Python

Functions

Chapter 16

Suresh techs

Suresh techs – Technology and Entertainment

Subscribe

Subscribe: You don’t miss our future videos

Doubts

Follow In Instagram #sureshtechs

- Link in the description
- Clear your doubts chatting with me

Real life functions?

- Marriage function
- Engagement function
- Birthday function

- Marriage function: everyone will get together to celebrate
- Here the task is to complete marriage function
- Similarly, we will create functions to perform specific tasks in any of the programming language

Syntax

```
def functionName(parameters):
        statements
```

Some functions

- def **doMarriage**(inviters, పెళ్లి కొడుకు, పెళ్లి కూతురు):

    Bojanalu

    Aatalu

    Pelli

    …

Write a program to check whether 19,18,22 are even or odd? Without function.

Write a program to check whether 19,18,22 are even or odd? With function.

Function definition

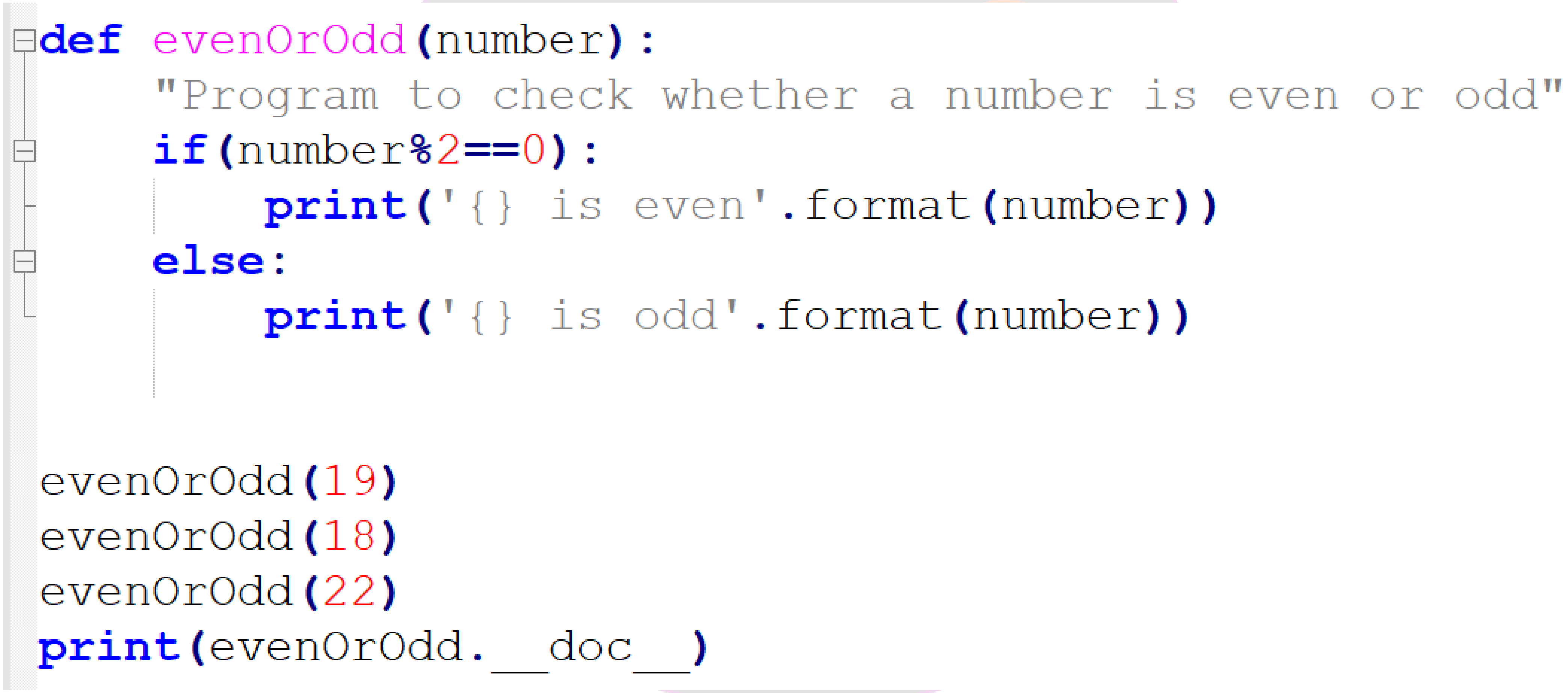
- Group of statements designed to perform specific task
- Idea behind functions is put commonly used statements together instead of writing same code again and again

Good practice (Docstring)

```
def functionName(parameters):
        "docstring"
        statements
```

- Used to describe the functionality of the function, it is optional but a good practice to use.
- functionName.__doc__

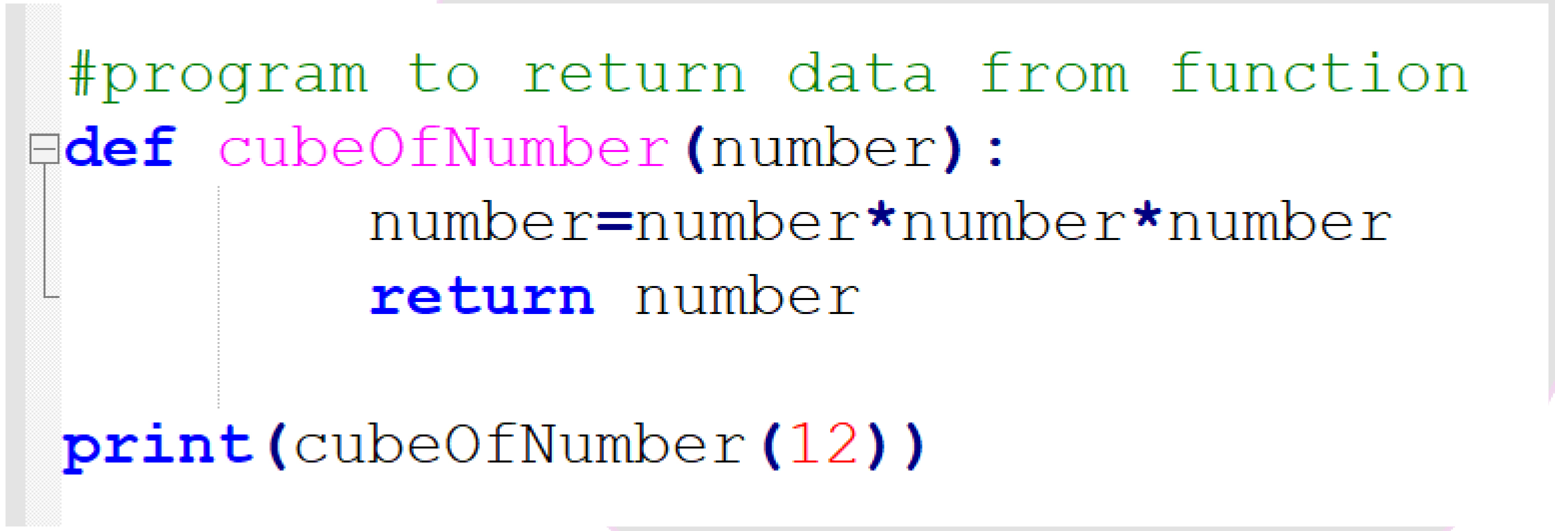
Change previous program to include doc string



```
def evenOrOdd(number):
    "Program to check whether a number is even or odd"
    if(number%2==0):
        print('{} is even'.format(number))
    else:
        print('{} is odd'.format(number))


evenOrOdd(19)
evenOrOdd(18)
evenOrOdd(22)
print(evenOrOdd.__doc__)
```

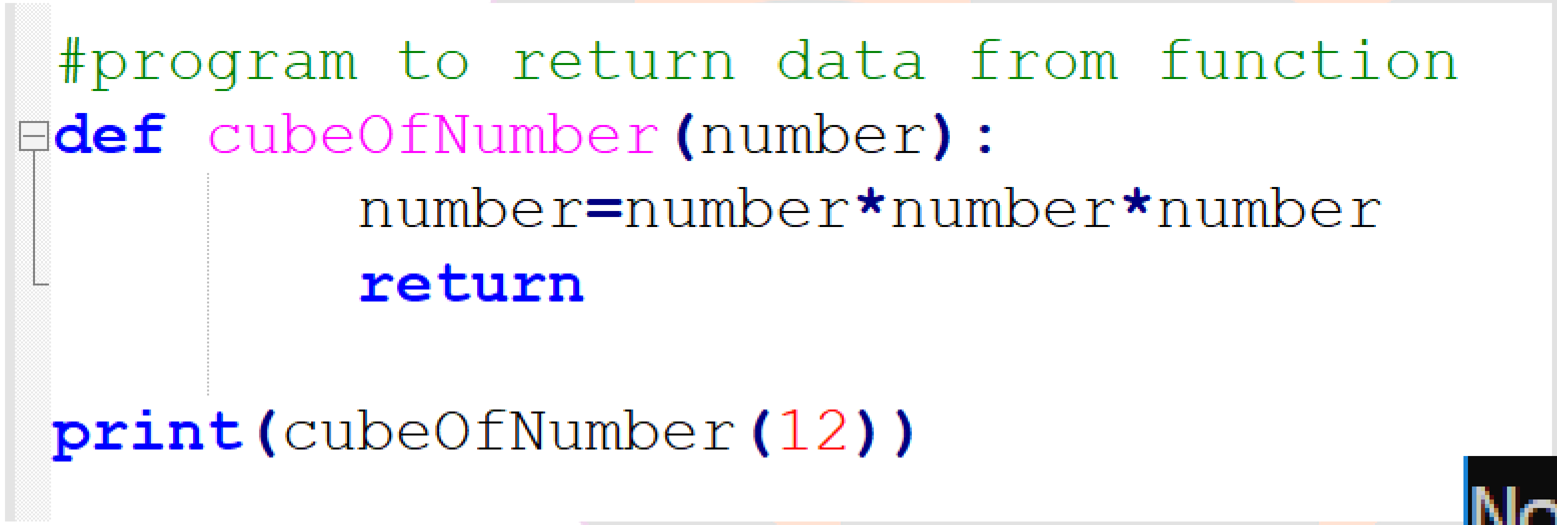
Returning from a function

- Used to return data and exit from the function

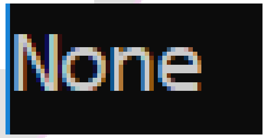


```
#program to return data from function
def cubeOfNumber(number):
        number=number*number*number
        return number

print(cubeOfNumber(12))
```

What if nothing is returned from the return statement?

