Python Assignment

1. What are the types of Applications?

* **Types of Applications**

Applications can be broadly classified as:

* **Web Applications** – Run in browsers (e.g., Gmail, Facebook).
* **Mobile Applications** – Designed for smartphones (e.g., WhatsApp, Instagram).
* **Desktop Applications** – Installed on computers (e.g., MS Word, Photoshop).
* **Enterprise Applications** – Large-scale business apps (e.g., ERP systems).
* **Embedded Applications** – Run on hardware devices (e.g., washing machine software).
* **Cloud Applications** – Hosted on the cloud (e.g., Google Drive).
* **Game Applications** – Built for entertainment (e.g., PUBG, Minecraft).

### ****2) What is Programming?****

Programming is the process of **writing instructions** (code) that a computer can understand and execute to perform specific tasks or solve problems.

### ****3) What is Python?****

Python is a **high-level, interpreted, object-oriented programming language** that is easy to learn, highly readable, and supports multiple paradigms like procedural, functional, and object-oriented programming. It is widely used in web development, data analysis, machine learning, automation, and more.

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

num = float(input("Enter a number: "))

if num > 0:

print("Positive number")

elif num < 0:

print("Negative number")

else:

print("Zero")

### ****5) Python program to get the Factorial of a given number****

num = int(input("Enter a number: "))

factorial = 1

if num < 0:

print("Factorial does not exist for negative numbers.")

elif num == 0:

print("The factorial of 0 is 1.")

else:

for i in range(1, num + 1):

factorial \*= i

print("The factorial of", num, "is", factorial)

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

n = int(input("Enter the number of terms: "))

a, b = 0, 1

print("Fibonacci sequence:")

for i in range(n):

print(a, end=" ")

a, b = b, a + b

**7) How memory is managed in Python?**

* Python uses **automatic memory management**.
* It has a **private heap space** where all objects and data structures are stored.
* The **Python memory manager** allocates and deallocates memory.
* **Garbage Collection** automatically removes unused objects to free memory.
* It also uses **reference counting** and a **cyclic garbage collector**.

**8) Purpose of continue statement in Python**

The continue statement is used inside loops to **skip the rest of the code** for the current iteration and move to the **next iteration** of the loop.

Example:

for i in range(5):

if i == 2:

continue

print(i)

(Here, when i is 2, it skips printing and continues with the next value.)

**9) Swap two numbers with and without a temporary variable**

**Using a temporary variable:**

a = int(input("Enter first number: "))

b = int(input("Enter second number: "))

temp = a

a = b

b = temp

print("After swapping: a =", a, ", b =", b)

**Without using a temporary variable:**

a = int(input("Enter first number: "))

b = int(input("Enter second number: "))

a, b = b, a

print("After swapping: a =", a, ", b =", b)

**10) Check if a number is even or odd**

num = int(input("Enter a number: "))

if num % 2 == 0:

print(num, "is Even")

else:

print(num, "is Odd")

**11) Test whether a passed letter is a vowel or not**

# Check if a letter is vowel or not

def is\_vowel(letter):

vowels = "aeiouAEIOU"

return letter in vowels

# Input

ch = input("Enter a letter: ")

if len(ch) == 1 and ch.isalpha():

if is\_vowel(ch):

print(f"{ch} is a vowel.")

else:

print(f"{ch} is not a vowel.")

else:

print("Please enter a single alphabet.")

**12) Sum of three integers (if two values are equal, sum = 0)**

*def sum\_three\_numbers(a, b, c):*

*if a == b or b == c or a == c:*

*return 0*

*else:*

*return a + b + c*

*print("Sum:", sum\_three\_numbers(10, 20, 30)) # 60*

*print("Sum:", sum\_three\_numbers(10, 10, 30)) # 0*

**13) Return true if two integers are equal or their sum or difference is 5**

def check\_numbers(x, y):

return x == y or abs(x - y) == 5 or (x + y) == 5

print(check\_numbers(10, 10)) # True (equal)

print(check\_numbers(7, 2)) # True (difference is 5)

print(check\_numbers(2, 3)) # True (sum is 5)

print(check\_numbers(4, 6)) # False

**14) Sum of first n positive integers**

def sum\_of\_n(n):

return n \* (n + 1) // 2 # Formula

n = int(input("Enter a positive number: "))

print(f"Sum of first {n} positive integers is:", sum\_of\_n(n))

