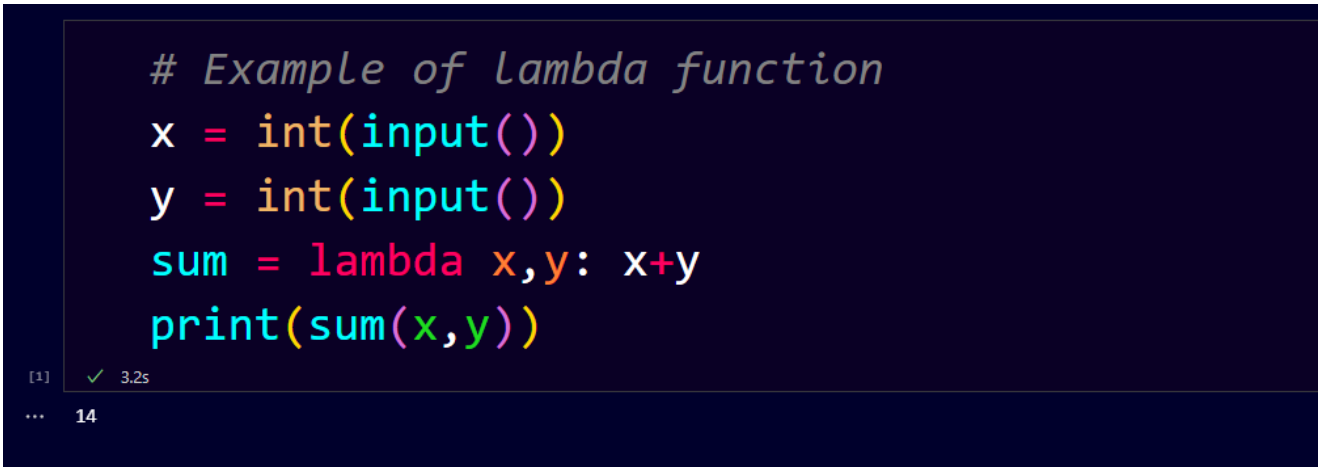
1. What is a lambda function in Python, and how does it differ from a regular function?

Solution:- The lambda function is use to calculate the simple operation and not the very complex operation. Lambda function starts with the lambda keyword and regular function starts with the def keyword

Example (Lambda):-



Example (User-Defined function):-

```python
# Example of user defined function
x = int(input())
y = int(input())
def add_number(x,y) :
    return x+y
print(add_number(x,y))
```

```
[4] ✓ 2.2s
... 14
```

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

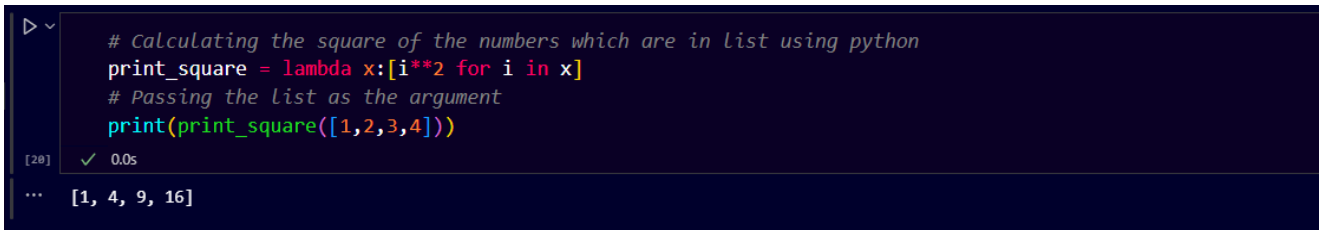
Solution:- The lambda function can take the multiple argument in the python

```python
# Multipication of four number using lambda function
a=int(input())
b=int(input())
c=int(input())
d=int(input())
multi = lambda a,b,c,d:a*b*c*d
print(multi(a,b,c,d))
```

```
[5] ✓ 5.5s
... 1260
```

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

Solution:- If the expression is small then we use the lambda function it provides the short code




```
# Calculating the square of the numbers which are in list using python
print_square = lambda x:[i**2 for i in x]
# Passing the list as the argument
print(print_square([1,2,3,4]))
```

