Task 1

Write function minMaxSum. Given five positive integers, find the minimum and maximum values that can be calculated by summing exactly four of the five integers. Then return the respective minimum and maximum values as a single line of two space-separated long integers.

```
# Function or lambda

minSum, maxSum = miniMaxSum([1,3,5,7,9])

print(f"Minimum sum is {minSum}\nMaximum sum is {maxSum}")
```

Input: A single line of five space-separated integers.

Output: Return two space-separated long integers denoting the respective minimum and maximum values that can be calculated by summing exactly four of the five integers. Output will be printed as it shown in the code above.

Sample Input:

```
[1,3,5,7,9]
```

Sample Output:

```
Minimum sum is 16
Maximum sum is 24
```

Task 2

You are given Point class. Initialize the given values. Write all the methods. __init__ method should initialize the Point class get_x method should return x. get_y method should return y. shift method should move the point in x and y axes(without return). distance method should return the distance between 2 points. __str__ method should contain how Point class should look like.

```
class Point(object):
    def __init__(self, x, y):
        pass

    def get_x(self):
        pass

    def get_y(self):
        pass
    def shift(self, dx, dy):
        pass

    def distance(self, other):
        pass

    def __str__(self):
        pass

point1 = Point(10,4)
point2 = Point(5,12)

print(point1.distance(point2))
point1.shift(10,1)
print(point1.distance(point2))
print(point2.__str__())
```

Task 3

Write function `factorial` by making function recursive. Factorial of a number is the product of all the integers from 1 to that number.

Input: integer `N`

Output: factorial of `N`

Sample Input:

```
6
```

Sample Output:

```
720
```

Sample Input:

```
9
```

Sample Output:

```
362880
```

Task 4

Create a class hierarchy for a clothing store using inheritance in Python

Class: Clothing

Attributes: name (str) size (str) color (str) price (float)

Methods: display_info(): displays the name, size, color, and price of the clothing item

---

Class: Shirt (inherits from Clothing)

Attributes: type (str) (e.g. casual, formal)

Methods: display_info(): displays the name, size, color, price, and type of the shirt

---

Class: Pants (inherits from Clothing)

Attributes: length (str) (e.g. short, regular, long)

Methods: display_info(): displays the name, size, color, price, and length of the pants

Task 5

Write a Python program to count float values in a mixed list using lambda.

Input: a mixed list containing values of different types

Output: number of floats in the mixed list

Example: Given a list: [1, 'abcd', 3.12, 1.2, 4, 'xyz', 5, 'pqr', 7, -5, -12.22] Result: 3

Task 6

Create a `Person` class. Each person has some `fives`, `tens`, and `twenties`. This class has a function that calculates the sum of money a person has. Your goal is to write a function that returns the name of a person with the most money.

Class: `Person`

Attributes: `name` (str), `fives` (int), `tens` (int), `twenties` (int)

Methods: `sum_money()`: returns sum of person's money

Method example: fives=2, tens=2, twenties=0 => 2 * 5 + 2 * 10 + 0 * 20 = 30

Sample example:

```
john = Person("John", 2, 2, 0)
alice = Person("Alice", 1, 3, 0)
mike = Person("Mike", 0, 0, 2)
```

Input sample 1: most_money([john, alice, mike])

Output sample 1: Mike

Input sample 1: most_money([john, alice])

Output sample 1: Alice

Task 7

Write a function `ispalindrome` that checks whether a string is a palindrome (a palindrome is a string that reads the same from left to right and from right to left).

NOTE: You have to solve these problems in two ways using for loop and while loop.

Sample Input 1:

```
level
```

Sample Output 1:

```
True
```

Sample Input 2:

```
meat
```

Sample Output 2:

```
False
```

Task 8

Create a `Person` class. It should contain an attribute of the class `count_of_object` (which will show the number of objects created) and instance attributes such as `first name`, `last name`, `age`, `occupation`, `gender`, and `nationality`. Also define methods such as `year_born(cur_year)`, `get_name()`, `get_nationality()`, `get_age()`, `get_occupation()`. Note that the `year_born(cur_year)` method returns the person's birth year.

