'Cherry': 2, 'Mango': 4}
for key,value in my_dict.items():
    print(f"{key}:{value}")
```

```
➦ Apple:3
    Banana:1
    Cherry:2
    Mango:4
```

52)How Do You Check the Presence of a Key in A Dictionary?

```
my_dict = {'Apple': 3, 'Banana': 1, 'Cherry': 2, 'Mango': 4}
if 'Cherry' in my_dict:
    print("Present")
else:
    print("Not Present")
```

```
➦ Present
```

53)Write a Python script to print a dictionary where the keys are numbers between 1 and 15.

```
my_dict = {i: f"{i}" for i in range(1, 16)}
print(my_dict)
```

```
➦ {1: '1', 2: '2', 3: '3', 4: '4', 5: '5', 6: '6', 7: '7', 8: '8', 9: '9', 10: '10', 11: '11', 12: '12', 13: '13', 14: '14'}
```

54)Write a Python program to check multiple keys exists in a dictionary.

```
my_dict = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
check_dict = ['a','d']
if all(key in my_dict for key in check_dict):
    print(True)
else:
    print(False)
```

```
➦ True
```

55)Write a Python script to merge two Python dictionaries.

```
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'd': 4}
dict1.update(dict2)
print(dict1)
```

```
➦ {'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

56)Write a Python program to map two lists into a dictionary Sample output: Counter ({'a': 400, 'b': 400,'d': 400, 'c': 300}).

```
keys = ['a', 'b', 'c', 'd']
values = [400, 400, 300, 400]
my_dict = dict(zip(keys,values))
print(my_dict)
```

```
➦ {'a': 400, 'b': 400, 'c': 300, 'd': 400}
```

57)Write a Python program to find the highest 3 values in a dictionary.

```
def highest_three_values(my_dict):
    highest_three = sorted(my_dict.values(),reverse=True)[:3]
    print(highest_three)
highest_three_values({'a': 15, 'b': 22, 'c': 9, 'd': 34, 'e': 18, 'f': 29})
```

```
➦ [34, 29, 22]
```

58)Write a Python program to combine values in python list of dictionaries. Sample data: [{ 'item': 'item1', 'amount': 400}, { 'item': 'item2', 'amount': 300}, o { 'item': 'item1', 'amount': 750}] Expected Output: • Counter ({'item1': 1150, 'item2': 300})

```
from collections import Counter
data = [{ 'item': 'item1', 'amount': 400}, { 'item': 'item2', 'amount': 300}, { 'item': 'item1', 'amount': 750}]
combine_values=Counter()
for i in data:
    item = i['item']
    amount = i['amount']
    combine_values[item]+=amount
print(combine_values)
```

```
➦ Counter({'item1': 1150, 'item2': 300})
```

59)Write a Python program to create a dictionary from a string. Note: Track the count of the letters from the string.

```
my_str = "Hello Python"
my_dict = {}
for i in my_str:
    if i in my_dict:
        my_dict[i]+=1
    else:
        my_dict[i]=1
print(my_dict)
```

```
➦ {'H': 1, 'e': 1, 'l': 2, 'o': 2, ' ': 1, 'P': 1, 'y': 1, 't': 1, 'h': 1, 'n': 1}
```

60)Sample string: 'w3resource' Expected output: • {'3': 1,'s': 1, 'r': 2, 'u': 1, 'w': 1, 'c': 1, 'e': 2, 'o': 1}

```
my_str = "w3resource"
my_dict = {}
for i in my_str:
    if i in my_dict:
        my_dict[i]+=1
    else:
        my_dict[i]=1
print(my_dict)
```

```
➦ {'w': 1, '3': 1, 'r': 2, 'e': 2, 's': 1, 'o': 1, 'u': 1, 'c': 1}
```

61)Write a Python function to calculate the factorial of a number (a nonnegative integer)

```
def factorial(n):
    fact=1
    for i in range(1,n+1):
        fact=fact*i
    print(fact)
factorial(5)
```

➞ 120

62)Write a Python function to check whether a number is in a given range.

```
r1 = int(input("Enter First Range:"))
r2 = int(input("Enter Second Range:"))
num = int(input("Enter a Number:"))
if num in range(r1,r2):
    print("Number in Range")
else:
    print("Number is Not in Range")
```

