Python Exercises-

<https://www.w3resource.com/python-exercises/unittest/index.php>

<https://www.w3resource.com/python-exercises/python-exception-handling-exercises.php>

<https://www.w3resource.com/python-exercises/oop/index.php>

'''

1. Write a Python unit test program to check if a given number is prime or not.

'''

import unittest

class prime:

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

self.num=num

def is\_prime(self):

res1=False

if self.num<2: return res1

elif self.num==2: res1=True

else:

for i in range(2,self.num):

if self.num%i==0: return False

res1=True

return res1

def dummy(self):

print('dummy func')

class TestPrime(unittest.TestCase):

def test\_prime(self):

ans1=prime(37)

ans2=prime(91)

self.assertEqual(ans1.is\_prime(),True)

self.assertEqual(ans2.is\_prime(),True) #FAILED

if \_\_name\_\_=='\_\_main\_\_':

unittest.main()

#Textbook

def is\_prime(number):

if number < 2:

return False

for i in range(2, int(number\*\*0.5) + 1):

if number % i == 0:

return False

return True

class PrimeNumberTestCase(unittest.TestCase):

def test\_prime\_numbers(self):

prime\_numbers = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]

#prime\_numbers = [2, 3, 4, 8, 11, 13, 17, 19, 23, 30, 31]

print("Prime numbers:",prime\_numbers)

for number in prime\_numbers:

self.assertTrue(is\_prime(number), f"{number} is not recognized as a prime number")

def test\_non\_prime\_numbers(self):

non\_prime\_numbers = [4, 6, 8, 10, 12, 14, 16, 18, 20]

#non\_prime\_numbers = [4, 6, 8, 9, 11, 12, 14, 17, 16, 18, 20]

print("Non prime numbers:",non\_prime\_numbers)

for number in non\_prime\_numbers:

self.assertFalse(is\_prime(number), f"{number} is incorrectly recognized as a prime number")

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

'''

2. Write a Python unit test program to check if a list is sorted

in ascending order.

'''

import unittest

def is\_ascending(lsts):

result=True

n=lsts[0]

for i in range(1,len(lsts)):

if lsts[i]<n:

result=False

break

else: n=lsts[i]

return result

print(is\_ascending([1,2,3,4,7]))

print(is\_ascending([3,4,6,2,9,10]))

class test\_mine(unittest.TestCase):

lst1=[1,2,3,4,7]

lst2=[3,4,6,2,9,10]

lst3=[0.1,0.6,1.3,2.5,18]

lst0=[lst1,lst2,lst3]

print(lst0)

def is\_asc(self):

for j in self.lst0:

self.assertTrue(is\_ascending(j),f"{j} is not ascending")

# syntax of self.assertTrue(condition, message=None)

# mwssage is the msg shown when it Fails

if \_\_name\_\_=='\_\_main\_\_':

unittest.main()

#似乎沒在跑? (說OK)

#Textbook

import unittest

def is\_sorted\_ascending(lst):

return all(lst[i] <= lst[i+1] for i in range(len(lst)-1))

class TestSortedAscending(unittest.TestCase):

def test\_sorted\_list(self):

#lst = [5, 7, 2, 8, 1, 9]

lst = [1, 2, 3, 4, 5, 6, 7]

print("Sorted list: ",lst)

self.assertTrue(is\_sorted\_ascending(lst), "The list is not sorted in ascending order")

def test\_unsorted\_list(self):

#lst = [1, 2, 3]

lst = [3,4,5,6,7,8]

print("Unsorted list: ",lst)

self.assertFalse(is\_sorted\_ascending(lst), "The list is sorted in ascending order")

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

#pynative 看不出來, 要在VS Code跑!

'''

3. Write a Python unit test program that checks if two lists are equal.

'''

import unittest

def tlae(lst1, lst2):

return lst1 == lst2

print(tlae([1, 2, 3], [1.0, 2.0, 3.0])) #True

class TestTwoListsAreEqual(unittest.TestCase):

def test\_lists\_are\_equal(self):

self.assertTrue(tlae([1, 2, 3], [1, 2, 3]), "Two lists are not equal")

self.assertTrue(tlae([1, 2, 3], [3, 2, 1]), "Two lists are not equal")

self.assertTrue(tlae([1, 2, 3], [1.0, 2.0, 3.0]), "Two lists are not equal")

self.assertTrue(tlae([9, 2, 3], [1.0, 2.0, 3.0]), "Two lists are not equal")

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

#Textbook - 前面只要有錯程式就停住了 後面也不會執行

import unittest

def lists\_are\_equal(nums1, nums2):

return nums1 == nums2

class TestListsEquality(unittest.TestCase):

def test\_equal\_lists(self):

nums1 = [10, 20, 30, 40]

nums2 = [10, 20, 30, 40]

#nums1 = [10, 20, 30, 40]

#nums2 = [10, 20, 30, 50]

print("\nEqual list test:\n",nums1,"\n",nums2)

self.assertTrue(lists\_are\_equal(nums1, nums2), "The lists are not equal")

def test\_unequal\_lists(self):

nums1 = [10, 20, 30, 40]

nums2 = [30, 20, 10, 40]

#nums1 = [10, 20, 30, 40]

#nums2 = [10, 20, 30, 40]

print("\nUnequal list test:\n",nums1,"\n",nums2)

self.assertFalse(lists\_are\_equal(nums1, nums2), "The lists are equal")

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

'''

