Codekata Report:

Name: ADITYA KUMAR JHA

Email: adityajha375911@gmail.com

Specialization: School of Computing Science & Engineering

Completion Year: 2027

Section: Section-59

1. Write a program to check whether a given number is prime or not. If prime display “True” else “False”.

Sample Input:

1

Sample Output:

False

Completion Status: Not Completed

Concepts Included:

galgotias - conditional statements

galgotias

Language Used: PYTHON 3

Source Code:

```
import math
num=int(input("enter the number"))
def is_prime(n):
if num < 2:
return False
i = 2
while i*i <= n:
if num % i == 0:
return False
i += 1
return True
```

ADITYA KUMAR JHA (adityajha375911@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

enter the number

Compilation Status: Failed

Execution Time:

0.01s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

enter the number

Compilation Status: Failed

Execution Time:

0.01s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

2. Write a program to find the largest of three numbers.

Sample Input:

5 8 7

Sample Output:

8

Completion Status: Not Completed

Concepts Included:

galgotias - conditional statements

galgotias

Language Used: PYTHON 3

Source Code:

```
num1=int(input("enter the first number"))
num2=int(input("enter the second number"))
num3=int(input("enter the third number"))
if num1>num2 and num1>num3:
print(num1,"is largest number")
if num2>num1 and num2>num3:
print(num2,"is largest number")
if num3>num1 and num3>num2:
print(num3,"is largest number")
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

enter the first number

Compilation Status: Failed

Execution Time:

0.01s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Output:

enter the first number

Compilation Status: Failed

Execution Time:

0.01s

3. Write a program to check whether a person is eligible for voting or not. If age is greater than or equal to 18 display “yes” else “no”

Sample Input:

18

Sample Output:

yes

Completion Status: Not Completed

Concepts Included:

galgotias - conditional statements

galgotias

Language Used: PYTHON 3

Source Code:

```
age=int(input("enter the age"))
if age>=18:
print("yes,eligible for voting")
else:
print("no,eligible for voting")
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Output:

enter the ageyes,eligible for voting

Compilation Status: Failed

Execution Time:

0.013s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

enter the ageno,eligible for voting

Compilation Status: Failed

Execution Time:

0.009s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

4. Write a program to print the reverse order in the given input list

Sample Input:

5
78 66 89 65 64

Sample Output:

64 65 89 66 78

Completion Status: Not Completed

Concepts Included:

galgotias - lists

galgotias

Language Used: PYTHON 3

Source Code:

```
def Reverse(lst):
new_lst = lst[::-1]
return new_lst


lst = [78,66,89,65,64]
print(Reverse(lst))
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

[64, 65, 89, 66, 78]

Compilation Status: Failed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

[64, 65, 89, 66, 78]

Compilation Status: Failed

Execution Time:

0.011s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

5. Write a program to iterate both lists simultaneously and display items from list1 in original order and items from list2 in reverse order.

Sample Input:

2 2
1 2
3 4

Sample Output:

1 4
2 3

Completion Status: Completed

Concepts Included:

galgotias - lists

galgotias

Language Used: PYTHON 3

Source Code:

```
def iterate_and_display(list1, list2):
for item1, item2 in zip(list1, reversed(list2)):
print(item1, item2)

# Input
size_list1, size_list2 = map(int, input().split())
list1 = list(map(int, input().split()))
list2 = list(map(int, input().split()))

# Output
iterate_and_display(list1, list2)
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 4
2 3

Compilation Status: Passed

Execution Time:

0.01s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

2 3

Compilation Status: Passed

Execution Time:

0.009s

6. Write a program to add two lists index-wise. Create a new list that contains the 0th index item from both the list, then the 1st index item, and so on till the last element. any leftover items will get added at the end of the new list.

Sample Input:

2 2
G V
U I

Sample Output:

GU VI

Completion Status: Not Completed

Concepts Included:

galgotias - lists

galgotias

Language Used: PYTHON 3

ADITYA KUMAR JHA (adityaha37511@gmail.com)

Source Code:

```
def add_lists_indexwise(list1, list2):
result_list = []
min_len = min(len(list1), len(list2))

for i in range(min_len):
result_list.append(list1[i] + list2[i])

# Add any leftover items
result_list.extend(list1[min_len:])
result_list.extend(list2[min_len:])

return ''.join(map(str, result_list))

# Input
size_list1, size_list2 = map(int, input().split())
list1 = input().split()
list2 = input().split()

# Output
result = add_lists_indexwise(list1, list2)
print(result)
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

GUVI

Compilation Status: Failed

Execution Time:

0.01s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Output:

QTWYEU

Compilation Status: Failed

Execution Time:

0.01s

7. Given a number N, print the Bitwise NOT of that number.Input Size : 1 <= N <= 10000Sample Testcase :INPUT5OUTPUT-6

Completion Status: Not Completed

Concepts Included:

bitwise

basics

Language Used: PYTHON 3

Source Code:

```
N=int(input("enter the number"))
result= -N
print("bitwise NOT OF",N,"is",result)
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

enter the numberbitwise NOT OF 3456 is -3456

Compilation Status: Failed

Execution Time:

0.009s

ADITYA KUMAR JHA (adityajha37591@gmail.com)

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

enter the numberbitwise NOT OF 10 is -10

Compilation Status: Failed

Execution Time:

0.009s

8. You are tasked with writing a program to calculate and print the area and circumference of a circle given its radius r units. Implement a class named 'Circle' without any parameter in its constructor.

Input:

The input consists of a single line containing an integer r representing the radius of the circle.

Output:

Output the area and circumference of the circle. Round the output to one decimal value.

Sample Input:

3

Sample Output:

28.318.8

Input Description:

r

Output Description:

Area
Circumference

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Note: Round the output to one decimal values

Completion Status: Completed

Concepts Included:

GU-INTRODUCTION TO OOP

Language Used: PYTHON 3

Source Code:

```
class circle:
def __init__(self):
pass
def area(self,r):
area=3.143*r*r
return area
def circumferance(self,r):
circumferance=2*3.14*r
return circumferance
obj=circle()
r=int(input())
a=obj.area(r)
c=obj.circumferance(r)
print("%0.1f" % a)
print("%0.1f" % c)
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

28.3
18.8

Compilation Status: Passed

Execution Time:

0.01s

TestCase2:

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Input:

< hidden >

Expected Output:

< hidden >

Output:

50.3
25.1

Compilation Status: Passed

Execution Time:

0.01s

9. You're required to write a program to calculate and print the area and perimeter of two rectangles given their sides. Implement a class named 'Rectangle' with methods named 'Area' and 'Perimeter', which return the area and perimeter, respectively. The length and breadth of the rectangles are passed as parameters to its constructor.

Input:

The input consists of two lines. Each line contains two integers representing the sides of the rectangles in the format (a, b) for the first rectangle and (c, d) for the second rectangle.

Output:

Output the area and perimeter of both rectangles separated by spaces in the formatArea1 Perimeter1Area2 Perimeter2.

Sample Input:

3 44 5

Sample Output:

12 1420 18

Input Description:

Get input sides of the two rectangles.
Rectangle 1 = (a , b)
Rectangle 1 = (c , d)

ADITYA KUMAR JHA (adityajha3559@gmail.com)

Output Description:

Area1 Perimeter1
Area2 Perimeter2

Completion Status: Completed

Concepts Included:

GU-INTRODUCTION TO OOP

Language Used: PYTHON 3

Source Code:

```
class rectangle:
def __init__(self,length,breadth):
self.l=length
self.b=breadth

def area(self):
area1=self.l*self.b
return area1

def per(self):
peri=2*(self.l+self.b)
return peri


a,b=map(int,input().split())
c,d=map(int,input().split())

rec1=rectangle(a,b)
rec2=rectangle(c,d)

print(rec1.area(),rec1.per())
print(rec2.area(),rec2.per())
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

12 14
20 18

Compilation Status: Passed

Execution Time:

0.01s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

20 18
25 20

Compilation Status: Passed

Execution Time:

0.01s

10. Write a code to get the input in the given format and print the output in the given format

Sample Input:

2 3 4 5 6 7 8

Sample Output:

2 3 4 5 6 7 8

Completion Status: Not Completed

Concepts Included:

Input/Output

Language Used: PYTHON 3

Source Code:

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

```
my_list=[2 3 4 5  6 7 8]
print(my_list)
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
File "script-3.8.1.py", line 1
my_list=[2 3 4 5  6 7 8]
^
SyntaxError: invalid syntax
```

Runtime Error (NZEC)

Compilation Status: Failed

Execution Time:

0.01s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
File "script-3.8.1.py", line 1
my_list=[2 3 4 5  6 7 8]
^
SyntaxError: invalid syntax
```

Runtime Error (NZEC)

Compilation Status: Failed

Execution Time:

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

0.009s

11. You are provided with a number, "N". Find its factorial.

Sample Input:

2

Sample Output:

2

Completion Status: Completed

Concepts Included:

absolute beginner

Language Used: PYTHON 3

Source Code:

```
N=int(input())
fact=1
for i in range(N):
fact*=(i+1)
print(fact)
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

24

Compilation Status: Passed

Execution Time:

0.009s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

6

Compilation Status: Passed

Execution Time:

0.009s

12. You are given A = Length of a rectangle & B = breadth of a rectangle. Find its area “C”.

(A and B are natural numbers)

Sample Input:

2
3

Sample Output:

6

Completion Status: Completed

Concepts Included:

absolute beginner

Language Used: PYTHON 3

Source Code:

```
A=int(input())
B=int(input())
area=A*B
print(area)
```

Compilation Details:

ADITYA KUMAR JHA (adityajha37591@gmail.com)

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

144

Compilation Status: Passed

Execution Time:

0.01s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

30

Compilation Status: Passed

Execution Time:

0.009s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

13. You are given with a number A i.e. the temperature in Celcius. Write a program to convert this into Fahrenheit.

Note: In case of decimal values, round-off to two decimal places.

Sample Input:

12

Sample Output:

53.60

Completion Status: Not Completed

Concepts Included:

absolute beginner

Language Used: PYTHON 3

Source Code:

```
celsius=int(input())
fahrenheit=(celsius+9/5)+32
print(fahrenheit)
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

33.8

Compilation Status: Failed

Execution Time:

0.009s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

54.8

Compilation Status: Failed

Execution Time:

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

0.009s

14. Write a code to get an integer N and print values from 1 till N i a separate line.

Sample Input:

5

Sample Output:

1
2
3
4
5

Completion Status: Not Completed

Concepts Included:

absolute beginner

basics

Looping

Language Used: PYTHON 3

Source Code:

```
N=int(input)
for i in range(1,N+1):
print(i)
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Traceback (most recent call last):

File "script-3.8.1.py", line 1, in <module>
N=int(input)
TypeError: int() argument must be a string, a bytes-like object or a number, not 'builtin_function_or_method'

Runtime Error (NZEC)

Compilation Status: Failed

Execution Time:

0.01s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Traceback (most recent call last):
File "script-3.8.1.py", line 1, in <module>
N=int(input)
TypeError: int() argument must be a string, a bytes-like object or a number, not 'builtin_function_or_method'

Runtime Error (NZEC)

Compilation Status: Failed

Execution Time:

0.01s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

15. Mani finds himself in a queue of(n) people but is uncertain about his exact position. However, he's confident that there are at least(a) people in front of him and at most(b) people behind him. Mani seeks to determine the number of different positions he could occupy in the queue.

Input:

The input consists of a single line containing three integers:(n),(a), and(b) ((0≤a,b<n≤100)). These represent the total number of people in the queue, the minimum number of people in front of Mani, and the maximum number of people behind Mani, respectively.

Output:

Print a single integer, the number of possible positions Mani could occupy in the queue.

Constraints:

The queue contains between 1 and 100 people.

Mani is certain there are at least 0 people in front of him and at most(n-1) people behind him.

Sample input:

5 2 3

Sample output:

3

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
n,a,b = map(int, input().split())

#..... YOUR CODE STARTS HERE .....

possible_positions = min(b + 1, n - a)

print(possible_positions)

#..... YOUR CODE ENDS HERE .....
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

< hidden >

Output:

3

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1

Compilation Status: Passed

Execution Time:

0.012s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

16. Mani finds himself in a queue of(n) people but is uncertain about his exact position. However, he's confident that there are at least(a) people in front of him and at most(b) people behind him. Mani seeks to determine the number of different positions he could occupy in the queue.

Input:

The input consists of a single line containing three integers:(n),(a), and(b) ((0≤a,b<n≤100)). These represent the total number of people in the queue, the minimum number of people in front of Mani, and the maximum number of people behind Mani, respectively.

Output:

Print a single integer, the number of possible positions Mani could occupy in the queue.

Constraints:

The queue contains between 1 and 100 people.

Mani is certain there are at least 0 people in front of him and at most(n-1) people behind him.

Sample input:

5 2 3

Sample output:

3

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
n,a,b = map(int, input().split())

#..... YOUR CODE STARTS HERE .....

possible_positions = min(b + 1, n - a)

print(possible_positions)

#..... YOUR CODE ENDS HERE .....
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

3

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1

Compilation Status: Passed

Execution Time:

0.012s

ADITYA KUMAR JHA (adityajha1.ajh@gmail.com)

17. Vishakan, a dedicated reader, has a limited time to indulge in his favorite activity: reading books in the library. Each book has its own unique charm, but Vishakan needs to optimize his time to read as many books as possible within his available time.

Input:

The first line contains two integers,(n) and (t) where, (1 <= n <= 10^5) and (1 <= t <= 10^9), denoting the number of books in the library and the total free minutes Vishakan has.

The second line contains(n) space-separated integers (a1, a2,...., an) where (1 <= ai <= 10^4), representing the time required to read each book.

Output:

Print a single integer, the maximum number of books Vishakan can read within his available time.

Constraints:

Vishakan has a maximum of(10^9) minutes to spend in the library.

Each book takes between 1 and 10,000 minutes to read.

Sample input:

3 32 2 3

Sample output:

1

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
import sys
from collections import deque, defaultdict

I = sys.stdin.readline

def ii():
return int(I().strip())

def li():
return list(map(int, I().strip().split()))

def mi():
return map(int, I().strip().split())

def main():
n, k = mi()
arr = li()

#..... YOUR CODE STARTS HERE .....


maxlen=0
s=sum(arr)
if s<=k:
print(n)
else:
curr=0
j=0
for i in range(n):
curr+=arr[i]
while curr>k:
curr-=arr[j]
j+=1
maxlen=max(maxlen,i-j+1)
print(maxlen)
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

```
#..... YOUR CODE ENDS HERE .....

if __name__ == '__main__':
main()
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1

Compilation Status: Passed

Execution Time:

0.014s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

0

Compilation Status: Passed

Execution Time:

0.014s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

18. On Earth, each plane has a special affinity for another plane. Given a group of n planes, numbered from 1 to n, where plane i likes plane fi, we aim to determine if any love triangle exists.

Input:

The first line contains a single integer n (2 ≤ n ≤ 5000) — representing the number o planes.

The second line contains n integers f1, f2, ..., fn (1 ≤ fi ≤ n, fi ≠ i), indicating that the i plane is fond of plane fi.

Output:

Output "YES" if there exists a love triangle among the planes. Otherwise, output "NO".

Sample input:

55 5 5 5 1

Sample output:

NO

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
class CycleChecker:
def __init__(self, f):
self.f = f
self.n = len(f)

def check_cycle(self):
#..... YOUR CODE STARTS HERE .....

for i in range(self.n):
a=i
b=self.f[i]-1
c=self.f[b]-1
d=self.f[c]-1
if d==a and a!=b and b!=c and c!=d:
return "YES"
return "NO"

#..... YOUR CODE ENDS HERE .....

if __name__ == "__main__":
n = int(input())
f = list(map(int, input().split()))

#..... YOUR CODE STARTS HERE .....
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

```
cycle_checker=CycleChecker(f)
result=cycle_checker.check_cycle()
print(result)

#..... YOUR CODE ENDS HERE .....
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

NO

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

YES

Compilation Status: Passed

Execution Time:

0.013s

ADITYA KUMAR JHA (adityajha37591@gmail.com)

19. Mani finds himself in a queue of(n) people but is uncertain about his exact position. However, he's confident that there are at least(a) people in front of him and at most(b) people behind him. Mani seeks to determine the number of different positions he could

occupy in the queue.

Input:

The input consists of a single line containing three integers:(n),(a), and(b) ((0≤a,b<n≤100)). These represent the total number of people in the queue, the minimum number of people in front of Mani, and the maximum number of people behind Mani, respectively.

Output:

Print a single integer, the number of possible positions Mani could occupy in the queue.

Constraints:

The queue contains between 1 and 100 people.

Mani is certain there are at least 0 people in front of him and at most(n-1) people behind him.

Sample input:

5 2 3

Sample output:

3

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
n,a,b = map(int, input().split())

#..... YOUR CODE STARTS HERE .....

possible_positions = min(b + 1, n - a)

print(possible_positions)

#..... YOUR CODE ENDS HERE .....
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

3

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1

Compilation Status: Passed

Execution Time:

0.012s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

20. Mani finds himself in a queue of(n) people but is uncertain about his exact position. However, he's confident that there are at least(a) people in front of him and at most(b) people behind him. Mani seeks to determine the number of different positions he could occupy in the queue.