➞ Enter First Range:23
Enter Second Range:78
Enter a Number:45
Number in Range

63)Write a Python function to check whether a number is perfect or not.

```
def perfect_number(n):
    if n<1:
        return False
    sum1 = sum(i for i in range(1,n) if n%i==0)
    return sum1==n
test_numbers = [6, 28, 496, 8128, 7, 15]
results = {n:perfect_number(n) for n in test_numbers}
print(results)
```

➞ {6: True, 28: True, 496: True, 8128: True, 7: False, 15: False}

64)Write a Python function that checks whether a passed string is palindrome or not.

```
str1 = input("Enter a String:")
str2 = str1[::-1]
if str1==str2:
    print("String is Palindrome")
else:
    print("String is Not Palindrome")
```

➞ Enter a String:neha
String is Not Palindrome

65)How Many Basic Types of Functions Are Available in Python?

In Python, there are two main types of functions:

i)Built-in Functions: These functions are pre-defined in Python and are available for use without requiring any additional code. They serve various purposes, such as performing mathematical operations, manipulating strings, working with lists, and more. Examples of built-in functions include `print()`, `len()`, and `sum()`.

ii)User-Defined Functions: These functions are defined by the user to perform specific tasks. You can create your own functions based on your requirements. For example, you can define a function to calculate the factorial of a number or to validate user input.


66)How can you pick a random item from a list or tuple?

i)List:

```
import random
my_list = [1, 2, 3, 4, 5]
random_item = random.choice(my_list)
print(random_item)
```

 1

```
import random
my_list = [1, 2, 3, 4, 5]
random_items = random.choices(my_list, k=3)
print(random_items)
# random.choices() can pick the same item multiple times. If you need unique items, you can use random.sample():
```

 [5, 5, 1]

ii)Tuple:

```
import random
my_tuple = (1, 2, 3, 4, 5)
random_item = random.choice(my_tuple)
print(random_item)
```

 4

67)How can you pick a random item from a range?

```
import random
random_item = random.randrange(10)
print(random_item)
random_item = random.randrange(2,10)
print(random_item)
random_item = random.randrange(1,20,2)
print(random_item)
```

 1
8
15


68)How can you get a random number in python?

```
import random
random_item = random.randrange(2,10)
print(random_item)
```

 9


69)How will you set the starting value in generating random numbers?

```
import random
random.seed(42)
random_number1 = random.random()
random_number2 = random.randint(1, 100)
random_number3 = random.uniform(1.0, 10.0)
print(random_number1)
print(random_number2)
print(random_number3)
```

 0.6394267984578837
4
7.673954497838496

70) How will you randomize the items of a list in place?

```
import random
my_list = [1, 2, 3, 4, 5]
random.shuffle(my_list)
print(my_list)
```

 [4, 3, 1, 5, 2]

71)What is File function in python? What are keywords to create and write file.

In Python, file handling allows you to work with files—reading from them, writing to them, and manipulating their content. To perform file operations, you will use the built-in `open()` function, which returns a file object.

i)open(): This function is used to open a file and returns a file object. The key is to use the right mode when opening:

ii)write(): This method of a file object is used to write a string to the file.


iii)close(): It's essential to close the file object after you're done writing to it to ensure that all changes are saved and resources are released.

What are keywords to create and write file.

-Create and write: 'w', 'x'

72)Write a Python program to read an entire text file.

```
# Mount Google Drive
from google.colab import drive
import pandas as pd
drive.mount('/content/drive')
file_path = '/content/drive/My Drive/Colab Notebooks/a.csv'
df = pd.read_csv(file_path)
print(df.head())
```

 Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)

```
      Hello World
0  This is for testing
1      Second Line
2      Third Line
3      Fourth Line
4      Fifth Line
```

```
from google.colab import drive
import pandas as pd


# Step 1: Mount Google Drive
drive.mount('/content/drive')

# Step 2: Read the CSV file
file_path_csv = '/content/drive/My Drive/Colab Notebooks/a.csv'
df = pd.read_csv(file_path_csv)

# Display the DataFrame (optional)
# print(df.head())

# Step 3: Write the DataFrame to a text file
file_path_txt = '/content/drive/My Drive/Colab Notebooks/a_converted.txt'
df.to_csv(file_path_txt, sep='\t', index=False)

# Print a message to confirm the file has been saved
print(f"File has been saved to {file_path_txt}")
with open(file_path_txt, 'r') as file:
    content = file.read()
print(content)
```

 Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)

```
File has been saved to /content/drive/My Drive/Colab Notebooks/a_converted.txt
Hello World
This is for testing
```


Second Line
Third Line
Fourth Line
Fifth Line

73)Write a Python program to append text to a file and display the text.

```
file_path = '/content/drive/My Drive/Colab Notebooks/a_converted.txt'
with open(file_path, "a") as f:
    f.write('This is a new line \n')
with open(file_path, "r") as f:
    content = f.read()
print(content)
```

➦ Hello World
This is for testing
Second Line
Third Line
Fourth Line
Fifth Line
This is a new line

74)Write a Python program to read first n lines of a file.

```
def read_first_n_lines(file_path, n):
    with open(file_path, 'r') as file:
        for i in range(n):
            line = file.readline()
            if not line:
                break
            print(line.strip())
file_path = '/content/drive/My Drive/Colab Notebooks/a_converted.txt'
n = 2
read_first_n_lines(file_path, n)
```

➦ Hello World
This is for testing

75)Write a Python program to read last n lines of a file.

```
from collections import deque
def read_last_n_lines(file_path, n):
    with open(file_path, 'r') as file:
        lines = deque(file, maxlen=n)
    return list(lines)
file_path = '/content/drive/My Drive/Colab Notebooks/a_converted.txt'
n = 2
last=read_last_n_lines(file_path,n)
for line in last:
    print(line.strip())
```

➦ Fifth Line
This is a new line

76)Write a Python program to read a file line by line and store it into a list.

```
def read_file_to_list(file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()
    return [line.strip() for line in lines]
file_path = '/content/drive/My Drive/Colab Notebooks/a_converted.txt'
read_file_to_list(file_path)
```

➦ ['Hello World',
'This is for testing',

```
'Second Line',
'Third Line',
'Fourth Line',
'Fifth Line',
'This is a new line']
```

77)Write a Python program to read a file line by line store it into a variable.

```
def read_file_to_variable(file_path):
    content = ""
    with open(file_path, 'r') as file:
        for line in file:
            content += line
    return content
file_path = '/content/drive/My Drive/Colab Notebooks/a_converted.txt'
read_file_to_variable(file_path)
```

```
↗ 'Hello World\nThis is for testing \nSecond Line \nThird Line \nFourth Line
\nFifth Line\nThis is a new line \n'
```

78)Write a python program to find the longest words.

```
def find_longest_words(file_path):
    with open(file_path, 'r') as f:
        content = f.read()
    import re
    words = re.findall(r'\b\w+\b', content)
    max_length = max(len(word) for word in words)
    longest_words = [word for word in words if len(word) == max_length]
    return longest_words
file_path = '/content/drive/My Drive/Colab Notebooks/a_converted.txt'
longest_words = find_longest_words(file_path)
for word in longest_words:
    print(word)
```

```
↗ testing
```

79)Write a Python program to count the number of lines in a text file.

```
def count_lines_in_file(file_path):
    with open(file_path, 'r') as file:
        line_count = sum(1 for _ in file)
    return line_count
file_path = '/content/drive/My Drive/Colab Notebooks/a_converted.txt'
count_lines_in_file(file_path)
```

```
↗ 7
```

80)Write a Python program to count the frequency of words in a file.


```
from collections import Counter
import re
def count_word_frequency(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        text = file.read()
    words = re.findall(r'\b\w+\b', text.lower())
    word_count = Counter(words)
    for word, count in word_count.items():
        print(f"{word}: {count}")
file_path = '/content/drive/My Drive/Colab Notebooks/a_converted.txt'
count_word_frequency(file_path)
```

```
↗ hello: 1
world: 1
this: 2
is: 2
for: 1
testing: 1
second: 1
```

```
line: 5
third: 1
fourth: 1
fifth: 1
a: 1
new: 1
```

81)Write a Python program to write a list to a file.

```
def write_list_to_file(file_path, items):
    with open(file_path, 'w') as file:
        for item in items:
            file.write(f"{item}\n")
    print(f"List successfully written to {file_path}")
file_path = '/content/drive/My Drive/Colab Notebooks/a_converted.txt'
items = ["Apple", "Banana", "Cherry"]
write_list_to_file(file_path, items)
```

 List successfully written to /content/drive/My Drive/Colab Notebooks/a_converted.txt

82)Write a Python program to copy the contents of a file to another file.