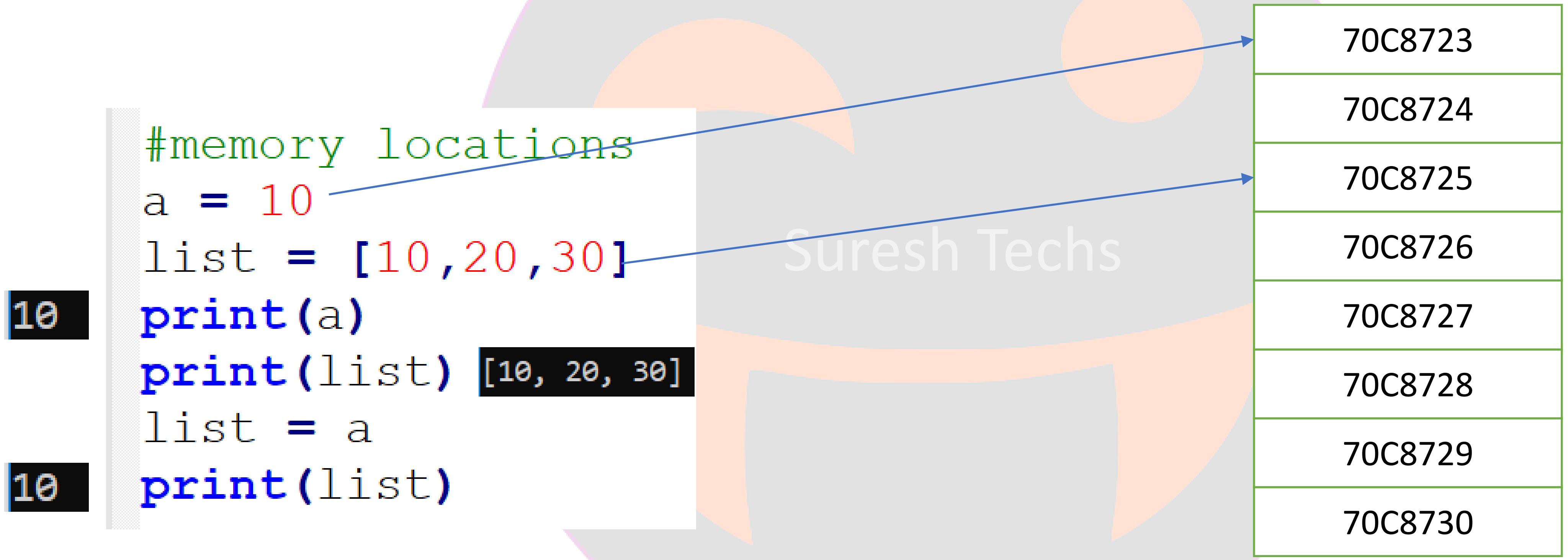


```
#program to return data from function
def cubeOfNumber(number):
        number=number*number*number
        return

print(cubeOfNumber(12))
```



```
None
```

Pass by reference, Pass by value

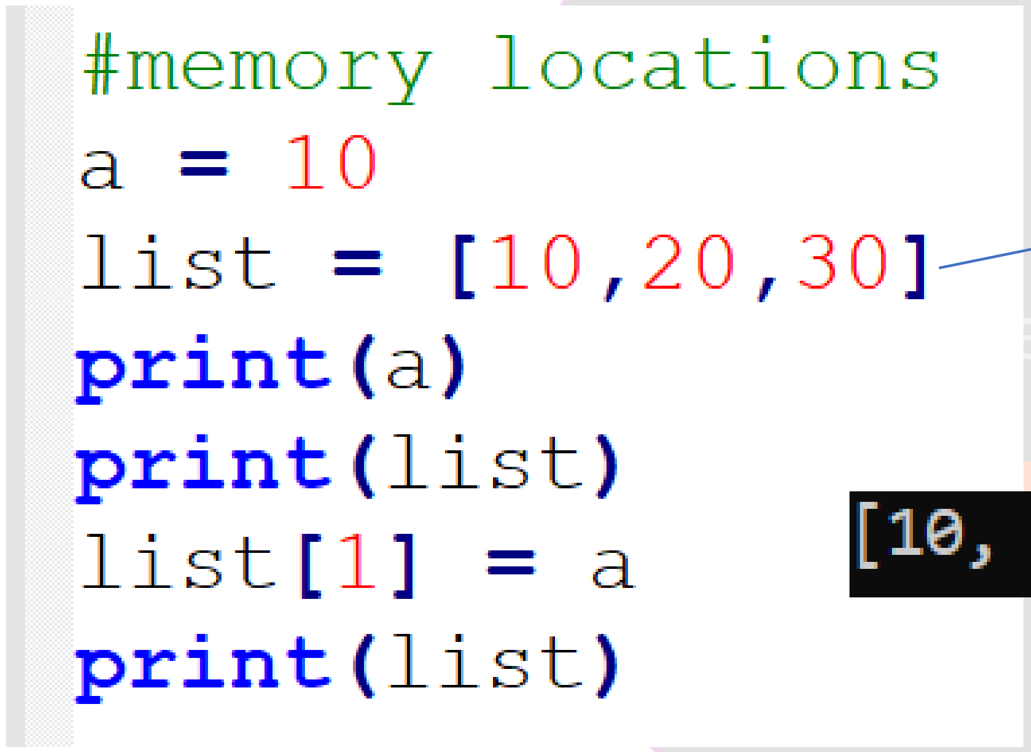


```
#memory locations
a = 10
list = [10,20,30]
print(a)
print(list)
list = a
print(list)
```

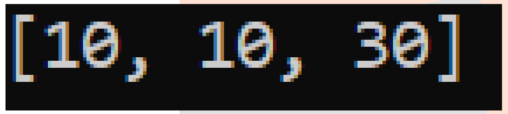
```
10
[10, 20, 30]
10
```

Memory Locations

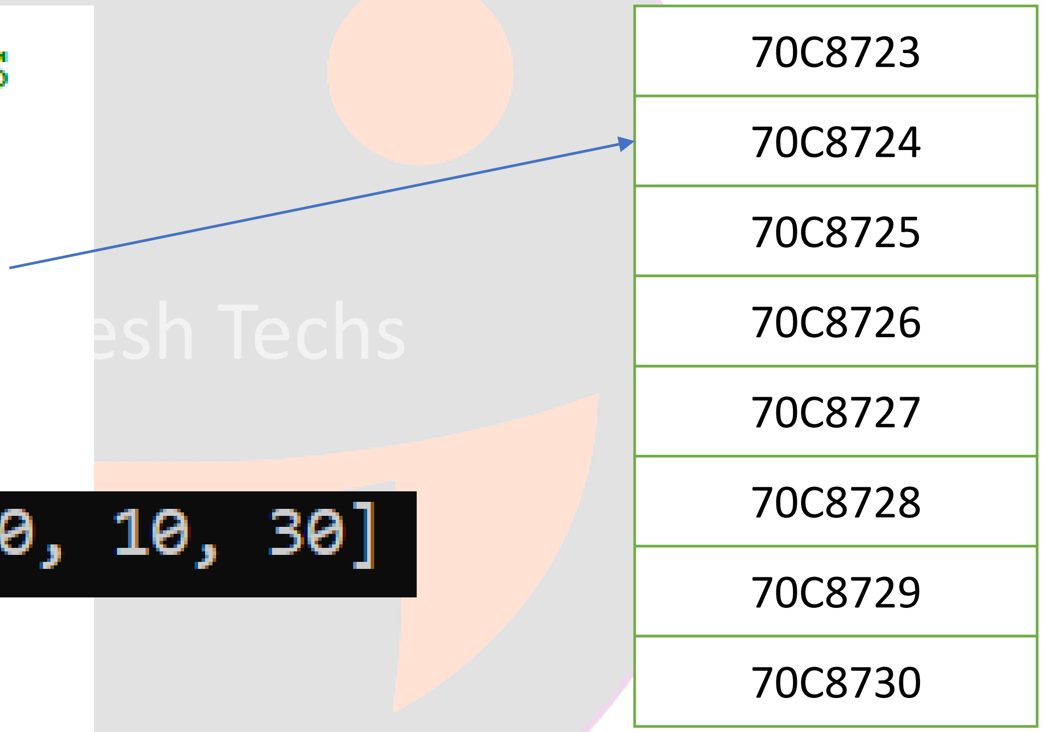
What is the output of last statement?



```python
#memory locations
a = 10
list = [10,20,30]
print(a)
print(list)
list[1] = a
print(list)
```