Input:

The input consists of a single line containing three integers:(n),(a), and(b) ((0≤a,b<n≤100)). These represent the total number of people in the queue, the minimum number of people in front of Mani, and the maximum number of people

behind Mani, respectively.

Output:

Print a single integer, the number of possible positions Mani could occupy in the queue.

Constraints:

The queue contains between 1 and 100 people.

Mani is certain there are at least 0 people in front of him and at most(n-1) people behind him.

Sample input:

5 2 3

Sample output:

3

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
n,a,b = map(int, input().split())

#..... YOUR CODE STARTS HERE .....

possible_positions = min(b + 1, n - a)

print(possible_positions)

#..... YOUR CODE ENDS HERE .....
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

3

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1

Compilation Status: Passed

Execution Time:

0.012s

ADITYA KUMAR JHA (adityajha37591@gmail.com)

21. Lucas, a university student, is working on a project where he needs to process a set of student records. He is given a list of students where each record contains the student's name and marks in three subjects: Math, Physics, Biology. Each subject's mark is within the range 0-100. He needs to return the name of the student with the highest average score.

If there is more than one student who has the highest average score, Lucas needs to return the first student in the list.

Input:

The first line contains an integer 'n' (1 <= n <= 1000) - representing the number of students.

The 'n' following lines each contain a string and three space-separated integers.

Output:

Print the name of the student who has the highest average score.

Sample Input:

5

John 85 90 82

Alice 90 91 92

Bob 80 79 81

Lucas 88 90 92

Maria 90 91 90

Sample Output:

Alice

Note:

The five students' average scores are as follows:

John's average is 85.67

Alice's average is 91

Bob's average is 80

Lucas's average is 90

Maria's average is 90.33

Alice has the highest average score and showcases the best performance among all students.

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
class Student:
def __init__(self, name, scores):
self._name = name  # protected attribute
```

```
self.__scores = scores  # private attribute

def calculate_average(self):
return sum(self.__scores) / len(self.__scores)

def get_name(self):  # public method to access protected attribute
return self._name


class TopStudentFinder:
def __init__(self):
self.max_avg = -1.0
self.top_student = ""

def find_top_student(self, students):
#..... YOUR CODE STARTS HERE .....



for student in students:
avg = student.calculate_average()
if avg>self.max_avg:
self.max_avg=avg
self.top_student = student.get_name()


#..... YOUR CODE ENDS HERE .....
def get_top_student(self):
return self.top_student


if __name__ == "__main__":
n = int(input())
students_data = []

for _ in range(n):
student_info = input().split()
name = student_info[0]
scores = list(map(int, student_info[1:]))
students_data.append(Student(name, scores))

#..... YOUR CODE STARTS HERE .....

top_student_finder=TopStudentFinder()
top_student_finder.find_top_student(students_data)
print(top_student_finder.get_top_student())

#..... YOUR CODE ENDS HERE ..
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Alice

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

John

Compilation Status: Passed

Execution Time:

0.012s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

22. Many years ago, the country of Berland was inhabited by a small population of(n) individuals, each with their own savings. The government aimed to increase the number of wealthy individuals by implementing a series of reforms.

Each reform involved the government selecting a subset of individuals, collecting all their savings, and then redistributing the total equally among the selected individuals. The ultimate goal was to ensure that each individual had at least(x) burles to be considered wealthy.

Given the initial savings of each individual and the minimum wealth threshold(x), your task is to calculate the maximum possible number of wealthy individuals after the reforms, if any were implemented.

Input:

The first line contains a single integer(T) where, (1 <= T <= 1000) — the number of

test cases.

For each test case, there are two lines:

- The second line contains two integers(n) and(x) where, (1 <= n <= 10^5, 1 <= x <= 10^9) — representing the number of individuals and the minimum wealth threshold.

- The third line contains(n) integers(a1, a2,...., an) where, (1 <= ai <= 10^9) — the initial savings of each individual.

Output:

Print(T) integers, one per test case, representing the maximum possible number of wealthy individuals after the reforms.

Constraints:

The total sum of(n) across all test cases does not exceed(10^5).

Sample input:

1 1 2 13

Sample output:

1

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
class TestCase:
def __init__(self, n, x, a):
self.n = n
self.x = x
self.a = a

def calculate_qualified_elements(self):
a_sorted = sorted(self.a, reverse=True)
p, q = 0, 0
#..... YOUR CODE STARTS HERE .....


for i in range(len(a_sorted)):
```

```
p += a_sorted[i]
if p/(i+1) >= self.x:
q += 1

#..... YOUR CODE ENDS HERE .....
return q


def process_test_cases():
test_cases = int(input())
results = []

for _ in range(test_cases):
n, x = map(int, input().split())
a = list(map(int, input().split()))
#..... YOUR CODE STARTS HERE .....


test_case = TestCase(n,x,a)
qualified_elements = test_case.calculate_qualified_elements()
results.append(qualified_elements)


#..... YOUR CODE ENDS HERE .....

return results


if __name__ == "__main__":
results = process_test_cases()
for result in results:
print(result)
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

2
4
0
3

Compilation Status: Passed

Execution Time:

0.012s

ADITYA KUMAR JHA (adityajha37591@gmail.com)

23. You have been given an array (a) containing (n) positive integers.

Your task is to find the minimum number of moves required to transform the array such that each element is divisible by (k) (given).

During each move, you can either increase (x) by 1 or choose one element (ai) from (a) and increase it by (x) (then increase (x) by 1). However, you can only perform the first operation once for each (i) from 1 to (n).

Let's solve (t) independent test cases.

Input:

The first line contains (t) (the number of test cases).

For each test case:

- The first line contains (n) and (k) (length of (a) and required divisor).

- The second line contains (n) integers (a1, a2, ..., an).

Output:

For each test case, print the minimum number of moves required.

Constraints:

(1 <= t <= 2 times 10^4)

(1 <= n <= 2 times 10^5)

(1 <= k <= 10^9)

(1 <= ai <= 10^9)

Sample input:

5 4 3 1 2 1 3 10 6 8 7 1 8 3 7 5 10 8 9 5 10 20 100 50 20 100500 10 25 24 24 24 24 24 24 24 24 24 24 8 8 1 2 3 4 5 6 7 8

Sample output:

6 18 0 227 8

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

```
class RemainderCalculator:
def __init__(self, n, k, a):
self.n = n
self.k = k
self.a = a

def calculate_remainders(self):
rem = {}
#..... YOUR CODE STARTS HERE .....

for num in self.a:
remainder = num%self.k
if remainder!=0:
rem[remainder]=rem.get(remainder,0)+1


#..... YOUR CODE ENDS HERE .....

return rem

def find_maximum_result(self, rem):
#..... YOUR CODE STARTS HERE .....
```

```
result =0
for r, count in rem.items():
moves =(self.k * (count - 1)) + (self.k - r)+1
result = max(result,moves)
return result

#..... YOUR CODE ENDS HERE .....


if __name__ == "__main__":
from sys import stdin, stdout, setrecursionlimit

setrecursionlimit(10**6)

t = int(stdin.readline())
for _ in range(t):
n, k = map(int, stdin.readline().split())
a = list(map(int, stdin.readline().split()))

#..... YOUR CODE STARTS HERE .....


calulator=RemainderCalculator(n,k,a)
remainders=calulator.calculate_remainders()
print(calulator.find_maximum_result(remainders))

#..... YOUR CODE ENDS HERE
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

6
18
0
227
8

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

6

Compilation Status: Passed

Execution Time:

0.012s

ADITYA KUMAR JHA (adityajha97691@gmail.com)

24. Richard Hendricks, Gilfoyle, and Dinesh, the brilliant minds behind Pied Piper, are on a mission to optimize their messaging platform's data compression algorithm. They aim to represent "Yes" and "No" messages more efficiently by converting them into a different base that requires fewer characters, thus reducing the overall message size.

Richard explains that they currently represent "Yes" messages as 1 and "No" messages as 0 in their messaging platform. However, he believes they can further compress this representation by changing the base to a more efficient one, achieving a four-fold decrease in the number of characters needed to represent the messages.

Gilfoyle immediately jumps in, suggesting a strategy to change the base of the representation to achieve their goal. With his expertise in cryptography and data compression, Gilfoyle proposes a solution to map the binary representation onto a different base that requires only one-fourth of the characters needed in the existing base.

Help Gilfoyle defend his case and determine the optimal base for representing "Yes" and "No" messages to achieve a four-fold decrease in the number of characters required.

Input:

A single line containing a string made up of 0's and 1's representing "Yes" and "No" messages.

Input Constraints: 1 ≤ N ≤ 57, where N is the number of characters (0's and 1's) in the string.

Output:

Single line denoting the value of the binary string in a base that uses four times few
characters than the existing base.

Sample input:

10110111000010010011011001010111111000111010010010000

Sample output:

16e126cafc7490

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
a = input()

#..... YOUR CODE STARTS HERE .....


hexadecimal=hex(int(a,2))

result=hexadecimal[2:]

print(result)

#..... YOUR CODE ENDS HERE .....
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

16e126cafc7490

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

b7093657e3a4

Compilation Status: Passed

Execution Time:

0.012s

25. Sanjay is a Stock Exchange Specialist. Any stocks that he buys, he has a capacity of for such greater price. He has created a algorithm to find the profit that can be obtained by trading certain stocks.

The machines will be generating a string, the string will be made up of integers from 0 to 9. Now each integer in this string will be a coded value of the profit. You have to multiply a digit N, with n-1,n-2,...till n-i>=1. Each digit in the string will be giving a profit of the particular amount thus obtained. Now summing up all the profits, it should be equal to the entire string considered as a integer input.

If they are equal, then the prediction was right, so print Right, if else print Wrong.

ADITYA KUMAR JHA (adityajha97.ej@gmail.com)

Input:

One single string input N, where N is the prediction by Victor

Input Constraints : 1<=length of N<=10^5

Output:

"Right" if the prediction is right and "Wrong" if the prediction is wrong

Sample input:

40583

Sample output:

Wrong

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
class NumberChecker:
def __init__(self, n):
self.n = n
self.sum = 0
self.temp = n

def check_factorial_sum(self):
while self.n:
#..... YOUR CODE STARTS HERE .....

digit = int(self.n % 10)
factorial =1
for i in range(1,digit +1):
factorial *= i

self.sum += factorial
self.n //= 10


#..... YOUR CODE ENDS HERE .....

if self.sum == self.temp:
return "Right"
else:
return "Wrong"


if __name__ == "__main__":
n = int(input())
number_checker = NumberChecker(n)
result = number_checker.check_factorial_sum()
print(result)
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Wrong

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Wrong

Compilation Status: Passed

Execution Time:

0.011s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

26. Gordon the wizard possesses an extraordinary collection of magical stones. Each stone has a unique weight ranging from 1 to n.

Gordon's peculiar ability allows him to fuse two stones together, resulting in a new stone with a weight equal to the sum of the weights of the two original stones. However, the two original stones vanish after fusion.

Gordon's ultimate goal is to maximize the number of stones that have the same weight. While it's not mandatory to make all stones uniform in weight, he desires to have as many stones of equal weight as possible.

Given the number of stones in Gordon's collection, determine how many stones of equal weight he can create using his magical fusion process.

Input:

The input consists of multiple test cases. The first line contains a single integer t (1 ≤ t ≤ 1000) — the number of test cases.

The following t lines each contain a single integer n (1 ≤ n ≤ 10^9), representing the number of stones in Gordon's collection.

Output:

For each test case, print a single integer, the maximum number of stones with equa
weight that Gordon can create.

Sample input:

110

Sample output:

5

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
class StoneCalculator:
def __init__(self, t):
self.t = t

def calculate_max_stones(self, n):
#..... YOUR CODE STARTS HERE .....

return n// 2

#..... YOUR CODE ENDS HERE .....
if __name__ == "__main__":
t = int(input())
stone_calculator = StoneCalculator(t)

for i in range(t):
n = int(input())
result = stone_calculator.calculate_max_stones(n)
print(result)
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

5

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

13

Compilation Status: Passed

Execution Time:

0.012s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

27. Sanjay wants to become the largest of bears, or at least to become larger than his brother Midhun.

Right now, Sanjay and Midhun weigh a and b respectively. It's guaranteed that Sanjay's weight is smaller than or equal to his brother's weight.

Sanjay eats a lot and his weight is tripled after every year, while Midhun's weight is doubled after every year.

After how many full years will Sanjay become strictly larger (strictly heavier) than Midhun?

Input:

The only line of the input contains two integers a and b ($1 \le a \le b \le 10$) — the weight of Sanjay and the weight of Midhun respectively.

Output:

Print one integer, denoting the integer number of years after which Sanjay will become strictly larger than Midhun.

Sample input:

4 7

Sample output:

2

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
class Calculation:
def __init__(self, x, y):
self.x = x
self.y = y

def find_iteration(self):
#..... YOUR CODE STARTS HERE .....

if self.x > self.y:
return 1

years = 0
while self.x <= self.y:
self.x *= 3
self.y *= 2
years += 1

return years

#..... YOUR CODE ENDS HERE .....

if __name__ == "__main__":
x, y = map(int, input().split())
calculation = Calculation(x, y)
result = calculation.find_iteration()
if result is not None:
print(result)
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

2

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1

Compilation Status: Passed

Execution Time:

0.011s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

28. A group of friends is planning a road trip to visit multiple destinations. Each friend owns a car, and they need to ensure that the total fuel consumption of their vehicles matches the distance they plan to travel. The distance to each destination is represented by a list of integers, where positive values indicate the distance in kilometers they need to travel, and negative values indicate the distance they need to backtrack.

The friends can adjust the fuel levels of their cars by adding or subtracting fuel, with each adjustment costing one coin. They want to minimize the total cost of adjusting their fuel levels to match the total distance they plan to travel.

What is the minimum cost the group will have to pay to ensure that the total fuel consumption of their vehicles equals the total distance they plan to travel?

Input:

The first line contains a single integer n (1 ≤ n ≤ 10^5) — the number of destinations the group plans to visit.

The second line contains n integers a1, a2, ..., an (-10^9 ≤ ai ≤ 10^9) — representing the distances to each destination. Positive values indicate the distance they need t travel, and negative values indicate the distance they need to backtrack.

Output:

Output a single number — the minimal number of coins you need to pay to make the product equal to 1.

Sample input:

40 0 0 0

Sample output:

4

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

```
class NumberProcessor:
def __init__(self, l):
self.l = l
self.s = 0
self.odd = 0
self.nz = 0

def process_numbers(self):
#..... YOUR CODE STARTS HERE .....

for i in range(len(self.l)):
if int(self.l[i]) < 0:
self.odd += 1
self.s += abs(-1 - int(self.l[i]))
elif int(self.l[i]) > 0:
self.s += (int(self.l[i]) - 1)
else:
self.nz += 1

if self.odd % 2 != 0:
if '0' in self.l:
return self.s + 1 * self.nz
else:
```

```
return self.s + 2
else:
if '0' in self.l:
return self.s + 1 * self.nz
else:
return self.s



#..... YOUR CODE ENDS HERE .....


if __name__ == "__main__":
n = int(input())
l = str(input()).split(" ")

number_processor = NumberProcessor(l)
result = number_processor.process_numbers()
print(result)
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

4

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

13

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Status: Passed

Execution Time:

0.012s

29. The Felicity Committee assigns a power value to each superhero, and the Avengers aim to maximize their team's average power through strategic operations.

Initially, the Avengers have n superheroes with powers a1, a2, ..., an. In each operation, they can either remove a superhero (if there are at least two) or increase a superhero's power by 1.

They are allowed a maximum of m operations, with each superhero limited to k power increases. Can you assist the Avengers in maximizing their team's average power

Input:

The first line contains three integers n, k and m (1≤n≤105, 1≤k≤105, 1≤m≤107) — the number of superheroes, the maximum number of times you can increase power of a particular superhero, and the total maximum number of operations.

The second line contains n integers a1,a2,…,an (1≤ai≤106) — the initial powers of the superheroes in the cast of avengers.

Output:

Output a single number — the maximum final average power.

Your answer is considered correct if its absolute or relative error does not exceed 10−6.

Formally, let your answer be a, and the jury's answer be b. Your answer is accepted if and only if |a−b|max(1,|b|)≤10−6.

Sample input:

2 4 64 7

Sample output:

11.0

ADITYA KUMAR JHA (adityajha8759@gmail.com)

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
n, k, m = map(int, input().split())
a = list(map(int, input().split()))

#..... YOUR CODE STARTS HERE .....

a.sort()
ans=sum(a)
r=0
for i in range(n):
if i<=m:
r=int(max(r,(ans+min(k*(n-i),m-i))/(n-i)))
ans-=a[i]

print(r)

#..... YOUR CODE ENDS HERE ..
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

11

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

5

Compilation Status: Passed

Execution Time:

0.012s

30. Sanjay is a student who hates maths. But he was a good programmer. There was a sum in second chapter which was given to him as a homework.Help him to finish his homework.