4. Write a Python unit test program to check if a string is a palindrome

'''

import unittest

def is\_palin(strs):

rev\_str=''.join(strs[-1::-1])

return strs==rev\_str

# print(is\_palin('atota'))

# print(is\_palin('agoda'))

class TestPalin(unittest.TestCase):

def test\_palin(self):

self.assertTrue(is\_palin('asantaatnasa'),'it is not palindrome')

self.assertTrue(is\_palin('racecar'),'it is not palindrome')

self.assertTrue(is\_palin('夏興夏'), 'it is not palindrome')

self.assertTrue(is\_palin('nvidia'),'it is not palindrome')

#重要! class name, def name 前面一定要都是'test'...

if \_\_name\_\_=='\_\_main\_\_':

unittest.main()

#Textbook

import unittest

def is\_palindrome(string):

return string == string[::-1]

class TestPalindrome(unittest.TestCase):

def test\_palindrome\_string(self):

#palindrome = "madam"

palindrome = "hello"

print("Test palindrome:",palindrome)

self.assertTrue(is\_palindrome(palindrome), "The string is not a palindrome")

def test\_non\_palindrome\_string(self):

#non\_palindrome = "hello"

non\_palindrome = "madam"

print("Test non palindrome:",non\_palindrome)

self.assertFalse(is\_palindrome(non\_palindrome), "The string is a palindrome")

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

#5. Write a Python unit test program to check if a file exists in a specified directory.

# in VS Code

import os

import unittest

#warmup

path1="d://test1.py"

print(os.path.exists(path1)) #True

def file\_exists(paths):

return (os.path.exists(paths))

class TestFileExists(unittest.TestCase):

def test\_fe(self):

pth1="d://test1.csv"

pth2="d://wine.csv"

self.assertTrue(file\_exists(pth1),'it does not exist fool!')

self.assertTrue(file\_exists(pth2),'it does not exist fool!')

if \_\_name\_\_=='\_\_main\_\_':

unittest.main()

#Textbook

import os

import unittest

def file\_exists(directory, filename):

file\_path = os.path.join(directory, filename)

return os.path.exists(file\_path)

class TestFileExists(unittest.TestCase):

def test\_existing\_file(self):

directory = '/path/txt'

filename = 'test1.txt'

self.assertTrue(file\_exists(directory, filename), "The file does not exist in the specified directory")

def test\_nonexistent\_file(self):

directory = '/path/txt'

filename = 'test2.txt'

self.assertFalse(file\_exists(directory, filename), "The file exists in the specified directory")

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

'''

6. Write a Python unit test that checks if a function handles

floating-point calculations accurately.

'''

import unittest

#warmup

n1=1/1428571

print(n1)

s=0

for i in range(1428571):

s+=n1

print(s) #0.9999999998780682

def accu\_sum(num):

divided=1/num

su=0

for j in range(num):

su+=divided

return su==num

print(accu\_sum(5)) #False!

#chatGPT said: dividing 1 by 5 should not cause rounding errors. However, the issue in the accu\_sum function is not related to the accuracy of the division itself, but rather to the accumulation of rounding errors in the loop'

#Binary Representation: Floating-point numbers in Python are represented in binary format. Some decimal values, such as 0.1 or 0.2, cannot be exactly represented in binary floating-point format, leading to small approximation errors.

class TestHandleFloatAccurately(unittest.TestCase):

def test\_hfa(self):

n1=5

n2=70

n3=200

n4=5000

self.assertTrue(accu\_sum(n1),'handle not accurately!')

self.assertTrue(accu\_sum(n2),'handle not accurately!')

self.assertTrue(accu\_sum(n3),'handle not accurately!')

self.assertTrue(accu\_sum(n4),'handle not accurately!')

if \_\_name\_\_=='\_\_main\_\_':

unittest.main()

#n1=5 就fail!

#Textbook

import unittest

class TestFloatingPointCalculations(unittest.TestCase):

def test\_addition(self):

result = 0.3 + 0.5

self.assertAlmostEqual(result, 0.8, places=6)

def test\_multiplication(self):

result = 0.3 \* 0.5

self.assertAlmostEqual(result, .15, places=6)

def test\_division(self):

result = 0.7 / 0.3

self.assertAlmostEqual(result, 2.333333, places=6)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

Note: unittest.TestCase裡面有很多’assert....’的submodule

在class TestFloatingPointCalculations(unittest.TestCase) 實際上的意思就是inherit了unittest.TestCase的submodule

'''

7. Write a Python unit test program to check if a function

handles multi-threading correctly

'''

#Textbook

import unittest

import threading

def perform\_task():

# Simulate some task

result = 0

for i in range(1, 100000):

result += i

return result

class Test\_Multi\_Threading(unittest.TestCase):

def test\_multi\_threading(self):

num\_threads = 10

threads = []

# Create and start multiple threads

for \_ in range(num\_threads):

t = threading.Thread(target=perform\_task)

threads.append(t)

t.start()

# Wait for all threads to finish

for t in threads:

t.join()

# Assert that all threads completed successfully

for t in threads:

self.assertFalse(t.is\_alive())

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

'''

8. Write a Python unit test program to check if a database

connection is successful.

