

## HomeWork\_2 (02/Nov./2022)

### 1. Problem1\_A\_Loops.py

```
while True:
    try:
        input_list=list(input("input a list of integers").split(" "))
        print(input_list)
        int_list=[int(i) for i in input_list]
        print(int_list)
        break
    except:
        print("your input must be integer numbers")

#return the list of integers that accept the division by 5:
out_list=[i for i in int_list if i%5==0]

print("The integers that accept division by 5 =",out_list)
```

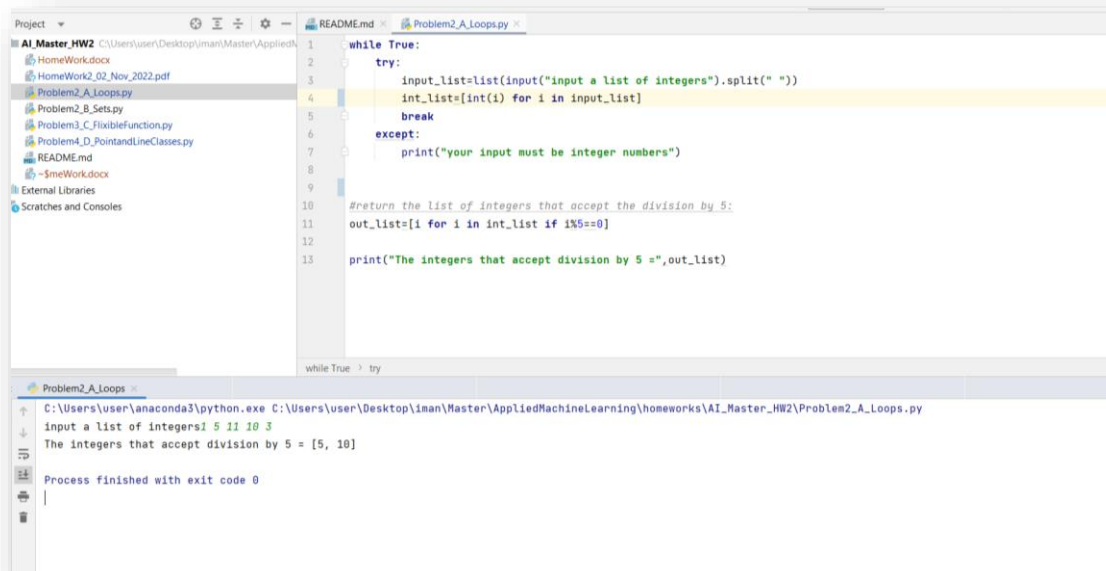


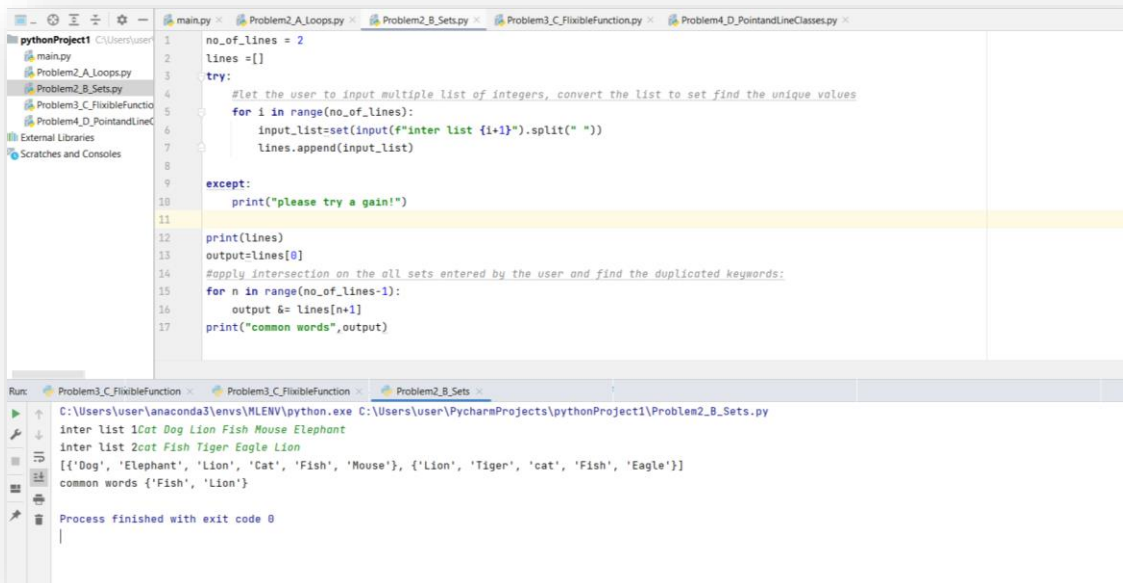
Figure 1: Output Sample of Problem1\_A

## 2. Problem2\_B\_Sets.py

```
no_of_lines = 2
lines = []
try:
    #let the user to input multiple list of integers, convert the list
    to set find the unique values
    for i in range(no_of_lines):
        input_list=set(input(f"inter list {i+1}").split(" "))
        lines.append(input_list)

except:
    print("please try a gain!")

print(lines)
output=lines[0]
#apply intersection on the all sets entered by the user and find the
duplicated keywords:
for n in range(no_of_lines-1):
    output &= lines[n+1]
print("common words",output)
```



The screenshot shows a Python IDE with a file explorer on the left and a console at the bottom. The file explorer lists several files: main.py, Problem2\_A\_Loops.py, Problem2\_B\_Sets.py (selected), Problem3\_C\_FlexibleFunction.py, and Problem4\_D\_PointandLineClasses.py. The console shows the execution of Problem2\_B\_Sets.py. The user input is "inter list 1Cat Dog Lion Fish Mouse Elephant" and "inter list 2cat Fish Tiger Eagle Lion". The output is a list of sets: [{"Dog", "Elephant", "Lion", "Cat", "Fish", "Mouse"}, {"Lion", "Tiger", "cat", "Fish", "Eagle"}]. The common words are {"Fish", "Lion"}.

```
1 no_of_lines = 2
2 lines = []
3 try:
4     #let the user to input multiple list of integers, convert the list
5     to set find the unique values
6     for i in range(no_of_lines):
7         input_list=set(input(f"inter list {i+1}").split(" "))
8         lines.append(input_list)
9
10 except:
11     print("please try a gain!")
12
13 print(lines)