You are given with a number n. You have to find the sum of numbers from 1 to n and subtract all the 2nd values from the sum. Help him to finish the exercise in the second

chapter. Write a program to finish the maths homework.

Input:

The first line contains an integer t, the total number of testcases.

The next t lines contain an integer n.

Input Constraints:1<=t<=1001<=n<=10^9

Output:

Print t lines of integer values

Sample input:

241000000000

Sample output:

-4499999998352516354

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
import math

class NumberCalculator:
#..... YOUR CODE STARTS HERE .....
```

```
def _init_(self):
pass
def calculate_result(self,n):
return n*(n+1)//2-2*(2**(int(math.log2(n))+1)-1)

#..... YOUR CODE ENDS HERE .....

if __name__ == "__main__":
number_calculator = NumberCalculator()
test_cases = int(input())

for i in range(test_cases):
n = int(input())
result = number_calculator.calculate_result(n)
print(result)
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

-4
499999998352516354

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

61
909
3285271

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Status: Passed

Execution Time:

0.011s

31. You are given a positive integer 'n'. Your task is to find all unique combinations of positive integers that multiply together to give 'n'. A combination is considered unique if there is no way to reorder it to get a different combination. The integers in each combination should be in non-decreasing order.

Your goal is to implement a function that generates these combinations for a given value of 'n'.

Input:

An integer 'n' (2 ≤ n ≤ 30), representing the number to factorize.

Output:

A list of lists, where each inner list contains a unique combination of integers greater than 1 that multiply to 'n'. The inner lists should be sorted in ascending order, and the outer list should also be sorted based on the first element of each inner list.

Constraints:

At least one valid combination exists for the given 'n'.

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Sample Input:

8

Sample Output:

[[2, 2, 2], [2, 4], [1, 8]]

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
class FactorFinder:
def __init__(self, n):
self.n = n

def find_factors(self):
def backtrack(remaining, start, path, result):
#..... YOUR CODE STARTS HERE .....

if remaining == 1:
if len(path) > 1:
result.append(path[:])
else:
result.append([1, path[0]])
return
for i in range(start, remaining + 1):
if remaining % i == 0:
path.append(i)
backtrack(remaining // i, i, path, result)
path.pop()


#..... YOUR CODE ENDS HERE .....

result = []
backtrack(self.n, 2, [], result)
return result

if __name__ == '__main__':
n = int(input())
factor_finder = FactorFinder(n)
print(factor_finder.find_factors())
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

[[2, 2, 2], [2, 4], [1, 8]]

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

[[1, 23]]

Compilation Status: Passed

Execution Time:

0.012s

ADITYA KUMAR JHA (adityajha1991@gmail.com)

32. Mani, renowned for his cryptographic prowess, has developed a novel encryption technique for transmitting positive integers securely to his acquaintances. In his method, Mani selects three secret integers — a, b, and c — from a specified range determined by lower and upper bounds, denoted as l and r. Subsequently, he employs the formula m = n * a + b - c to generate the encrypted value of the integer n.

However, an adversary has managed to intercept the parameters l, r, and m, posing a challenge to decrypt the original values of a, b, and c. The task at hand is to decipher any valid combination of a, b, and c that adheres to the following criteria:

The values of a, b, and c must be integers.

They must fall within the inclusive range [l, r].

There exists at least one positive integer n such that n * a + b - c equals the intercepted value m.

The input comprises multiple test cases, each presenting the intercepted values of l, r, and m. It is assured that at least one feasible solution exists, and any valid combination of a, b, and c may be outputted if multiple solutions are available.

Sample input:

2010 12 4325 49 15 7 398 9 4416 17 5030 40 975601 801 1000100 102 909599 799 1000503 997 9194 383 590000 100000 70999975000 100000 124999375000 499999 625001375000 500000 624999300000 400000 499999250000 500000 170000 80000 227277025770000 80000 999995334490000 100000 9999955820

Sample output:

11 11 1225 25 495 6 79 8 917 16 1735 35 40800 801 601101 102 102599 601 799503 503 997194 194 383100000 100000 9000199999 100000 75000375000 375000 499999499999 500000 375000399999 400000 300000250000 250001 50000070007 73002 8000070009 80000 7219890003 92499 100000

Completion Status: Completed

Concepts Included:

gu - introduction to oops: unit 1

Language Used: PYTHON 3

Source Code:

```
import sys

def get_ints():
return map(int, sys.stdin.readline().split())

for _ in range(int(input())):
lst = list(get_ints())

#..... YOUR CODE STARTS HERE .....

l,r,m=lst[0],lst[1],lst[2]
for i in range(l,r+1):
x1=m%i
x2=i-(m%i)
if i<=m and x1<=(r-l):
a=i
b=r
c=r-x1
i=r+2
break
elif x2<=(r-l):
a=i
b=r-x2
c=r
i=r+2
break
print(a,end=" ")
print(b,end=" ")
print(c)



#..... YOUR CODE ENDS HERE .....
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

11 11 12
25 25 49
5 6 7
9 8 9
17 16 17
35 35 40
800 801 601
101 102 102
599 601 799
503 503 997
194 194 383
100000 100000 90001
99999 100000 75000
375000 375000 499999
499999 500000 375000
399999 400000 300000
250000 250001 500000
70007 73002 80000
70009 80000 72198
90003 92499 100000

Compilation Status: Passed

Execution Time:

0.094s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1 500000 500000
500000 500000 500000
499999 500000 375000
499999 500000 375000
499999 500000 375000

499999 500000 375000
499999 500000 375000
499999 500000 375000
499999 500000 375000
499999 500000 375000
499999 500000 375000
499999 500000 375000
499999 500000 375000
499999 500000 375000
499999 500000 375000
499999 500000 375000
499999 500000 375000
499999 500000 375000
499999 500000 375000
499999 500000 375000

Compilation Status: Passed

Execution Time:

0.836s

ADITYA KUMAR JHA (adityajha0515.11@gmail.com)

33. In a college, there are two types of students: Undergraduate and Graduate. Both types of students have a method get_fee() that returns the tuition fee. However, the tuition fee is calculated differently for Undergraduate and Graduate students. For Undergraduate students, the tuition fee is a fixed amount of $5000. For Graduate students, the tuition fee is the number of credits taken times $50.

Please write a Python program that defines classes for Undergraduate and Graduate students, both inheriting from a common Student class. The Student class should have the get_fee() method, which should be overridden in the Undergraduate and Graduate classes.

Sample Input:

60

Sample Output:

50003000

Explanation:

The Undergraduate class overrides the get_fee() method and returns a fixed amount of $5000. The Graduate class also overrides the get_fee() method and returns the number of credits taken times $50. In the sample input, the Graduate student has taken 60 credits, so the fee is 60 * $50 = $3000.

Constraints:

The number of credits for a Graduate student is a positive integer.

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
class Student:
#..... YOUR CODE STARTS HERE .....
def get_fee(self):
pass


#..... YOUR CODE ENDS HERE .....

class Undergraduate(Student):
def get_fee(self):
#..... YOUR CODE STARTS HERE .....

return 5000


#..... YOUR CODE ENDS HERE .....

class Graduate(Student):
def __init__(self, credits):
#..... YOUR CODE STARTS HERE .....


self.credits = credits

#..... YOUR CODE ENDS HERE .....

def get_fee(self):
#..... YOUR CODE STARTS HERE .....

return self.credits * 50

#..... YOUR CODE ENDS HERE .....

credits = int(input())

undergraduate = Undergraduate()
graduate = Graduate(credits)
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

programs = [undergraduate, graduate]
for program in programs:
print(program.get_fee())

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

5000
2000

Compilation Status: Passed

Execution Time:

0.014s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

5000
5000

Compilation Status: Passed

Execution Time:

0.011s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

34. In a B.Tech course, there are several subjects, each with a different method of evaluation. Some subjects have lab work, some have project work, and some have only theory exams. Define an interface Subject with the methods has_lab(), has_project(), and

get_theory_marks(). Then, define three classes LabSubject, ProjectSubject, and TheorySubject that implement this interface according to the following rules:

LabSubject returns True for has_lab(), False for has_project(), and a fixed score of 70 for get_theory_marks().

ProjectSubject returns False for has_lab(), True for has_project(), and a fixed score of 80 for get_theory_marks().

TheorySubject returns False for both has_lab() and has_project(), and a fixed score of 90 for get_theory_marks().

Sample Input:

LabSubject().has_lab()ProjectSubject().has_project()TheorySubject().get_theory_marks()

Sample Output:

TrueTrue90

Explanation:

The LabSubject class returns True for has_lab(), indicating that the subject has lab work. The ProjectSubject class returns True for has_project(), indicating that the subject has project work. The TheorySubject class returns 90 for get_theory_marks(), indicating that the theory marks for the subject are 90.

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
from abc import ABC, abstractmethod

class Subject(ABC):
#..... YOUR CODE STARTS HERE .....
def get_theory_marks(self):
pass


#..... YOUR CODE ENDS HERE .....
```

```
class LabSubject(Subject):
def has_lab(self):
#..... YOUR CODE STARTS HERE .....
return True


#..... YOUR CODE ENDS HERE .....

def has_project(self):
#..... YOUR CODE STARTS HERE .....

return False

#..... YOUR CODE ENDS HERE .....

def get_theory_marks(self):
#..... YOUR CODE STARTS HERE .....

return 70

#..... YOUR CODE ENDS HERE .....

class ProjectSubject(Subject):
def has_lab(self):
#..... YOUR CODE STARTS HERE .....
return False


#..... YOUR CODE ENDS HERE .....

def has_project(self):
#..... YOUR CODE STARTS HERE .....

return True

#..... YOUR CODE ENDS HERE .....

def get_theory_marks(self):
#..... YOUR CODE STARTS HERE .....
return 80


#..... YOUR CODE ENDS HERE .....

class TheorySubject(Subject):
def has_lab(self):
#..... YOUR CODE STARTS HERE .....
return False


#..... YOUR CODE ENDS HERE .....

def has_project(self):
#..... YOUR CODE STARTS HERE .....
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

```
return False

#..... YOUR CODE ENDS HERE .....

def get_theory_marks(self):
#..... YOUR CODE STARTS HERE .....
return 90


#..... YOUR CODE ENDS HERE .....

def call_func(cls, func):
value = None

if (func == "has_lab()"):
value = cls.has_lab()

elif(func == "has_project()"):
value = cls.has_project()

elif(func == "get_theory_marks()"):
value = cls.get_theory_marks()

return value

# Getting input from the user
lab_class, lab_func = input().strip().split(".")
project_class, project_func = input().strip().split(".")
theory_class, theory_func = input().strip().split(".")

if (lab_class == 'LabSubject()'):
lab = LabSubject()
print(call_func(lab, lab_func))

if(project_class == "ProjectSubject()"):
project = ProjectSubject()
print(call_func(project, project_func))

if(theory_class == "TheorySubject()"):
theory = TheorySubject()
print(call_func(theory, theory_func))
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

True
True
90

Compilation Status: Passed

Execution Time:

0.018s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

False
80
False

Compilation Status: Passed

Execution Time:

0.017s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

35. In a school, there are several types of staff members: Teachers, Janitors, and Administrators. All staff members have a method get_salary() that returns their monthly salary. However, the salary is calculated differently for each type of staff member. For Teachers, the salary is a fixed amount of $3000. For Janitors, the salary is a fixed amount of $2000. For Administrators, the salary is a fixed amount of $4000.

Write a Python program that defines an abstract class StaffMember with the abstract method get_salary(). Then, define three classes Teacher, Janitor, and Administrator that inherit from the StaffMember class and implement the get_salary() method.

Sample Input:

Teacher().get_salary()Janitor().get_salary()Administrator().get_salary()

Sample Output:

300020004000

Explanation:

In this sample, the Teacher class returns a salary of $3000, the Janitor class returns a salary of $2000, and the Administrator class returns a salary of $4000. This demonstrates how each class correctly implements the get_salary() method from the StaffMember abstract base class.

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

ADITYA KUMAR JHA (adityajha37591@gmail.com)

```
from abc import ABC, abstractmethod

class StaffMember(ABC):
@abstractmethod
def get_salary(self):
pass

#..... YOUR CODE STARTS HERE .....
class Teacher(StaffMember):
def get_salary(self):
return 3000

class Janitor(StaffMember):
def get_salary(self):
return 2000

class Administrator(StaffMember):
def get_salary(self):
return 4000


#..... YOUR CODE ENDS HERE .....

def call_func(cls, func):
value = None
cls_obj = None

if (cls == "Teacher()"):
cls_obj = Teacher()
```

```
elif(cls == "Janitor()"):
cls_obj = Janitor()

elif(cls == "Administrator()"):
cls_obj = Administrator()

if (func == "get_salary()"):
value = cls_obj.get_salary()

return value

for _ in range(3):
cls, func = input().strip().split(".")
print(call_func(cls, func))
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

3000
2000
4000

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

3000
4000
2000

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Status: Passed

Execution Time:

0.018s

36. In a sports team, each player has a name and a number of goals scored. The number of goals scored should always be a non-negative integer. If a negative value or a non-integer value is assigned to the number of goals, it should be automatically set to 0.

Please write a Python class Player that has name and goals as attributes. Use the property decorator to ensure that goals is always a non-negative integer.

Sample Input 1:

John, 510

Sample Output 1:

John510

Sample Input 2:

John, 10-5

Sample Output 2:

John100

Explanation:

In the sample input, a Player object is created with the name "John" and 5 goals. The goals property is then updated to 10, -5, and "hello". When the goals property is set to a negative value or a non-integer value, the goals.setter method automatically sets goals to 0. This is why the output is 5, 10, 0, 0. This demonstrates how the property decorator can be used to control the values of an attribute.

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
class Player:
def __init__(self, name, goals=0):
#..... YOUR CODE STARTS HERE .....
```

```
self._name = name
self._goals = self._validate_goals(goals)

@property
def goals(self):
return self._goals

@goals.setter
def goals(self, value):
self._goals = self._validate_goals(value)

#..... YOUR CODE ENDS HERE .....

#..... YOUR CODE STARTS HERE .....

def _validate_goals(self, value):
try:
value = int(value)
if value < 0:
return 0
else:
return value
except ValueError:
return 0

#..... YOUR CODE ENDS HERE .....

def clean_input(value):
return str(value.strip())

name, goal = map(clean_input, input().strip().split(","))
next_goal = input()

player = Player(name, goal)
print(name)
print(player.goals)
player.goals =next_goal
print(player.goals)
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Alice
7
15

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Bob
0
0

Compilation Status: Passed

Execution Time:

0.011s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

37. You are developing a book management system for a bookstore. The system has a function get_book_details(isbn: str) -> dict that takes an ISBN number as input and returns a dictionary with book details. The dictionary contains the following keys: 'title', 'author', 'price', and 'quantity'.

The bookstore has the following books:

"Python Programming" by John Doe, ISBN: "1234567890", Price: 500, Quantity: 10

"Data Structures and Algorithms" by Jane Doe, ISBN: "9876543210", Price: 600, Quantity: 5

"Machine Learning" by Sam Smith, ISBN: "1111111111", Price: 700, Quantity: 2

However, not all books are available in the store at all times. If a book is not available, the function should raise a ValueError with a message "Book not found".

Write a Python function handle_book_request(isbn: str) -> str that calls

get_book_details(isbn: str) -> dict and handles any ValueError that might be raised. The function should return a string message about the status of the request.

If the book is found, the function should return a message in the following format: "Book found: {title} by {author}. Price: {price}, Quantity: {quantity}". If the book is not found, it should return the message "Book not found".

Constraints:

ISBN is a string of length 10 or 13.

The title, author are strings and price, quantity are integers.

You can't install any external libraries.

Sample Input:

1234567890

Sample Output:

Book found: Python Programming by John Doe. Price: 500, Quantity: 10

Explanation:

In the sample output, that message is returned when the book is found in the bookstore. The details of the book, including the title, author, price, and quantity, are displayed in the message. This is done by catching the book details returned by the get_book_details function in a try block.

A message, "Book not found", is returned If the book is not found in the bookstore. This happens when the get_book_details function raises a ValueError. In the except block, this error is caught and the error message is returned. This demonstrates how error handling works in Python - when an error occurs, instead of the program crashing, the error is caught and handled gracefully. In this case, a meaningful message is returned to the user. This is a key aspect of robust software design.

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Completion Status: Completed

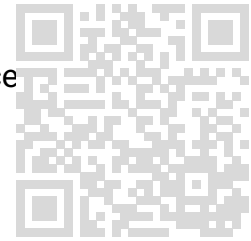
Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
def get_book_details(isbn):
books = {
"1234567890": {"title": "Python Programming", "author": "John Doe", "price": 500,
"quantity": 10},
```



```
"9876543210": {"title": "Data Structures and Algorithms", "author": "Jane Doe", "price
600, "quantity": 5},
"1111111111": {"title": "Machine Learning", "author": "Sam Smith", "price": 700,
"quantity": 2},
}
if isbn in books:
return books[isbn]
else:
raise ValueError("Book not found")

def handle_book_request(isbn):
#..... YOUR CODE STARTS HERE .....
try:
book_details = get_book_details(isbn)
return f"Book found: {book_details['title']} by {book_details['author']}. Price:
{book_details['price']}, Quantity: {book_details['quantity']}"
except ValueError as e:
return str(e)


#..... YOUR CODE ENDS HERE .....

isbn = input()
response = handle_book_request(isbn)
print(response)
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Book found: Python Programming by John Doe. Price: 500, Quantity: 10

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Book found: Data Structures and Algorithms by Jane Doe. Price: 600, Quantity: 5

Compilation Status: Passed

Execution Time:

0.012s

38. You are developing a product management system for an e-commerce platform. The system has a function get_product_details(product_id: str) -> dict that takes a product ID as input and returns a dictionary with product details. The dictionary contains the following keys: 'name', 'category', 'price', and 'stock'.

The e-commerce platform has the following products:

"Wireless Mouse", Product ID: "123456", Category: "Electronics", Price: 500, Stock: 10

"Running Shoes", Product ID: "789012", Category: "Sports", Price: 600, Stock: 5

"Coffee Maker", Product ID: "345678", Category: "Home Appliances", Price: 700, Stock: 2

However, not all products are available on the platform at all times. If a product is not available, the function should raise an Exception with a message "Product not found".

Write a Python function handle_product_request(product_id: str) -> str that calls get_product_details(product_id: str) -> dict and handles any Exception that might be raised. The function should return a string message about the status of the request.

If the product is found, the function should return a message in the following format: "Product found: {name}, Category: {category}, Price: {price}, Stock: {stock}". If the product is not found, it should return the message "Product not found".

Constraints:

Product ID is a string of length 6.

The name, category are strings and price, stock are integers.

You can't install any external libraries.

Sample Input:

901234

ADITYA KUMAR JHA (adityajha27591@gmail.com)

Sample Output:

Request handledProduct not found

Explanation:

"Product not found": This message is returned when the product with the ID "901234" is not found on the platform. In this case, the get_product_details function raises an Exception with the message "Product not found". This exception is caught in the handle_product_request function and the error message is returned. This demonstrates how exception handling works in Python - when an exception occurs, instead of the program crashing, the exception is caught and handled gracefully. In this case, a meaningful message is returned to the user.

In all cases, the finally block is executed after the try and except blocks, indicating that the product request has been processed (Request handled), regardless of whether an exception was raised or not. This is a key aspect of robust software design and demonstrates the use of the finally keyword in Python exception handling.

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
def get_product_details(product_id):
products = {
"123456": {"name": "Wireless Mouse", "category": "Electronics", "price": 500, "stock":
10},
"789012": {"name": "Running Shoes", "category": "Sports", "price": 600, "stock": 5},
"345678": {"name": "Coffee Maker", "category": "Home Appliances", "price": 700,
"stock": 2},
}
if product_id in products:
return products[product_id]
else:
raise Exception("Product not found")

def handle_product_request(product_id):
#..... YOUR CODE STARTS HERE .....
try:
product_details = get_product_details(product_id)
return f"Product found: {product_details['name']}, Category:
{product_details['category']}, Price: {product_details['price']}, Stock:
{product_details['stock']}"
except Exception as e:
return str(e)
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