**15) Calculate the length of a string**

# Calculate string length

text = input("Enter a string: ")

print("Length of the string is:", len(text))

**16) Count character frequency in a string**

def char\_frequency(s):

freq = {}

for char in s:

freq[char] = freq.get(char, 0) + 1

return freq

# Example

text = "hello world"

print(char\_frequency(text))

✅ Output (for "hello world"):  
{'h': 1, 'e': 1, 'l': 3, 'o': 2, ' ': 1, 'w': 1, 'r': 1, 'd': 1}

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

* In Python, **negative indexes** allow you to access elements from the **end of a sequence** (like string, list, or tuple).
* -1 refers to the **last element**, -2 is the **second last**, and so on.  
  ✅ Example:

text = "Python"

print(text[-1]) # 'n'

print(text[-2]) # 'o'

**18) Count occurrences of a substring in a string**

text = "Python is easy, and Python is powerful"

substring = "Python"

count = text.count(substring)

print(f"The substring '{substring}' occurs {count} times.")

✅ Output: The substring 'Python' occurs 2 times.

**19) Count occurrences of each word in a sentence**

def word\_count(sentence):

words = sentence.split()

freq = {}

for word in words:

freq[word] = freq.get(word, 0) + 1

return freq

# Example

sentence = "Python is easy and Python is powerful"

print(word\_count(sentence))

**20) Join two strings with a space and swap the first two characters**

def swap\_strings(s1, s2):

new\_s1 = s2[:2] + s1[2:]

new\_s2 = s1[:2] + s2[2:]

return new\_s1 + " " + new\_s2

# Example

print(swap\_strings("hello", "world"))

**21) Add 'in' or 'ly' depending on conditions**

def add\_suffix(s):

if len(s) < 3:

return s

elif s.endswith("ing"):

return s + "ly"

else:

return s + "in"

# Example

print(add\_suffix("play")) # "playin"

print(add\_suffix("playing")) # "playingly"

print(add\_suffix("go")) # "go"

**22) Reverse string if its length is multiple of 4**

def reverse\_if\_multiple\_of\_4(s):

return s[::-1] if len(s) % 4 == 0 else s

# Example

print(reverse\_if\_multiple\_of\_4("abcd")) # "dcba"

print(reverse\_if\_multiple\_of\_4("python")) # "python"

**23) First 2 and last 2 characters**

def first\_last\_chars(s):

if len(s) < 2:

return ""

return s[:2] + s[-2:]

# Example

print(first\_last\_chars("Python")) # "Pyon"

print(first\_last\_chars("A")) # ""

**24) Insert string in the middle of another string**

def insert\_middle(main, insert):

middle = len(main) // 2

return main[:middle] + insert + main[middle:]

# Example

print(insert\_middle("Python", "123")) # "Py123thon"

**25) What is a List? How to reverse it?**

* A **List** in Python is an ordered, mutable collection of items.  
  Example:
* my\_list = [1, 2, 3, 4, 5]
* Ways to **reverse a list**:
* # Method 1: Using reverse()
* my\_list.reverse()
* print(my\_list)
* # Method 2: Using slicing
* print(my\_list[::-1])

**26) Remove last object from a list**

my\_list = [10, 20, 30, 40]

# Method 1: Using pop()

my\_list.pop()

print(my\_list) # [10, 20, 30]

# Method 2: Using slicing

my\_list = my\_list[:-1]

print(my\_list) # [10, 20]

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

* list1[-1] means the **last element** of the list.
* So:

list1 = [2, 33, 222, 14, 25]

print(list1[-1]) # 25

✅ Answer: 25

**28) Difference between append() and extend()**

* **append(x)** → adds the whole object x as a **single element** at the end of the list.
* **extend(iterable)** → adds each element of the iterable to the list (merges lists).

