1. import pandas as pd

s =pd.Series([1,2,3,4,5],index= ['a','b','c','d','e'])

print(s['f'])

A. KeyError B. IndexError C. ValueError D. None of the above mentioned

2. Which of the following is the correct extension of the Python file?

A. .python B. .pl C. .py D. .p

3. What will be the value of the following Python expression? 4 + 3 % 5

A. 7 B. 4 C. 2 D. 1

4. What are the values of the following Python expressions?

2\*\*(3\*\*2)

(2\*\*3)\*\*2

2\*\*3\*\*2

A. 512, 64, 512 C. 64, 512, 64

B. 512, 512, 512 D. 64, 64, 64

5. What will be the output of the following Python function?

len(["hello",2, 4, 6])

A. Error B. 4 C. 6

D. 3

6. What will be the output of the following Python code snippet?

for i in [1, 2, 3, 4] [::-1]:

print (i)

A. 4 3 2 1 B. 1 2 3 4 C. Error D. None

7. What is the purpose of NumPy in Python?

A. To do numerical calculations C. Both A and B

B. To do scientific computing D. None of the above

8. Observe the following code and identify what will be the outcome, where np is a numpy array?

a=np.array([1,2,3,4,5,6]) print(a)

A. [1 2 3 4 5] C. [0 1 2 3 4 5 6]

B. [1 2 3 4 5 6] D. None of the above

9. What will be the minimum number of arguments require to pass in pandas series?

A. 2 C. 4

B. 3 D. None of the above

10. NumPy is often used along with packages like?

A. Node.js C. Scipy

B. Matplotlib D. Both B and C

11. Which type of Programming does Python support?

A.Object oriented programming C. Structured programming

B. Functional programming D. All of the mentioned

12. A series object is size mutable.

A. True B. False

13. Which of the following is the truncation division operator in Python?

A. | C. /

B. // D. %

14. What will be the output of the following Python code?

x = 'abcd'

for i in x:

print(i.upper())

A. aBCD C. error

B. a b c d D. ABCD

15. What will be the output of the following Python program?

i = 0

while i < 5:

print(i)

i += 1

if i == 3:

break

else:

print(0)

A. error C. 0 1 2

B. 0 1 2 0 D. none of the above

16. What will be the output of the following Python program?

def addItem(listParam):

listParam += [1]

mylist = [1, 2, 3, 4]

addItem(mylist)

print(len(mylist))

A. 5 C. 2

B. 8 D. 1

17. What will be the output of following code?

import pandas as pd

series1 = pd.Series([10,20,30,40,50])

print (series1)

A.

0 10

1 20

2 30

3 40

4 50

dtype: int64

C.

0 10

1 20

2 30

3 40

4 50

dtype: float32

B.

1 10

2 20

3 30

4 40

5 50

dtype: int64

D. None of the above

18. A Dataframe object is value mutable.

A. True C. False

19. Amongst which of the following can be used to create various inputs using pandas DataFrame.

A. Lists, dict C. Numpy ndarrays and Another DataFrame

B. Series D. All of the above mentioned

20. NumPY stands for?

A. Numerical Python C. Numbering Python

B. Number in python D. None of the above

21.What will be the output of the following Python expression if x=56.236?

print("%.2f"%x)

a) 56.236

b) 56.23

c) 56.0000

d) 56.24

22. What will be the output of the following Python function?

len(["hello",2, 4, 6])

a) Error

b) 6

c) 4

d) 3

23. What will be the output of the following Python code?

x = 'abcd'

for i in x:

print(i.upper())

a)aBCD

b) a b c d

c) error

d)ABCD

24. What will be the output of the following Python code snippet?

for i in [1, 2, 3, 4][::-1]:

print (i)

a) 4 3 2 1

b) error

c) 1 2 3 4

d) none of the mentioned

25. What will be the output of the following Python statement?

1. >>>"a"+"bc"

a) bc

b) abc

c) a

d) bca

26. What arithmetic operators cannot be used with strings in Python?

a) \*

b) –

c) +

d) All of the mentioned

27. What will be the output of the following Python code?

print("abc. DEF".capitalize())

a) Abc. def

b) abc. def

c) Abc. Def

d) ABC. DEF

28. Which of the following statements is used to create an empty set in Python?

a) ( )

b) [ ]

c) { }

d) set()

29. Which of the following Python statements will result in the output: 6?

A = [[1, 2, 3],

[4, 5, 6],

[7, 8, 9]]

a) A[2][1]

b) A[1][2]

c) A[3][2]

d) A[2][3]

30. What is the maximum possible length of an identifier in Python?

a) 79 characters

b) 31 characters

c) 63 characters

d) none of the mentioned

31. What will be the output of the following Python program?

i = 0

while i < 5:

print(i)

i += 1

if i == 3:

break

else:

print(0)

a) error

b) 0 1 2 0

c) 0 1 2

d) none of the mentioned

Part B

Write in detail about branching statements in python.