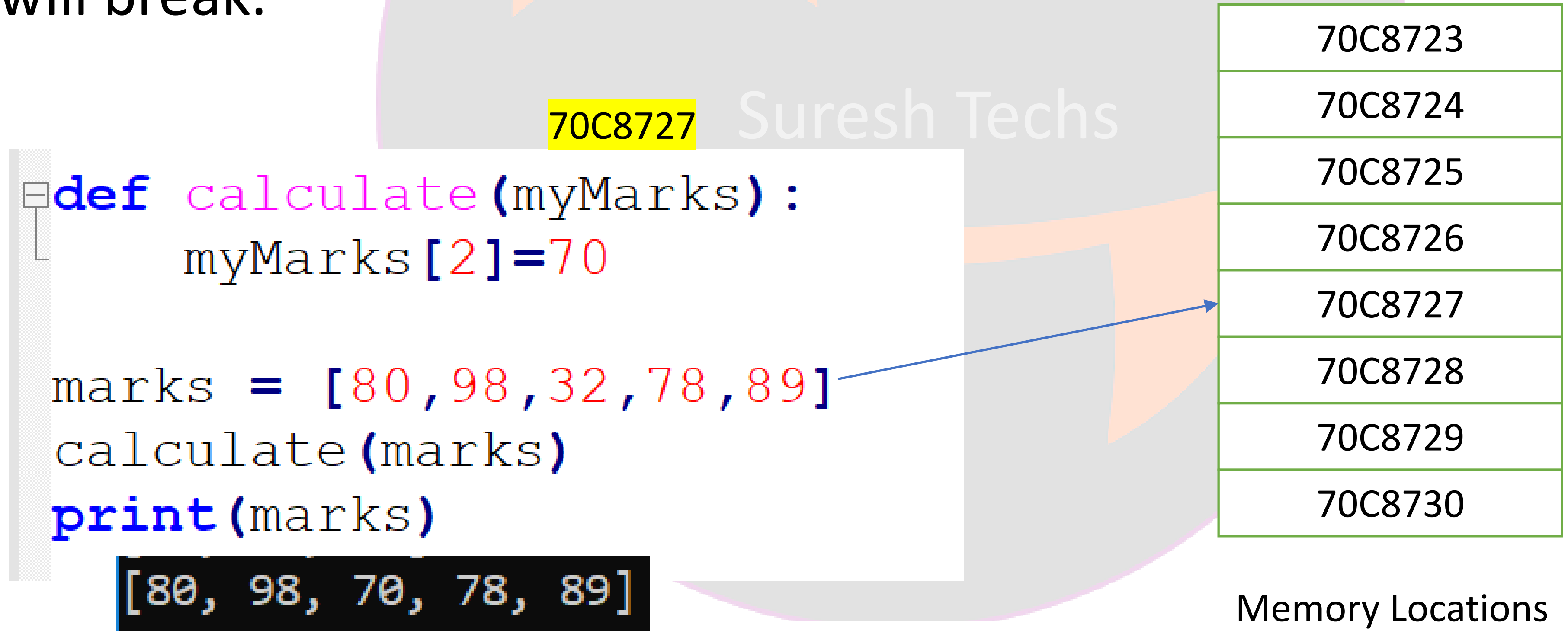
```
[10, 10, 30]
```



Memory Locations

Important

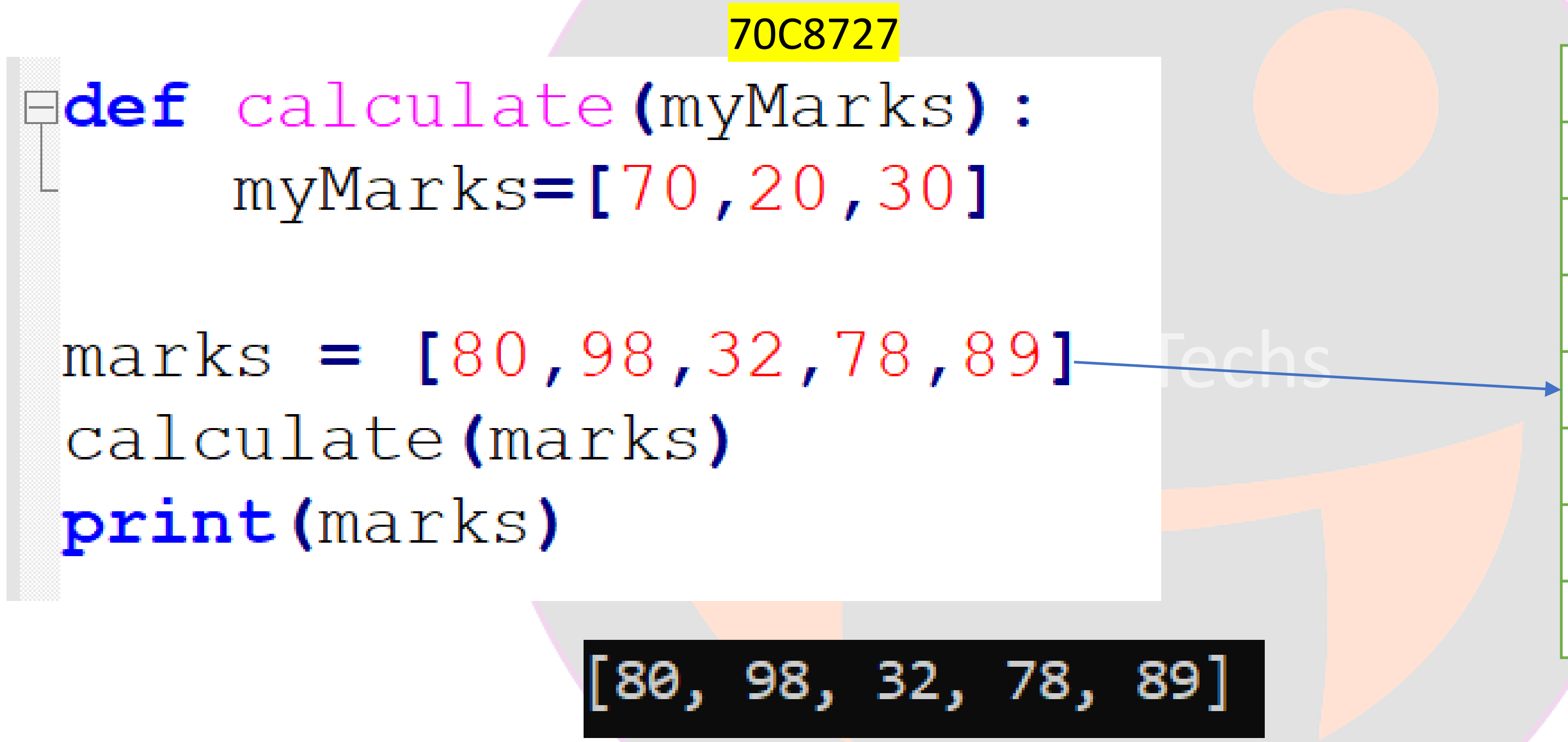
- If the reference passed to a function gets changed to something else, then the connection between the passed and received parameter will break.



```
def calculate(myMarks):
    myMarks[2]=70

marks = [80,98,32,78,89]
calculate(marks)
print(marks)
```

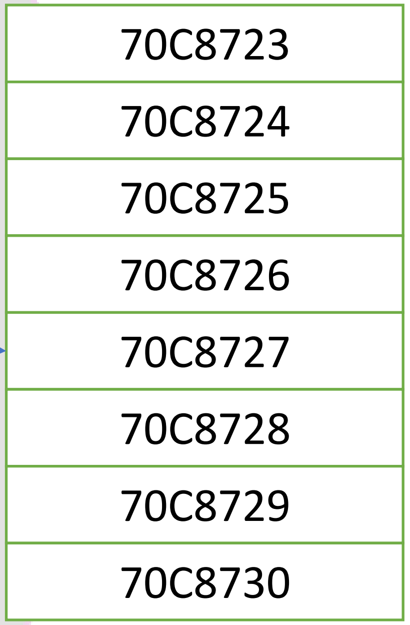
```
[80, 98, 70, 78, 89]
```

What will be the output of this program?


70C8727

```
def calculate(myMarks):
    myMarks=[70,20,30]

marks = [80,98,32,78,89]
calculate(marks)
print(marks)
```

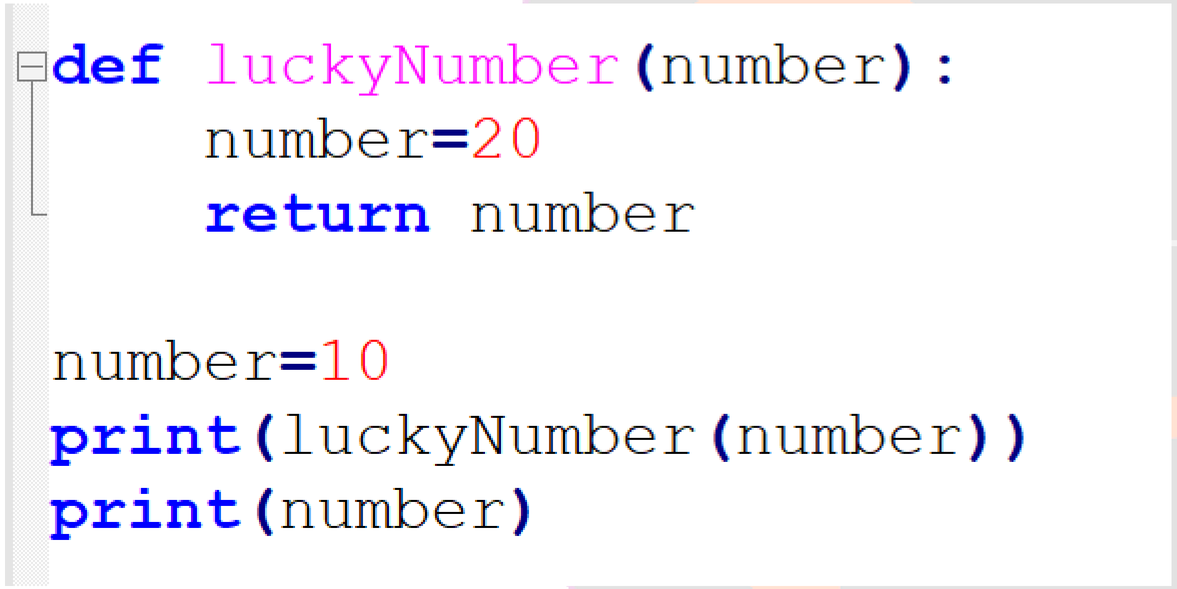
```
[80, 98, 32, 78, 89]
```



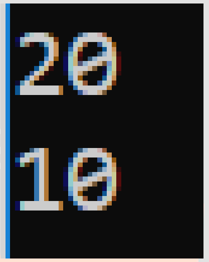
Memory Locations

If the reference passed to a function gets changed to something else, then the connection between the passed and received parameter will break.

What is the output of this program?

