"""
Anonymous Functions / Lambda Functions
Syntax:
lambda <parameters> : expression

Uses of Lambda Functions
1. They are useful to create a function when the function body is a single expression.
2. They are used to pass a function as an argument and the function body is a single expression.

Lambda functions in Python are small, anonymous functions defined using the lambda keyword. Unlike regular functions defined with def, they do not require a name and are typically used for short, one-time operations.
Key characteristics of lambda functions:
Anonymous: They are not explicitly named like functions defined with def.
Single Expression: A lambda function can only contain a single expression, which is implicitly returned. It cannot have multiple statements or a return keyword.
Syntax: The syntax is lambda arguments: expression.
Conciseness: They offer a concise way to define simple functions, particularly when used as arguments to higher-order functions like map(), filter(), and sorted().
Example:
Python

# A lambda function to add two numbers
add = lambda x, y: x + y
print(add(5, 3)) # Output: 8

# Using lambda with filter() to get even numbers from a list
numbers = [1, 2, 3, 4, 5, 6]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers) # Output: [2, 4, 6]
When to use lambda functions:
For simple, inline functions that are used only once.
As arguments to higher-order functions (map(), filter(), sorted(), reduce()) where a full function definition would be overly verbose.
In situations where a function is needed temporarily and defining a named function seems unnecessary.
"""

def func(a, b):
    return a + b

lambda a, b : a + b

"""
map:
applies a function to each and every item in the iterable
and returns the value.
Syntax:
map(<function>, <iterable>)

map returns a map object

filter:

```
applies a function to each and every item in the iterable
and returns only the values which are True.
Syntax:
filter(<function>, <iterable>)
filter returns filter object

reduce:
applies a function to each and every item in the iterable
and returns the single final computed value.
Syntax:
reduce(<function>, <iterable>)
"""


"""
t1 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
l1 = []
for item in t1:
    l1.append(item ** 2)
t2 = tuple(l1)
print(t2)


def calculate_square(num):
    return num ** 2


t1 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
t2 = map(calculate_square, t1)
print(t2)
t2 = tuple(t2)
print(t2)


def add_ten(num):
    return num + 10


l1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
l2 = map(add_ten, l1)
print(l2)
l2 = list(l2)
print(l2)


# filter
l1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
l2 = []
for item in l1:
    if (item % 2) == 0:
        l2.append(item)
print(l2)


def is_even(num):
    return (num % 2) == 0


l1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
l2 = filter(is_even, l1)
print(l2)
l2 = list(l2)
print(l2)


def is_odd(num):
    return (num % 2) == 1


l2 = filter(is_odd, l1)
print(l2)
l2 = list(l2)
print(l2)


# Reduce
l1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(sum(l1))


result = 0
for item in l1:
    result += item
print(result)


from functools import reduce
def add(x, y):
    return x + y


result = reduce(add, l1)
print(result)


def multiply(x, y):
    return x * y


result = reduce(multiply, l1)
print(result)
"""


l1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
l2 = list(map(lambda num: num ** 2, l1))
print(l2)

l2 = list(map(lambda num: num + 10, l1))
print(l2)


l2 = list(filter(lambda num: (num % 2) == 0, l1))
print(l2)
l2 = list(filter(lambda num: (num % 2) == 1, l1))
print(l2)
```

```
from functools import reduce
result = reduce(lambda x, y: x + y, l1)
print(result)

result = reduce(lambda x, y: x * y, l1)
print(result)


"""
l1 = []
for _ in range(5):
    temp = int(input("Enter Number: "))
    l1.append(temp)
print(l1)
"""


values = input("Enter Values: ")
print(type(values))
print(values)
values_list = values.split(' ')
print(values_list)
for index, value in enumerate(values_list):
    values_list[index] = int(value)
print(values_list)

values_list = list(map(int, input("Enter Values: ").split(' ')))
print(values_list)
```