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

Solution:- Advantages:-1-> Reduce the length of the code

2-> Generally code ends in the single line

Disadvantage:- 1-> It is not possible to compute the complex if elif and else statement

2-> It is also not possible to compute the complex looping statement

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

Solution:- Yes it is possible to access the value of the lambda function outside the scope of the lambda function

Example:-

Here lambda function is taking the argument x which is outside the scope of the lambda function

```
def outer_function() :
    x=10
    lambda_function = lambda y:x+y
    return lambda_function
outer_fun = outer_function()
result = outer_fun(5)
print(result)
```

```
0.0s
15
```

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

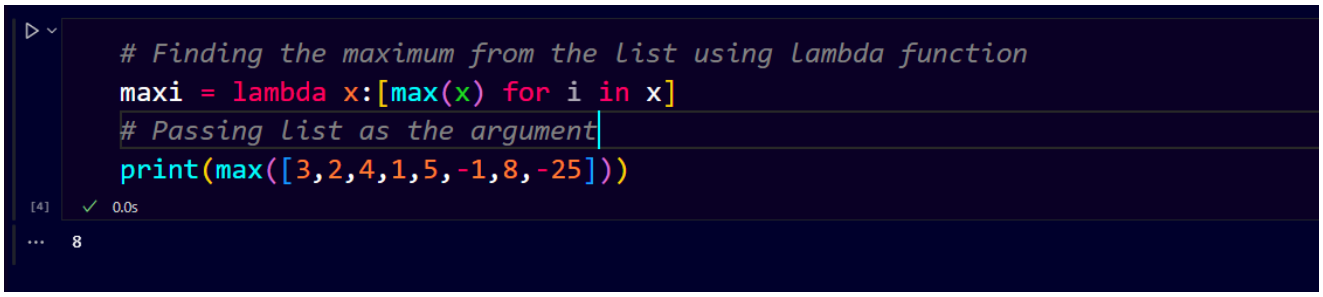
Solution:- Calculating the square of 10

```
# Printing the square of the given number
x = int(input())
square = lambda x:x**2
print(square(x))
```

```
[11] 2.0s
100
```

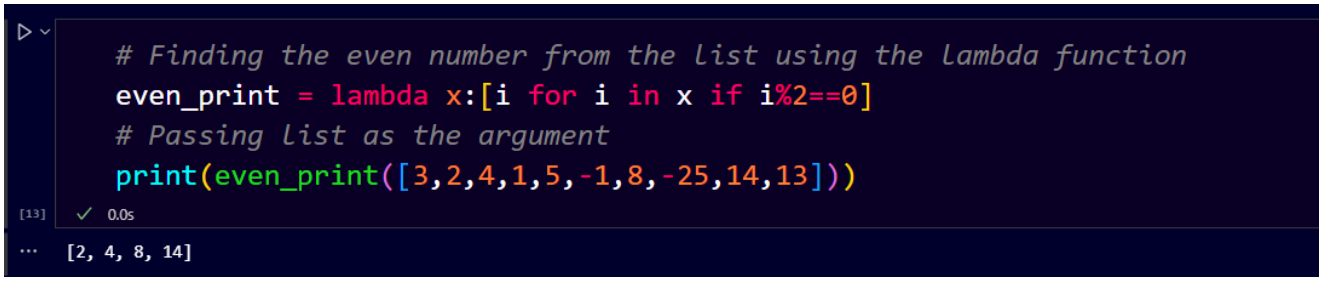
7. Create a lambda function to find the maximum value in a list of integers.

Solution:-



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

Solution:-



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

Solution:-

```python
# Sorting according to the length of the string
lst = ["abc","abcd","a","ab","c","cc","d"]
dict={}
for i in lst :
    dict[i] = len(i)
sorted_keys = sorted(dict,key=lambda k:dict[k])
print(sorted_keys)
```