If Statement

The **if** statement allows you to execute a block of code only if a particular condition is true

x = 5

if x > 0:

print("x is positive")

elif Statement

The **elif** statement allows you to check additional conditions after the initial **if** statement.

x = 5

if x < 0:

print("x is negative")

elif x > 0:

print("x is positive")

else:

print("x is zero")

else Statement

The **else** statement allows you to execute a block of code if none of the previous conditions are true

x = 10

if x < 0:

print("x is negative")

elif x > 0 and x <= 10:

print("x is between 1 and 10")

elif x > 10 and x <= 20:

print("x is between 11 and 20")

else:

print("x is greater than 20")

2. Elaborate the three Pandas objects with suitable examples.

Pandas is a popular data manipulation library in Python that provides three main objects to work with: **Series**, **DataFrame**, and **Index**

Series:

A Series is a one-dimensional labeled array that can hold any data type such as integers, floats, strings, or Python objects. It has two main components: an index and a data array. Here's an example of creating a Series:

import pandas as pd

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

index = ['a', 'b', 'c', 'd', 'e']

series = pd.Series(data, index=index)

print(series)

DataFrame:

A DataFrame is a two-dimensional labeled data structure that can hold data of different types such as integers, floats, strings, or Python objects. It has three main components: rows, columns, and an index.

import pandas as pd

data = {'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],

'age': [25, 30, 35, 40, 45],

'gender': ['F', 'M', 'M', 'M', 'F']}

df = pd.DataFrame(data)

print(df)

Index:

An Index is an immutable sequence used to label the rows and columns of a Series or DataFrame. It can hold any hashable data type such as integers, strings, or tuples.

import pandas as pd

index = pd.Index(['a', 'b', 'c', 'd', 'e'])

print(index)

Explain various ways of querying dataframe.

Querying a DataFrame involves selecting specific rows and/or columns based on certain criteria. There are several ways to query a DataFrame in Pandas.

Indexing and Slicing:

Indexing and Slicing can be used to select specific rows or columns of a DataFrame based on their position. We can use the .loc[] or .iloc[] methods to select rows and columns by label or by integer position.

import pandas as pd

df = pd.read\_csv('example.csv')

# select first row

print(df.iloc[0])

# select first two rows

print(df.iloc[:2])

# select specific columns by label

print(df.loc[:, ['column1', 'column2']])

Query Method:

The query() method allows us to filter rows of a DataFrame using a string expression. For example:

import pandas as pd

df = pd.read\_csv('example.csv')

# select rows where column1 > 10 and column2 is 'value'

print(df.query('column1 > 10 and column2 == "value"'))

Groupby Method:

The groupby() method allows us to group rows of a DataFrame based on a column or a set of columns.

import pandas as pd

df = pd.read\_csv('example.csv')

# group by column1 and calculate mean of column2 for each group

print(df.groupby('column1')['column2'].mean())

Merge Method:

The merge() method allows us to combine two DataFrames based on a common column. We can perform an inner join, outer join, left join, or right join to combine the DataFrames. For example:

import pandas as pd

df1 = pd.read\_csv('example1.csv')

df2 = pd.read\_csv('example2.csv')

# inner join two DataFrames based on column1

merged\_df = pd.merge(df1, df2, on='column1', how='inner')

Explain applying various functions in dataframe.

In Pandas, we can apply various functions to a DataFrame or a Series to manipulate, transform or aggregate data.

apply(): The apply() function is used to apply a function along an axis of the DataFrame or Series. We can pass a lambda function or a predefined function to apply() to perform element-wise operation on each row or column of the DataFrame. For example:

import pandas as pd

df = pd.read\_csv('example.csv')

# apply a lambda function to the column1

df['column1'] = df['column1'].apply(lambda x: x + 10)

# define a function to apply to column2

def func(x):

if x < 0:

return 'negative'

else:

return 'positive'

df['column2'] = df['column2'].apply(func)

map(): The map() function is used to map values of a Series to another set of values using a dictionary or a function. For example:

import pandas as pd

df = pd.read\_csv('example.csv')

# map values of column1 using a dictionary

mapping = {0: 'zero', 1: 'one', 2: 'two', 3: 'three'}

df['column1'] = df['column1'].map(mapping)

# map values of column2 using a function

def func(x):

if x == 'value1':

return 'A'

elif x == 'value2':

return 'B'

else:

return 'C'

df['column2'] = df['column2'].map(func)

agg(): The agg() function is used to aggregate data in a DataFrame. We can pass a list of functions or a dictionary of column names and functions to agg() to perform multiple aggregation operations on the DataFrame. For example:

import pandas as pd

df = pd.read\_csv('example.csv')

# calculate multiple aggregation operations on column1 and column2

agg\_dict = {'column1': ['mean', 'max'], 'column2': ['sum', 'count']}

df\_agg = df.agg(agg\_dict)

Write in detail about if, for and while in python.