Example:

```
actor1 = Person(fisrt_name="Brad", last_name="Pitt", age=50, occupation="Actor",
gender="Man", nationality="American)

actor1.year_born(cur_year=2023)   Result will be   1973
```

```
actor1.count_of_object   Result will be  1


actor2 = Person(fisrt_name="Tom", last_name="Cruise", age=61, occupation="Actor",
gender="Man", nationality="American)

actor2.year_born(cur_year=2023)   Result will be   1962

actor2.count_of_object   Result will be   2
```

Task 9

You are given the array of number `a = [1,2,3,4,5]`. You need to apply following commands on array `a`:

Command	**Description**
`map ARITHMETIC_OPERATOR OPERAND`	replace each `ELEM_i` with result of operation `ELEM_i ARITHMETIC_OPERATOR OPERAND`
`filter CONDITIONAL_OPERATOR OPERAND`	remain only elements of array which satisfy condition `ELEM_i CONDITIONAL_OPERATOR OPERAND`. For example, command `filter > 4` means remain only element greater than 4
`fold ARITHMETIC_OPERATOR`	combine all elements by applying operation `ELEM_1 ARITHMETIC_OPERATOR ELEM_2 ... ELEM_n`, where `n` number of elements in array `a`. For example, after applying command `fold *` on array `a` the resulting array will be `[1*2*3*4*5] = [120]`
`progression ARITHMETIC_OPERATOR`	replace each `ELEM_i` with result of operation `ELEM_1 ARITHMETIC_OPERATOR ELEM_2 ... ELEM_i`. For example, after applying command `progression +` on array `a` the value of the 3-rd element of resulting array will be `1 + 2 + 3 = 6`

where

- `ARITHMETIC_OPERATOR` is one of `+`, `-`, `*`, `/`,
- `CONDITIONAL_OPERATOR` is one of `>`, `<`, `<=`, `>=`,
- `ELEM_i` is the i-th element of array `a`,
- `OPERAND` is any rational number.

NOTE: use lambda function

Input Format:

Single line containing command

Output Format:

Result of applying command on array a on a single line

Samples:

Input	Output
`map * 3`	`[3,6,9,12,15]`
`filter > 3`	`[4,5]`
`progression +`	`[1,3,6,10,15]`
`fold +`	`[15]`

Task 10

Create a class Polygon. Constructor of class Polygon should take array of numbers, each number represents length of edge. Add validation of edges to check that such polygon can exist (length of each edge must be less than sum of others), otherwise print corresponding error message.

Add Set method that takes array of numbers of same length and sets new edges. Also validate new array of edges. Create private static method to separate common validation logic of polygon existance.

Add Print method that prints polygon in human readable format if polygon is valid, otherwise print error message. For example, `It's 5-gon with length of edges: 5, 6, 7, 8, 9.`

Create a RightTriangle class inherited from Polygon. Class constructors should take 3 parameters corresponding to the length of each edge of triangle. Use constructor of parent class. Add validation specific to right triangle.

Realize Set and Print methods with adding corresponding validation and reuse validation from parent class' method.

```python
class Polygon:
    def __init__(...): # add parameters
        # your code here

    # realize Set method

    # realize Print method

class RightTriangle: # add inheritance of Polygon class
    def __init__(...): # add parameters
        # your code here
    # realize Set method

    # realize Print method
```

Task 11

You are given a positive integer `number`. Write function which returns list of prime numbers whose indeces are also prime numbers.

Note: index counts from 1 (2 -> 1, 3 -> 2, ..., 17 -> 7, ...)

Example:

Input: `number = 11`

Output:

```
[3, 5, 11]
```

Explanation

First prime number is 2, index 1 Second prime number is 3, index 2 - prime Third prime number is 5, index 3 Fourth prime number is 7, index 4 - prime Fifth prime number is 11, index 5 - prime

Prime nubmers are 2, 3, 5, ...

So result is [3, 5, 11]

Task 12

Create class `Person` with constructor that accept two parameters `name` and `age`. This constructor should has default value 0 for age.
In this class `Person` has 5 fields:
name - str, name of the person
age - int, age of the person, default - 0
partner - Person, partner of the person, default - None
children - list of Person, childrean that person has, default - []
status - str, status from ['Single', 'Married', 'Traitor', 'Dead'], default - 'Single'

You also should add properties for this fields:
name - get_name() -> return name
set_name(name) -> set not empty string name
age - get_age() -> return age
set_name() -> set non negative integer age
children - get_children() -> return string in format: if 0 child -> "0 child"
if more than 0 -> children separeted by ','
status - get_status() -> return status
set_status(status) -> set status from ['Single', 'Married', 'Traitor', 'Dead']
partner - get_partner() -> return partner
set_partner(person) -> set parner, if partner is not null, set status 'Married', otherwise 'Single'