[38] ✓ 0.0s

```
['a', 'c', 'd', 'ab', 'cc', 'abc', 'abcd']
```

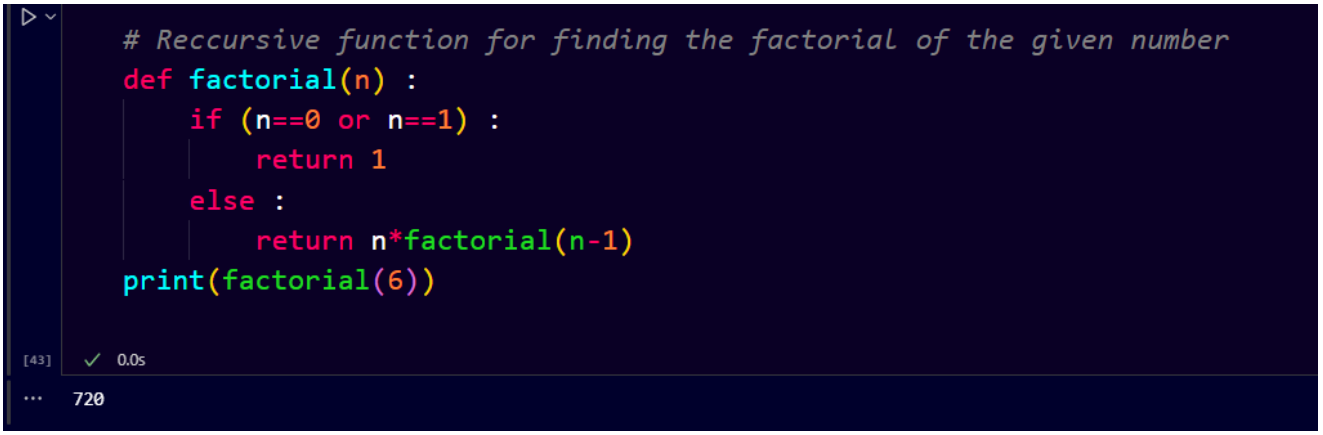
10. Create a lambda function that takes two lists as input and returns a new list containing the common elements between the two lists.

Solution:-

```python
# Printing the comman element of the both list using the lambda function
list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
common_elements = lambda x, y: [item for item in x if item in y]
result = common_elements(list1, list2)
print(result)
```

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

Solution:-

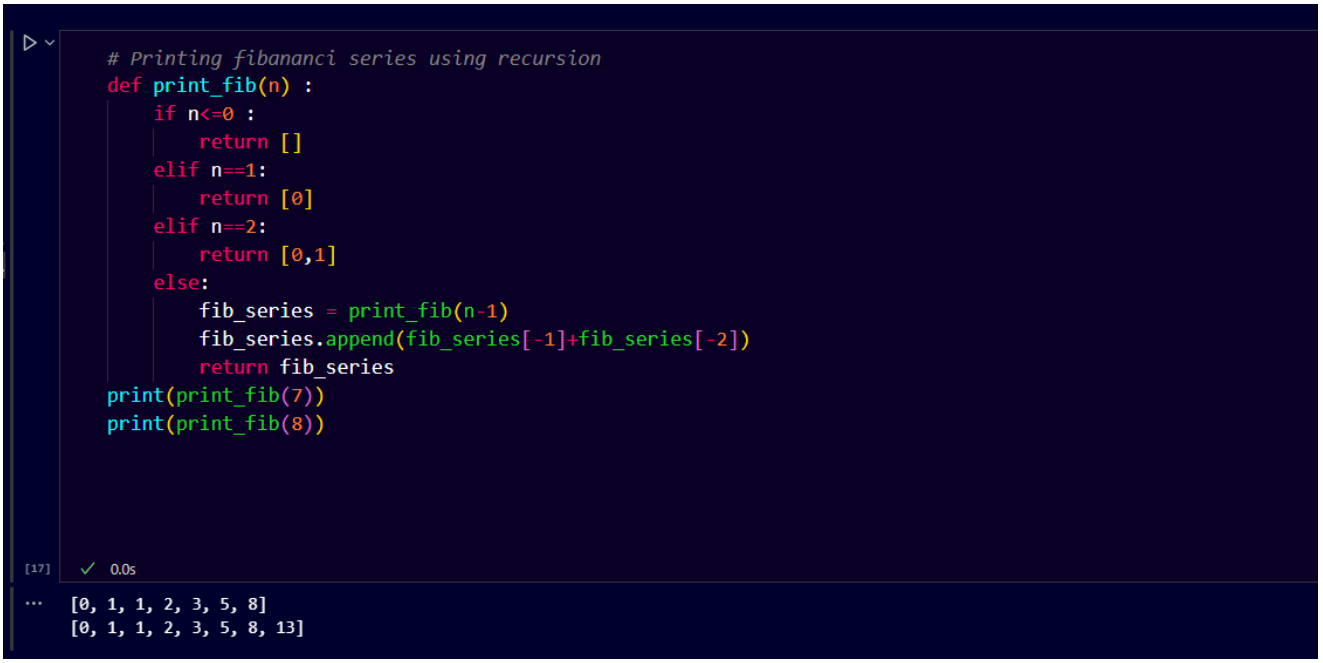


```python
# Reccursive function for finding the factorial of the given number
def factorial(n) :
    if (n==0 or n==1) :
        return 1
    else :
        return n*factorial(n-1)
print(factorial(6))
```

```
[43]  ✓ 0.0s
...  720
```