```
finally:
print("Request handled")


#..... YOUR CODE ENDS HERE .....

# Get input from the user
user_input = input()

# Call the handle_product_request function with user input
result = handle_product_request(user_input)
print(result)
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Request handled
Product found: Wireless Mouse, Category: Electronics, Price: 500, Stock: 10
```

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Request handled
Product found: Running Shoes, Category: Sports, Price: 600, Stock: 5
```

Compilation Status: Passed

Execution Time:

ADITYA KUMAR JHA (adityajha37591@gmail.com)

0.012s

39. You are developing a simple calculator in Python. The calculat has a function calculate(operation: str, num1: float, num2: float) -> float that takes an operation and two numbers as input and returns the result of the operation. The operation can be one of the following: 'add', 'subtract', 'multiply', 'divide'.

However, not all operations are valid. If an invalid operation is passed, the function should raise a ValueError with a message "Invalid operation".

Also, if the operation is 'divide' and the second number is 0, the function should raise a ZeroDivisionError with a message "Cannot divide by zero".

Write a Python function handle_calculation(operation: str, num1: float, num2: float) -> str that calls calculate(operation: str, num1: float, num2: float) -> float and handles any ValueError or ZeroDivisionError that might be raised. The function should return a string message about the status of the calculation.

Constraints:

num1 and num2 are floats.

You can't install any external libraries.

Sample Input:

add, 1.0, 2.0

Sample Output:

Result: 3.0

Explanation:

The result of the operation is displayed in the message.

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
def calculate(operation, num1, num2):
```

```
if operation == 'add':
return num1 + num2
elif operation == 'subtract':
return num1 - num2
elif operation == 'multiply':
return num1 * num2
elif operation == 'divide':
if num2 == 0:
raise ZeroDivisionError("Cannot divide by zero")
return num1 / num2
else:
raise ValueError("Invalid operation")

def handle_calculation(operation, num1, num2):
#..... YOUR CODE STARTS HERE .....

try:
result = calculate(operation, num1, num2)
return f"Result: {result}"
except ValueError as e:
return str(e)
except ZeroDivisionError as e:
return str(e)

#..... YOUR CODE ENDS HERE .....

def main():
inputs = input()
operation, num1, num2 = map(str.strip, inputs.split(","))
num1 = float(num1)
num2 = float(num2)

message = handle_calculation(operation, num1, num2)
print(message)

if __name__ == "__main__":
main()
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Result: 6.0

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Result: 5.0

Compilation Status: Passed

Execution Time:

0.012s

ADITYA KUMAR (adityaha37591@gmail.com)

40. You are developing a stock management system for a market. The system has a function get_stock(item: str) -> int that takes an item name as input and returns the quantity of that item in stock.

The market has the following items in stock:

"Apples" - 50

"Oranges" - 30

"Grapes" - 0

However, not all items are available in the market at all times. If an item is not available, the function should raise a custom exception ItemNotFoundError with a message "Item not found".

Write a Python function handle_stock_request(item: str) -> str that calls get_stock(item: str) -> int and handles any ItemNotFoundError that might be raised. The function should return a string message about the status of the request.

If the item is found and the stock is greater than 0, the function should return a message in the following format: "Stock for {item}: {stock}". If the item is not found or the stock is 0, it should return the message "Item not found".

Constraints:

Item is a string.

You can't install any external libraries.

Sample Input:

Apples

Sample Output:

Stock for Apples: 50

Explanation:

In the sample outputs, the message is returned when the item is found in the market and the stock is greater than 0. The stock quantity of the item is displayed in the message. This is done by catching the stock quantity returned by the get_stock function in a try block.

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
class ItemNotFoundError(Exception):
pass

def get_stock(item):
stock = {
"Apples": 50,
"Oranges": 30,
"Grapes": 0,
}
if item in stock:
return stock[item]
else:
raise ItemNotFoundError("Item not found")

def handle_stock_request(item):
#..... YOUR CODE STARTS HERE .....
try:
return f"Stock for {item}: {get_stock(item)}"
except ItemNotFoundError:
return f"Item{item} not found in stock"


#..... YOUR CODE ENDS HERE .....
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

```
user_input = input()

print(handle_stock_request(user_input))
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Stock for Apples: 50

Compilation Status: Passed

Execution Time:

0.017s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Stock for Oranges: 30

Compilation Status: Passed

Execution Time:

0.018s

ADITYA KUMAR JHA (adityajha37591@gmail.com)

41. Design a Python program that uses multiple inheritance to model an e-commerce platform. The platform sells two types of products: physical products and digital products. Both types of products have a name and a price. However, a physical product also has a shipping weight, while a digital product has a file size.

Create classes to represent both types of products and use multiple inheritance to create a third class that represents a product that is both physical and digital (for example, a video game that can be purchased as a physical disc or downloaded digitally).

Write a method in each class to print the details of the product.

Sample Input:

Book, 9.99, 1.0Ebook, 6.99, 2Video Game, 59.99, 0.2, 25

Sample Output:

Physical Product: Book, Price: $9.99, Shipping Weight: 1.0 kgDigital Product: Ebook, Price: $6.99, File Size: 2 MBHybrid Product: Video Game, Price: $59.99, Shipping Weight: 0.2 kg, File Size: 25 GB

Explanation:

In the sample output, each line represents the details of a product:

The first line is the details of a physical product. It's a book with a price of $9.99 and a shipping weight of 1.0 kg.

The second line is the details of a digital product. It's an Ebook with a price of $6.99 and a file size of 2 MB.

The third line is the details of a hybrid product. It's a video game with a price of $59.99, a shipping weight of 0.2 kg, and a file size of 25 GB.

These details are printed by calling the print_details() method of each product object. The method is defined in each class and prints the details specific to that class. For the HybridProduct class, which inherits from both PhysicalProduct and DigitalProduct, the print_details() method prints the details of both the physical and digital aspects of the product.

ADITYA KUMAR JHA (adityajha3215911@gmail.com)

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
class Product:
def __init__(self, name, price):
#..... YOUR CODE STARTS HERE .....
self.name= name
self.price= price
```

```
#..... YOUR CODE ENDS HERE .....

class PhysicalProduct(Product):
def __init__(self, name, price, weight):
#..... YOUR CODE STARTS HERE .....

self.weight= weight
super().__init__(name,price)


#..... YOUR CODE ENDS HERE .....

def print_details(self):
#..... YOUR CODE STARTS HERE .....
print(f"Physical Product: {self.name}, Price: ${self.price}, Shipping Weight:
{self.weight} kg")


#..... YOUR CODE ENDS HERE .....

class DigitalProduct(Product):
def __init__(self, name, price, file_size):
#..... YOUR CODE STARTS HERE .....
self.name=name
self.price=price
self.file_size= file_size


#..... YOUR CODE ENDS HERE .....

def print_details(self):
#..... YOUR CODE STARTS HERE .....
print(f"Digital Product: {self.name}, Price: ${self.price}, File Size: {self.file_size} MB")


#..... YOUR CODE ENDS HERE .....

class HybridProduct(PhysicalProduct, DigitalProduct):
def __init__(self, name, price, weight, file_size):
#..... YOUR CODE STARTS HERE .....
self.name=name
self.price=price
self.weight= weight
self.file_size= file_size


#..... YOUR CODE ENDS HERE .....

def print_details(self):
#..... YOUR CODE STARTS HERE .....
print(f"Hybrid Product: {self.name}, Price: ${self.price}, Shipping Weight: {self.weight}
kg, File Size: {self.file_size} GB")
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

#..... YOUR CODE ENDS HERE .....

```
def clean_input(value):
return str(value.strip())


if __name__ == '__main__':
pp_name, pp_price, shipping_weight = map(clean_input, input().strip().split(','))
physical_product = PhysicalProduct(pp_name, pp_price, shipping_weight)

dp_name, dp_price, file_size = map(clean_input, input().strip().split(','))
digital_product = DigitalProduct(dp_name, dp_price, file_size)

hp_name, hp_price, hp_shipping_weight, hp_file_size = map(clean_input,
input().strip().split(','))
hybrid_product = HybridProduct(hp_name, hp_price, hp_shipping_weight, hp_file_size)

physical_product.print_details()
digital_product.print_details()
hybrid_product.print_details()
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Physical Product: Laptop, Price: $599.99, Shipping Weight: 2.5 kg
Digital Product: Software, Price: $199.99, File Size: 500 MB
Hybrid Product: Video Game, Price: $59.99, Shipping Weight: 0.2 kg, File Size: 25 GB

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Physical Product: Camera, Price: $299.99, Shipping Weight: 1.0 kg
Digital Product: Photography Course, Price: $49.99, File Size: 2000 MB
Hybrid Product: Music Album, Price: $19.99, Shipping Weight: 0.1 kg, File Size: 2 GB

Compilation Status: Passed

Execution Time:

0.012s

42. You are a college professor and you have the scores of your students in a list. You want to find out which students have passed the course. A student passes the course if they score 60 or more. Write a Python function passed_students(scores) that takes a list of scores and returns a list of scores that are 60 or more. Use the filter function to filter out the passing scores.

Input:

scores: a list of integers representing the scores of the students.

Output:

The function should return a list of integers representing the scores of the students who have passed.

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Sample Input:

[55, 60, 65, 70, 75, 80, 85, 90, 95, 100]

Sample Output:

[60, 65, 70, 75, 80, 85, 90, 95, 100]

Explanation:

The filter function applies a function to every item in the scores list and returns a new list with the items for which the function returns True. In this case, the function is a lambda function that checks if a score is 60 or more. This demonstrates how the filter function can be used to filter items in a list based on a condition. In the context of grading students in a college, this can be useful to find out which students have passed the course based on their scores.

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
import json
from io import StringIO

def passed_students(scores):
#..... YOUR CODE STARTS HERE .....

return list(filter(lambda x: x >= 60, scores))

#..... YOUR CODE ENDS HERE .....

if __name__ == "__main__":
scores = json.load(StringIO(input().strip()))
passed_students = passed_students(scores)

print(passed_students)
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

[60, 65, 70, 75, 80, 85, 90, 95, 100]

Compilation Status: Passed

Execution Time:

0.018s

TestCase2:

Input:

< hidden >

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Expected Output:

< hidden >

Output:

[60, 65, 70]

Compilation Status: Passed

Execution Time:

0.019s

43. You are a college professor and you have two lists: one with the names of your students and another with their corresponding scores. You want to associate each student with their score. Write a Python function associate_students(names, scores) that takes a list of names and a list of scores and returns a list of tuples where each tuple contains a name and the corresponding score. Use the zip function to associate the names with the scores.

Input:

names: a list of strings representing the names of the students.

scores: a list of integers representing the scores of the students.

Output:

The function should return a list of tuples where each tuple contains a name and the corresponding score.

ADITYA KUMAR JHA(adityajha37591@gmail.com)

Sample Input:

['Alice', 'Bob', 'Charlie', 'Dave'], [85, 90, 78, 92]

Sample Output:

[('Alice', 85), ('Bob', 90), ('Charlie', 78), ('Dave', 92)]

Explanation:

The zip function takes two or more iterable objects (like lists or strings), aggregates them in a tuple, and returns it. In this case, it takes a name from the names list and a score from the scores list and puts them together in a tuple. This demonstrates how the zip function can be used to combine two lists into a list of tuples. In the context of grading students in a college, this can be useful to associate each student with their score.

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
import re

def associate_students(names, scores):
#..... YOUR CODE STARTS HERE .....

return list(zip(names, scores))

#..... YOUR CODE ENDS HERE .....

def clean_input(value):
value = value.strip()
return re.sub(r'[^A-Z0-9a-z,]', '', value).split(',')

if __name__ == "__main__":
names, scores = list(map(clean_input, input().strip().split('],')))
scores = list(map(int, scores))
print(associate_students(names, scores))
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

[('Alice', 85), ('Bob', 90), ('Charlie', 78), ('Dave', 92)]

Compilation Status: Passed

Execution Time:

0.017s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

[('Eve', 88), ('Frank', 92), ('Grace', 95), ('Heidi', 90)]

Compilation Status: Passed

Execution Time:

0.017s

44. You are a college professor and you have the scores of your students in a list. You want to find the total score of all students. Write a Python function total_score(scores) that takes a list of scores and returns the total score. Use the reduce function to calculate the total score.

Input:

scores: a list of integers representing the scores of the students.

Output:

The function should return an integer representing the total score of all students.

Sample Input:

[85, 90, 78, 92]

Sample Output:

345

Explanation:

The reduce function applies a function of two arguments cumulatively to the items of an iterable, from left to right, so as to reduce the iterable to a single output. In this case, it takes two scores at a time and adds them, continuing this process until all scores have been added together. This demonstrates how the reduce function can be used to reduce a list to a single result. In the context of grading students in a college, this can be useful to calculate the total score of all students.

ADITYAKUMAR JHA (adityajha37591@gmail.com)

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
import json
from io import StringIO
from functools import reduce

def total_score(scores):
#..... YOUR CODE STARTS HERE .....

return reduce(lambda x, y: x + y, scores)

#..... YOUR CODE ENDS HERE .....

def clean_input(value):
return json.load(StringIO(value))

if __name__ == "__main__":
scores = clean_input(input().strip())

print(total_score(scores))
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

345

Compilation Status: Passed

Execution Time:

0.019s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

365

Compilation Status: Passed

Execution Time:

0.019s

45. You are given a string that represents a list of movies and their release years. The format of the string is "<movie title> (<release year>)". Write a Python function get_movies(data) that takes this string and returns a list of tuples where each tuple contains the movie title and the release year. Use regular expressions to parse the data.

Input:

data: a string representing a list of movies and their release years.