**if** statement: The **if** statement is used for decision making in Python. It checks whether a certain condition is true or false and executes the code accordingly.

num = -5

if num > 0:

print("The number is positive")

elif num == 0:

print("The number is zero")

else:

print("The number is negative")

The **for** loop is used for iterating over a sequence (such as a list, tuple, or string) or a collection (such as a dictionary or a set). It executes a set of statements repeatedly for each item in the sequence or collection.

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

for num in numbers:

print(num)

The **while** loop is used for executing a set of statements repeatedly as long as a condition is true. It is used when we do not know the number of iterations in advance.

n = 10

a, b = 0, 1

count = 0

while count < n:

print(a)

a, b = b, a + b

count += 1

Write a program to calculate percentile and variance using NUMPY.

A percentile is a measure used in statistics that indicates the value below which a certain percentage of observations or data points in a dataset falls. For example, the 50th percentile (also known as the median) is the value below which 50% of the observations or data points fall. Percentiles are often used to describe the distribution of a dataset, and they can help identify outliers or extreme values.

Variance is a statistical measure that describes how much the data points in a dataset deviate from the mean or average value of the dataset. It is calculated by taking the average of the squared differences between each data point and the mean. A high variance indicates that the data points are spread out over a wide range of values, while a low variance indicates that the data points are clustered closely around the mean.

import numpy as np

# Create an array of random numbers

data = np.random.rand(100)

# Calculate the 75th percentile of the data

percentile = np.percentile(data, 75)

# Calculate the variance of the data

variance = np.var(data)

# Print the results

print("75th percentile:", percentile)

print("Variance:", variance)

Differentiate NumPy and Pandas with example.

NumPy and Pandas are two of the most widely used Python libraries for data analysis, but they serve different purposes. NumPy is mainly used for numerical computing, while Pandas is used for data manipulation and analysis. Here are some examples to differentiate NumPy and Pandas:

Creating an array of data:

NumPy allows you to create arrays of numerical data:

import numpy as np

a = np.array([1, 2, 3, 4])

print(a)

Output: **[1 2 3 4]**

Pandas allows you to create a DataFrame, which is a 2-dimensional table:

import pandas as pd

data = {'name': ['Alice', 'Bob', 'Charlie'], 'age': [25, 30, 35]}

df = pd.DataFrame(data)

print(df)

name age

0 Alice 25

1 Bob 30

2 Charlie 35

Performing mathematical operations:

NumPy provides a range of mathematical functions and operations, such as addition, subtraction, multiplication, and division:

a = np.array([1, 2, 3, 4])

b = np.array([5, 6, 7, 8])

print(a + b)

print(a - b)

print(a \* b)

print(a / b)

Pandas provides methods for performing mathematical operations on DataFrames, such as mean, sum, min, max:

data = {'name': ['Alice', 'Bob', 'Charlie'], 'age': [25, 30, 35]}

df = pd.DataFrame(data)

print(df.mean())

print(df.sum())

print(df.min())

print(df.max())

Write in detail about try, except, else and finally in exception handling.

The **try** statement is used to enclose a block of code that may raise an exception. The code inside the **try** block is executed normally, and if an exception is raised, the control is passed to the nearest **except** block.

The **except** statement is used to catch and handle exceptions raised by the **try** block. It specifies the type of exception that the block can handle. If the exception is of the specified type or its subclass, the code inside the **except** block is executed.

The **else** statement is used in conjunction with the **try** statement to specify a block of code that should be executed only if no exception is raised in the **try** block.

The **finally** statement is used to specify a block of code that should be executed irrespective of whether an exception is raised or not. This block is executed even if an exception is raised, and it is generally used to release resources such as files, network connections, etc.

filename = 'sample.txt'

try:

file = open(filename, 'r')

except FileNotFoundError:

print(f"Error: {filename} not found")

else:

contents = file.read()

print(contents)

file.close()

finally:

print("File operation complete")

Part C

Explain in detail about Set and operations on it using methods.