Example:

list1 = [1, 2, 3]

list1.append([4, 5])

print(list1) # [1, 2, 3, [4, 5]]

list1 = [1, 2, 3]

list1.extend([4, 5])

print(list1) # [1, 2, 3, 4, 5]

**29) Largest, smallest, and sum of all numbers in a list**

def list\_stats(numbers):

largest = max(numbers)

smallest = min(numbers)

total = sum(numbers)

return largest, smallest, total

# Example

nums = [10, 25, 3, 99, 56]

print(list\_stats(nums)) # (99, 3, 193)

**30) How will you compare two lists?**

You can compare lists in different ways:

1. **Check if equal (same elements, same order)**

list1 = [1, 2, 3]

list2 = [1, 2, 3]

print(list1 == list2) # True

1. **Check if they have the same elements (ignoring order)**

list1 = [1, 2, 3]

list2 = [3, 2, 1]

print(sorted(list1) == sorted(list2)) # True

1. **Element-wise comparison**

for a, b in zip(list1, list2):

print(a == b)

**31) Count strings with length ≥ 2 and same first & last character**

def match\_words(words):

count = 0

for word in words:

if len(word) >= 2 and word[0] == word[-1]:

count += 1

return count

# Example

words = ["abc", "xyz", "aba", "1221", "aa"]

print(match\_words(words)) # 3

**32) Remove duplicates from a list**

def remove\_duplicates(lst):

return list(set(lst))

# Example

nums = [1, 2, 2, 3, 4, 4, 5]

print(remove\_duplicates(nums)) # [1, 2, 3, 4, 5]

**33) Check if a list is empty or not**

def is\_empty(lst):

return len(lst) == 0

print(is\_empty([])) # True

print(is\_empty([1, 2, 3])) # False

**34) Check if two lists have at least one common member**

def common\_member(list1, list2):

for item in list1:

if item in list2:

return True

return False

# Example

print(common\_member([1, 2, 3], [4, 5, 6])) # False

print(common\_member([1, 2, 3], [3, 4, 5])) # True

**35) List of first & last 5 squares between 1 and 30**

squares = [x\*\*2 for x in range(1, 31)]

print(squares[:5]) # First 5

print(squares[-5:]) # Last 5

✅ Output:  
[1, 4, 9, 16, 25] and [676, 729, 784, 841, 900]

**36) Return new list with unique elements**

def unique\_list(lst):

return list(set(lst))

# Example

print(unique\_list([1, 2, 2, 3, 4, 4, 5])) # [1, 2, 3, 4, 5]

**37) Convert list of characters into a string**

chars = ['P', 'y', 't', 'h', 'o', 'n']

result = ''.join(chars)

print(result) # "Python"

**38) Select an item randomly from a list**

import random

items = [10, 20, 30, 40, 50]

print(random.choice(items)) # Random element

**39) Find the second smallest number in a list**

def second\_smallest(nums):

unique\_nums = sorted(set(nums))

if len(unique\_nums) < 2:

return None

return unique\_nums[1]

# Example

print(second\_smallest([5, 1, 8, 2, 3, 1])) # 2

**40) Get unique values from a list**

nums = [1, 2, 2, 3, 4, 4, 5]

unique = list(set(nums))

print(unique) # [1, 2, 3, 4, 5]

**41) Check whether a list contains a sublist**

def is\_sublist(lst, sub):

for i in range(len(lst) - len(sub) + 1):

if lst[i:i+len(sub)] == sub:

return True

return False

# Example

print(is\_sublist([1, 2, 3, 4, 5], [2, 3])) # True

print(is\_sublist([1, 2, 3, 4, 5], [3, 6])) # False

**42) Split a list into different variables**

data = [10, 20, 30]

a, b, c = data

print(a, b, c) # 10 20 30

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

* **Tuple**: An **ordered, immutable** collection of items in Python. Declared using ().  
  Example:
* tup = (1, 2, 3)
* **Difference between List and Tuple**:

| **Feature** | **List ( [] )** | **Tuple ( () )** |
| --- | --- | --- |
| Mutability | Mutable (can be changed) | Immutable (cannot be changed) |
| Syntax | list1 = [1,2,3] | tuple1 = (1,2,3) |
| Performance | Slower | Faster |
| Use case | When items need to change | When items must stay fixed |