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

Solution:-

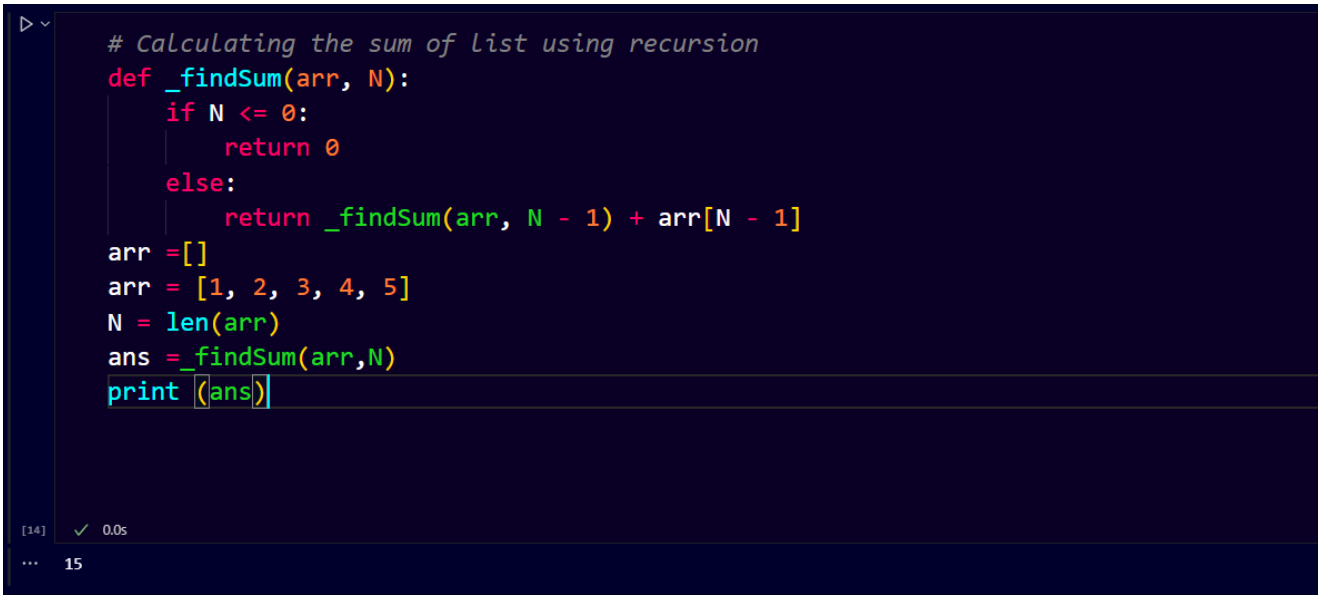


```python
# Printing fibananci series using recursion
def print_fib(n) :
    if n<=0 :
        return []
    elif n==1:
        return [0]
    elif n==2:
        return [0,1]
    else:
        fib_series = print_fib(n-1)
        fib_series.append(fib_series[-1]+fib_series[-2])
        return fib_series
print(print_fib(7))
print(print_fib(8))
```

```
[17]  ✓ 0.0s
...  [0, 1, 1, 2, 3, 5, 8]
     [0, 1, 1, 2, 3, 5, 8, 13]
```