Output:

The function should return a list of tuples where each tuple contains a movie title and the release year.

ADITYA KUMAR JHA (adityajha3751@gmail.com)

Sample Input:

The Shawshank Redemption (1994), The Godfather (1972), The Dark Knight (2008)

Sample Output:

[('The Shawshank Redemption', '1994'), ('The Godfather', '1972'), ('The Dark Knight', '2008')]

Explanation:

The re.findall function applies the regular expression pattern to the data string and returns a list of tuples where each tuple contains the movie title and the release year.

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
import re

def get_movies(data):
#..... YOUR CODE STARTS HERE .....

# Define the regular expression pattern
pattern = r'([\w\s]+) \((\d{4})\)'

# Find all matches using the pattern
matches = re.findall(pattern, data)

# Create a list of tuples from the matches
movies = [(title.strip(), year) for title, year in matches]

return movies

#..... YOUR CODE ENDS HERE .....

if __name__ == "__main__":
data = input().strip()
print(get_movies(data))
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

[('The Shawshank Redemption', '1994'), ('The Godfather', '1972'), ('The Dark Knight', '2008')]

Compilation Status: Passed

Execution Time:

0.017s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

[('Pulp Fiction', '1994'), ('Fight Club', '1999'), ('Forrest Gump', '1994')]

Compilation Status: Passed

Execution Time:

0.018s

46. You are managing an e-commerce website and you have a list of products. Each product is represented as a dictionary with two keys: 'name' and 'price'. You want to create a list of all products that cost more than $50. Write a Python function expensive_products(products) that takes a list of products and returns a list of the names of products that cost more than $50. Use a list comprehension to create the list.

Input:

products: a list of dictionaries. Each dictionary represents a product and has two keys: 'name' (a string) and 'price' (a float).

Output:

The function should return a list of strings representing the names of products that cost more than $50.

Sample Input:

[{'name': 'Product 1', 'price': 49.99}, {'name': 'Product 2', 'price': 50.01}, {'name': 'Product 3', 'price': 50.00}]

Sample Output:

['Product 2']

ARYA KUMAR JHA (aaryaha2709@gmail.com)

Explanation:

The list comprehension generates a new list containing the names of all products that cost more than $50. It does this by iterating over each product in the products list and checking if the product's price is greater than $50. If it is, the product's nam is added to the new list. This demonstrates how list comprehensions can be used to create new lists based on existing lists in Python. In the context of managing an e-commerce website, this can be useful to quickly find all products that meet a certain price criterion.

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
import json
from io import StringIO
import re

def expensive_products(products):
#..... YOUR CODE STARTS HERE .....

return [product['name'] for product in products if product['price'] > 50]

#..... YOUR CODE ENDS HERE .....

def convert_to_list_of_dicts(data):
data_without_brackets = re.sub(r'[\[\]]', '', data).split('},')

lst = []

data = [d + '}' if '}' not in d else d for d in data_without_brackets]

for d in data:
if ('{' in d and '}' in d):
d = d.strip().replace("'", '"')
lst.append(json.load(StringIO(d)))

return lst

if __name__ == "__main__":
products = convert_to_list_of_dicts(input().strip())
print(expensive_products(products))
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

['Product 2']

Compilation Status: Passed

Execution Time:

0.019s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

['Product A', 'Product B']

Compilation Status: Passed

Execution Time:

0.019s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

47. Design a Python program that uses hierarchical inheritance to model electronic devices. There are two types of electronic devices: Mobiles and Laptops. Both Mobiles and Laptops have a brand name and a price. However, a Mobile also has a screen size, while a Laptop has a screen resolution.

Create classes to represent both types of devices and use hierarchical inheritance to avoid code duplication. Write a method in each class to print the details of the device.

Sample Input:

Samsung Galaxy S21, 799.99, 6.2 inchesDell XPS 13, 999.99, 1920 x 1080

Sample Output:

Mobile: Samsung Galaxy S21, Price: $799.99, Screen Size: 6.2 inchesLaptop: Dell X 13, Price: $999.99, Screen Resolution: 1920 x 1080

Explanation:

In the sample output, each line represents the details of an electronic device:

The first line is the details of a mobile. It's a Samsung Galaxy S21 with a price of $799.99 and a screen size of 6.2 inches.

The second line is the details of a laptop. It's a Dell XPS 13 with a price of $999.99 and a screen resolution of 1920 x 1080.

These details are printed by calling the print_details() method of each device object. The method is defined in each class and prints the details specific to that device.

This is an example of hierarchical inheritance, where the Mobile and Laptop classes inherit from a common base class (ElectronicDevice). The base class defines properties and methods common to all electronic devices (like name and price), while the derived classes add properties and methods specific to mobiles and laptops (like screen_size and screen_resolution).

Constraints:

The name of the mobile or laptop is a string and can contain any printable ASCII characters.

The price of the mobile or laptop is a float and can be any positive number.

The screen size of the mobile is a string and can contain any printable ASCII characters.

The screen resolution of the laptop is a string and can contain any printable ASCII characters.

ADITYA KUMAR (aditya.ha37591@gmail.com)

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
class ElectronicDevice:
def __init__(self, name, price):
#..... YOUR CODE STARTS HERE .....
self.name=name
self.price=price
```

```
#..... YOUR CODE ENDS HERE .....

class Mobile(ElectronicDevice):
def __init__(self, name, price, screen_size):
#..... YOUR CODE STARTS HERE .....
super().__init__(name,price)
self.screen_size=screen_size


#..... YOUR CODE ENDS HERE .....

def print_details(self):
#..... YOUR CODE STARTS HERE .....
return(f"Mobile: {self.name}, Price: ${self.price}, Screen Size: {self.screen_size}" )


#..... YOUR CODE ENDS HERE .....

class Laptop(ElectronicDevice):
def __init__(self, name, price, screen_resolution):
#..... YOUR CODE STARTS HERE .....
super().__init__(name,price)
self.screen_resolution=screen_resolution


#..... YOUR CODE ENDS HERE .....

def print_details(self):
#..... YOUR CODE STARTS HERE .....
return(f"Laptop: {self.name}, Price: ${self.price}, Screen Resolution:
{self.screen_resolution}")


#..... YOUR CODE ENDS HERE .....

# Get sample input from the user
mobile_input = input().split(", ")
laptop_input = input().split(", ")

# Create objects
mobile = Mobile(*mobile_input)
laptop = Laptop(*laptop_input)

# Print details
print(mobile.print_details())
print(laptop.print_details())
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Mobile: iPhone 13, Price: $1099.00, Screen Size: 6.1 inches
Laptop: MacBook Pro, Price: $2399.00, Screen Resolution: 3072 x 1920

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Mobile: Google Pixel 6, Price: $599.00, Screen Size: 6.4 inches
Laptop: Dell XPS 15, Price: $1749.99, Screen Resolution: 3840 x 2400

Compilation Status: Passed

Execution Time:

0.012s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

48. Design a Python program that uses single level inheritance to model a school. The school has two types of people: students and teachers. Both students and teachers have a name and an age. However, a student also has a department, while a teacher has a subject they teach.

Create classes to represent both a student and a teacher, and use inheritance to avoid code duplication. Write a method in each class to print the details of the student or teacher.

Sample Input:

Ravi Kumar, 20, Computer ScienceDr. Srinivasan, 45, Data Structures and Algorithms

Sample Output:

Student: Ravi Kumar, Age: 20, Department: Computer ScienceTeacher: Dr. Srinivasa Age: 45, Subject: Data Structures and Algorithms

Explanation:

In the sample output, each line represents the details of a person in the school:

The first line is the details of a student. It's Ravi Kumar who is 20 years old and studying Computer Science.

The second line is the details of a teacher. It's Dr. Srinivasan who is 45 years old and teaches Data Structures and Algorithms.

These details are printed by calling the print_details() method of each person object. The method is defined in each class and prints the details specific to that person.

This is an example of single level inheritance, where the Student and Teacher classes inherit from a common base class (Person). The base class defines properties and methods common to all people (like name and age), while the derived classes add properties and methods specific to students and teachers (like grade and subject).

Constraints:

The name of the student or teacher is a string and can contain any printable ASCII characters.

The age of the student or teacher is an integer and can be any positive integer.

The department of the student is a string and can contain any printable ASCII characters.

The subject of the teacher is a string and can contain any printable ASCII characters.

ADITYA KUMAR JHA (adityajha87591@gmail.com)

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
class Person:
def __init__(self, name, age):
self.name = name
self.age = age

def print_details(self):
```

```
pass


class Student(Person):
def __init__(self, name, age, department):
#..... YOUR CODE STARTS HERE .....
self.name=name
self.age=age
self.department=department


#..... YOUR CODE ENDS HERE .....

def print_details(self):
#..... YOUR CODE STARTS HERE .....
print(f"Student: {self.name}, Age: {self.age}, Department: {self.department}")


#..... YOUR CODE ENDS HERE .....


class Teacher(Person):
def __init__(self, name, age, subject):
#..... YOUR CODE STARTS HERE .....
self.name=name
self.age=age
self.subject=subject


#..... YOUR CODE ENDS HERE .....

def print_details(self):
#..... YOUR CODE STARTS HERE .....
print(f"Teacher: {self.name}, Age: {self.age}, Subject: {self.subject}")


#..... YOUR CODE ENDS HERE .....


def main():
# Sample Input
student_info = input().split(", ")
teacher_info = input().split(", ")

# Create objects
student = Student(*student_info)
teacher = Teacher(*teacher_info)

# Print details
student.print_details()
teacher.print_details()


if __name__ == "__main__":
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

main()

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Student: Ravi Kumar, Age: 20, Department: Computer Science
Teacher: Dr. Srinivasan, Age: 45, Subject: Data Structures and Algorithms

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Student: Priya, Age: 21, Department: Electronics and Communication
Teacher: Prof. Ramanujan, Age: 50, Subject: Digital Signal Processing

Compilation Status: Passed

Execution Time:

0.011s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

49. Consider an e-commerce application that uses a class hierarchy to represent different types of products. We have a base class Product and two subclasses Book and Electronics. Both Book and Electronics classes override a method get_details() from the Product class. Now, we have a new product type EBook that is both a Book and an Electronics. The EBook class inherits from both Book and Electronics.

Write a Python function mro_sequence(cls) that takes a class cls as input and retur
the Method Resolution Order (MRO) for that class as a list of class names.

Sample Input:

EBook

Sample Output:

['EBook', 'Book', 'Electronics', 'Product', 'object']

Explanation:

In Python, the Method Resolution Order (MRO) is the order in which the base classes are searched when executing a method. Python uses an algorithm called C3 Linearization, or just C3, to compute this order. In the given example, the MRO for class EBook is EBook -> Book -> Electronics -> Product -> object.

Constraints:

The input class cls is a valid Python class.

The class hierarchy can have multiple levels of inheritance but does not contain any circular inheritance.

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
class Product:
def get_details(self):
pass

class Book(Product):
def get_details(self):
pass

class Electronics(Product):
def get_details(self):
pass

class EBook(Book, Electronics):
pass

class Apparel(Product):
```

```
pass

class SmartWatch(Apparel, Electronics):
pass

class Furniture(Product):
pass

class SmartFurniture(Furniture, Electronics):
pass

class Organic(Product):
pass

class Grocery(Product):
pass

class OrganicGrocery(Grocery, Organic):
pass

def mro_sequence(cls):
#..... YOUR CODE STARTS HERE .....
return [c.__name__ for c in cls.mro()]


#..... YOUR CODE ENDS HERE .....

class_name = input()

cls = globals()[class_name]

# Print the MRO sequence
print(mro_sequence(cls))
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

['EBook', 'Book', 'Electronics', 'Product', 'object']

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

['SmartWatch', 'Apparel', 'Electronics', 'Product', 'object']

Compilation Status: Passed

Execution Time:

0.012s

50. Consider an election scenario where we have different roles such as Citizen, Politician, and PublicServant. Both Politician and PublicServant are subclasses of Citizen. Now, we have a new role Mayor who is both a Politician and a PublicServant. The Mayor class inherits from both Politician and PublicServant.

Each of these classes has a method role_duties() that describes the duties of the role. However, when a Mayor object calls this method, which method should it execute? This is known as the Diamond Problem.

Write a Python function mro_sequence(cls) that takes a class cls as input and returns the Method Resolution Order (MRO) for that class as a list of class names.

ADITYA KUMAR SAH (adityahacker.sah@gmail.com)

Sample Input:

Mayor

Sample Output:

['Mayor', 'Politician', 'PublicServant', 'Citizen', 'object']

Explanation:

In Python, the Method Resolution Order (MRO) is the order in which the base classes are searched when executing a method. Python uses an algorithm called C3 Linearization, or just C3, to compute this order. In the given example, the MRO for class Mayor is Mayor -> Politician -> PublicServant -> Citizen -> object. This means that if a Mayor object calls the role_duties() method, it will first look for this method in the Mayor class. If it doesn't find it there, it will look in the Politician class, then in the PublicServant class, and finally in the Citizen class.

Constraints:

The input class cls is a valid Python class.

The class hierarchy can have multiple levels of inheritance but does not contain any circular inheritance.

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
class Citizen:
def role_duties(self):
return "Citizen duties"

class Politician(Citizen):
def role_duties(self):
return "Politician duties"

class PublicServant(Citizen):
def role_duties(self):
return "Public servant duties"

class Mayor(Politician, PublicServant):
def role_duties(self):
return "Mayor duties"

class Voter(Citizen):
def role_duties(self):
return "Voter duties"

class Candidate(Citizen):
def role_duties(self):
return "Candidate duties"

class ElectedOfficial(Voter, Candidate):
def role_duties(self):
return "Elected official duties"


class Activist(Citizen):
def role_duties(self):
return "Activist duties"
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

```
class ActivistPolitician(Activist, Politician):
def role_duties(self):
return "Activist politician duties"

class Worker(Citizen):
def role_duties(self):
return "Worker duties"

class WorkingPolitician(Worker, Politician):
def role_duties(self):
return "Working politician duties"

def mro_sequence(cls):
#..... YOUR CODE STARTS HERE .....
return [c.__name__ for c in cls.mro()]


#..... YOUR CODE ENDS HERE .....

class_name = input()

cls = globals()[class_name]

# Print the MRO sequence
print(mro_sequence(cls))
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

['Mayor', 'Politician', 'PublicServant', 'Citizen', 'object']

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

['ElectedOfficial', 'Voter', 'Candidate', 'Citizen', 'object']

Compilation Status: Passed

Execution Time:

0.012s

51. Design a Python program that uses multilevel inheritance to model the administrative divisions of Tamil Nadu. Tamil Nadu is divided into districts, and each district is divided into cities. All divisions have a name and a population. However, a district also has a number of cities, and Tamil Nadu has a number of districts.

Create classes to represent Tamil Nadu, a district, and a city, and use multilevel inheritance to avoid code duplication. Write a method in each class to print the details of the division.

Sample Input:

Coimbatore, 1064000Coimbatore District, 3472578, 2Tamil Nadu, 77841267, 38

Sample Output:

City: Coimbatore, Population: 1064000District: Coimbatore District, Population: 3472578, Number of Cities: 2State: Tamil Nadu, Population: 77841267, Number of Districts: 38

Explanation:

Multilevel inheritance is used to model the hierarchical relationship between a state, its districts, and the cities within those districts.

The City class is the base level in this hierarchy. It has properties like name and population.

The District class inherits from the City class, making it the next level in the hierarchy. It has all the properties of a City, plus an additional property: the number of cities. When we create a District object and call its print_details() method, it prints the details specific to that district, including the number of cities.

The TamilNadu class inherits from the District class, making it the top level in the hierarchy. It has all the properties of a District, plus an additional property: the number of districts. When we create a TamilNadu object and call its print_details() method, it prints the details specific to the state, including the number of districts.

ADITYA KUMAR JHA (adityajha3759@gmail.com)

This is an example of multilevel inheritance, where a derived class inherits from a base class, which in turn inherits from another base class. In this case, TamilNadu a derived class that inherits from District, which is a derived class that inherits from City.

Constraints:

The name of the city, district, or state is a string and can contain any printable ASCII characters.

The population of the city, district, or state is an integer and can be any positive integer.

The number of cities in a district is an integer and can be any positive integer.

The number of districts in Tamil Nadu is an integer and can be any positive integer.

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
class City():
def __init__(self, name, population):
#..... YOUR CODE STARTS HERE .....
self.name=name
self.population=population


#..... YOUR CODE ENDS HERE .....

def print_details(self):
#..... YOUR CODE STARTS HERE .....
print(f"City: {self.name}, Population: {self.population}")


#..... YOUR CODE ENDS HERE .....


class District(City):
def __init__(self, name, population, num_cities):
#..... YOUR CODE STARTS HERE .....
super().__init__(name,population)
self.num_cities=num_cities
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

```
#..... YOUR CODE ENDS HERE .....

def print_details(self):
#..... YOUR CODE STARTS HERE .....
print(f"District: {self.name}, Population: {self.population}, Number of Cities:
{self.num_cities}")