'''

#Textbook

import unittest

import sqlite3

class TestDatabaseConnection(unittest.TestCase):

def test\_database\_connection(self):

# Create a database connection

conn = sqlite3.connect(':memory:')

#通常情况下，SQLite数据库会将数据存储在磁盘文件中。但是，在某些情况下，我们可以让SQLite数据库始终驻留在内存中，这就是所谓的内存数据库（In-Memory Database）。内存数据库将所有数据存储在RAM中，而不是磁盘上，这种方式可以提高查询速度，并且使得一些临时性的数据操作更加高效。

cursor = conn.cursor()

# Execute a simple query

cursor.execute("SELECT 1")

# Fetch the result

result = cursor.fetchone()

# Close the database connection

cursor.close()

conn.close()

# Assert that the result is as expected

self.assertEqual(result, (1,))

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

#在给定的代码中，sqlite3.connect(':memory:')被用来创建一个内存数据库连接。然后，我们创建了一个名为cursor的游标对象，并使用cursor.execute("SELECT 1")执行了一个简单的查询。最后，我们使用cursor.fetchone()获取了查询结果，并将其与期望结果(1,)进行比较。由于内存数据库是在RAM中创建的，因此在运行测试时不需要实际的数据库文件。因此，这个测试可以成功地运行，而不会失败。

#可以使用内置的sqlite3模块来创建和操作SQLite数据库。要创建一个内存数据库，只需要在连接字符串中指定特殊的数据库名称，例如':memory:'

#example:

import sqlite3

# Connect to an in-memory database

conn = sqlite3.connect(':memory:')

# Create a table

conn.execute('CREATE TABLE users (id INTEGER PRIMARY KEY, name TEXT)')

# Insert some data

conn.execute('INSERT INTO users (name) VALUES (?)', ('Alice',))

conn.execute('INSERT INTO users (name) VALUES (?)', ('Bob',))

# Query the data

cursor = conn.execute('SELECT \* FROM users')

for row in cursor:

print(row)

# Close the connection

conn.close()

'''

9. Write a Python unit test program to check if a database query returns the expected results.

'''

import sqlite3

import unittest

class TestDatabaseQuery(unittest.TestCase):

def test\_dbq(self):

conn=sqlite3.connect(':memory:')

cursor=conn.cursor()

cursor.execute('SELECT 1')

result=cursor.fetchall()

cursor.close()

conn.close()

self.assertEqual(result,[(1,)])

#if fetchone: the result will be (1,); if fetchall: the result will be [(1,)]

if \_\_name\_\_=='\_\_main\_\_':

unittest.main()

#Textbook

import unittest

import sqlite3

class TestDatabaseQuery(unittest.TestCase):

def setUp(self):

# Create a database connection and insert test data

self.conn = sqlite3.connect(':memory:')

self.cursor = self.conn.cursor()

self.cursor.execute("CREATE TABLE employees (id INTEGER PRIMARY KEY, name TEXT, salary REAL)")

self.cursor.execute("INSERT INTO employees (name, salary) VALUES ('Ylva Guiomar', 1800.0)")

self.cursor.execute("INSERT INTO employees (name, salary) VALUES ('Scott Gregorius', 2100.0)")

self.conn.commit()

def tearDown(self):

# Close the database connection

self.cursor.close()

self.conn.close()

def test\_database\_query(self):

# Execute the query

self.cursor.execute("SELECT name, salary FROM employees ORDER BY name")

results = self.cursor.fetchall()

# Define the expected results

expected\_results = [('Scott Gregorius', 2100.0), ('Ylva Guiomar', 1800.0)]

# Assert that the results match the expected results

self.assertEqual(results, expected\_results)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

'''

10. Write a Python unit test program to check if a function

correctly parses and validates input data.

'''

import unittest

def pavid(data):

print('the parse result is:')

if str(data).isnumeric():

return float(data)

else:

return data

class TestParseValidate(unittest.TestCase):

def test\_pav(self):

self.assertTrue(pavid(2.673)==2.673, 'not correctly handled!')

self.assertTrue(pavid('what')=='what', 'not correctly handled')

self.assertTrue(pavid(38)==38, 'not correctly handled')

if \_\_name\_\_=='\_\_main\_\_':

unittest.main()

#Textbook

import unittest

def parse\_and\_validate\_input(data):

# Function to parse and validate input data

if isinstance(data, str) and data.isnumeric():

return int(data) > 0

return False

class TestInputParsing(unittest.TestCase):

def test\_valid\_input(self):

data = "100"

result = parse\_and\_validate\_input(data)

self.assertTrue(result)

def test\_invalid\_input(self):

data = "Hello"

result = parse\_and\_validate\_input(data)

self.assertFalse(result)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

Exception-handling

'''

1. Write a Python program to handle a ZeroDivisionError exception

when dividing a number by zero.

exception ZeroDivisionError:

Raised when the second argument of a division or modulo operation

is zero. The associated value is a string indicating the type

of the operands and the operation.

'''

def calc(a,b,method):

if method=='add':

print(a+b)

elif method=='subtract':

print(a-b)

elif method=='multiply':

print(a\*b)

elif method=='divide':

try:

print(a/b)

except: ZeroDivisionError

print('Divide by Zero')

else: print('method not correct, quit')

calc(9,4,'add')

calc(3,0,'divide')

calc(4,8,'multiply')

#Textbook

def divide\_numbers(x, y):

try:

result = x / y

print("Result:", result)

except ZeroDivisionError:

print("The division by zero operation is not allowed.")

# Usage

numerator = 100

denominator = 0

divide\_numbers(numerator, denominator)

'''

2. Write a Python program that prompts the user to input

an integer and raises a ValueError exception if the input

is not a valid integer.

exception ValueError:

Raised when an operation or function receives an argument

that has the right type but an inappropriate value,

and the situation is not described by a more precise exception

such as IndexError.

'''

print('Input an integer:')

try:

user\_input=int(input())

print(f"you input an integer {user\_input}")

except ValueError:

print('what you input is not an integer!')