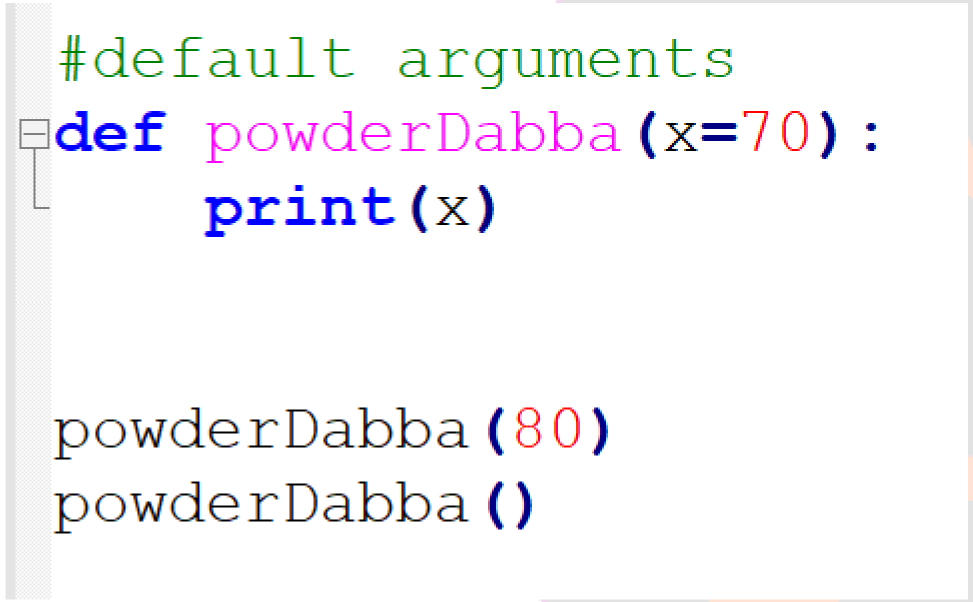


```python
def luckyNumber(number):
    number=20
    return number

number=10
print(luckyNumber(number))
print(number)
```

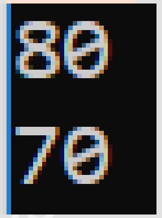


```
20
10
```

Default arguments



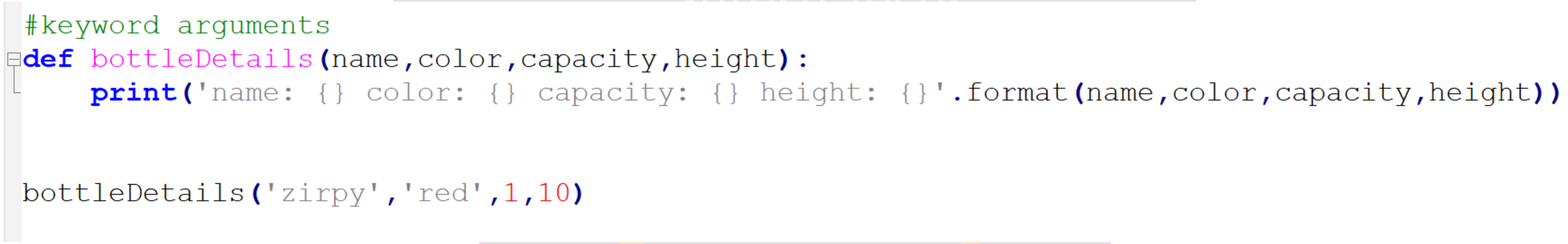
```
#default arguments
def powderDabba(x=70):
    print(x)


powderDabba(80)
powderDabba()
```



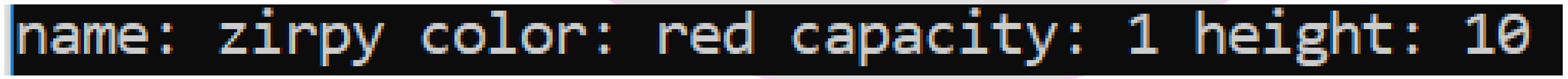
```
80
70
```

Keyword arguments(kwargv)

- We can specify argument name and value together while calling a function

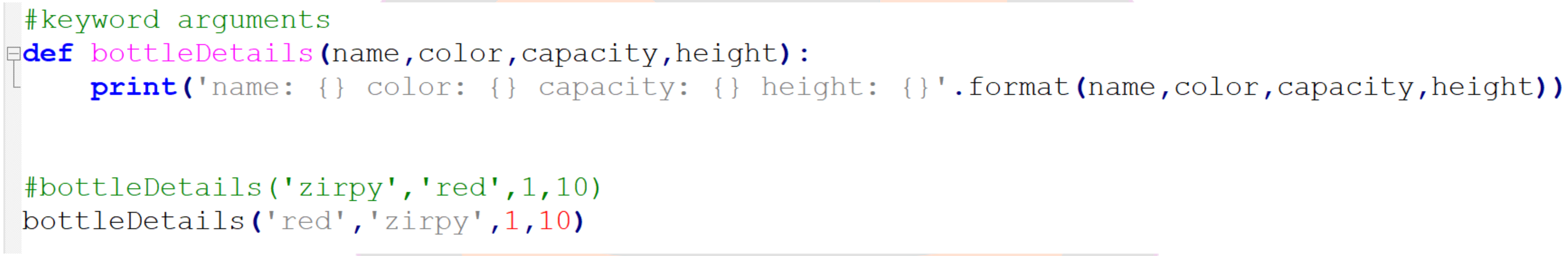


```
#keyword arguments
def bottleDetails(name,color,capacity,height):
    print('name: {} color: {} capacity: {} height: {}'.format(name,color,capacity,height))


bottleDetails('zirpy','red',1,10)
```

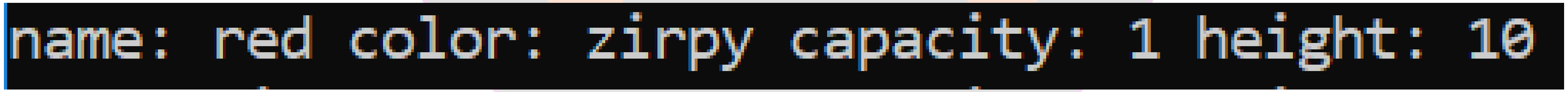


```
name: zirpy color: red capacity: 1 height: 10
```

Keyword arguments(kwargv)

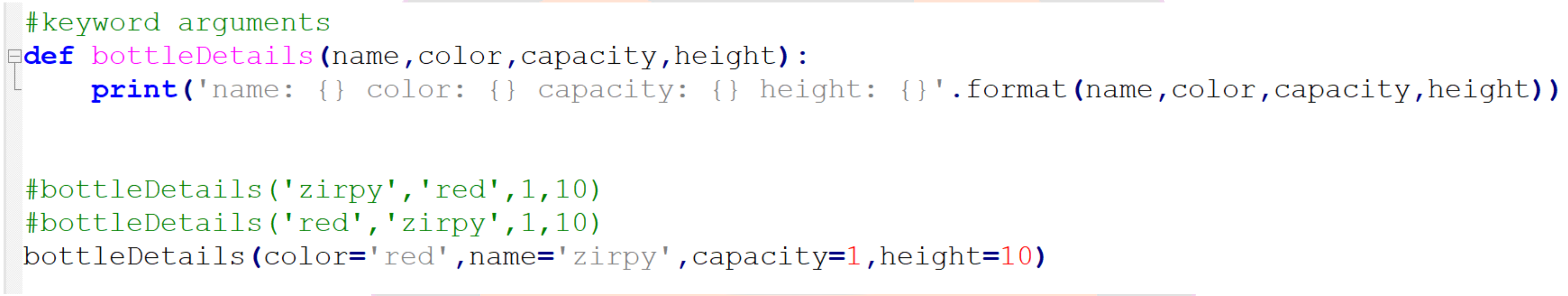


```
#keyword arguments
def bottleDetails(name,color,capacity,height):
    print('name: {} color: {} capacity: {} height: {}'.format(name,color,capacity,height))


#bottleDetails('zirpy','red',1,10)
bottleDetails('red','zirpy',1,10)
```

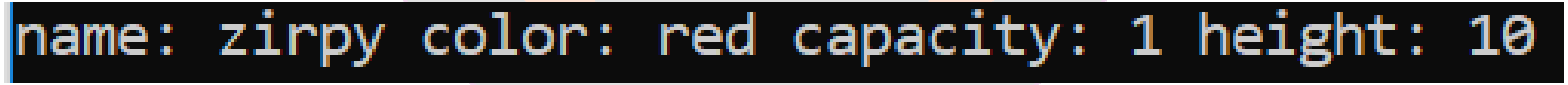


```
name: red color: zirpy capacity: 1 height: 10
```

Keyword arguments(kwargv)



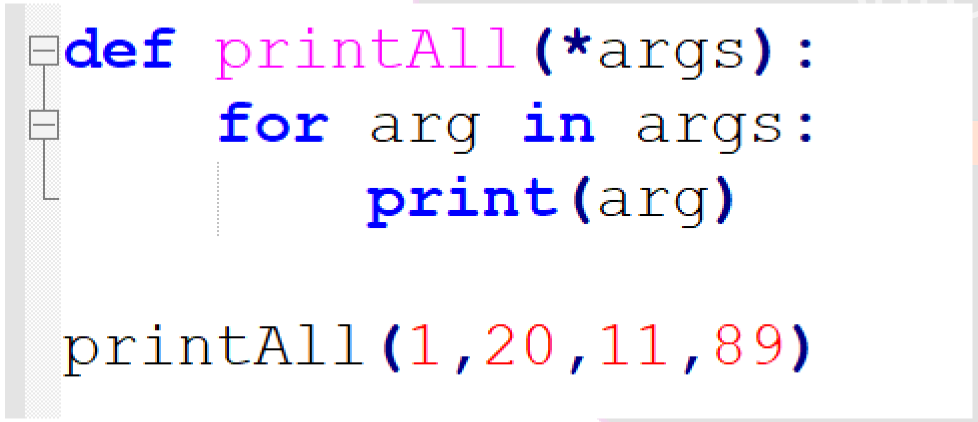
```
#keyword arguments
def bottleDetails(name,color,capacity,height):
    print('name: {} color: {} capacity: {} height: {}'.format(name,color,capacity,height))


#bottleDetails('zirpy','red',1,10)
#bottleDetails('red','zirpy',1,10)
bottleDetails(color='red',name='zirpy',capacity=1,height=10)
```



