1. What is a lambda function in Python, and how does it differ from a regular function?

In Python, a lambda function also known as an anonymous function is a one-line, and anonymous function that can take any number of arguments but can only have one expression. Lambda functions are defined using the 'lambda' keyword and are commonly used for short functions that don't need a formal function definition like regular functions.

2. Can a lambda function in Python have multiple arguments? If yes, how can you define and use them?

Lambda functions can have as many arguments as needed. Example given as follows:

In [1]:
```
lam = lambda a, b, c: a + b + c
result=lam(2,3,4)
result
```

Out[1]: 9

3. How are lambda functions typically used in Python? Provide an example use case.

Lambda functions are particularly useful in functional programming scenarios where you need to pass a function as an argument to other functions. Eg:

In [3]:
```
num=[1, 2, 3, 4, 5]
sqr=map(lambda x: x**2, num)
list(sqr)
```

Out[3]: [1, 4, 9, 16, 25]

4. What are the advantages and limitations of lambda functions compared to regular functions in Python?

Advantages: Conciseness: Lambda functions allow you to define small, one-line functions without the need for a formal function definition. Anonymous: Lambda functions are anonymous, meaning they don't require a separate name. Functional programming: Lambda functions are particularly useful in functional programming paradigms and when using higher-order functions like map(), filter(), and sorted().

Disadvantages: Single expression: Lambda functions can only contain a single expression, limiting their use for more complex functions. Readability: Overuse of lambda functions can decrease code readability. Limited functionality: Due to their simplicity, lambda functions are not suitable for functions that require complex statements.

5. Are lambda functions in Python able to access variables defined outside of their own scope? Explain with an example.

Yes, lambda functions in Python can access variables defined outside of their own scope. When a lambda function is created, it retains access to the variables present in the scope where it was defined, including global variables.

In [4]:
```
# Eg for the same is given as follows:
a=23
multiply = lambda x: a * x
multiply(6)
```

Out[4]: 138

6. Write a lambda function to calculate the square of a given number.

In [6]:
```
sqr = lambda x : x*x
sqr(7)
```

Out[6]: 49

7. Create a lambda function to find the maximum value in a list of integers.

In [16]:
```
list_int=[13,54,67,96,25,46]
find_max = lambda x : max(x)
find_max(list_int)
```

Out[16]: 96

8. Implement a lambda function to filter out all the even numbers from a list of integers.

In [21]:
```
list_int=[13,54,67,96,25,46]
even = filter(lambda x: x%2==0,list_int)
list(even)
```

Out[21]: [54, 96, 46]

9. Write a lambda function to sort a list of strings in ascending order based on the length of each string.

In [24]:
```
list_str=["supra","GTR","Nissan","lamborgini"]
sorted_list=sorted(list_str,key=lambda x:len(x))
sorted_list
```

Out[24]:
```
['GTR', 'supra', 'Nissan', 'lamborgini']
```

10. Create a lambda function that takes two lists as input and returns a new list containing the common elements between the two lists.

In [29]:
```
a=[1,2,3,4,5]
b=[7,6,8,3,1]
common=lambda a, b: list(set(a).intersection(b))
common(a,b)
```

Out[29]:
```
[1, 3]
```

11. Write a recursive function to calculate the factorial of a given positive integer.

In [32]:
```
def factorial(n):
    if n==0:
        return 1
    else:
        return n*factorial(n - 1)

factorial(5)  #calling function
```

Out[32]:
```
120
```

12. Implement a recursive function to compute the nth Fibonacci number.

In [33]:
```
def fib_series(n):
    if n==1:
        return 0
    elif n==2:
        return 1
    else:
        return fib_series(n-1)+fib_series(n-2)

fib_series(9)
```

Out[33]:
```
21
```

13. Create a recursive function to find the sum of all the elements in a given list.

In [35]:
```
def sum_recursive(lst):
    if not lst:
        return 0
    else:
        return lst[0] + sum_recursive(lst[1:])

l = [1, 2, 3, 4, 5]
sum_recursive(l)
```

Out[35]: 15

14. Write a recursive function to determine whether a given string is a palindrome.

In [37]:
```
def is_palindrome(s):
    s = s.lower()
    if len(s) <= 1:
        return True
    elif s[0] != s[-1]:
        return False
    else:
        return is_palindrome(s[1:-1])
```

In [41]:
```
a='Hritik'
b="Nitin"

print(is_palindrome(a))
print(is_palindrome(b))
```

```
False
True
```

15. Implement a recursive function to find the greatest common divisor (GCD) of two positive integers.

In [44]:
```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

gcd(33,88)
```

Out[44]: 11