```
# lambda p1, p2: expression     $Here, p1 and p2 are the parameters which are passed to the lambda function. You can add as many or few parameters as you need.


def addition(x, y):
    return x + y


result = addition(10, 5)
print(result)

# You cannot write multiple statements in the body of a lambda function.
adder = lambda x, y: x + y
print(adder(1, 2))

# What a lambda returns
string = 'some kind of a useless lambda'
print(lambda string: print(string))


# A REGULAR FUNCTION
def guru(funct, *args):


    funct(*args)


def printer_one(arg):


    return print(arg)


def printer_two(arg):


    print(arg)
# CALL A REGULAR FUNCTION
guru(printer_one, 'printer 1 REGULAR CALL')
guru(printer_two, 'printer 2 REGULAR CALL \n')
# CALL A REGULAR FUNCTION THRU A LAMBDA
guru(lambda: printer_one('printer 1 LAMBDA CALL'))
guru(lambda: printer_two('printer 2 LAMBDA CALL'))
```

```python

f=(lambda x: x + x)(2)  #take 2 as x and compute
print(f)

# lambdas in filter()
# The filter function is used to select some particular elements from a sequence of elements. The sequence can be any iterator like lists, sets, tuples, etc.

sequences = [10,2,8,7,5,4,3,11,0, 1]
filtered_result = filter (lambda x: x > 4, sequences)
print(list(filtered_result))


# lambdas in map()
# the map function is used to apply a particular operation  to every element in a sequence. Like filter(), it also takes 2 parameters:

sequences = [10,2,8,7,5,4,3,11,0, 1]
filtered_result = map (lambda x: x*x, sequences)
print(list(filtered_result))

# Step 1) Perform the defined operation on the first 2 elements of the sequence.
#
# Step 2) Save this result
#
# Step 3) Perform the operation with the saved result and the next element in the sequence.
from functools import reduce
sequences = [1,2,10,20,70]
product = reduce (lambda x, y: x*y, sequences)
print(product)
```