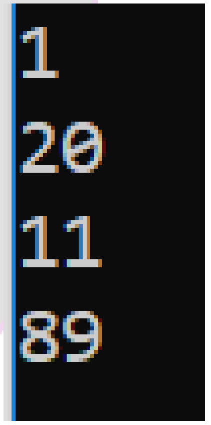
```
name: zirpy color: red capacity: 1 height: 10
```

Variable number of arguments

- *args(normal arguments / Non keyword arguments)
- **kwargs(keyword arguments)

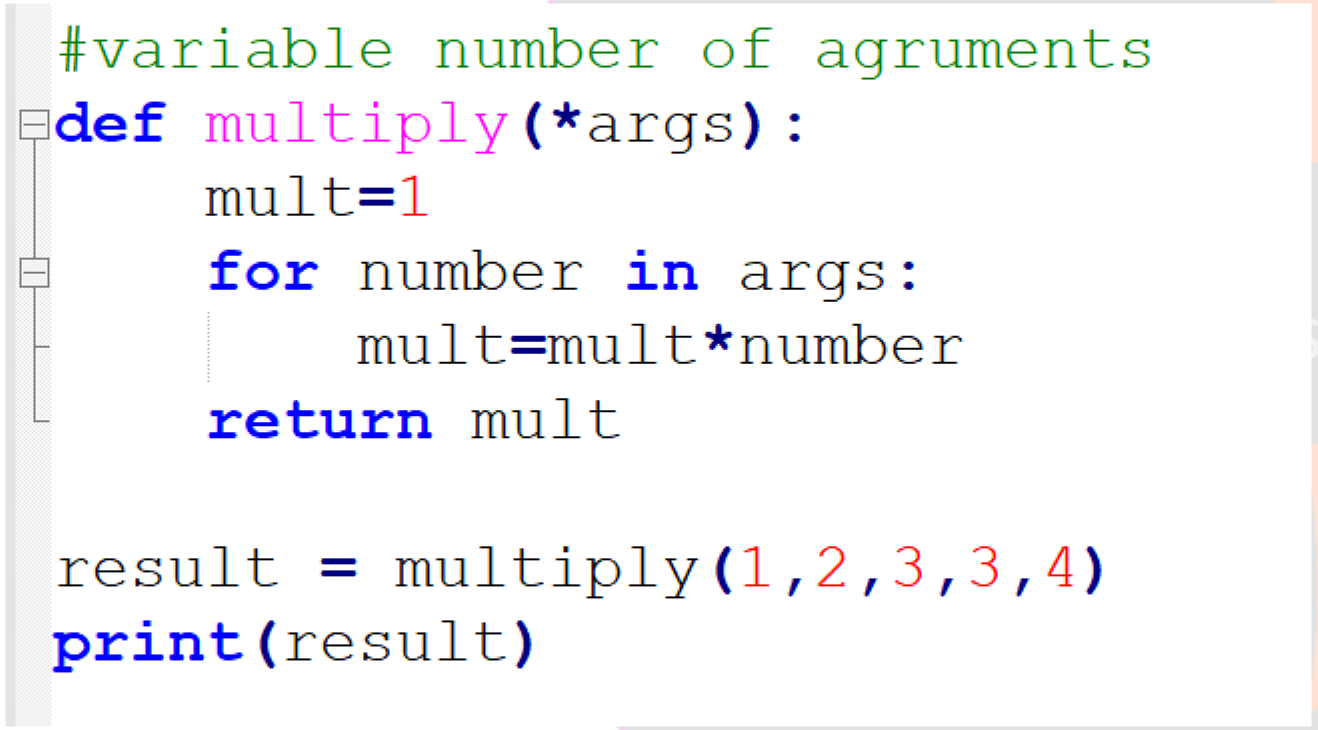


```python
def printAll(*args):
    for arg in args:
        print(arg)

printAll(1,20,11,89)
```

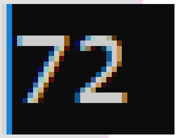


```
1
20
11
89
```

What is the output of this program?

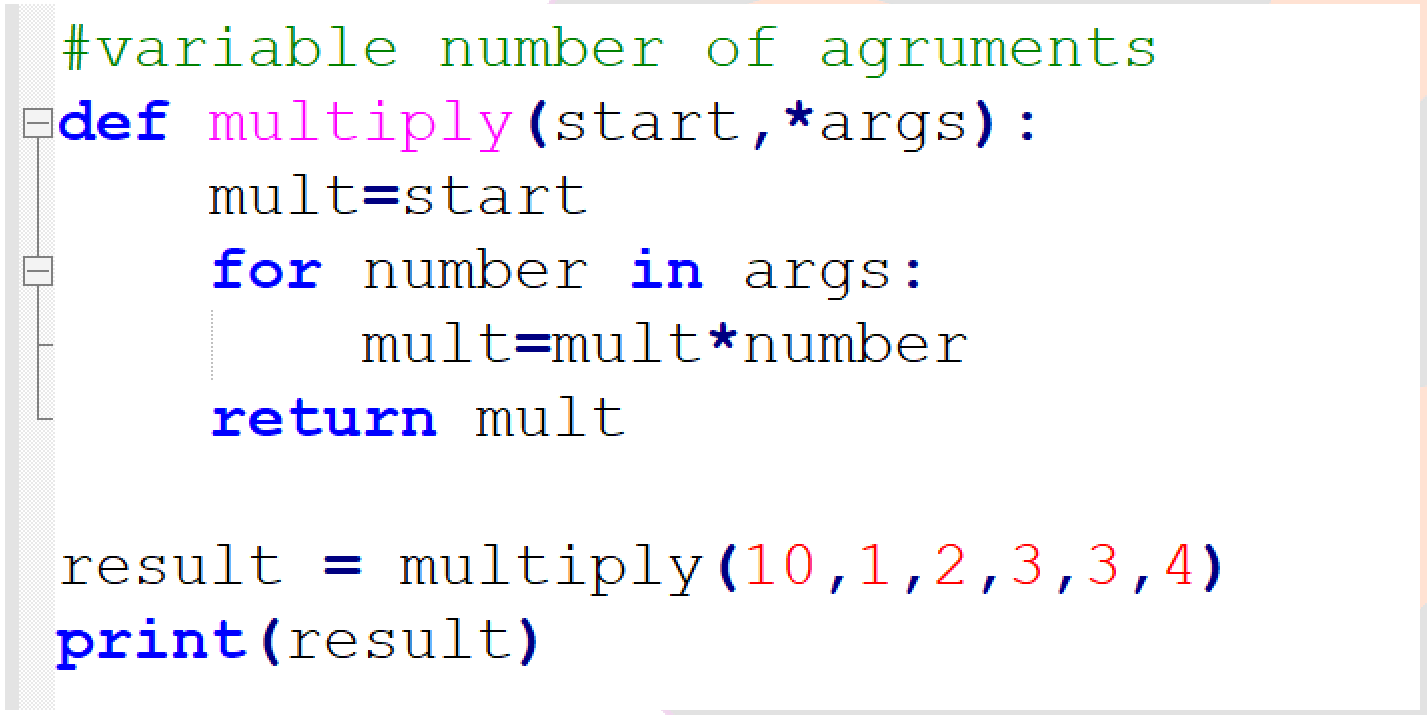


```
#variable number of agruments
def multiply(*args):
    mult=1
    for number in args:
        mult=mult*number
    return mult

result = multiply(1,2,3,3,4)
print(result)
```



```
72
```

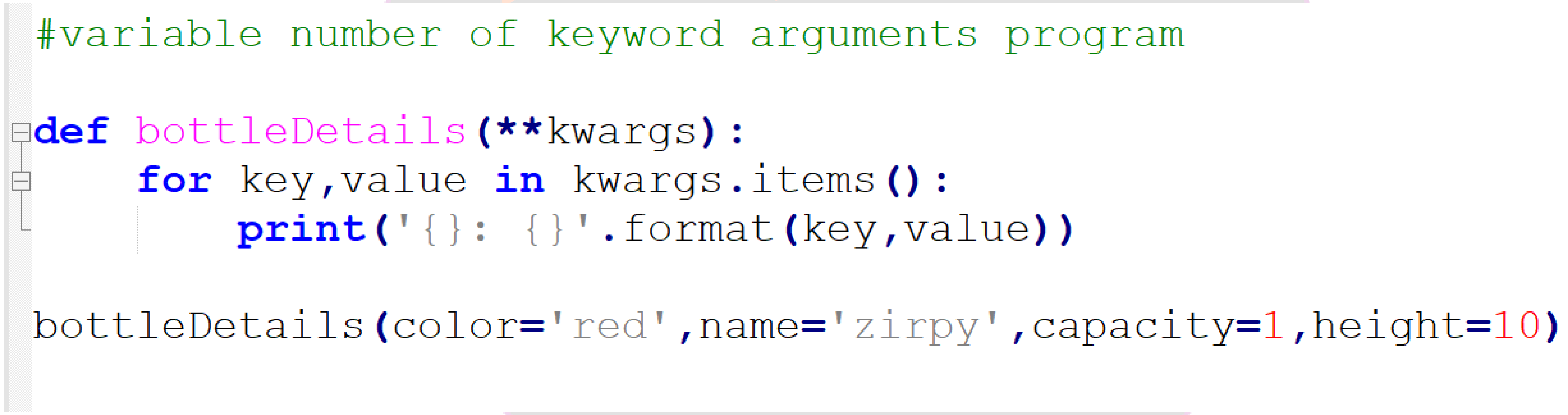
What is the output of this program?



```
#variable number of agruments
def multiply(start,*args):
    mult=start
    for number in args:
        mult=mult*number
    return mult

result = multiply(10,1,2,3,3,4)
print(result)
```