#..... YOUR CODE ENDS HERE .....


class TamilNadu(District):
def __init__(self, name, population, num_districts):
#..... YOUR CODE STARTS HERE .....
self.name=name
self.population=population
self.num_districts=num_districts


#..... YOUR CODE ENDS HERE .....

def print_details(self):
#..... YOUR CODE STARTS HERE .....
print(f"State: {self.name}, Population: {self.population}, Number of Districts:
{self.num_districts}")


#..... YOUR CODE ENDS HERE .....

def clean_input(value):
return str(value.strip())

def main():
city_name, city_population = map(clean_input, input().strip().split(","))
city = City(city_name, int(city_population))

district_name, district_population, num_cities = map(clean_input,
input().strip().split(","))
district = District(district_name, int(district_population), int(num_cities))

state_name, state_population, num_districts = map(clean_input,
input().strip().split(","))
state = TamilNadu(state_name, int(state_population), int(num_districts))

city.print_details()
district.print_details()
state.print_details()


if __name__ == "__main__":
main()
```

ADITYA KUMAR JHA (adityajha3793011@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

City: Chennai, Population: 4681087
District: Chennai District, Population: 7100000, Number of Cities: 1
State: Tamil Nadu, Population: 77841267, Number of Districts: 38

Compilation Status: Passed

Execution Time:

0.013s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

City: Coimbatore, Population: 2154944
District: Coimbatore District, Population: 3472578, Number of Cities: 1
State: Tamil Nadu, Population: 77841267, Number of Districts: 38

Compilation Status: Passed

Execution Time:

0.016s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

52. Design a Python program for a bookstore that uses classes and inheritance. The bookstore sells two types of items: books and magazines. Both books and magazines have a title and a price. However, a book also has an author, while a magazine has an issue number. Create classes to represent both items and use inheritance to avoid code duplication.

Write a method in each class to print the details of the book or magazine. Also, write a method in each class to change the price of the book or magazine.

Sample Input:

The Great Gatsby, F. Scott Fitzgerald, 15.99Time, 5.99, 2312.994.99

Sample Output:

Book: The Great Gatsby by F. Scott Fitzgerald, priced at $15.99Magazine: Time, Issue No. 23, priced at $5.99Book: The Great Gatsby by F. Scott Fitzgerald, priced at $12.99Magazine: Time, Issue No. 23, priced at $4.99

Explanation:

In the sample output, each line represents the details of an item in the bookstore:

The first line is the details of a book. It's "The Great Gatsby" by F. Scott Fitzgerald, priced at $15.99.

The second line is the details of a magazine. It's "Time", Issue No. 23, priced at $5.99.

The third line is the updated details of the book after changing the price. The price of "The Great Gatsby" has been updated to $12.99.

The fourth line is the updated details of the magazine after changing the price. The price of "Time", Issue No. 23, has been updated to $4.99.

These details are printed by calling the print_details() method of each item object. The method is defined in each class and prints the details specific to that item.

This is an example of single level inheritance, where the Book and Magazine classes inherit from a common base class (Item). The base class defines properties and methods common to all items (like name and price), while the derived classes add properties and methods specific to books and magazines (like author and issue number).

The change_price() method in each class demonstrates the concept of reusability in object-oriented programming. Instead of writing separate functions to change the price of a book and a magazine, we write a single method in the base class (Item) that both Book and Magazine inherit. This allows us to change the price of any item, regardless of whether it's a book or a magazine.

Constraints:

The title of the book or magazine is a string and can contain any printable ASCII characters.

The author of the book is a string and can contain any printable ASCII characters.

The issue number of the magazine is an integer and can be any positive integer.

The price of the book or magazine is a float and can be any positive number.

ADITYA KUMAR JHA (adityajha03759@gmail.com)

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
class Item:
def __init__(self, title, price):
#..... YOUR CODE STARTS HERE .....
self.title=title
self.price=price


#..... YOUR CODE ENDS HERE .....

def change_price(self, new_price):
#..... YOUR CODE STARTS HERE .....
self.new_price=new_price
self.price=self.new_price


#..... YOUR CODE ENDS HERE .....

def print_details(self):
#..... YOUR CODE STARTS HERE .....
print(f"Item: {self.title}, priced at ${self.price}")


#..... YOUR CODE ENDS HERE .....


class Book(Item):
def __init__(self, title, author, price):
#..... YOUR CODE STARTS HERE .....
super().__init__(title,price)
self.author=author


#..... YOUR CODE ENDS HERE .....

def print_details(self):
#..... YOUR CODE STARTS HERE .....
print(f"Book: {self.title} by {self.author}, priced at ${self.price}")


#..... YOUR CODE ENDS HERE .....


class Magazine(Item):
def __init__(self, title, price, issue):
#..... YOUR CODE STARTS HERE .....
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

```
super().__init__(title,price)
self.issue=issue


#..... YOUR CODE ENDS HERE .....

def print_details(self):
#..... YOUR CODE STARTS HERE .....
print(f"Magazine: {self.title}, Issue No. {self.issue}, priced at ${self.price}")


#..... YOUR CODE ENDS HERE .....


def clean_input(ip):
return str(ip.strip())

book_title, book_author, book_price = map(clean_input, input().strip().split(","))
mz_title, mz_price, mz_issue = map(clean_input, input().strip().split(","))

book1 = Book(book_title, book_author, book_price)
magazine1 = Magazine(mz_title, mz_price, mz_issue)

# Printing initial details
book1.print_details()
magazine1.print_details()

# Changing prices
book1.change_price(input())
magazine1.change_price(input())

# Printing updated details
book1.print_details()
magazine1.print_details()
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Book: To Kill a Mockingbird by Harper Lee, priced at $10.99
Magazine: National Geographic, Issue No. 144, priced at $6.99
Book: To Kill a Mockingbird by Harper Lee, priced at $9.99
Magazine: National Geographic, Issue No. 144, priced at $5.99
```

Compilation Status: Passed

Execution Time:

0.017s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Book: 1984 by George Orwell, priced at $8.99
Magazine: Forbes, Issue No. 102, priced at $7.99
Book: 1984 by George Orwell, priced at $7.99
Magazine: Forbes, Issue No. 102, priced at $6.99

Compilation Status: Passed

Execution Time:

0.011s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

53. Design a Python program for an e-commerce platform that sells products. Each product on the platform has a name, price, and seller. There are two types of products: new and used. Used products have an additional attribute: condition.

Create classes to represent a Product, NewProduct, and UsedProduct. Use inheritance and the super() function to avoid code duplication. Write a method in each class to print the details of the product.

Sample Input:

Laptop, 1200.99, ElectronicsHubSmartphone, 450.50, TechResell, Good

Sample Output:

New Product: Laptop, Price: $1200.99, Seller: ElectronicsHubUsed Product: Smartphone, Price: $450.50, Seller: TechResell, Condition: Good

Explanation:

In the sample output, each line represents the details of a product:

The first line is the details of a new product. It's a Laptop with a price of $1200.99 sold by ElectronicsHub.

The second line is the details of a used product. It's a Smartphone with a price of $450.50 sold by TechResell and its condition is Good.

These details are printed by calling the print_details() method of each product obje
The method is defined in each class and prints the details specific to that product.

This is an example of single level inheritance, where the NewProduct and UsedProduct classes inherit from a common base class (Product). The base class defines properties and methods common to all products (like name, price, and seller), while the derived classes add properties and methods specific to new and used products. The super() function is used to call a method in the parent class from the child class.

Constraints:

The name of the product and the seller is a string and can contain any printable ASCII characters.

The price of the product is a float and can be any positive number.

The condition of the used product is a string and can contain any printable ASCII characters.

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
class Product:
def __init__(self, name, price, seller):
#..... YOUR CODE STARTS HERE .....
self.name=name
self.price=price
self.seller=seller


#..... YOUR CODE ENDS HERE .....

def print_details(self):
#..... YOUR CODE STARTS HERE .....
pass


#..... YOUR CODE ENDS HERE .....

class NewProduct(Product):
```

```
def print_details(self):
#..... YOUR CODE STARTS HERE .....
return(f"New Product: {self.name}, Price: ${self.price}, Seller: {self.seller}")


#..... YOUR CODE ENDS HERE .....

class UsedProduct(Product):
def __init__(self, name, price, seller, condition):
#..... YOUR CODE STARTS HERE .....
super().__init__(name,price,seller)
self.condition=condition


#..... YOUR CODE ENDS HERE .....

def print_details(self):
#..... YOUR CODE STARTS HERE .....
return(f"Used Product: {self.name}, Price: ${self.price}, Seller: {self.seller}, Condition:
{self.condition}")


#..... YOUR CODE ENDS HERE .....

# Get input from the user
product1_info = input().split(", ")
product2_info = input().split(", ")

# Create product objects
product1 = NewProduct(*product1_info)
if len(product2_info) == 4:
product2 = UsedProduct(*product2_info)
else:
product2 = None

# Print details
print(product1.print_details())
if product2:
print(product2.print_details())
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

New Product: Laptop, Price: $1200.99, Seller: ElectronicsHub
Used Product: Smartphone, Price: $450.50, Seller: TechResell, Condition: Good

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

New Product: Camera, Price: $699.99, Seller: CameraStore
Used Product: Tablet, Price: $299.99, Seller: Gadgets4Less, Condition: Fair

Compilation Status: Passed

Execution Time:

0.011s

ADITYA KUMAR JHA (adityajha37591@gmail.com)

54. You are developing a Python program for managing various shapes in a geometry application. The program should support different types of shapes, such as circles, rectangles, and triangles. Each shape has its own unique attributes and methods.

Write a Python script that demonstrates method overloading in the context of different shape classes. Your program should accomplish the following tasks:

Define a base class called Shape.

Implement different subclasses for various shapes, such as Circle, Rectangle, and Triangle.

Each shape class should have methods to calculate its area and perimeter.

Demonstrate method overloading by implementing different versions of the area and perimeter methods for each shape class.

Ensure the following operations can be performed:

Calculate the area and perimeter of a circle.

Calculate the area and perimeter of a rectangle.

Calculate the area and perimeter of a triangle.

Your program should continuously prompt the user for shape inputs until the "quit" command is entered.

Provide appropriate error handling for invalid inputs. Display a message in the following format for invalid inputs: "Invalid input. Please enter a valid shape (circle/ rectangle/triangle)."

Sample Input:

circle5rectangle4 6triangle3 4 5quit

Sample Output:

Area of the circle: 78.54Perimeter of the circle: 31.42Area of the rectangle: 24.0Perimeter of the rectangle: 20.0Area of the triangle: 6.0Perimeter of the triangle: 12.0Quitting the program.

Explanation:

The user selects a shape and provides the necessary dimensions.

The program calculates the area and perimeter based on the provided inputs and shape type.

The process continues until the user enters "quit." If the user enters an invalid shape, an appropriate error message is displayed.

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
import math

class Shape:
#..... YOUR CODE STARTS HERE .....
def area(self):
pass

def perimeter(self):
pass
```

```
#..... YOUR CODE ENDS HERE .....

class Circle(Shape):
#..... YOUR CODE STARTS HERE .....

def area(self,radius):
self.radius=radius
return math.pi*self.radius*self.radius

def perimeter(self,radius):
self.radius=radius
return (2*math.pi*self.radius)


#..... YOUR CODE ENDS HERE .....

class Rectangle(Shape):
#..... YOUR CODE STARTS HERE .....

def area(self,lenght,breath):
self.lenght=lenght
self.breath=breath
return self.lenght*self.breath

def perimeter(self,lenght,breath):
self.lenght=lenght
self.breath=breath
return (2*(self.lenght+self.breath))



#..... YOUR CODE ENDS HERE .....

class Triangle(Shape):
#..... YOUR CODE STARTS HERE .....
def perimeter(self,a,b,c):
self.a=a
self.b=b
self.c=c
return self.a+self.b+self.c

def area(self,a,b,c):
self.a=a
self.b=b
self.c=c
s= (self.a+self.b+self.c)/2
return (math.sqrt(s*(s-self.a)*(s-self.b)*(s-self.c)))


#..... YOUR CODE ENDS HERE .....

def main():
shape_classes = {'circle': Circle(), 'rectangle': Rectangle(), 'triangle': Triangle()}
while True:
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

```
shape = input().lower()
if shape == 'quit':
print("Quitting the program.")
break
elif shape not in shape_classes:
print("Invalid input. Please enter a valid shape (circle/rectangle/triangle).")
continue

dimensions = list(map(float, input().split()))
print("Area of the {}: {:.2f}".format(shape, shape_classes[shape].area(*dimensions)))
print("Perimeter of the {}: {:.2f}".format(shape,
shape_classes[shape].perimeter(*dimensions)))

if __name__ == "__main__":
main()
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Area of the circle: 78.54
Perimeter of the circle: 31.42
Area of the rectangle: 24.00
Perimeter of the rectangle: 20.00
Area of the triangle: 6.00
Perimeter of the triangle: 12.00
Quitting the program.
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Area of the circle: 153.94
Perimeter of the circle: 43.98
Area of the rectangle: 15.00
Perimeter of the rectangle: 16.00
Area of the triangle: 24.00
Perimeter of the triangle: 24.00
Quitting the program.

Compilation Status: Passed

Execution Time:

0.012s

55. You are tasked with creating a Python program to manage the borrowing of resources from a library. The library offers both books and DVDs. Write a Python script that allows users to borrow items from the library. Faculty members can only borrow DVDs, while students can borrow both books and DVDs.

Here are the operations the program should support:

Allow a user to borrow an item from the library.

Display a message indicating whether the borrowing was successful or not, based on the user's status and the type of item they are trying to borrow.

Quit the program.

Ensure the following message formats:

For borrowing a book: Book "{book_title}" successfully borrowed by {self.name}.

For borrowing a DVD by a faculty member: DVD "{title}" successfully borrowed by {self.name}.

For informing a faculty member that they can only borrow DVDs: Faculty members can only borrow DVDs.

Please enter the following details for borrowing:

Name of the borrower

Status of the borrower (faculty/student)

Title of the item to borrow

Sample Input:

AlicefacultyThe Great Gatsbyquit

ADITYA KUMAR JHA (adityaha37591@gmail.com)

Sample Output:

Faculty members can only borrow DVDs.Quitting the program.

Explanation:

Alice, a faculty member, attempts to borrow "The Great Gatsby", which is a book. However, the program informs her that faculty members can only borrow DVDs. Then, the program quits as Alice does not attempt any further operations.

Please ensure to include "DVD" in the title if you want to borrow a DVD, as shown in the sample input.

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
class User:
def __init__(self, name, user_type):
#..... YOUR CODE STARTS HERE .....
self.name=name
self.user_type=user_type


#..... YOUR CODE ENDS HERE .....

def borrow_book(self, book_title):
#..... YOUR CODE STARTS HERE .....
self.book_title=book_title
return f'Book \"{self.book_title}\" successfully borrowed by {self.name}.'

#..... YOUR CODE ENDS HERE .....

class Faculty(User):
def __init__(self, name):
#..... YOUR CODE STARTS HERE .....
self.name=name


#..... YOUR CODE ENDS HERE .....

def borrow_book(self, title):
#..... YOUR CODE STARTS HERE .....
self.title=title
return f'DVD \"{self.title}\" successfully borrowed by {self.name}.'
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

```
#..... YOUR CODE ENDS HERE .....

def main():
while True:
name = input()
user_type = input().lower()
if user_type == "faculty":
user = Faculty(name)
else:
user = User(name, user_type)

item_title = input()
if user_type == "faculty":
if "DVD" in item_title:
print(user.borrow_book(item_title))
else:
print("Faculty members can only borrow DVDs.")
else:
print(user.borrow_book(item_title))

choice = input().lower()
if choice != "yes":
print("Quitting the program.")
break

if __name__ == "__main__":
main()
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Faculty members can only borrow DVDs.
Quitting the program.

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Book "The Lord of the Rings: The Fellowship of the Ring" successfully borrowed by David.
Quitting the program.

Compilation Status: Passed

Execution Time:

0.011s

56. You're developing a Python program for a virtual pet simulation game. The game features various types of pets, each with its own unique abilities and needs. You need to implement a system to manage different types of pets and their actions. The program should support pets such as cats, dogs, and birds.

Write a Python script that demonstrates method overloading in the context of different pet classes. Your program should accomplish the following tasks:

Define a base class called Pet.

Implement different subclasses for various types of pets, such as Cat, Dog, and Bird.

Each pet class should have methods to perform actions such as eating, sleeping, and making sounds.

Demonstrate method overloading by implementing different versions of the eat(), sleep(), and make_sound() methods for each pet class.

Additionally, integrate some unique behaviors for each type of pet:

Cats should be able to purr when making a sound.

Dogs should be able to wag their tails when making a sound.

Birds should be able to chirp when making a sound.

Ensure the following operations can be performed:

Feed a pet and display a message indicating it's eating.

Put a pet to sleep and display a message indicating it's sleeping.

Prompt a pet to make a sound and display the appropriate message based on its type.

ADITYA KUMAR JHA (adityajha8791@gmail.com)

Your program should continuously prompt the user for pet actions until the "quit" command is entered.

Provide appropriate error handling for invalid inputs. Display a message in the following format for invalid inputs: "Invalid input. Please enter a valid action (feed/sleep/make_sound/quit)."

Sample Input:

cateatdogmake_soundbirdsleepquit

Sample Output:

Cat is eating.Dog is wagging its tail.Bird is sleeping.Quitting the program.

Explanation:

The user selects a pet and an action to perform.

The program executes the action based on the selected pet type and displays an appropriate message.

The process continues until the user enters "quit." If the user enters an invalid action, an appropriate error message is displayed.

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Completion Status: Completed

Concepts Included:

gu - inheritance and exception handling: unit 2

Language Used: PYTHON 3

Source Code:

```
class Pet:
#..... YOUR CODE STARTS HERE .....

def eat(self):
pass
def sleep(self):
pass
def make_sound(self):
pass

#..... YOUR CODE ENDS HERE .....


class Cat(Pet):
#..... YOUR CODE STARTS HERE .....
def eat(self):
```

```
print("Cat is eating.")
def sleep(self):
print("Cat is sleeping.")
def make_sound(self):
print("Cat is purring.")