A set is a collection of distinct elements, where each element is unique and unordered. Sets are often used in mathematics and computer science to represent collections of items, such as numbers, letters, or other objects. In Python, sets are represented using curly braces {} or the set() function.

Creating a Set:

To create a set, you can use the set() function or curly braces {} with elements inside them. For example:

# using set() function

my\_set = set([1, 2, 3, 4, 5])

# using curly braces

my\_set = {1, 2, 3, 4, 5}

Basic operations on Sets:

Adding Elements to a Set: You can add elements to a set using the add() method. If the element already exists in the set, it will not be added again.

my\_set = {1, 2, 3}

my\_set.add(4)

print(my\_set) # Output: {1, 2, 3, 4}

Removing Elements from a Set: You can remove elements from a set using the remove() method. If the element does not exist in the set, it will raise a KeyError.

my\_set = {1, 2, 3, 4}

my\_set.remove(3)

print(my\_set) # Output: {1, 2, 4}

Union of Sets: The union of two sets is a new set that contains all the elements from both sets. In Python, you can use the union() method or the pipe (|) operator to combine sets.

set1 = {1, 2, 3}

set2 = {3, 4, 5}

set3 = set1.union(set2)

print(set3) # Output: {1, 2, 3, 4, 5}

Intersection of Sets: The intersection of two sets is a new set that contains only the elements that are in both sets. In Python, you can use the intersection() method or the ampersand (&) operator to find the common elements in sets.

set1 = {1, 2, 3}

set2 = {3, 4, 5}

set3 = set1.intersection(set2)

print(set3) # Output: {3}

Difference of Sets: The difference of two sets is a new set that contains only the elements that are in the first set but not in the second set. In Python, you can use the difference() method or the minus (-) operator to find the difference between sets.

set1 = {1, 2, 3}

set2 = {3, 4, 5}

set3 = set1.difference(set2)

print(set3) # Output: {1, 2}

Subset and Superset of Sets: You can check if a set is a subset or superset of another set using the issuperset() and issubset() methods.

set1 = {1, 2, 3}

set2 = {1, 2}

print(set2.issubset(set1)) # Output: True

print(set1.issuperset(set2)) # Output: True

Symmetric Difference of Sets: The symmetric difference of two sets is a new set that contains only the elements that are in either of the sets, but not in both. In Python, you can use the symmetric\_difference() method or the caret (^) operator to find the symmetric difference between sets.

set1 = {1, 2, 3, 4}

set2 = {3, 4, 5, 6}

sym\_diff = set1 ^ set2

print(sym\_diff)

{1, 2, 5, 6}

set1 = {1, 2, 3, 4}

set2 = {3, 4, 5, 6}

sym\_diff = set1.symmetric\_difference(set2)

print(sym\_diff)

Explain with example about various ways of constructing Pandas Data Frame object. Write about working in CSV files.

Pandas is a powerful data manipulation library that provides easy-to-use data structures for data analysis. The most commonly used data structure in Pandas is the DataFrame, which is a two-dimensional tabular data structure with rows and columns.

There are various ways to construct a Pandas DataFrame object:

From a dictionary: The easiest way to create a DataFrame is from a dictionary. In this method, the keys of the dictionary represent the column names, and the values represent the data in the columns.

Example:

import pandas as pd

data = {'Name': ['John', 'Emma', 'Michael', 'Sophia'],

'Age': [25, 30, 28, 32],

'Country': ['USA', 'UK', 'Australia', 'Canada']}

df = pd.DataFrame(data)

print(df)

Output:

Name Age Country

0 John 25 USA

1 Emma 30 UK

2 Michael 28 Australia

3 Sophia 32 Canada

From a list of lists: Another way to create a DataFrame is from a list of lists. In this method, each inner list represents a row, and the outer list represents the entire data frame.

Example:

import pandas as pd

data = [['John', 25, 'USA'], ['Emma', 30, 'UK'], ['Michael', 28, 'Australia'], ['Sophia', 32, 'Canada']]

df = pd.DataFrame(data, columns=['Name', 'Age', 'Country'])

print(df)

Output:

Name Age Country

0 John 25 USA

1 Emma 30 UK

2 Michael 28 Australia

3 Sophia 32 Canada

From a CSV file: Pandas also provides the ability to create a DataFrame from a CSV file. In this method, the CSV file is read using the read\_csv() function and converted into a DataFrame.