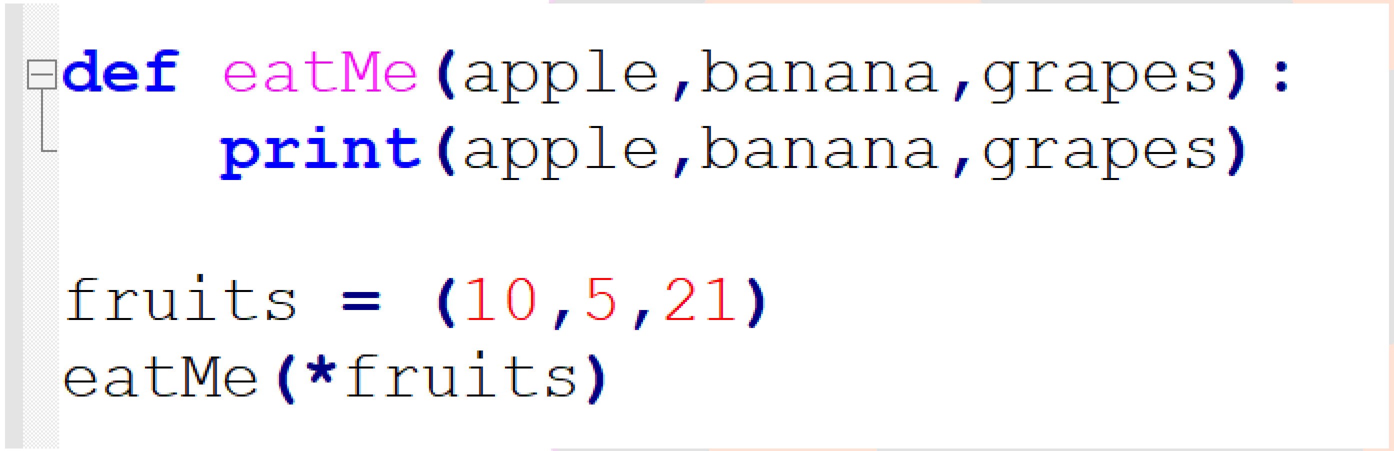
**kwargs(keyword arguments)

- Used to pass variable number of keyword arguments



```python
#variable number of keyword arguments program

def bottleDetails(**kwargs):
    for key,value in kwargs.items():
        print('{}: {}'.format(key,value))

bottleDetails(color='red',name='zirpy',capacity=1,height=10)
```

Passing *args, **kwargs from caller



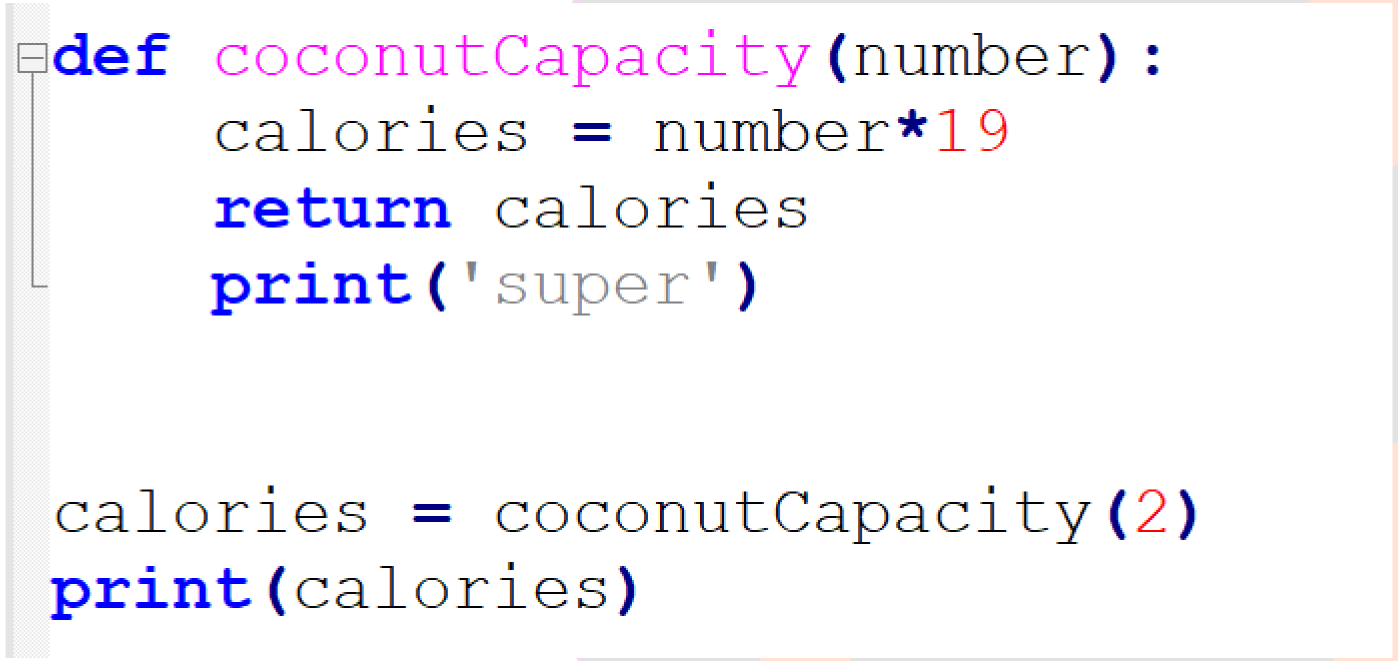
```
def eatMe(apple,banana,grapes):
    print(apple,banana,grapes)

fruits = (10,5,21)
eatMe(*fruits)
```

Yield, generators

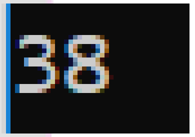
- Generators?
  - **Generator function**: If the body of the function contains **yield** then that function becomes **generator function**
  - **Generator object**: generator functions return a generator object. You can get values from generator object either by calling next(__next__) method or using a for in loop

Note: using __next__() is not a good option as we need to call again and again. We will use looping techniques most often!

yield?

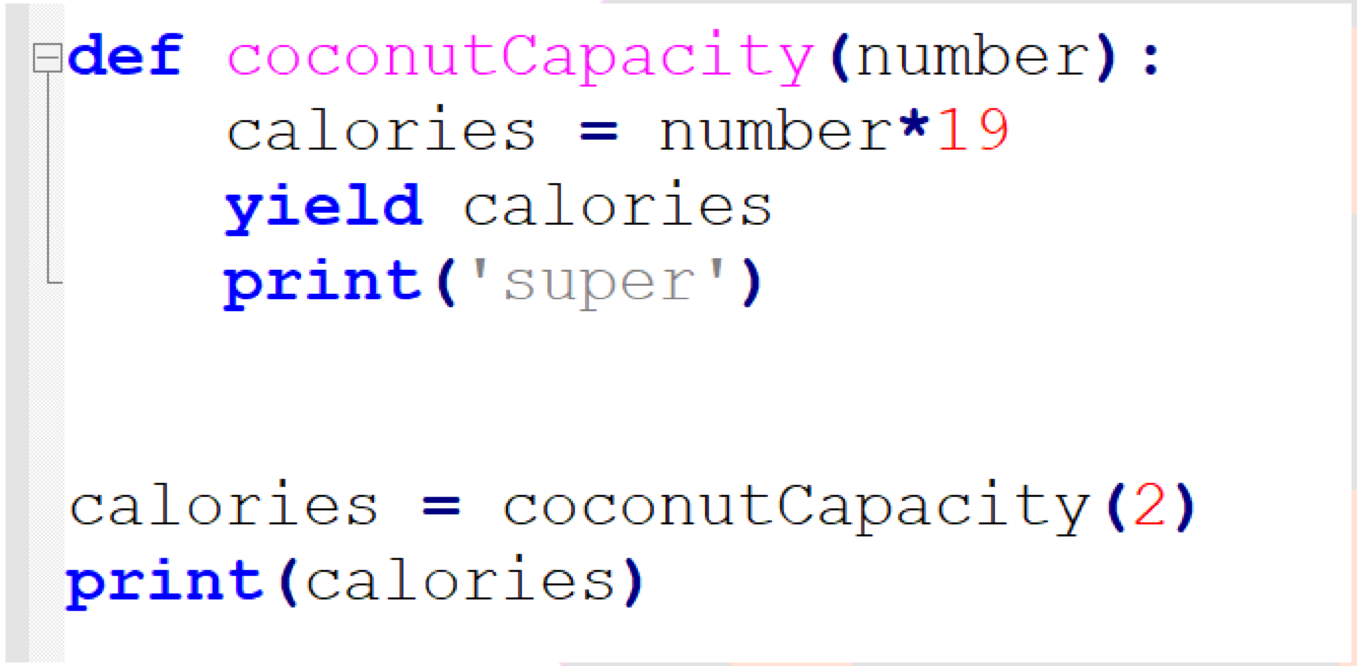


```python
def coconutCapacity(number):
    calories = number*19
    return calories
    print('super')


calories = coconutCapacity(2)
print(calories)
```

