GLOBAL TREND PYTHON ASSESSMENT

Jayalakshmi .G (Internshala)

1. **Implement a Python class MaxHeap that supports the following operations: insert, delete, and get\_max. Ensure the operations maintain the properties of a max-heap.**

**Code:**

class MaxHeap:

def \_\_init\_\_(self):

self.heap = []

def insert(self, val):

self.heap.append(val)

self.\_heapify\_up(len(self.heap) - 1)

def delete(self, val):

try:

index = self.heap.index(val)

self.heap[index] = self.heap[-1]

self.heap.pop()

self.\_heapify\_down(index)

except ValueError:

print(f"Value {val} not found in the heap")

def get\_max(self):

if self.heap:

return self.heap[0]

return None

def \_heapify\_up(self, index):

parent\_index = (index - 1) // 2

if parent\_index >= 0 and self.heap[index] > self.heap[parent\_index]:

self.heap[index], self.heap[parent\_index] = self.heap[parent\_index], self.heap[index]

self.\_heapify\_up(parent\_index)

def \_heapify\_down(self, index):

left\_child\_index = 2 \* index + 1

right\_child\_index = 2 \* index + 2

largest = index

if left\_child\_index < len(self.heap) and self.heap[left\_child\_index] > self.heap[largest]:

largest = left\_child\_index

if right\_child\_index < len(self.heap) and self.heap[right\_child\_index] > self.heap[largest]:

largest = right\_child\_index

if largest != index:

self.heap[index], self.heap[largest] = self.heap[largest], self.heap[index]

self.\_heapify\_down(largest)

# Example usage

heap = MaxHeap()

heap.insert(10)

heap.insert(20)

heap.insert(5)

print(heap.get\_max()) # Output: 20

heap.delete(20)

print(heap.get\_max()) # Output: 10

**Output:**

20

10

1. **Write a Python function that takes a list of URLs, attempts to download their content, and retries up to 3 times if an error occurs. Use appropriate error handling to manage different types of exceptions.**

**Code:**

import requests

from time import sleep

def download\_urls(urls):

results = {}

for url in urls:

attempts = 0

success = False

while attempts < 3 and not success:

try:

response = requests.get(url)

response.raise\_for\_status()

results[url] = response.text

success = True

except requests.exceptions.RequestException as e:

print(f"Error downloading {url}: {e}")

attempts += 1

sleep(1)

return results

# Example usage

urls = ['http://example.com', 'http://invalid-url']

print(download\_urls(urls))

1. **Write a Python script that trains a simple linear regression model using scikit-learn. Use a dataset of your choice, split it into training and testing sets, and evaluate the model's performance.**

**Code:**

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

# Example dataset

X = np.random.rand(100, 1)

y = 3 \* X.squeeze() + 2 + np.random.randn(100)

# Split the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train the model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Evaluate the model

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

print(f"Mean Squared Error: {mse}")

**Output:**

Mean Squared Error: 1.3388083876862573

.

4.. **Using pandas, write a Python function to clean and preprocess a given DataFrame, which involves handling missing values, normalizing numerical columns, and encoding categorical columns.**

**Code:**

import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.impute import SimpleImputer

def preprocess\_data(df):

# Define preprocessing for numeric columns (impute missing values and scale)

numeric\_features = df.select\_dtypes(include=['int64', 'float64']).columns

numeric\_transformer = Pipeline(steps=[

('imputer', SimpleImputer(strategy='median')),

('scaler', StandardScaler())])

# Define preprocessing for categorical features (impute missing values and one-hot encode)

categorical\_features = df.select\_dtypes(include=['object']).columns

categorical\_transformer = Pipeline(steps=[

('imputer', SimpleImputer(strategy='constant', fill\_value='missing')),

('onehot', OneHotEncoder(handle\_unknown='ignore'))])

# Combine preprocessing steps

preprocessor = ColumnTransformer(

transformers=[

('num', numeric\_transformer, numeric\_features),

('cat', categorical\_transformer, categorical\_features)])

# Apply preprocessing

df\_processed = preprocessor.fit\_transform(df)