#Textbook

def get\_integer\_input(prompt):

try:

value = int(input(prompt))

return value

except ValueError:

print("Error: Invalid input, input a valid integer.")

# Usage

n = get\_integer\_input("Input an integer: ")

print("Input value:", n)

'''

3. Write a Python program that opens a file and handles

a FileNotFoundError exception if the file does not exist.

'''

import subprocess

print(subprocess.run('ls')) #run, script.py

lst1=['run','script.py','notexist.py']

for i in lst1:

try:

with open(i,'r') as f:

print(f"File exists and its name is: {f.name}")

except FileNotFoundError:

print('file not exist!')

#Textbook

def open\_file(filename):

try:

file = open(filename, 'r')

contents = file.read()

print("File contents:")

print(contents)

file.close()

except FileNotFoundError:

print("Error: File not found.")

# Usage

file\_name = input("Input a file name: ")

open\_file(file\_name)

'''

4. Write a Python program that prompts the user to input

two numbers and raises a TypeError exception if the inputs

are not numerical.

exception TypeError:

Raised when an operation or function is applied to an object

of inappropriate type. The associated value is a string

giving details about the type mismatch.

'''

try:

a,b=input('input two numbers and seperated by ",":\n').split(',')

a=float(a)

b=float(b)

print(f"you input {a} and {b}")

except Exception as e:

print('not all two numbers are numerical!')

#Textbook

def get\_numeric\_input(prompt):

while True:

try:

value = float(input(prompt))

return value

except ValueError:

print("Error: Invalid input. Please Input a valid number.")

# Usage

n1 = get\_numeric\_input("Input the first number: ")

n2 = get\_numeric\_input("Input the second number: ")

result = n1 \* n2

print("Product of the said two numbers:", result)

'''

5. Write a Python program that opens a file and handles a

PermissionError exception if there is a permission issue.

exception PermissionError:

Raised when trying to run an operation without the adequate access rights -

for example filesystem permissions. Corresponds to errno EACCES, EPERM, and ENOTCAPABLE.

'''

try:

with open('/usr/bin/alias','r') as f:

print(f.read())

except Exception as e:

print(e) # [Errno 2] No such file or directory: '/usr/bin/alias'

#Textbook

def open\_file(filename):

try:

with open(filename, 'w') as file:

contents = file.read()

print("File contents:")

print(contents)

except PermissionError:

print("Error: Permission denied to open the file.")

# Usage

file\_name = input("Input a file name: ")

open\_file(file\_name)

# immediately after opening the file, it is attempted to read its contents using the read method. This is not valid because the file was opened in write mode, which does not allow reading.

'''

6. Write a Python program that executes an operation on

a list and handles an IndexError exception if the index

is out of range.

exception IndexError:

Raised when a sequence subscript is out of range. (Slice indices are silently truncated to fall in the allowed range; if an index is not an integer, TypeError is raised.)

'''

def find\_middle(lsts):

mid=len(lsts)/2

return lsts[mid]

ex\_list=[1,2,3,4,5]

try:

print(find\_middle(ex\_list))

except Exception as e:

print(e) # list indices must be integers or slices, not float

#Textbook

def test\_index(data, index):

try:

result = data[index]

# Perform desired operation using the result

print("Result:", result)

except IndexError:

print("Error: Index out of range.")

nums = [1, 2, 3, 4, 5, 6, 7]

index = int(input("Input the index: \n"))

test\_index(nums, index)

'''

7. Write a Python program that prompts the user to input

a number and handles a KeyboardInterrupt exception

if the user cancels the input.

exception KeyboardInterrupt:

Raised when the user hits the interrupt key (normally Control-C or Delete).

During execution, a check for interrupts is made regularly.

The exception inherits from BaseException so as to

not be accidentally caught by code that catches Exception

and thus prevent the interpreter from exiting.

'''

# for i in range(100):

# try:

# print('input anything:\n')

# userinput=input()

# print(f"you input {userinput}")

# except KeyboardInterrupt:

# print('user interupt it!')

# in pynative,會沒完沒了,因為要一次把所有值input進去

#Textbook

try:

n = int(input("Input a number: "))

print("You entered:", n)

except KeyboardInterrupt:

print("Input canceled by the user.")

'''

8. Write a Python program that executes division and handles

an ArithmeticError exception if there is an arithmetic error.

exception ArithmeticError:

The base class for those built-in exceptions that are raised for

various arithmetic errors: OverflowError, ZeroDivisionError,

FloatingPointError.

'''

def calc(a,b,operand):

str1='a'+operand+'b'

return eval(str1)

print(calc(2,3,'+'))

a,b,operand=input('input two number and operand type:\n').split(',')

try:

cal\_str=a+operand+b

print(eval(cal\_str))

except ArithmeticError as e:

print(e)

#Textbook

def division(dividend, divisor):

try:

result = dividend / divisor

print("Result:", result)

except ArithmeticError:

print("Error: Arithmetic error occurred!")

#Usage

dividend = float(input("Input the dividend: "))

divisor = float(input("Input the divisor: "))

division(dividend, divisor)

'''

9. Write a Python program that opens a file and handles a

UnicodeDecodeError exception if there is an encoding issue.

exception UnicodeDecodeError:

Raised when a Unicode-related error occurs during decoding.

It is a subclass of UnicodeError.