#..... YOUR CODE ENDS HERE .....

class Dog(Pet):
#..... YOUR CODE STARTS HERE .....
def eat(self):
print("Dog is eating.")
def sleep(self):
print("Dog is sleeping.")
def make_sound(self):
print("Dog is wagging its tail.")


#..... YOUR CODE ENDS HERE .....

class Bird(Pet):
#..... YOUR CODE STARTS HERE .....
def eat(self):
print("Bird is eating.")
def sleep(self):
print("Bird is sleeping.")
def make_sound(self):
print("Bird is chirping.")


#..... YOUR CODE ENDS HERE .....

def main():
while True:
pet_type = input("").lower()
if pet_type == 'quit':
print("Quitting the program.")
break
elif pet_type not in ['cat', 'dog', 'bird']:
print("Invalid input. Please enter a valid pet type (cat/dog/bird).")
continue

pet_action = input("").lower()
if pet_action == 'quit':
print("Quitting the program.")
break
elif pet_action not in ['eat', 'sleep', 'make_sound']:
print("Invalid input. Please enter a valid action (eat/sleep/make_sound/quit).")
continue

if pet_type == 'cat':
cat = Cat()
if pet_action == 'eat':
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

```
cat.eat()  # Output: "Cat is eating."
elif pet_action == 'sleep':
cat.sleep()  # Output: "Cat is sleeping."
elif pet_action == 'make_sound':
cat.make_sound()  # Output: "Cat is purring."
elif pet_type == 'dog':
dog = Dog()
if pet_action == 'eat':
dog.eat()  # Output: "Dog is eating."
elif pet_action == 'sleep':
dog.sleep()  # Output: "Dog is sleeping."
elif pet_action == 'make_sound':
dog.make_sound()  # Output: "Dog is wagging its tail."
elif pet_type == 'bird':
bird = Bird()
if pet_action == 'eat':
bird.eat()  # Output: "Bird is eating."
elif pet_action == 'sleep':
bird.sleep()  # Output: "Bird is sleeping."
elif pet_action == 'make_sound':
bird.make_sound()  # Output: "Bird is chirping."

if __name__ == "__main__":
main()
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
Cat is eating.
Dog is wagging its tail.
Bird is sleeping.
Quitting the program.
```

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Dog is eating.
Bird is sleeping.
Quitting the program.

Compilation Status: Passed

Execution Time:

0.011s

57. You are managing an e-commerce website and you have a list of products. Each product is represented as a dictionary with two keys: 'name' and 'category'. You want to create a list of all products that belong to the category 'Electronics'. Write a Python function electronics_products(products) that takes a list of products and returns a list of tuples where each tuple contains the product name and the category. Use a tuple comprehension to create the list.

ADITYA KUMAR JHA (adityajha7911@gmail.com)

Input:

products: a list of dictionaries. Each dictionary represents a product and has two keys: 'name' (a string) and 'category' (a string).

Output:

The function should return a list of tuples where each tuple contains a product name and the category.

Sample Input:

[{'name': 'Product 1', 'category': 'Books'}, {'name': 'Product 2', 'category': 'Electronics'}, {'name': 'Product 3', 'category': 'Electronics'}]

Sample Output:

[('Product 2', 'Electronics'), ('Product 3', 'Electronics')]

Explanation:

The tuple comprehension generates a new list containing tuples of the names and categories of all products that belong to the 'Electronics' category. It does this by

iterating over each product in the products list and checking if the product's catego is 'Electronics'. If it is, a tuple containing the product's name and category is added the new list. This demonstrates how tuple comprehensions can be used to create new lists based on existing lists in Python. In the context of managing an e-commerce website, this can be useful to quickly find all products that belong to a certain category.

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

ADITYA KUMAR JHA (adityajha37591@gmail.com)

```
import json
from io import StringIO
import re

def electronics_products(products):
#..... YOUR CODE STARTS HERE .....

return [(product['name'], product['category']) for product in products if
product['category'] == 'Electronics']

#..... YOUR CODE ENDS HERE .....

def convert_to_list_of_dicts(data):
data_without_brackets = re.sub(r'[\[\]]', '', data).split('},')

lst = []

data = [d + '}' if '}' not in d else d for d in data_without_brackets]

for d in data:
if ('{' in d and '}' in d):
d = d.strip().replace("'", '"')
lst.append(json.load(StringIO(d)))

return lst

if __name__ == "__main__":
products = convert_to_list_of_dicts(input().strip())
print(electronics_products(products))
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

[('Product 2', 'Electronics'), ('Product 3', 'Electronics')]

Compilation Status: Passed

Execution Time:

0.019s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

[('iPhone 13', 'Electronics'), ('Samsung Galaxy S21', 'Electronics')]

Compilation Status: Passed

Execution Time:

0.018s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

58. You are managing an e-commerce website and you have a list of products. Each product is represented as a dictionary with three keys: 'name', 'category', and 'brand'. You want to create a set of all unique brands present in your product list. Write a Python function unique_brands(products) that takes a list of products and returns a set of unique brands. Use a set comprehension to create the set.

Input:

products: a list of dictionaries. Each dictionary represents a product and has three keys: 'name' (a string), 'category' (a string), and 'brand' (a string).

Output:

The function should return a set of unique brands.

Sample Input:

[{'name': 'Product 1', 'category': 'Books', 'brand': 'Brand A'}, {'name': 'Product 2', 'category': 'Electronics', 'brand': 'Brand B'}, {'name': 'Product 3', 'category': 'Electronics', 'brand': 'Brand B'}]

Sample Output:

['Brand A', 'Brand B']

Explanation:

The set comprehension generates a new set containing unique brands of all products. It does this by iterating over each product in the products list and adding the product's brand to the new set. Since sets in Python only contain unique elements, any duplicate brands are automatically removed. This demonstrates how set comprehensions can be used to create new sets based on existing lists in Python. In the context of managing an e-commerce website, this can be useful to quickly find all unique product brands.",

ADITYA KUMAR JHA (adityajha3759@gmail.com)

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
import json
from io import StringIO
import re

def unique_brands(products):
#..... YOUR CODE STARTS HERE .....

return sorted({product['brand'] for product in products})

#..... YOUR CODE ENDS HERE .....

def convert_to_list_of_dicts(data):
data_without_brackets = re.sub(r'[\[\]]', '', data).split('},')

lst = []
```

```
data = [d + '}' if '}' not in d else d for d in data_without_brackets]

for d in data:
if ('{' in d and '}' in d):
d = d.strip().replace("'", '"')
lst.append(json.load(StringIO(d)))

return lst

if __name__ == "__main__":
products = convert_to_list_of_dicts(input().strip())
print(sorted(unique_brands(products)))
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

['Brand A', 'Brand B']

Compilation Status: Passed

Execution Time:

0.019s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

['Apple', 'Bloomsbury', 'Samsung']

Compilation Status: Passed

Execution Time:

0.019s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

59. You are managing an e-commerce website and you have a list of products. Each product is represented as a dictionary with three keys: 'name', 'category', and 'stock'. The 'stock' key represents the number of units of the product available in your inventory. You want to create a dictionary where the keys are the product names and the values are the stock count. Write a Python function product_stock(products) that takes a list of products and returns a dictionary of product names and their stock count. Use a dictionary comprehension to create the dictionary.

Input:

products: a list of dictionaries. Each dictionary represents a product and has three keys: 'name' (a string), 'category' (a string), and 'stock' (an integer).

Output:

The function should return a dictionary where the keys are product names and the values are their stock count.

Sample Input:

[{'name': 'Product 1', 'category': 'Books', 'stock': 20}, {'name': 'Product 2', 'category': 'Electronics', 'stock': 15}, {'name': 'Product 3', 'category': 'Electronics', 'stock': 30}]

Sample Output:

{'Product 1': 20, 'Product 2': 15, 'Product 3': 30}

Explanation:

The dictionary comprehension generates a new dictionary containing product names as keys and their stock count as values. It does this by iterating over each product in the products list and adding an entry to the new dictionary where the key is the product's name and the value is the product's stock count. This demonstrates how dictionary comprehensions can be used to create new dictionaries based on existing lists in Python. In the context of managing an e-commerce website, this can be useful to quickly find the stock count of a product given its name.

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Source Code:

```
def product_stock(products):
#..... YOUR CODE STARTS HERE .....

return {product['name']: product['stock'] for product in products}

#..... YOUR CODE ENDS HERE .....

products_input = eval(input())

result = product_stock(products_input)
print(result)
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

{'iPhone 13': 10, 'Harry Potter': 20, 'Samsung Galaxy S21': 15}

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

{'MacBook Pro': 5, 'The Alchemist': 30, 'Dell XPS': 7}

Compilation Status: Passed

Execution Time:

0.011s

ADITYA KUMAR JHA (adityajha37591@gmail.com)

60. You are managing an e-commerce website and you have a list of products. Each product is represented as a dictionary with three keys: 'name', 'category', and 'price'. You want to calculate the total price of all products in a specific category. Write a Python function total_price(products, category) that takes a list of products and a category, and returns the total price of all products in that category. Use the sum function from the built-in Python math module to calculate the total price.

Input:

products: a list of dictionaries. Each dictionary represents a product and has three keys: 'name' (a string), 'category' (a string), and 'price' (a float).

category: a string representing the category of products for which the total price should be calculated.

Output:

The function should return a float representing the total price of all products in the specified category.

Sample Input:

[{'name': 'Product 1', 'category': 'Books', 'price': 19.99}, {'name': 'Product 2', 'category': 'Electronics', 'price': 299.99}, {'name': 'Product 3', 'category': 'Electronics', 'price': 499.99}], 'Electronics'

Sample Output:

799.98

Explanation:

The generator expression generates a sequence of prices for all products in the specified category. It does this by iterating over each product in the products list and checking if the product's category matches the specified category. If it does, the product's price is added to the sequence. The fsum function from the math module then calculates the sum of this sequence, which is the total price of all products in the specified category. This demonstrates how modules and packages can be used in Python to provide additional functionality, such as mathematical operations. In the context of managing an e-commerce website, this can be useful to quickly calculate the total price of all products in a certain category.

Completion Status: Completed

Concepts Included:

ADITYA KUMAR JHA (adityajha37591@gmail.com)

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
import json
from io import StringIO
import re
from math import fsum

def total_price(products, category):
#..... YOUR CODE STARTS HERE .....

return fsum(product['price'] for product in products if product['category'] == category)


#..... YOUR CODE ENDS HERE .....

def convert_to_list_of_dicts(data):
data_without_brackets = re.sub(r'[\[\]]', '', data).split('},')

lst = []

data = [d + '}' if '}' not in d else d for d in data_without_brackets]

for d in data:
if ('{' in d and '}' in d):
d = d.strip().replace("'", '"')
lst.append(json.load(StringIO(d)))

return lst

if __name__ == "__main__":
ip = input().strip().replace("],", ']<DIVIDE>')
products, category = ip.split("<DIVIDE>")
products = convert_to_list_of_dicts(products)
category = category.replace("'", '').strip()

print(total_price(products, category))
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1799.98

Compilation Status: Passed

Execution Time:

0.02s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

2499.98

Compilation Status: Passed

Execution Time:

0.018s

ADITYA JHA (adityajha37591@gmail.com)

61. You are managing a bank's software system. Every time a customer makes a transaction, the transaction amount is recorded. You want to create a function that calculates the square root of the transaction amount. Write a Python function transaction_sqrt(transaction_amount) that takes a transaction amount and returns the square root of the transaction amount. Use the sqrt function from the built-in Python math module to calculate the square root.

Input:

transaction_amount: a float representing the transaction amount.

Output:

The function should return a float representing the square root of the transaction amount.

Sample Input:

100.0

Sample Output:

10.0

Explanation :

The sqrt function from the math module calculates the square root of a number. In the context of managing a bank's software system, this can be useful to perform various calculations related to the transaction amount.

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
from math import sqrt

def transaction_sqrt(transaction_amount):
#..... YOUR CODE STARTS HERE .....

return sqrt(transaction_amount)

#..... YOUR CODE ENDS HERE .....

transaction_amount = float(input())
result = transaction_sqrt(transaction_amount)
print(result)
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

10.0

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

7.0

Compilation Status: Passed

Execution Time:

0.012s

62. You are managing a bank's software system. You want to calculate the monthly interest for a given principal amount, rate of interest, and time period. Write a Python function calculate_interest(principal, rate, time) to calculate and return the monthly interest.

Input:

principal: a float representing the principal amount.

rate: a float representing the annual rate of interest in percentage.

time: an integer representing the time period in months.

Output:

The function should return a float representing the monthly interest.

Sample Input:

1000.0, 5, 12

Sample Output:

ADITYA KUMAR JHA (adityajha37591@gmail.com)

50.0

Explanation:

The calculate_interest function takes three parameters: principal, rate, and time, representing the principal amount, annual rate of interest (in percentage), and time period (in months) respectively. Inside the function, it first converts the annual rate to a monthly rate by dividing it by 100 and then by 12 (since there are 12 months in a year).

Then, it calculates the monthly interest by multiplying the principal amount with the monthly rate and the time period.

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
def calculate_interest(principal, rate, time):
#..... YOUR CODE STARTS HERE .....

monthly_rate = rate / (100 * 12)  # Convert annual rate to monthly rate
monthly_interest = principal * monthly_rate * time
return monthly_interest

#..... YOUR CODE ENDS HERE .....


if __name__ == "__main__":
principal, rate, time = list(map(float, input().strip().split(',')))

# Calculate and print monthly interest
print(calculate_interest(principal, rate, time))
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

50.0

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

280.0

Compilation Status: Passed

Execution Time:

0.011s

ADITYA JHA (adityajha37591l@gmail.com)

63. You are working on a Python project that involves performing complex mathematical operations. For this project, you need to calculate the square root of a list of transaction amounts. So write a Python function calculate_sqrt(transaction_amounts) that takes a list of transaction amounts and returns a list of their square roots.

Sample Input:

[100.0, 200.0, 300.0, 400.0]

Sample Output:

[10.0, 14.142135623730951, 17.320508075688775, 20.0]

Explanation:

The calculate_sqrt function uses the sqrt function from the math module to calculate the square root of each transaction amount. The square root of a number is a value that, when multiplied by itself, gives the original number. In this case, we're calculating the square root of each transaction amount in the list.

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
import math

def calculate_sqrt(transaction_amounts):
#..... YOUR CODE STARTS HERE .....

return [math.sqrt(amount) for amount in transaction_amounts]

#..... YOUR CODE ENDS HERE .....

if __name__ == "__main__":
transaction_amounts = eval(input().strip())
print(calculate_sqrt(transaction_amounts))
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

[10.0, 14.142135623730951, 17.320508075688775, 20.0]

Compilation Status: Passed

Execution Time:

0.011s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

[22.360679774997898, 24.49489742783178, 26.457513110645905, 28.284271247461902]

Compilation Status: Passed

Execution Time:

0.011s

64. You are a college professor and you have the scores of your students in a list. You want to grade the students based on their scores. Write a Python function grade_students(scores) that takes a list of scores and returns a list of grades. Use the map function to apply the grading scheme to all scores.

The grading scheme is as follows:

Score >= 90: 'A'

80 <= Score < 90: 'B'

70 <= Score < 80: 'C'

60 <= Score < 70: 'D'

Score < 60: 'F'

Input:

scores: a list of integers representing the scores of the students.

Output:

The function should return a list of strings representing the grades of the students.

Sample Input:

[95,55]

Sample Output:

['A','F']

Explanation:

The map function applies the get_grade function to every item in the scores list. The get_grade function takes a score and returns a grade based on the grading scheme. This demonstrates how the map function can be used to apply a function to every item in a list. In the context of grading students in a college, this can be useful to

ADITYA KUMAR JHA (adityajha37591@gmail.com)

grade all students based on their scores.