14 output=lines[0]
15 #apply intersection on the all sets entered by the user and find the
16 duplicated keywords:
17 for n in range(no_of_lines-1):
18     output &= lines[n+1]
19 print("common words",output)
```

Run: Problem3\_C\_FlexibleFunction - Problem3\_C\_FlexibleFunction - Problem2\_B\_Sets -

C:\Users\user\Anaconda3\envs\MLENV\python.exe C:\Users\user\PycharmProjects\pythonProject1\Problem2\_B\_Sets.py

inter list 1Cat Dog Lion Fish Mouse Elephant

inter list 2cat Fish Tiger Eagle Lion

[{'Dog', 'Elephant', 'Lion', 'Cat', 'Fish', 'Mouse'}, {'Lion', 'Tiger', 'cat', 'Fish', 'Eagle'}]

common words {'Fish', 'Lion'}

Process finished with exit code 0

Figure 2:Output Sample of Problem2\_B

### 3. Problem3\_C\_FlexibleFunction.py

```
def find_min_max(*args,**kwargs):
    #find the Max or Min value in the list

    res=[]
    for key in kwargs.keys():
        if kwargs[key][1]=="L":

            res.append(max(kwargs[key][0]))
        elif kwargs[key][1]=="S":
            res.append(min(kwargs[key][0]))

    return res


while True:
    try:
        no_of_lines = 4
        lines = ""
        for i in range(no_of_lines):
            lines += input() + "\n"

        #let the user to input the list of integers:
        input_list=lines.split("\n")[0:4]

        op1=input_list[1]

        op2 = input_list[3]
        #convert the list of strings to list of floats:
        int_list1 = [float(i) for i in input_list[0].split(" ")]
        int_list2 = [float(i) for i in input_list[2].split(" ")]

        #let the user to choose the operant L, Large, large, l=Max and
        S, Small, small, s=Min:
        #input_op=input("type L to find the max or S to find the
        min:") [0].upper()

        except:
            print("your input must be numbers")
            continue

        #return result to the user:

        output=find_min_max(int_list1=[int_list1,op1],int_list2=[int_list2,op2])
        print(output)
        status = list(input("press Q to quite or C to continue"))[0].upper()
        if status=="Q":
            break
```

The screenshot displays a Python IDE with the following components:

- Project Explorer:** Lists files including `AI_Master_HW2`, `HomeWork.docx`, `HomeWork2_02_Nov_2022.pdf`, `Problem2_A_Loops.py`, `Problem2_B_Sets.py`, `Problem3_C_FlexibleFunction.py`, `Problem4_D_PointandLineClasses.py`, `README.md`, `~$meWork.docx`, `External Libraries`, and `Scratches and Consoles`.
- Code Editor:** Shows the code for `Problem3_C_FlexibleFunction.py`. The code defines a function that takes a list of strings and processes them based on a user-defined number of lines (4 in this case). It splits the input into two lists of integers and prints them.

```
no_of_lines = 4
lines = ""
for i in range(no_of_lines):
    lines += input() + "\n"

#Let the user to input the list of integers:
input_list=lines.split("\n")[0:4]
#list1=input_list[0].split(" ")
op1=input_list[1]
#list2=input_list[2].split(" ")
op2 = input_list[3]
#convert the list of strings to list of floats:
int_list1 = [float(i) for i in input_list[0].split(" ")]
int_list2 = [float(i) for i in input_list[2].split(" ")]

#print(int_list1,int_list2)
```
- Console:** Shows the execution of the program. It prompts the user to enter 4 lines of input. The first line is `1 2 3`, which is split into `[3.0, 1.5]`. The second line is `1.5 2.5 3.5`, which is split into `[1.5, 4.0]`. The program then prints the results of the splits.

```
1 2 3
L
1.5 2.5 3.5 4.5
S
[3.0, 1.5]
press Q to quite or C to continue
1.5 2.5 3.5
S
1 2 3 4
L
[1.5, 4.0]
press Q to quite or C to continue
```

Figure 3:Output Sample of Problem3\_C (Case\_0)

#### 4. Problem4\_D\_PointandLineClasses.py

```
import math
#Point Class
class Point():
    def __init__(self,x,y):
        self.x=x
        self.y=y
#Line Class
class Lin):
    def __init__(self,line_start,line_end):
        #
        super(Lin,self).__init__([line_start[0],line_end[0]],[line_start[1],line_end
[1]])
        #create two points instances
        self.point1=Point(line_start[0],line_start[1])
        self.point2=Point(line_end[0],line_end[1])
        print(self.point1.x,self.point1.y)
        print(self.point2.x,self.point2.y)

    #claculate the length of line
    def line_length(self):
        length=math.sqrt((self.point2.x-self.point1.x)**2+(self.point2.y-
self.point1.y)**2)
        return length
while True:
    try:
        #take the coordinates from user as input
        input_coordinate=list(input("please input the coordination of teh
line x1,y1,x2,y2 respectively:").split(" "))
        #check the length of input values(must be 4)
        if len(input_coordinate)<4:
            print("please input 4 values of type number")
            continue
        #if the input values more that 4 elements it takes the first 4
elements:
        if len(input_coordinate)>=4:
            input_coordinate=input_coordinate[0:4]
        #convert the list of strings to list of floats:
        input_coordinate=[float(i) for i in input_coordinate]
    except:
        print("check the input to be 4 numbers")
        continue
    #create new line instance from the line class and find the length by
calling line_length function.

new_line=Lin([input_coordinate[0],input_coordinate[1]],[input_coordinate[2],
input_coordinate[3]])
length=new_line.line_length()
print("the length of line =",length)
```

The screenshot displays the PyCharm IDE interface. The top pane shows the source code for `Problem4_D_PointandLineClasses.py`. The code defines a `Point` class and a `Line` class that inherits from `Point`. The `Line` class takes start and end coordinates and calculates the length of the line using the distance formula.

```
1 import math
2 #Point Class
3 class Point():
4     def __init__(self,x,y):
5         self.x=x
6         self.y=y
7 #Line Class
8 class Line(Point):
9     def __init__(self, line_start, line_end):
10         # super(Line, self).__init__([line_start[0], line_end[0]], [line_start[1], line_end[1]])
11         #create two points instances
12         self.point1=Point(line_start[0],line_start[1])
13         self.point2=Point(line_end[0],line_end[1])
14         print(self.point1.x,self.point1.y)
15         print(self.point2.x,self.point2.y)
16
17
18         #calculate the length of line
```

The bottom pane shows the Run console output for the script. It displays the results of three test cases where the length of a line is calculated based on user input coordinates.

```
Run: C:\Users\user\anaconda3\envs\MLENV\python.exe C:\Users\user\PycharmProjects\pythonProject1\Problem4_D_PointandLineClasses.py
please input the coordination of teh line x1,y1,x2,y2 respectively:0 0 1 1
the length of line = 1.4142135623730951
please input the coordination of teh line x1,y1,x2,y2 respectively:0 0 0 1
the length of line = 1.0
please input the coordination of teh line x1,y1,x2,y2 respectively:-1 -1 1 1
the length of line = 2.8284271247461903
please input the coordination of teh line x1,y1,x2,y2 respectively:check the input to be 4 numbers
```

Figure 4: :Output Sample of Problem4\_D