Example:

Suppose we have a CSV file named data.csv with the following contents:

Copy code

Name,Age,Country

John,25,USA

Emma,30,UK

Michael,28,Australia

Sophia,32,Canada

The following code reads the CSV file and creates a DataFrame:

import pandas as pddf = pd.read\_csv('data.csv')

print(df)

Output:

Name Age Country

0 John 25 USA

1 Emma 30 UK

2 Michael 28 Australia

3 Sophia 32 Canada

What is Dictionary? Explain Python dictionaries in detail discussing its operations and methods.

A dictionary is a collection of key-value pairs that allows you to store and retrieve data based on the key. In Python, dictionaries are implemented as hash tables, which means they are optimized for fast lookups and insertions.

Python dictionaries are created using curly braces {} or the built-in dict() function. The keys and values in a dictionary can be of any data type, and you can mix and match data types as needed. Here's an example of creating a dictionary:

my\_dict = {"apple": 2, "banana": 3, "orange": 1}

In this example, "apple", "banana", and "orange" are the keys, and 2, 3, and 1 are the values.

You can access the value associated with a key by using square brackets [] and passing the key inside the brackets. Here's an example:

print(my\_dict["apple"]) # Output: 2

You can also add new key-value pairs to a dictionary using square brackets []. Here's an example:

my\_dict["pear"] = 4

print(my\_dict) # Output: {"apple": 2, "banana": 3, "orange": 1, "pear": 4}

To remove a key-value pair from a dictionary, you can use the del keyword and pass the key to be deleted. Here's an example:

del my\_dict["orange"]

print(my\_dict) # Output: {"apple": 2, "banana": 3, "pear": 4}

Python dictionaries provide a number of useful methods to manipulate their contents. Here are some of the most commonly used methods:

dict.get(key, default=None): Returns the value associated with the key, or default if the key is not found.

dict.keys(): Returns a list of all the keys in the dictionary.

dict.values(): Returns a list of all the values in the dictionary.

dict.items(): Returns a list of tuples, where each tuple contains a key-value pair.

dict.update(other\_dict): Updates the dictionary with the key-value pairs from other\_dict.

dict.pop(key, default=None): Removes and returns the value associated with the key, or default if the key is not found.

dict.clear(): Removes all key-value pairs from the dictionary.

Here's an example of using some of these methods:

my\_dict = {"apple": 2, "banana": 3, "orange": 1}

print(my\_dict.get("apple")) # Output: 2

print(list(my\_dict.keys())) # Output: ["apple", "banana", "orange"]

print(list(my\_dict.values())) # Output: [2, 3, 1]

print(list(my\_dict.items())) # Output: [("apple", 2), ("banana", 3), ("orange", 1)]

my\_dict.update({"pear": 4})

print(my\_dict) # Output: {"apple": 2, "banana": 3, "orange": 1, "pear": 4}

print(my\_dict.pop("orange")) # Output: 1

my\_dict.clear()

print(my\_dict) # Output: {}

Explain about the five NumPy Array attributes.

NumPy is a Python library that provides a multidimensional array object, which is the fundamental data structure used for numerical computing in Python. NumPy arrays are homogeneous and can be of any dimension. Here are the five key attributes of NumPy arrays:

shape: The shape attribute returns a tuple that contains the dimensions of the array. For a one-dimensional array, the shape will be (n,), for a two-dimensional array, the shape will be (m, n), and for a three-dimensional array, the shape will be (l, m, n), and so on.

ndim: The ndim attribute returns the number of dimensions of the array. For example, a one-dimensional array has ndim=1, a two-dimensional array has ndim=2, and so on.

size: The size attribute returns the total number of elements in the array. For example, for a two-dimensional array with shape (m, n), the size will be m \* n.

dtype: The dtype attribute returns the data type of the elements in the array. NumPy arrays can hold elements of different data types such as int, float, and complex.

itemsize: The itemsize attribute returns the size (in bytes) of each element in the array. For example, an array of 32-bit integers will have itemsize=4. This attribute is useful for determining the memory requirements of an array.

import numpy as np

# create a two-dimensional array

a = np.array([[1, 2, 3], [4, 5, 6]])

# print the array

print("Array:")

print(a)

# print the shape of the array

print("Shape:")

print(a.shape)

# print the number of dimensions of the array

print("Number of dimensions:")

print(a.ndim)

# print the total number of elements in the array

print("Total number of elements:")

print(a.size)

# print the data type of the elements in the array

print("Data type:")

print(a.dtype)

# print the size (in bytes) of each element in the array

print("Size of each element (in bytes):")

print(a.itemsize)

Output:

Array:

[[1 2 3]

[4 5 6]]

Shape:

(2, 3)

Number of dimensions:

2

Total number of elements:

6

Data type:

int64

Size of each element (in bytes):

8