Content of unicode.txt:

(有各國的文字)

'''

#in VS Code:

paths=["d://robots.txt","d://joke\_raw.txt"]

try:

for i in paths:

with open(i,'r') as f:

#with open(i,'r',encoding='UTF-8') as f: --> this will be ok!

print(f.read())

except UnicodeDecodeError as e:

print(e)

#Textbook

def open\_file(filename):

encoding = input("Input the encoding (ASCII,UTF-16,UTF-8) for the file: ")

try:

with open(filename, 'r', encoding=encoding) as file:

contents = file.read()

print("File contents:")

print(contents)

except UnicodeDecodeError:

print("Error: Encoding issue occurred while reading the file.")

# Usage

file\_name = input("Input the file name: ")

open\_file(file\_name)

'''

10. Write a Python program that executes a list operation

and handles an AttributeError exception if the attribute

does not exist.

exception AttributeError:

Raised when an attribute reference (see Attribute references)

or assignment fails. (When an object does not support

attribute references or attribute assignments at all,

TypeError is raised.)

'''

#for example, use the module of 'str' onto the list

#warmup

str1='john'

print(str1.startswith('j')) #True

print(str1.startswith('q')) #False

print(str1.find('h')) #2

print(str1.replace('j','k')) #kohn

def lst\_op(lsts,ope,arg):

try:

if ope=='append':

lsts.append(arg)

return lsts

elif ope=='remove':

lsts.remove(arg)

return lsts

except Exception as e:

return e

print(lst\_op([1,2,3,4,5],'append',1))

print(lst\_op([1,2,3,4,5],'remove',3))

print(lst\_op([1,2,3,4,5],'find',4)) #None (why?)

#Textbook

def test\_list\_operation(nums):

try:

r = len(nums) # Trying to access the length attribute

print("Length of the list:", r)

except Exception as e:

print("Error: The list does not have a 'length' attribute.")

print(e)

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

nums1=13

test\_list\_operation(nums)

test\_list\_operation(nums1)

OOP (Object-Oriented-Programming)

'''

1. Write a Python program to create a class representing a Circle.

Include methods to calculate its area and perimeter.

'''

class Circle:

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

self.radius=radius

def area(radius):

print(f"the area with the radius {radius} is {3.14\*radius\*\*2}")

def perimeter(radius):

print(f"the perimeter with the radius {radius} is {2\*3.14\*radius}")

Circle.area(3)

Circle.perimeter(5)

#Textbook

import math

class Circle:

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

self.radius = radius

def calculate\_circle\_area(self):

return math.pi \* self.radius\*\*2

def calculate\_circle\_perimeter(self):

return 2 \* math.pi \* self.radius

# Example usage

radius = float(input("Input the radius of the circle: "))

circle = Circle(radius)

area = circle.calculate\_circle\_area()

perimeter = circle.calculate\_circle\_perimeter()

print("Area of the circle:", area)

print("Perimeter of the circle:", perimeter)

'''

2. Write a Python program to create a person class.

Include attributes like name, country and date of birth.

Implement a method to determine the person's age.

'''

from datetime import datetime as dt

#warmup

today = dt.now()

dob1 = '1985-07-24'

dob1\_tobj = dt.strptime(dob1, '%Y-%m-%d')

print(dob1\_tobj)

age = today - dob1\_tobj

age\_day=age.days #convert into date-object

print(f"you are {int(age\_day/365)} yrs old")

class Person:

def \_\_init\_\_(self,name,country,DOB):

self.name=name

self.country=country

self.DOB=DOB

def info(self):

DOB\_tobj=dt.strptime(self.DOB, '%Y-%m-%d')

age\_day=(today-DOB\_tobj).days

print(f"your name is {self.name} and come from {self.country} and you are {int(age\_day/365)} yrs old now")

p1=Person('Charles','USA','1997-08-16')

p2=Person('Emiko','Japan','1976-12-19')

p3=Person('Chopin','Poland','1810-3-1')

p1.info()

p2.info()

p3.info()

#Textbook

from datetime import date

class Person:

def \_\_init\_\_(self, name, country, date\_of\_birth):

self.name = name

self.country = country

self.date\_of\_birth = date\_of\_birth

def calculate\_age(self):

today = date.today()

age = today.year - self.date\_of\_birth.year

if today < date(today.year, self.date\_of\_birth.month, self.date\_of\_birth.day):

age -= 1

return age

# Example usage

person1 = Person("Ferdi Odilia", "France", date(1962, 7, 12))

person2 = Person("Shweta Maddox", "Canada", date(1982, 10, 20))

person3 = Person("Elizaveta Tilman", "USA", date(2000, 1, 1))

# Accessing attributes and calculating age

print("Person 1:")

print("Name:", person1.name)

print("Country:", person1.country)

print("Date of Birth:", person1.date\_of\_birth)

print("Age:", person1.calculate\_age())

print("\nPerson 2:")

print("Name:", person2.name)

print("Country:", person2.country)

print("Date of Birth:", person2.date\_of\_birth)

print("Age:", person2.calculate\_age())

print("\nPerson 3:")

print("Name:", person3.name)

print("Country:", person3.country)

print("Date of Birth:", person3.date\_of\_birth)

print("Age:", person3.calculate\_age())

'''

3. Write a Python program to create a calculator class.

Include methods for basic arithmetic operations.

'''

class calc:

# def \_\_init\_\_(self, a,b):

# self.a=a

# self.b=b

def add(a,b):

return a+b

def sub(a,b):

return a-b

def mul(a,b):

return a\*b

def div(a,b):

try:

return a/b

except Exception as e:

return e

print(calc.add(3,6))

print(calc.sub(8,4))

print(calc.mul(2,7))

print(calc.div(9,0))

#Textbook

class Calculator:

def add(self, x, y):

return x + y

def subtract(self, x, y):

return x - y

def multiply(self, x, y):

return x \* y

def divide(self, x, y):

if y != 0:

return x / y

else:

return ("Cannot divide by zero.")