**44) Create a tuple with different data types**

my\_tuple = (1, "Hello", 3.14, True)

print(my\_tuple)

✅ Output: (1, 'Hello', 3.14, True)

**45) Unzip a list of tuples into individual lists**

pairs = [(1, 'a'), (2, 'b'), (3, 'c')]

numbers, letters = zip(\*pairs)

print(list(numbers)) # [1, 2, 3]

print(list(letters)) # ['a', 'b', 'c']

**46) Convert a list of tuples into a dictionary**

pairs = [(1, 'a'), (2, 'b'), (3, 'c')]

dictionary = dict(pairs)

print(dictionary) # {1: 'a', 2: 'b', 3: 'c'}

**47) Create a dictionary using tuples in Python**

You can directly use dict() with tuples of key-value pairs:

my\_dict = dict(((1, 'apple'), (2, 'banana'), (3, 'cherry')))

print(my\_dict) # {1: 'apple', 2: 'banana', 3: 'cherry'}

**48) Sort dictionary by value (ascending & descending)**

my\_dict = {'a': 3, 'b': 1, 'c': 2}

# Ascending

asc = dict(sorted(my\_dict.items(), key=lambda x: x[1]))

print("Ascending:", asc)

# Descending

desc = dict(sorted(my\_dict.items(), key=lambda x: x[1], reverse=True))

print("Descending:", desc)

✅ Output:

Ascending: {'b': 1, 'c': 2, 'a': 3}

Descending: {'a': 3, 'c': 2, 'b': 1}

**49) Concatenate multiple dictionaries into a new one**

dict1 = {1: 'a', 2: 'b'}

dict2 = {3: 'c', 4: 'd'}

dict3 = {5: 'e'}

new\_dict = {\*\*dict1, \*\*dict2, \*\*dict3}

print(new\_dict) # {1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e'}

**50) Check if a key already exists in a dictionary**

my\_dict = {'name': 'John', 'age': 25}

key = 'age'

if key in my\_dict:

print(f"Key '{key}' exists.")

else:

print(f"Key '{key}' does not exist.")

**51) Traverse through a dictionary object**

my\_dict = {'a': 1, 'b': 2, 'c': 3}

# Traverse keys and values

for key, value in my\_dict.items():

print(key, ":", value)

✅ Output:

a : 1

b : 2

c : 3

**52) Check presence of a key in a dictionary**

* Using **in operator**:

my\_dict = {'x': 10, 'y': 20}

print('x' in my\_dict) # True

print('z' in my\_dict) # False

* Using **.get() method**:

print(my\_dict.get('x') is not None) # True

**53) Print a dictionary where keys are numbers between 1 and 15**

(Values will be squares of keys for example)

my\_dict = {x: x\*\*2 for x in range(1, 16)}

print(my\_dict)

✅ Output (partial):

{1: 1, 2: 4, 3: 9, ..., 15: 225}

**54) Check if multiple keys exist in a dictionary**

def check\_keys(dictionary, keys):

return all(key in dictionary for key in keys)

# Example

my\_dict = {'a': 1, 'b': 2, 'c': 3}

print(check\_keys(my\_dict, ['a', 'b'])) # True

print(check\_keys(my\_dict, ['a', 'd'])) # False

**55) Merge two dictionaries**

dict1 = {'a': 100, 'b': 200}

dict2 = {'c': 300, 'd': 400}

# Method 1: Using unpacking

merged = {\*\*dict1, \*\*dict2}

print(merged)

# Method 2: Using update()

dict1.update(dict2)

print(dict1)