You alse should add methods:
procreate(person1, person2) - class method, return new person with name Child of `person1.name` and `person2.name`, add new person to children for person1 and person2
get_full_information() - return string in format Person `name` with age `self.age` and status `self.status` has partner `self.partner` and `self.children`

You also should add your own implementations for `+, -, *, del`:
+ - add partners to person1 and person2, set statuts 'Married' if both are single, otherwise 'In love' if person doesn't have partner, oterwise status 'Traitor'
- - remove parthers for persons, and set status 'Single' if partner is a person that removes
* - invoke method procreate(p1, p2)
del - set status 'Dead', and set persons partner None
string representation - Person `name`

Example

```
a = Person('a')
b = Person('b', 0)
d = Person('d', 10)

print(a) # Person a
print(b) # Person b
print(d) # Person d

print(a.get_full_information()) # Person a with age 0 and status Single has
partner None and 0 child

a + b

print(a.get_full_information()) # Person a with age 0 and status Married has
partner Person b and 0 child

c = a * b
a + d
a - d

print(a.get_full_information()) # Person a with age 0 and status Traitor has
partner Person b and a has Person Child of a and b children
print(b.get_full_information()) # Person b with age 0 and status Married has
partner Person a and b has Person Child of a and b children
print(d.get_full_information()) # Person d with age 10 and status Single has
partner None and 0 child
print(c.get_full_information()) # Person Child of a and b with age 0 and status
Single has partner None and 0 child
```

Task 13

Design a class hierarchy for a simple weather simulation system. There are three types of weather conditions: sunny, cloudy, and rainy. The system should be able to handle the following functionality:

- A weather condition can be created with a specific temperature.
- A weather condition can return its temperature.
- A weather condition can return a string representation of itself.
- A weather condition can return a message about what to wear given its temperature.

In addition, the system should be able to handle the following:

- Compute the average temperature of a list of weather conditions.
- Compute the average temperature of a list of weather conditions given a specific condition.
- Filter a list of weather conditions based on a specific condition.

```python
class WeatherCondition:
    def __init__(self, temperature):
        self.temperature = temperature

    def __str__(self):
        raise NotImplementedError("__str__ not implemented")

    def what_to_wear(self):
        raise NotImplementedError("what_to_wear not implemented")

class Sunny(WeatherCondition):
    def __str__(self):
        # your code here

    def what_to_wear(self):
        # your code here


class Cloudy(WeatherCondition):
    def __str__(self):
        # your code here


    def what_to_wear(self):
        # your code here

class Rainy(WeatherCondition):
    def __str__(self):
        # your code here

    def what_to_wear(self):
        # your code here


average_temp = lambda ...

average_temp_by_condition = lambda ...

filter_conditions = lambda ...
```

conditions = [Sunny(20), Cloudy(1), Rainy(10), Sunny(14), Cloudy(-20), Rainy(5)]

# Compute the average temperature of all conditions

```
avg_temp = average_temp(conditions)
print("Average temperature of all conditions:", avg_temp)
```

# Compute the average temperature of sunny conditions

```
avg_temp_sunny = average_temp_by_condition(conditions, Sunny)
print("Average temperature of sunny conditions:", avg_temp_sunny)
```

# Filter sunny conditions

```
sunny_conditions = filter_conditions(conditions, Sunny)
print("Sunny conditions:")
for c in sunny_conditions:
    print(c.what_to_wear())
```

Task 14

Implement a task that uses lambda functions to process a list of dictionaries containing information about employees. The list of dictionaries should have the following keys: `name`, `age`, `salary`, and `department`.

Implement the following operations using lambda functions:

- Filter employees based on age (above or equal to 25 and below or equal to 40)
- Sort the employees based on salary in ascending order
- Group the employees based on the department they work in
- Calculate the average salary for each department

```
employee_data = [{'name': 'John Doe', 'age': 30, 'salary': 50000, 'department': 'IT'},
                 {'name': 'Jane Doe', 'age': 35, 'salary': 55000, 'department': 'HR'},
                 {'name': 'Jim Smith', 'age': 40, 'salary': 60000, 'department': 'IT'},
                 {'name': 'Amy Johnson', 'age': 32, 'salary': 53000, 'department': 'HR'},
                 {'name': 'Tom Brown', 'age': 38, 'salary': 57000, 'department': 'IT'}]

def process_employee_data(employee_data: List[Dict[str, int]]):
    # your code here
```

```
result = process_employee_data(employee_data)
print(result)

# Output: {'IT': 55000.0, 'HR': 54000.0}
```