# Example usage

calculator = Calculator()

# Addition

result = calculator.add(7, 5)

print("7 + 5 =", result)

# Subtraction

result = calculator.subtract(34, 21)

print("34 - 21 =", result)

# Multiplication

result = calculator.multiply(54, 2)

print("54 \* 2 =", result)

# Division

result = calculator.divide(144, 2)

print("144 / 2 =", result)

# Division by zero (raises an error)

result = calculator.divide(45, 0)

print("45 / 0 =", result)

'''

4. Write a Python program to create a class that represents

a shape. Include methods to calculate its area and perimeter.

Implement subclasses for different shapes like circle, triangle,

and square.

'''

class shape:

def circle(self,radius):

area=3.14\*radius\*\*2

perimeter=2\*3.14\*radius

print(f"the circle with radius' {radius} area is {area}, and its perimeter is {perimeter}")

def triangle(self,w,h):

area=w\*h\*0.5

print(f"the triangle with width {w} and height {h} area is {area}")

def square(self,side):

area=side\*\*2

perimeter=side\*4

print(f"the square with side {side} area is {area}, and perimeter is {perimeter}")

s=shape()

s.circle(3)

s.triangle(5,12)

s.square(4)

#Textbook

import math

class Shape:

def calculate\_area(self):

pass

def calculate\_perimeter(self):

pass

class Circle(Shape):

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

self.radius = radius

def calculate\_area(self):

return math.pi \* self.radius\*\*2

def calculate\_perimeter(self):

return 2 \* math.pi \* self.radius

class Rectangle(Shape):

def \_\_init\_\_(self, length, width):

self.length = length

self.width = width

def calculate\_area(self):

return self.length \* self.width

def calculate\_perimeter(self):

return 2 \* (self.length + self.width)

class Triangle(Shape):

def \_\_init\_\_(self, base, height, side1, side2, side3):

self.base = base

self.height = height

self.side1 = side1

self.side2 = side2

self.side3 = side3

def calculate\_area(self):

return 0.5 \* self.base \* self.height

def calculate\_perimeter(self):

return self.side1 + self.side2 + self.side3

# Example usage

r = 7

circle = Circle(r)

circle\_area = circle.calculate\_area()

circle\_perimeter = circle.calculate\_perimeter()

print("Radius of the circle:",r)

print("Circle Area:", circle\_area)

print("Circle Perimeter:", circle\_perimeter)

l = 5

w = 7

rectangle = Rectangle(l, w)

rectangle\_area = rectangle.calculate\_area()

rectangle\_perimeter = rectangle.calculate\_perimeter()

print("\nRectangle: Length =",l," Width =",w)

print("Rectangle Area:", rectangle\_area)

print("Rectangle Perimeter:", rectangle\_perimeter)

base = 5

height = 4

s1 = 4

s2 = 3

s3 = 5

print("\nTriangle: Base =",base," Height =",height," side1 =",s1," side2 =",s2," side3 =",s3)

triangle = Triangle(base,height,s1,s2,s3)

triangle\_area = triangle.calculate\_area()

triangle\_perimeter = triangle.calculate\_perimeter()

print("Triangle Area:", triangle\_area)

print("Triangle Perimeter:", triangle\_perimeter)

'''

5. Write a Python program to create a class representing a binary search tree.

Include methods for inserting and searching for elements in the binary tree.

'''

idx\_dict={}

class bst:

def \_\_init\_\_(self,data,ti):

self.data=data

self.ti=ti

#tree information, eg:'1110', 1st~3rd layer:yes, 4th layer:no

def insert(data,ti):

idx\_dict[data]=ti

def search(data):

try:

print(f"you search {data} and its located at {idx\_dict[data]}")

except KeyError:

print('data does not exist')

bst.insert('a','110')

bst.insert('b','11110')

bst.insert('c','0')

bst.insert('d','10')

bst.search('b')

bst.search('c')

bst.search('x')

#Textbook:different meaning of 'binary tree

'''

Binary search tree:

In computer science, a binary search tree (BST), also called an ordered or sorted

binary tree, is a rooted binary tree data structure with the key of each

internal node being greater than all the keys in the respective node's left subtree

and less than the ones in its right subtree. The time complexity of operations on the

binary search tree is directly proportional to the height of the tree.

'''

class Node:

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

self.value = value

self.left = None

self.right = None

def \_\_str\_\_(self):

return str(self.value)

class BinarySearchTree:

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

self.root = None

def insert(self, value):

if self.root is None:

self.root = Node(value)

else:

self.\_insert\_recursive(self.root, value)

def \_insert\_recursive(self, node, value):

if value < node.value:

if node.left is None:

node.left = Node(value)

else:

self.\_insert\_recursive(node.left, value)

elif value > node.value:

if node.right is None:

node.right = Node(value)

else:

self.\_insert\_recursive(node.right, value)

def search(self, value):

return self.\_search\_recursive(self.root, value)

def \_search\_recursive(self, node, value):

if node is None or node.value == value:

return node

if value < node.value:

return self.\_search\_recursive(node.left, value)

else:

return self.\_search\_recursive(node.right, value)