✅ Output:  
{'a': 100, 'b': 200, 'c': 300, 'd': 400}

**56) Map two lists into a dictionary (with Counter example)**

from collections import Counter

keys = ['a', 'b', 'c', 'd']

values = [400, 400, 300, 400]

result = dict(zip(keys, values))

print(result) # {'a': 400, 'b': 400, 'c': 300, 'd': 400}

# Using Counter to match sample output

print(Counter(result))

✅ Output:

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

Counter({'a': 400, 'b': 400, 'd': 400, 'c': 300})

**57) Find the highest 3 values in a dictionary**

my\_dict = {'a': 100, 'b': 200, 'c': 50, 'd': 400, 'e': 300}

top3 = sorted(my\_dict.values(), reverse=True)[:3]

print("Top 3 values:", top3)

✅ Output:  
Top 3 values: [400, 300, 200]

**58) Combine values in list of dictionaries**

from collections import Counter

data = [

{'item': 'item1', 'amount': 400},

{'item': 'item2', 'amount': 300},

{'item': 'item1', 'amount': 750}

]

counter = Counter()

for d in data:

counter[d['item']] += d['amount']

print(counter)

✅ Output:  
Counter({'item1': 1150, 'item2': 300})

**59) Create a dictionary from a string (count letters)**

from collections import Counter

s = "w3resource"

letter\_count = Counter(s)

print(letter\_count)

**60) Example with given string**

s = "w3resource"

letter\_count = {ch: s.count(ch) for ch in s}

print(letter\_count)

✅ Output:

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

**61) Factorial of a number (non-negative integer)**

def factorial(n):

if n == 0 or n == 1:

return 1

else:

return n \* factorial(n-1)

# Example

print(factorial(5)) # 120

**62) Check whether a number is in a given range**

def in\_range(n, start, end):

return start <= n <= end

print(in\_range(5, 1, 10)) # True

print(in\_range(15, 1, 10)) # False

**63) Check whether a number is perfect**

👉 A **perfect number** is a number equal to the sum of its proper divisors (excluding itself).  
Example: 6 = 1 + 2 + 3

def is\_perfect(n):

if n < 2:

return False

divisors = [i for i in range(1, n) if n % i == 0]

return sum(divisors) == n

print(is\_perfect(6)) # True

print(is\_perfect(28)) # True

print(is\_perfect(12)) # False

**64) Check if a string is palindrome**

👉 Palindrome = same forwards and backwards.

def is\_palindrome(s):

return s == s[::-1]

print(is\_palindrome("madam")) # True

print(is\_palindrome("python")) # False

**65) How many basic types of functions are available in Python?**

* **Built-in functions** → Already provided by Python (e.g., len(), print(), max()).
* **User-defined functions** → Functions created by programmers using def or lambda.

👉 So, there are **two basic types**:

1. **Built-in functions**
2. **User-defined functions**

**66) Pick a random item from a list or tuple**

import random

items = [10, 20, 30, 40]

print(random.choice(items)) # Random from list

tup = (1, 2, 3, 4)

print(random.choice(tup)) # Random from tuple

**67) Pick a random item from a range**

import random

print(random.choice(range(1, 11))) # Random number between 1 and 10

**68) Get a random number in Python**

import random

# Random float between 0 and 1

print(random.random())

# Random integer between given range

print(random.randint(1, 100))

**69) Set the starting value (seed) for random numbers**

👉 Use **random.seed()** to reproduce results.

import random

random.seed(10)

print(random.randint(1, 100)) # Always gives the same output for seed=10

**70) Randomize the items of a list in place**

import random

items = [1, 2, 3, 4, 5]

random.shuffle(items)

print(items) # Items shuffled in place

**71) What is File function in Python? Keywords to create/write file**

* **File functions** are used to perform operations like read, write, and append.
* In Python, you use **open()** to open/create a file.
  + "w" → write (creates new file or overwrites existing)
  + "a" → append
  + "r" → read

Example:

f = open("example.txt", "w")

f.write("Hello, world!")

f.close()

**72) Read an entire text file**

with open("example.txt", "r") as f:

data = f.read()

print(data)