# Convert the result back to a DataFrame with appropriate column names

df\_processed = pd.DataFrame(df\_processed, columns=numeric\_features.tolist() +

preprocessor.named\_transformers\_['cat']['onehot'].get\_feature\_names\_out(categorical\_features).tolist())

return df\_processed

# Example usage

data = {

'age': [25, np.nan, 35, 45],

'salary': [50000, 60000, np.nan, 80000],

'city': ['New York', 'San Francisco', 'Los Angeles', np.nan]

}

df = pd.DataFrame(data)

print("Original DataFrame:")

print(df)

df\_processed = preprocess\_data(df)

print("\nPreprocessed DataFrame:")

print(df\_processed)

**Output:**

Original DataFrame:

age salary city

0 25.0 50000.0 New York

1 NaN 60000.0 San Francisco

2 35.0 NaN Los Angeles

3 45.0 80000.0 NaN

Preprocessed DataFrame:

age salary ... city\_San Francisco city\_missing

0 -1.414214 -1.147079 ... 0.0 0.0

1 0.000000 -0.229416 ... 1.0 0.0

2 0.000000 -0.229416 ... 0.0 0.0

3 1.414214 1.605910 ... 0.0 1.0

[4 rows x 6 columns]

**5.Write a Python function to compute the nth Fibonacci number using recursion.**

**Code:**

def fibonacci(n):

if n <= 0:

return 0

elif n == 1:

return 1

else:

return fibonacci(n - 1) + fibonacci(n - 2)

# Example usage

print(fibonacci(10))

**Output:**

55

**6. Write a Python function that divides two numbers and handles the case where the divisor is zero by returning a custom error message.**

**Code:**

def safe\_divide(a, b):

try:

result = a / b

except ZeroDivisionError:

return "Error: Division by zero is not allowed."

return result

# Example usage

print(safe\_divide(10, 2)) # Output: 5.0

print(safe\_divide(10, 0)) # Output: Error: Division by zero is not allowed.

**Output:**

5.0

Error: Division by zero is not allowed.

**7.** **Write a Python decorator that measures the execution time of a function and logs it. Apply this decorator to a function that performs a computationally expensive task.**

**Code:**

import time

import logging

# Set up logging

logging.basicConfig(level=logging.INFO)

def time\_logger(func):

def wrapper(\*args, \*\*kwargs):

start\_time = time.time()

result = func(\*args, \*\*kwargs)

end\_time = time.time()

execution\_time = end\_time - start\_time

logging.info(f"Executed {func.\_\_name\_\_} in {execution\_time:.4f} seconds")

return result

return wrapper

@time\_logger

def expensive\_function():

# Simulate a computationally expensive task

time.sleep(2)

# Example usage

expensive\_function()

**Output:**

INFO:root:Executed expensive\_function in 2.0006 seconds

**8. Write a Python function that takes two numbers and an operator (as a string) and performs the corresponding arithmetic operation (addition, subtraction, multiplication, or division).**

**Code:**

def calculate(a, b, operator):

if operator == '+':

return a + b

elif operator == '-':

return a - b

elif operator == '\*':

return a \* b

elif operator == '/':

try:

return a / b

except ZeroDivisionError:

return "Error: Division by zero is not allowed."

else:

return "Error: Invalid operator."

# Example usage

print(calculate(10, 5, '+')) # Output: 15

print(calculate(10, 5, '-')) # Output: 5

print(calculate(10, 5, '\*')) # Output: 50

print(calculate(10, 5, '/')) # Output: 2.0

print(calculate(10, 0, '/')) # Output: Error: Division by zero is not allowed.

**Output:**

15

5

50

2.0

Error: Division by zero is not allowed.

**9. Write a Python function that generates a random password. The password should contain a mix of uppercase letters, lowercase letters, digits, and special characters.**

**Code:**

import random

import string

def generate\_password(length=12):

characters = string.ascii\_letters + string.digits + string.punctuation

password = ''.join(random.choice(characters) for i in range(length))

return password

# Example usage

print(generate\_password()) # Example output: 'aB3!dE7@i#'

**Output:**

t?|C6>\_#9)KA

**10. Write a Python function that takes a 2D list (matrix) and returns its transpose.**

**Code:**

def transpose(matrix):

return list(map(list, zip(\*matrix)))

# Example usage

matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

print(transpose(matrix))

**Output:**

[[1, 4, 7], [2, 5, 8], [3, 6, 9]]