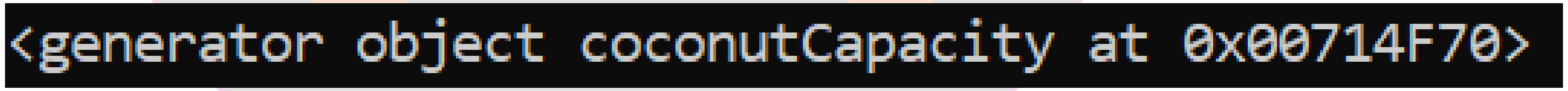


```
38
```

yield



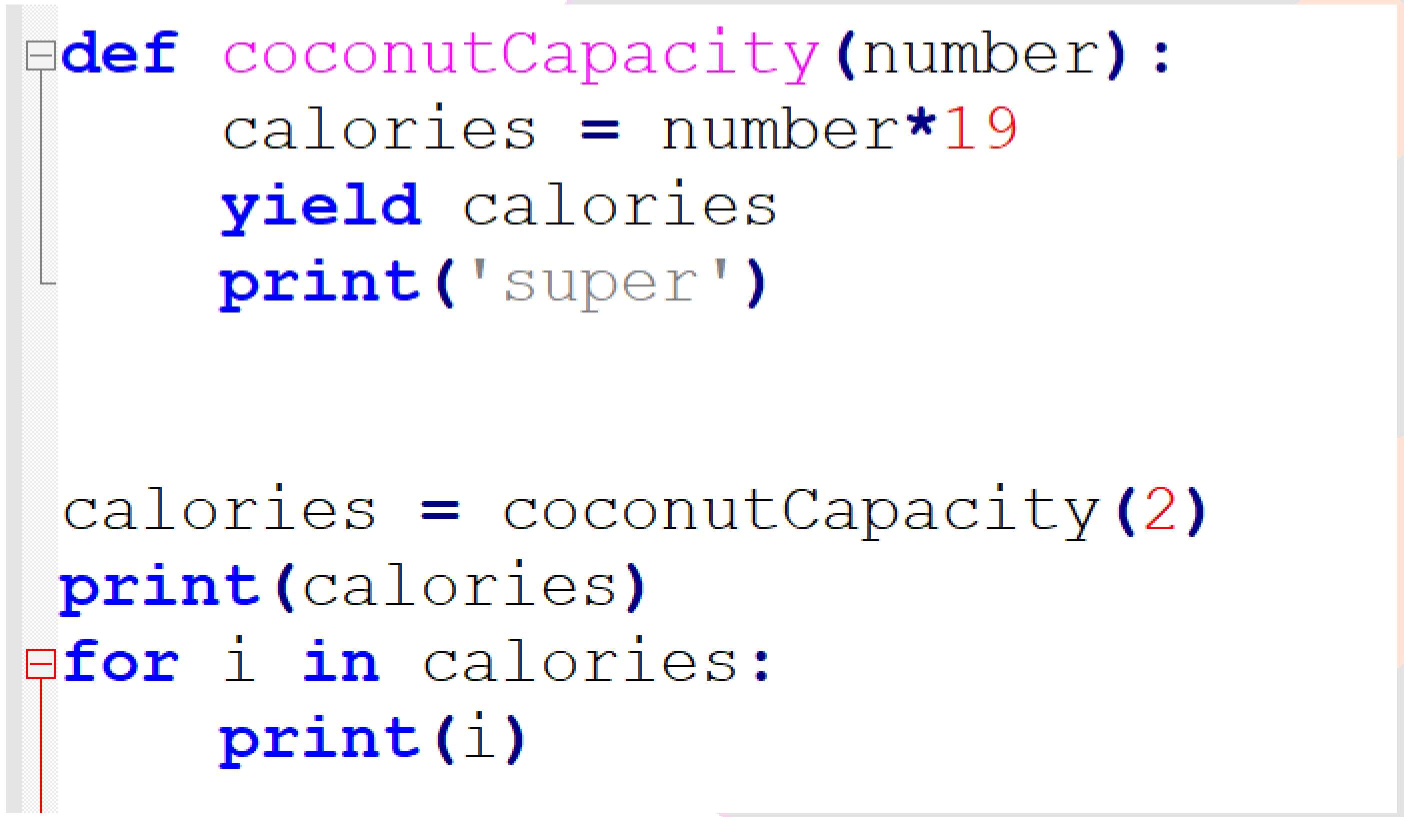
```
def coconutCapacity(number):
    calories = number*19
    yield calories
    print('super')


calories = coconutCapacity(2)
print(calories)
```



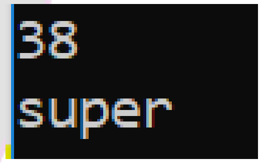
```
<generator object coconutCapacity at 0x00714F70>
```

Yield suspends functions execution and sends value back to the caller and then resumes where it left off.

yield



```python
def coconutCapacity(number):
    calories = number*19
    yield calories
    print('super')


calories = coconutCapacity(2)
print(calories)
for i in calories:
    print(i)
```



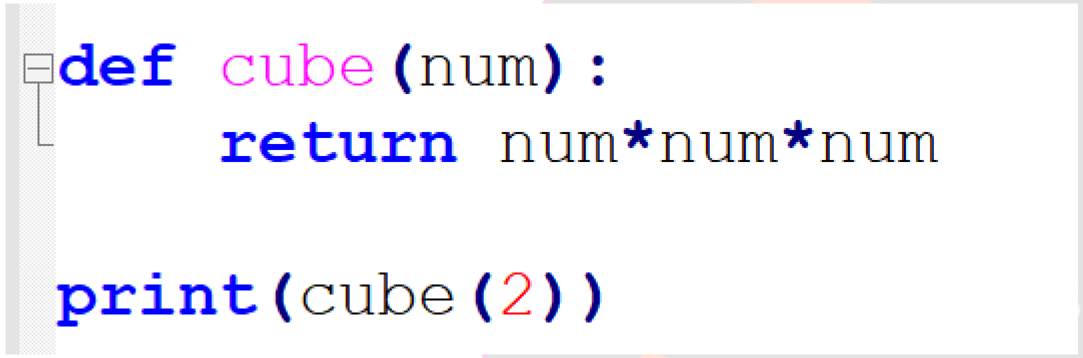
```
38
super
```

Anonymous function?(lambda)

- Anonymous meaning?
- Ex: Anonymous donation
- Ex: Anonymous author

- Anonymous: **unknown name or origin.**

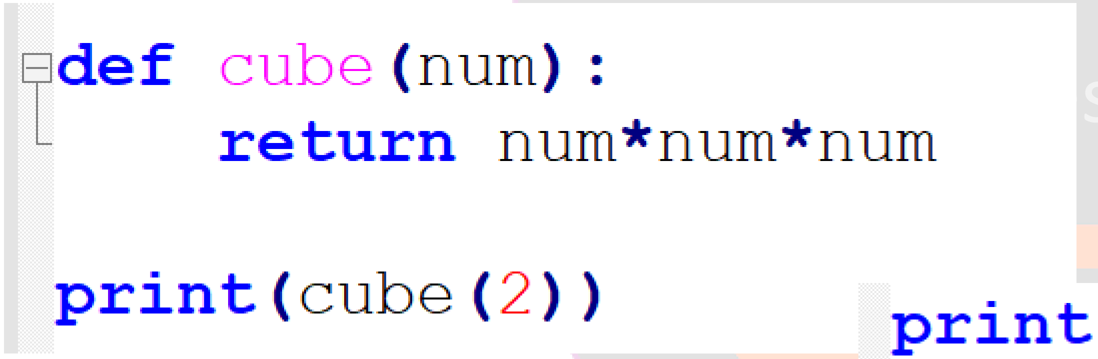
Normal function



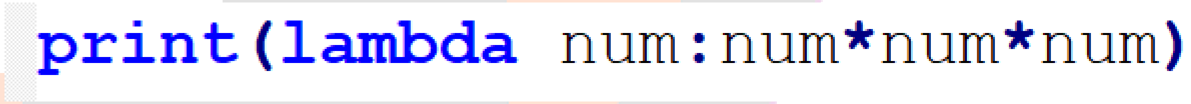
```
def cube(num):
    return num*num*num

print(cube(2))
```

lambda

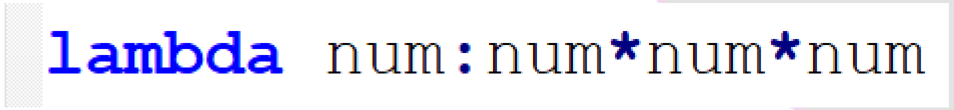
- lambda arguments : expression
- Function can have any number of arguments but only one expression



```
def cube(num):
    return num*num*num

print(cube(2))
```

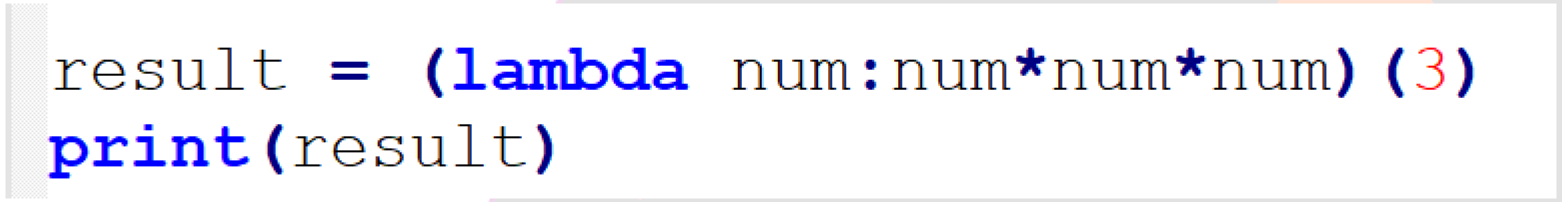


```
print(lambda num:num*num*num)
```

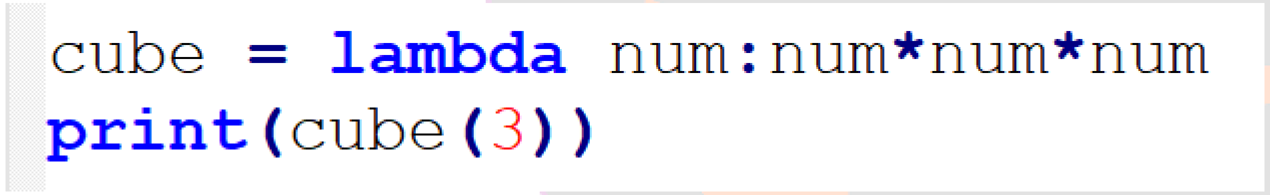


```
lambda num:num*num*num
```

Calling lambda function



```
result = (lambda num:num*num*num)(3)
print(result)
```



```
cube = lambda num:num*num*num
print(cube(3))
```