**73) Append text to a file and display**

with open("example.txt", "a") as f:

f.write("\nThis is appended text.")

with open("example.txt", "r") as f:

print(f.read())

**74) Read first n lines of a file**

def read\_first\_n(filename, n):

with open(filename, "r") as f:

for i in range(n):

print(f.readline(), end="")

read\_first\_n("example.txt", 3)

**75) Read last n lines of a file**

def read\_last\_n(filename, n):

with open(filename, "r") as f:

lines = f.readlines()

for line in lines[-n:]:

print(line, end="")

read\_last\_n("example.txt", 2)

**76) Read a file line by line and store into a list**

with open("example.txt", "r") as f:

lines = f.readlines()

print(lines) # Each line as list element

**77) Read a file line by line and store into a variable**

with open("example.txt", "r") as f:

content = ""

for line in f:

content += line

print(content)

**78) Find the longest words in a file**

def longest\_word(filename):

with open(filename, "r") as f:

words = f.read().split()

max\_len = max(len(word) for word in words)

return [word for word in words if len(word) == max\_len]

print(longest\_word("example.txt"))

**79) Count number of lines in a text file**

with open("example.txt", "r") as f:

count = sum(1 for \_ in f)

print("Number of lines:", count)

**80) Count frequency of words in a file**

from collections import Counter

with open("example.txt", "r") as f:

words = f.read().split()

freq = Counter(words)

print(freq)

**81) Write a list to a file**

data = ["Python\n", "is\n", "awesome\n"]

with open("list.txt", "w") as f:

f.writelines(data)

**82) Copy contents of one file to another**

with open("example.txt", "r") as f1, open("copy.txt", "w") as f2:

for line in f1:

f2.write(line)

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

* **Error** → Problem in the program that stops execution.
  + Types: **Syntax errors** (wrong code structure) & **Exceptions** (runtime errors like division by zero).
* **Exception handling** → Managing runtime errors using try-except.

Example:

try:

x = 10 / 0

except ZeroDivisionError:

print("Cannot divide by zero")

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

* A try block can have **multiple except blocks** (one for each error type).
* Built-in exception classes:
  + ZeroDivisionError
  + ValueError
  + TypeError
  + KeyError
  + IndexError

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

* The **else block** executes **only if no exception occurs** inside the try block.

Example:

try:

x = 5 / 1

except ZeroDivisionError:

print("Error!")

else:

print("No error, else block executed") # Runs here

**86) Can one block of except statements handle multiple exceptions?**

👉 Yes ✅  
You can catch multiple exceptions in one except using a tuple.

try:

x = int("abc") # Raises ValueError

except (ValueError, TypeError) as e:

print("Error:", e)

**87) When is the finally block executed?**

* The **finally block** is executed **no matter what happens** in the try-except block—whether an exception occurs or not.
* It is typically used to **release resources**, like closing a file.

Example:

try:

x = 10 / 2

except ZeroDivisionError:

print("Cannot divide by zero")

finally:

print("This will always execute")

✅ Output:

This will always execute

**88) What happens when '1' == 1 is executed?**

* '1' is a **string**, 1 is an **integer**.
* Python **does not automatically convert types**, so '1' == 1 evaluates to **False**.

print('1' == 1) # False

**89) Handling exceptions with Try/Except/Finally**

* try → Code that might raise an exception.
* except → Handle the exception.
* finally → Code that always runs, e.g., closing a file.

Example:

try:

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

data = f.read()

except FileNotFoundError:

print("File not found!")

finally:

print("Execution complete, closing resources")

✅ Output (if file not found):

File not found!

Execution complete, closing resources

**90) Program to allow only odd numbers**

def input\_odd():

try:

num = int(input("Enter an odd number: "))

if num % 2 == 0:

raise ValueError("Even number entered! Please enter an odd number.")

print("You entered:", num)

except ValueError as e:

print("Error:", e)

input\_odd()

Example:

Enter an odd number: 4

Error: Even number entered! Please enter an odd number.

Bottom of Form

Bottom of Form