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

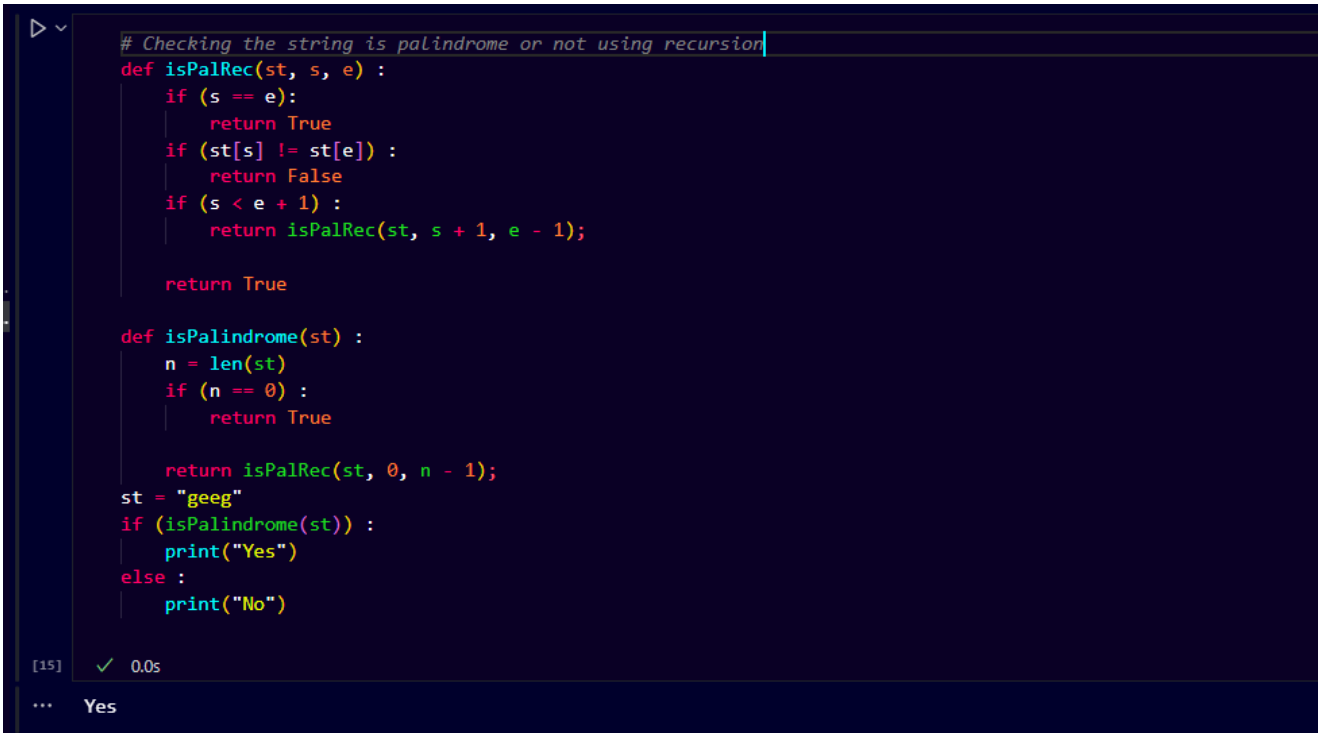
Solution:-




```python
# Calculating the sum of list using recursion
def _findSum(arr, N):
    if N <= 0:
        return 0
    else:
        return _findSum(arr, N - 1) + arr[N - 1]
arr =[]
arr = [1, 2, 3, 4, 5]
N = len(arr)
ans =_findSum(arr,N)
print (ans)
```

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

Solution:-



```python
# Checking the string is palindrome or not using recursion
def isPalRec(st, s, e) :
    if (s == e):
        return True
    if (st[s] != st[e]) :
        return False
    if (s < e + 1) :
        return isPalRec(st, s + 1, e - 1);

    return True

def isPalindrome(st) :
    n = len(st)
    if (n == 0) :
        return True

    return isPalRec(st, 0, n - 1);
st = "geeg"
if (isPalindrome(st)) :
    print("Yes")
else :
    print("No")
```

```
[15]  ✓ 0.0s
...  Yes
```

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

Solution:-

```python
# GCD of the two number using recursion
def gcd(a,b) :
    if (b==0) :
        return a
    else :
        return gcd (b,a%b)
print(gcd(98,56))
```

[16] ✓ 0.0s

```
14
```