# Example usage

bst = BinarySearchTree()

bst.insert(5)

bst.insert(3)

bst.insert(7)

bst.insert(2)

bst.insert(4)

bst.insert(6)

bst.insert(8)

print("Searching for elements:")

print(bst.search(4)) # Found, returns the node (4)

print(bst.search(9)) # Not found, returns None

'''

6. Write a Python program to create a class representing a stack

data structure. Include methods for pushing and popping elements

'''

#A stack is an abstract data type that serves as a collection of elements, with two main operations: Push, which adds an element to the collection, and Pop, which removes the most recently added element that was not yet removed.

class sds:

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

self.db=db

def push(self,ele):

self.db.append(ele)

def pop(self):

self.db.pop()

def query(self):

print(self.db)

db1=sds([1,2,3,4])

db1.push(5)

db1.push(9)

db1.query() #[1,2,3,4,5,9]

db1.pop()

db1.query() #[1,2,3,4,5]

#Textbook

class Stack:

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

self.items = []

def push(self, item):

self.items.append(item)

def pop(self):

if not self.is\_empty():

return self.items.pop()

else:

return "Cannot pop from an empty stack."

def is\_empty(self):

return len(self.items) == 0

def size(self):

return len(self.items)

def peek(self):

if not self.is\_empty():

return self.items[-1]

else:

return "Empty stack."

# Example usage

stack = Stack()

stack.push(0)

stack.push(1)

stack.push(2)

stack.push(3)

stack.push(4)

print("Stack size:", stack.size())

print("Top element:", stack.peek())

popped\_item = stack.pop()

print("\nPopped item:", popped\_item)

print("\nStack size:", stack.size())

print("Top element:", stack.peek())

#----------------------------------------

stack1 = Stack()

print("\nStack size:", stack1.size())

popped\_item = stack1.pop()

print("\nPopped item:", popped\_item)

'''

7. Write a Python program to create a class representing a

linked list data structure. Include methods for displaying

linked list data, inserting and deleting nodes.

'''

#Textbook

class Node:

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

self.data = data

self.next = None

#the next attribute in the Node class could potentially cause confusion with the next() method used to retrieve the next item from an iterator. However, in the context of a linked list, the next attribute is simply a reference to the next node in the list and is not related to the next() method used for iteration.

class LinkedList:

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

self.head = None

def display(self):

current = self.head

while current:

print(current.data, end=" ")

current = current.next

print()

def insert(self, data):

new\_node = Node(data)

if not self.head:

self.head = new\_node

else:

current = self.head

while current.next:

current = current.next

current.next = new\_node

def delete(self, data):

if not self.head:

return

if self.head.data == data:

self.head = self.head.next

return

current = self.head

prev = None

while current and current.data != data:

prev = current

current = current.next

if current:

prev.next = current.next

# Example usage

linked\_list = LinkedList()

linked\_list.insert(1)

linked\_list.insert(2)

linked\_list.insert(3)

linked\_list.insert(4)

print("Initial Linked List:")

linked\_list.display()

linked\_list.insert(5)

print("After insert a new node (4):")

linked\_list.display()

linked\_list.delete(2)

print("After delete a existing node (2):")

linked\_list.display()

#warmup

loop1=[i for i in range(5)]

# print(loop1.next())

#Note: Lists, tuples, dictionaries, and sets are iterable objects in Python, but they are not iterator objects

iter1=iter([1,2,3,4,5])

print(next(iter1))

print(next(iter1))

'''We already have Iterable objects, why we still need 'Iterator'?

Iterators in Python are objects that allow you to iterate over collections of data, such as lists, tuples, dictionaries, and sets. The main purpose of iterators is to provide a way to access the elements of a collection one at a time, without having to load the entire collection into memory at once. This is particularly useful when working with large data sets or streams of data that cannot be loaded into memory all at once.

'''

#warmup- class內本來就可隨便定義變數名稱

class fuck:

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

self.you='good'

self.me='bad'

def check(self, feeling):

if feeling=='good':

print(self.you==feeling)

elif feeling=='bad':

print('i feel bad cause u fuck me off')

else: return None

f=fuck()

f.check('good')

f.check('bad')

'''

8. Write a Python program to create a class representing a

shopping cart. Include methods for adding and removing items,

and calculating the total price.

'''

class scart:

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

self.buy\_list=[]

self.total=0

def add\_item(self,item,price):

self.buy\_list.append(item)

self.total+=price

def remove\_item(self,item,price):

self.buy\_list.remove(item)

self.total-=price

def cal\_price(self):

print('you need to pay:',self.total)

kmart=scart()

kmart.add\_item('salad',12)

kmart.add\_item('notebook',5)

kmart.add\_item('oven',230)

kmart.cal\_price()

kmart.remove\_item('notebook',5)

kmart.cal\_price()

#Textbook

class ShoppingCart:

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

self.items = []

def add\_item(self, item\_name, qty):

item = (item\_name, qty)

self.items.append(item)

def remove\_item(self, item\_name):

for item in self.items:

if item[0] == item\_name:

self.items.remove(item)

break

def calculate\_total(self):

total = 0

for item in self.items:

total += item[1]

return total

# Example usage

cart = ShoppingCart()

cart.add\_item("Papaya", 100)

cart.add\_item("Guava", 200)

cart.add\_item("Orange", 150)

print("Current Items in Cart:")

for item in cart.items:

print(item[0], "-", item[1])

total\_qty = cart.calculate\_total()

print("Total Quantity:", total\_qty)

cart.remove\_item("Orange")

print("\nUpdated Items in Cart after removing Orange:")

for item in cart.items:

print(item[0], "-", item[1])

total\_qty = cart.calculate\_total()

print("Total Quantity:", total\_qty)