Start coding or [generate](#) with AI.

83)Explain Exception handling? What is an Error in Python?

Exception handling is a crucial concept in Python that allows you to gracefully handle errors during program execution.

Errors in Python:

i)syntax errors: These occur when the interpreter encounters incorrect syntax in your code (e.g., misspelled keywords, missing colons, unbalanced parentheses).


ii)Exceptions: These are raised during program execution due to internal events that change the normal flow of the program. Unlike syntax errors, exceptions don't necessarily stop the program; they alter its behavior.

Common Types of Exceptions:SyntaxError , TypeError , NameError , IndexError , KeyError , ValueError , AttributeError , ZeroDivisionError , ImportError.

84)How many except statements can a try-except block have? Name Some built-in exception classes:

A try-except block in Python can have multiple except statements to handle different exceptions. There is no strict limit to the number of except statements you can have, so you can add as many as you need to handle different types of exceptions.

```
from math import factorial
num = int(input("enter a n :"))
try:
    ans = factorial(num)
    print(ans)
except NameError as e:
    print(e)
except IndexError as e:
    print(e)
except ValueError as msg:
    print(msg)
except Exception as e:
    print(e)
```

 enter a n :5
120

85)When will the else part of try-except-else be executed?

The else block in a try-except-else construct is executed only if no exceptions occur within the try block.

```
try:
    result = 10 / 2
except ZeroDivisionError:
    print("Oops! You tried to divide by zero.")
else:
    print("No exceptions occurred. This runs!")
```

➞ No exceptions occurred. This runs!

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
else:
    print("Division was successful. Result:", result)
```

➞ Cannot divide by zero!

86)Can one block of except statements handle multiple exception?

Yes, a single except block can handle multiple exceptions in Python. You can specify multiple exception types within the same except block, separated by parentheses.

```
try:
    x = int(input("Enter a number: "))
    result = 10 / x
except (ValueError, ZeroDivisionError) as e:
    print(f"An error occurred: {e}")
```

➞ Enter a number: 0
An error occurred: division by zero

87)When is the finally block executed?

The finally block in a try-except-finally construct is executed always, regardless of whether an exception was raised or not in the try block. This includes cases where an exception is caught by an except block, an exception is not caught and propagates out of the block, or no exception occurs at all.

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print("Caught division by zero")
finally:
    print("This will always be executed")
```

➞ Caught division by zero
This will always be executed

88)What happens when „1“== 1 is executed?

When the expression "1" == 1 is executed in Python, it evaluates to False.


```
print('1' == 1)
```

➞ False

89)How Do You Handle Exceptions with Try/Except/Finally in Python? Explain with coding snippets.

```
mylist = [10,20,30,40]
```

```
try:  
    print(mylist[4])  
except:  
    print('error')
```

 error

```
try:  
    result = 10 / 2  
except ZeroDivisionError:  
    print("Oops! You tried to divide by zero.")  
finally:  
    print("This always runs, no matter what.")
```

 This always runs, no matter what.

90)Write python program that user to enter only odd numbers, else will raise an exception.

```
class EvenNumberError(Exception):  
  
    def __init__(self, number):  
        self.number = number  
        self.message = f"{number} is an even number. Please enter an odd number."  
        super().__init__(self.message)  
  
def get_odd_number():  
    while True:  
        try:  
            number = int(input("Enter an odd number: "))  
            if number % 2 == 0:  
                raise EvenNumberError(number)  
            print(f"Thank you for entering an odd number: {number}")  
            break  
        except EvenNumberError as e:  
            print(e)  
        except ValueError:  
            print("Invalid input. Please enter a valid integer.")
```