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
import json
from io import StringIO

def grade_students(scores):
def get_grade(score):
#..... YOUR CODE STARTS HERE .....

if score >= 90:
return 'A'
elif 80 <= score < 90:
return 'B'
elif 70 <= score < 80:
return 'C'
elif 60 <= score < 70:
return 'D'
else:
return 'F'

#..... YOUR CODE ENDS HERE .....

return list(map(get_grade, scores))

if __name__ == "__main__":
scores = json.load(StringIO(input().strip()))
grades = grade_students(scores)

print(grades)
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

['A', 'B', 'C', 'D', 'F']

Compilation Status: Passed

Execution Time:

0.018s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

['A', 'A', 'B', 'C', 'D']

Compilation Status: Passed

Execution Time:

0.03s

ADITYA JHA (adityajha37591@gmail.com)

65. In an e-commerce application, you have a list of n products. Each product has a unique ID and a price. You need to implement a generator function get_products_in_budget(budget, products) that yields products one by one, sorted by price, until the total price of yielded products exceeds the budget.

Input:

budget: an integer representing the total budget.

products: a list of tuples. Each tuple contains two elements - the first element is the product ID (a string), and the second element is the product price (an integer).

Output:

The function should yield tuples. Each tuple contains two elements - the first element is the product ID (a string), and the second element is the product price (an integer).

Constraints:

1 <= n <= 10^5

1 <= budget <= 10^9

1 <= price <= 10^6

Sample Input:

100[("p1", 30), ("p2", 50), ("p3", 40), ("p4", 60)]

Sample Output:

('p1', 30)('p3', 40)

Explanation:

The generator function get_products_in_budget yields products one by one, sorted by price, until the total price of yielded products exceeds the budget. In this case, the products "p1" and "p3" are yielded, with total price 30+40=70, which is the maximum possible total price not exceeding the budget of 100.

This question highlights the importance and performance benefits of generators. Generators allow us to generate as we go along, instead of holding everything in memory. In the context of an e-commerce application, this can be particularly useful when dealing with a large number of products, as it allows us to efficiently process each product one at a time, reducing memory usage.

ADITYA KUMAR JHA (adityajha3550@gmail.com)

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
import re

def get_products_in_budget(budget, products):
#..... YOUR CODE STARTS HERE .....

products.sort(key=lambda x: x[1])

total_price = 0

# Yield products one by one until total price exceeds budget
for product_id, price in products:
if total_price + price <= budget:
yield product_id, price
total_price += price
```

```
else:
break

#..... YOUR CODE ENDS HERE .....

def replace_non_alphanumeric(text, replacement=''):
return re.sub(r'[^a-zA-Z0-9,]', replacement, text)

def clean_input(value):
value = replace_non_alphanumeric(value).split(',')

id, price = list(map(lambda x: x.strip(), value))

return (id, int(price))

if __name__ == "__main__":
budget = int(input())
products = input()

products = list(map(clean_input, products.strip().replace('[', '').replace(']', '').split('),')))

for product in get_products_in_budget(budget, products):
print(product)
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

```
('p1', 30)
('p3', 40)
```

Compilation Status: Passed

Execution Time:

0.018s

TestCase2:

Input:

< hidden >

Expected Output:

ADITYA KUMAR JHA (adityajha37591@gmail.com)

< hidden >

Output:

('p1', 10)
('p2', 20)

Compilation Status: Passed

Execution Time:

0.017s

66. You are organizing a marriage function and you have a list of tasks to be done. Each task has a time duration. You want to keep track of how long each task takes. Write a Python decorator time_it that takes a function and prints the time taken by that function. Use this decorator to decorate the function do_task(task), which simulates doing the task by sleeping for the duration of the task.

The output of the do_task function should be in the format: f"{task[0]} took {round(end - start, 2)} seconds"

Input:

task: a tuple. The first element is the task name (a string), and the second element is the task duration in seconds (an integer).

Output:

The function should print the task name and the time taken by the task in the format: f"{task[0]} took {round(end - start, 2)} seconds"

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Sample Input:

Decorating the hall, 2

Sample Output:

Decorating the hall took 2.0 seconds

Explanation:

The decorator time_it is used to measure the time taken by the function do_task. The do_task function simulates doing a task by sleeping for the duration of the task. The decorator prints the task name and the time taken by the task. This demonstrates how decorators can be used to modify the behavior of a function, in this case, by adding timing functionality. In the context of organizing a marriage function, this can be useful to keep track of how long each task takes.

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
import time

def time_it(func):
#..... YOUR CODE STARTS HERE .....

def wrapper(task):
start_time = time.time()
result = func(task)
end_time = time.time()
duration = end_time - start_time
print(f"{task[0]} took {round(duration, 2)} seconds")
return result
return wrapper

#..... YOUR CODE ENDS HERE .....

@time_it
def do_task(task):
#..... YOUR CODE STARTS HERE .....

task_name, task_duration = task
time.sleep(task_duration)

#..... YOUR CODE ENDS HERE .....


if __name__ == "__main__":
task_name, duration = list(map(lambda x: x.strip(), input().strip().split(',')))
task = [task_name, int(duration)]

do_task(task)
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Decorating the hall took 2.0 seconds

Compilation Status: Passed

Execution Time:

0.012s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Preparing the food took 3.0 seconds

Compilation Status: Passed

Execution Time:

0.012s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

67. You are organizing a marriage function and you have a list of tasks to be done. Each task has a time duration. You want to keep track of the total time taken by all tasks. Write a Python class MarriageFunction with a class method do_task that simulates doing a task and keeps track of the total time taken by all tasks.

Input:

task: a tuple. The first element is the task name (a string), and the second element is the task duration in seconds (an integer).

Output:

The function should print the task name and the time taken by the task in the format: f"{task[0]} took {round(end - start, 2)} seconds

Sample Input:

[("Decorating the hall", 2)]

Sample Output:

Decorating the hall took 2.0 seconds

Explanation:

The class MarriageFunction has a class method do_task which simulates doing a task by sleeping for the duration of the task. The method adds the time taken by the task to the total_duration class variable and prints the task name and the time taken by the task. This demonstrates how class variables can be used to keep track of state across multiple method calls. In the context of organizing a marriage function, this can be useful to keep track of how long all tasks take in total.

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
import time
import re

class MarriageFunction:
total_duration = 0

@classmethod
def do_task(cls, task):
#..... YOUR CODE STARTS HERE .....

start_time = time.time()
time.sleep(task[1])
end_time = time.time()
duration = end_time - start_time
cls.total_duration += duration
print(f"{task[0]} took {round(duration, 2)} seconds")

#..... YOUR CODE ENDS HERE .....

def replace_non_alphanumeric(text, replacement=''):
return re.sub(r'[^a-zA-Z0-9, ]', replacement, text)

def clean_input(value):
value = replace_non_alphanumeric(value).split(',')

id, price = list(map(lambda x: x.strip(), value))
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

```
return (id, int(price))

if __name__ == "__main__":
tasks = input()

tasks = list(map(clean_input, tasks.strip().replace('[', '').replace(']', '').split('),')))

for task in tasks:
MarriageFunction.do_task(task)
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Decorating the hall took 2.0 seconds

Compilation Status: Passed

Execution Time:

0.018s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Preparing the food took 3.0 seconds

Compilation Status: Passed

Execution Time:

0.018s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

68. You are developing a student management system for a B.Tech

program. The system has a function get_student_grade(student_id: str) -> str that takes a student ID as input and returns the grade of that student.

The program has the following students enrolled:

Student ID: "BT202101", Grade: "A"

Student ID: "BT202102", Grade: "B"

However, not all students are enrolled in the program at all times. If a student is not enrolled, the function should raise a StudentNotFoundError with a message "Student not found".

Write a Python function proxy_get_student_grade(student_id: str) -> str that acts as a proxy to the get_student_grade(student_id: str) -> str function. The proxy function should handle any StudentNotFoundError that might be raised and return a default grade 'N/A' if the student is not found.

Constraints:

Student ID is a string.

You can't install any external libraries.

Sample Input:

BT202101

Sample Output:

A

Explanation:

In the sample outputs, the first grade is returned when the student is found in the program. The grade of the student is displayed in the output. This is done by catching the grade returned by the get_student_grade function in a try block.

This demonstrates how proxy functions work in Python - a proxy function acts as an interface to another function and can add additional behavior (like error handling) without changing the original function's code. In this case, the proxy function adds error handling to the get_student_grade function, allowing it to return a default grade when a student is not found. This is a key aspect of robust software design.

In Python, functions are first-class objects, which means they can be passed around and used as arguments just like any other object (string, int, float, list, etc.). Higher-order functions are a kind of function that takes one or more functions as arguments, returns a function, or both. This property allows us to create proxy functions like proxy_get_student_grade.

Completion Status: Completed

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
class StudentNotFoundError(Exception):
pass

def get_student_grade(student_id):

students = {
"BT202101": "A",
"BT202102": "B",
}
if student_id in students:
return students[student_id]
else:
raise StudentNotFoundError("Student not found")

def proxy_get_student_grade(student_id):
#..... YOUR CODE STARTS HERE .....

try:
return get_student_grade(student_id)
except StudentNotFoundError:
return 'N/A'

#..... YOUR CODE ENDS HERE .....

student_id = input()

print(proxy_get_student_grade(student_id))
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

A

Compilation Status: Passed

Execution Time:

0.017s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

B

Compilation Status: Passed

Execution Time:

0.011s

69. You are developing a calculator in Python. The calculator has four operations: add, subtract, multiply, and divide. Each operation is a function that takes two numbers and returns the result of the operation.

Write a Python function calculate(operation: Callable[[float, float], float], num1: float, num2: float) -> float that takes an operation function and two numbers as input and returns the result of the operation. The operation function is a higher-order function that can be any of the four operation functions.

Constraints:

num1 and num2 are floats.

You can't install any external libraries.

ADITYA KUMAR JHA (adityaa05911@gmail.com)

Sample Input:

add, 1.0, 2.0

Sample Output:

3.0

Explanation:

In the sample outputs, each number is the result of a calculation performed by the calculate function. The calculate function is a higher-order function, which means it accepts other functions as arguments and/or returns a function as its result. In this

case, calculate accepts an operation function as an argument and applies it to the two number arguments.

The operation functions (add, divide) are defined using lambda functions, which are small anonymous functions defined with the lambda keyword in Python. They can take any number of arguments but can only have one expression. In this case, each lambda function takes two arguments and returns the result of a specific arithmetic operation.

This demonstrates the power and flexibility of higher-order functions and lambda functions in Python. They allow us to write more modular and concise code by treating functions as first-class objects, meaning that functions can be passed around and used as arguments or return values, just like any other objects (strings, numbers, lists, etc.).

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
from typing import Callable

def calculate(operation: Callable[[float, float], float], num1: float, num2: float) -> float:
#..... YOUR CODE STARTS HERE .....

return operation(num1, num2)

#..... YOUR CODE ENDS HERE .....

def clean_input(value):
return str(value.strip())

add = lambda x, y: x + y
subtract = lambda x, y: x - y
multiply = lambda x, y: x * y
divide = lambda x, y: x / y if y != 0 else None


operation, num1, num2 = map(clean_input, input().strip().split(','))

func = None

if (operation == 'add'):
func = add

elif (operation == 'subtract'):
func = subtract
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

```
elif (operation == 'multiply'):
func = multiply

elif (operation == 'divide'):
func = divide

print(calculate(func, float(num1), float(num2)))
```

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

3.0

Compilation Status: Passed

Execution Time:

0.023s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

1.5

Compilation Status: Passed

Execution Time:

0.023s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

70. You are developing a student management system for a B.Tech program. The system has a list of students, where each student is represented as a dictionary with the following keys: 'id', 'name',

'course', and 'grade'.

Write a Python function iterate_students(students: List[Dict[str, Union[str, int]]]) -> None that takes a list of students as input and prints each student's details using a iterator. The details should be printed in the following format: "Student ID: {id}, Nam {name}, Course: {course}, Grade: {grade}".

Constraints:

The id, name, course are strings and grade is an integer.

You can't install any external libraries.

Sample Input:

[

{"id": "BT202101", "name": "John Doe", "course": "Computer Science", "grade": "A"},

{"id": "BT202102", "name": "Jane Doe", "course": "Electrical Engineering", "grade": "B"},

]

Sample Output:

Student ID: BT202101, Name: John Doe, Course: Computer Science, Grade: AStudent ID: BT202102, Name: Jane Doe, Course: Electrical Engineering, Grade: B

Explanation:

In the sample output, each line is the result of iterating over the list of students and printing each student's details. This is done using a for loop, which in Python, creates an iterator for the list and iterates over it.

Iterators in Python are objects that can be iterated (or looped) over. An object is called iterable if we can get an iterator from it. Most built-in containers in Python like: list, tuple, string, etc. are iterables. The iter() function (which in turn calls the iter() method) returns an iterator from them.

In this case, the list of students is an iterable and the for loop creates an iterator that iterates over each student in the list. For each student, the student's details are printed to the console. This demonstrates the use of iterators in Python to efficiently loop over an iterable object.

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
import sys
import json
from io import StringIO

def iterate_students(students):
#..... YOUR CODE STARTS HERE .....

for student in students:
print(f"Student ID: {student['id']}, Name: {student['name']}, Course: {student['course']},
Grade: {student['grade']}")

#..... YOUR CODE ENDS HERE .....

def clean_input(value):
value = value.strip()
if (len(value)):
if ('}'  not in value):
value += '}'

if ('{' in value and '}' in value):
return json.load(StringIO(value))
return False

if __name__ == "__main__":
students_list = ""

for line in sys.stdin:
line = line.strip()
students_list += line

students_list = list(map(clean_input, students_list.replace('[', '').replace(']', '').split('},')))
students_list = list(filter(lambda student: isinstance(student, dict), students_list))

iterate_students(students_list)
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Student ID: BT202101, Name: John Doe, Course: Computer Science, Grade: A
Student ID: BT202102, Name: Jane Doe, Course: Electrical Engineering, Grade: B

Compilation Status: Passed

Execution Time:

0.019s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

Student ID: BT202103, Name: Alice Smith, Course: Mechanical Engineering, Grade: C
Student ID: BT202104, Name: Bob Johnson, Course: Civil Engineering, Grade: D

Compilation Status: Passed

Execution Time:

0.02s

ADITYA KUMAR JHA (adityajha37591@gmail.com)

71. You are implementing a simple calculator. Write a Python class Calculator with static methods add, subtract, multiply, and divide that perform the respective operations.

Input:

num1: a float, the first number.

num2: a float, the second number.

Output:

The function should return the result of the operation.

Sample Input:

Calculator.add(5, 3)

Sample Output:

8

Explanation:

The class Calculator has static methods add, subtract, multiply, and divide which perform the respective operations. Static methods, marked with the @staticmethod decorator, don't take a self or cls parameter. This means you can't modify object state or class state within a static method. However, they're useful when you need to perform a utility function that doesn't modify the state of the object or class, like in this case where we're performing simple mathematical operations.

Completion Status: Completed

Concepts Included:

gu - functional programming: unit 3

Language Used: PYTHON 3

Source Code:

```
class Calculator:
@staticmethod
def add(num1, num2):
#..... YOUR CODE STARTS HERE .....

return num1+num2

#..... YOUR CODE ENDS HERE .....

@staticmethod
def subtract(num1, num2):
#..... YOUR CODE STARTS HERE .....

return num1-num2

#..... YOUR CODE ENDS HERE .....

@staticmethod
def multiply(num1, num2):
#..... YOUR CODE STARTS HERE .....

return num1*num2

#..... YOUR CODE ENDS HERE .....

@staticmethod
def divide(num1, num2):
#..... YOUR CODE STARTS HERE .....

return num1/num2

#..... YOUR CODE ENDS HERE .....

def call_func(func, args):
```

ADITYA KUMAR JHA (adityajha37591l@gmail.com)

```
value = None
num1, num2 = args

if (func == "add"):
value = Calculator.add(num1, num2)

elif (func == "subtract"):
value = Calculator.subtract(num1, num2)

elif (func == "multiply"):
value = Calculator.multiply(num1, num2)

elif (func == "divide"):
value = Calculator.divide(num1, num2)

return value


if __name__ == "__main__":
func_call = list(map(lambda x: str(x.strip()), input().strip().split(':')))
class_name, func = func_call

if (class_name == 'Calculator'):
parans_pos = func.index('(')

name = func[:parans_pos].strip()
args = tuple(map(lambda x: int(x.strip()), func[parans_pos+1:
len(func)-1].strip().split(',')))

print(call_func(name, args))
```

ADITYA KUMAR JHA (adityajha37591@gmail.com)

Compilation Details:

TestCase1:

Input:

< hidden >

Expected Output:

< hidden >

Output:

8

Compilation Status: Passed

Execution Time:

0.016s

TestCase2:

Input:

< hidden >

Expected Output:

< hidden >

Output:

2

Compilation Status: Passed

Execution Time:

0.016s

ADITYA KUMAR JHA (adityajha37591l@gmail.com)