'''

9. Write a Python program to create a class representing a

stack data structure. Include methods for pushing, popping and

displaying elements

'stack data structure' follows 'LIFO' (last in first out)

'''

class stack:

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

self.ds=[]

def push(self,data):

self.ds.append(data)

def pop(self):

if self.ds==None: pass

else:

self.ds.pop()

def display(self):

print(self.ds)

poker=stack()

poker.display()

poker.push(5)

poker.push(12)

poker.push(13)

poker.push('joker')

poker.display()

poker.pop()

poker.display()

poker.pop()

poker.display()

#Textbook

class Stack:

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

self.items = []

def push(self, item):

self.items.append(item)

def pop(self):

if not self.is\_empty():

return self.items.pop()

else:

raise IndexError("Cannot pop from an empty stack.")

def is\_empty(self):

return len(self.items) == 0

def display(self):

print("Stack items:", self.items)

# Example usage

stack = Stack()

stack.push(10)

stack.push(20)

stack.push(30)

stack.push(40)

stack.push(50)

stack.display()

popped\_item = stack.pop()

print("Popped item:", popped\_item)

popped\_item = stack.pop()

print("Popped item:", popped\_item)

stack.display()

'''

10. Write a Python program to create a class representing a

queue data structure. Include methods for enqueueing and

dequeueing elements.

Queue data structure follows FIFO (first in first out)

'''

class qds:

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

self.ds=[]

def enqueue(self,data):

self.ds.append(data)

def dequeue(self):

if self.ds==None: return None

else:

self.ds.pop(0)

def display(self):

print(self.ds)

freeway=qds()

freeway.enqueue('APT2812')

freeway.enqueue('9A8216')

freeway.enqueue('44148A')

freeway.display()

freeway.dequeue()

freeway.display()

#Textbook

class Queue:

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

self.items = []

def enqueue(self, item):

self.items.append(item)

def dequeue(self):

if not self.is\_empty():

return self.items.pop(0)

else:

raise IndexError("Cannot dequeue from an empty queue.")

def is\_empty(self):

return len(self.items) == 0

# Example usage

queue = Queue()

queue.enqueue(10)

queue.enqueue(20)

queue.enqueue(30)

queue.enqueue(40)

queue.enqueue(50)

print("Current Queue:", queue.items)

dequeued\_item = queue.dequeue()

print("Dequeued item:", dequeued\_item)

dequeued\_item = queue.dequeue()

print("Dequeued item:", dequeued\_item)

print("Updated Queue:", queue.items)

'''

11. Write a Python program to create a class representing a bank.

Include methods for managing customer accounts and transactions

'''

class bank:

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

self.CA=CA #Customer Account

def open\_acct(self,customer):

self.CA[customer]=0

def transaction(self,customer,var):

if customer not in self.CA.keys():

print('Account does not exist!')

self.CA[customer]=self.CA[customer]+var

def display(self,who):

if who=='all': print(self.CA)

else:

if who in self.CA:

print(who, ':', self.CA[who])

else: print('Account does not exist!')

HCBC=bank({})

HCBC.open\_acct('John')

HCBC.transaction('John',100)

HCBC.display('all')

PSMC=bank({'Frank':500})

PSMC.open\_acct('Jeff')

PSMC.transaction('Jeff',200)

PSMC.open\_acct('Martin')

PSMC.transaction('Martin',350)

PSMC.transaction('Frank',-1000)

PSMC.display('all')

PSMC.display('Frank')

PSMC.display('Vincent')

#Textbook

class Bank:

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

self.customers = {}

def create\_account(self, account\_number, initial\_balance=0):

if account\_number in self.customers:

print("Account number already exists.")

else:

self.customers[account\_number] = initial\_balance

print("Account created successfully.")

def make\_deposit(self, account\_number, amount):

if account\_number in self.customers:

self.customers[account\_number] += amount

print("Deposit successful.")

else:

print("Account number does not exist.")

def make\_withdrawal(self, account\_number, amount):

if account\_number in self.customers:

if self.customers[account\_number] >= amount:

self.customers[account\_number] -= amount

print("Withdrawal successful.")

else:

print("Insufficient funds.")

else:

print("Account number does not exist.")

def check\_balance(self, account\_number):

if account\_number in self.customers:

balance = self.customers[account\_number]

print(f"Account balance: {balance}")

else:

print("Account number does not exist.")

# Example usage

bank = Bank()

acno1= "SB-123"

damt1 = 1000

print("New a/c No.: ",acno1,"Deposit Amount:",damt1)

bank.create\_account(acno1, damt1)

acno2= "SB-124"

damt2 = 1500

print("New a/c No.: ",acno2,"Deposit Amount:",damt2)

bank.create\_account(acno2, damt2)

wamt1 = 600

print("\nDeposit Rs.",wamt1,"to A/c No.",acno1)

bank.make\_deposit(acno1, wamt1)

wamt2 = 350

print("Withdraw Rs.",wamt2,"From A/c No.",acno2)

bank.make\_withdrawal(acno2, wamt2)

print("A/c. No.",acno1)

bank.check\_balance(acno1)

print("A/c. No.",acno2)

bank.check\_balance(acno2)

wamt3 = 1200

print("Withdraw Rs.",wamt3,"From A/c No.",acno2)

bank.make\_withdrawal(acno2, wamt3)

acno3 = "SB-134"

print("A/c. No.",acno3)

bank.check\_balance(acno3) # Non-existent account number