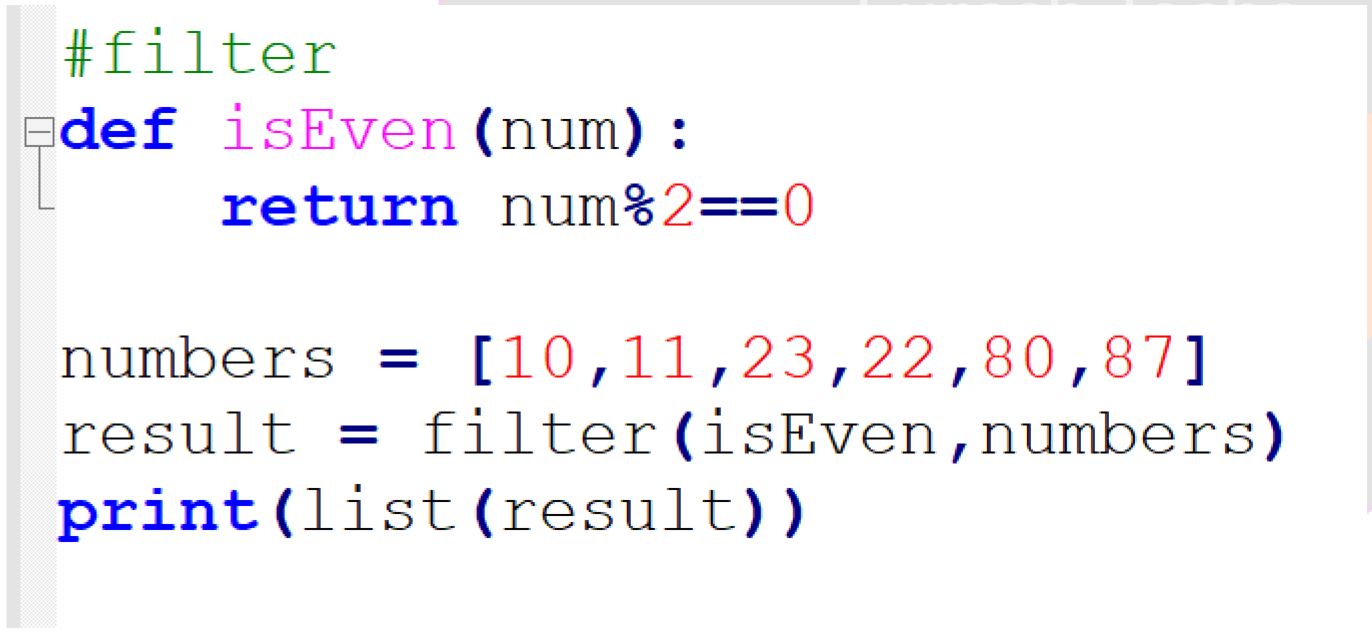
Filter, map, reduce

- Filter: filters the list based on some condition
- Map: iterates through all items in the list and returns list with new items
- Reduce: returns a single value

- Note: All of these methods takes a function and list

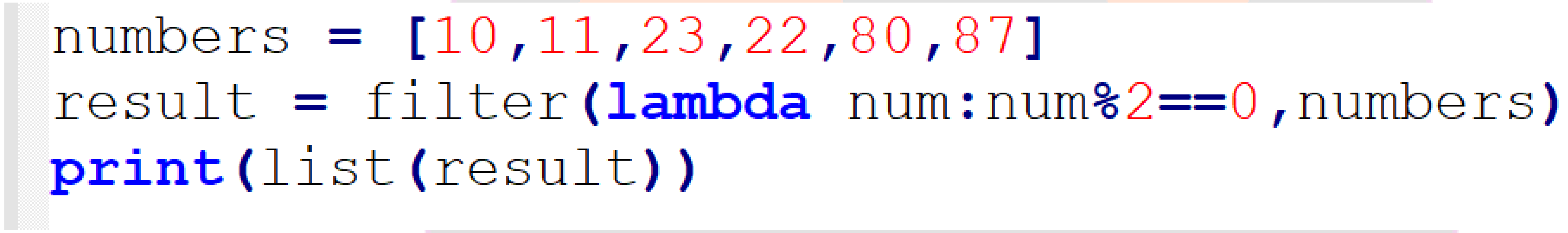
Filter

- Takes a **function** and iterable and creates a filtered list based on satisfying condition
- filter(function, iterable)



```python
#filter
def isEven(num):
    return num%2==0

numbers = [10,11,23,22,80,87]
result = filter(isEven,numbers)
print(list(result))
```

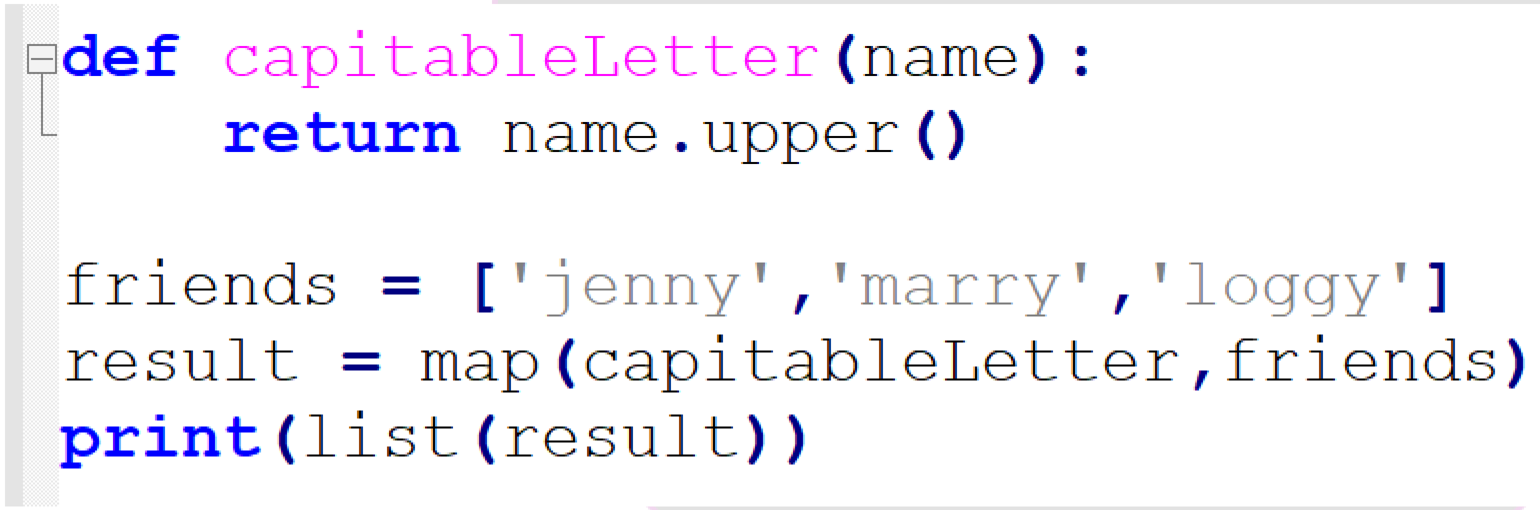
Let’s use anonymous function instead of isEven



```
numbers = [10,11,23,22,80,87]
result = filter(lambda num:num%2==0,numbers)
print(list(result))
```

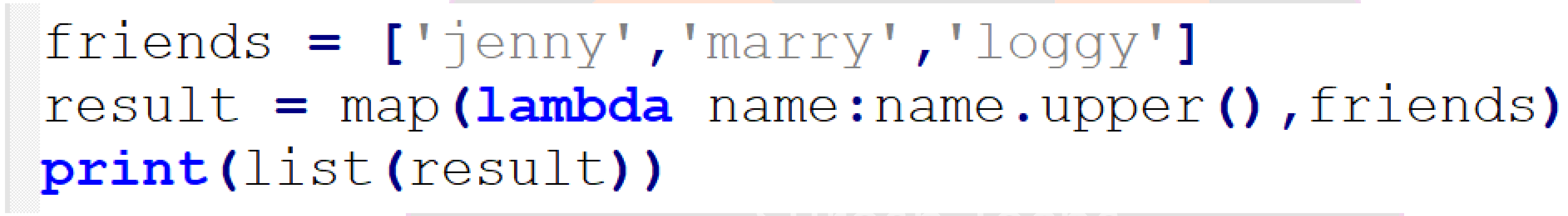
Map

- map(function, iterable)



```
def capitableLetter(name):
    return name.upper()

friends = ['jenny','marry','loggy']
result = map(capitableLetter,friends)
print(list(result))
```

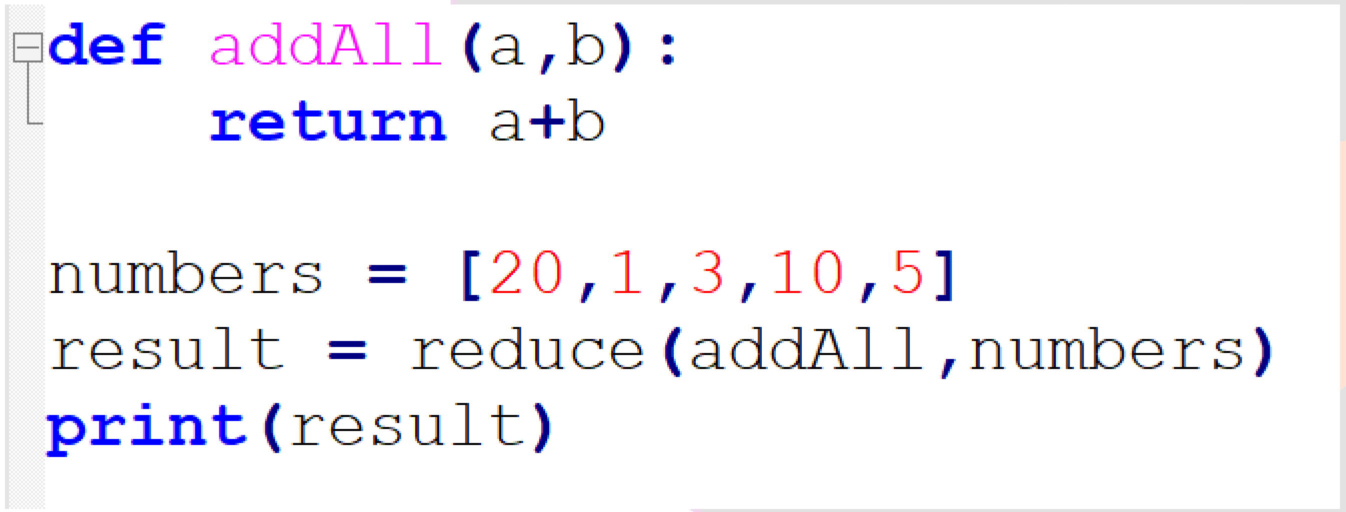
Let’s use lambda



```python
friends = ['jenny','marry','loggy']
result = map(lambda name:name.upper(),friends)
print(list(result))
```

reduce

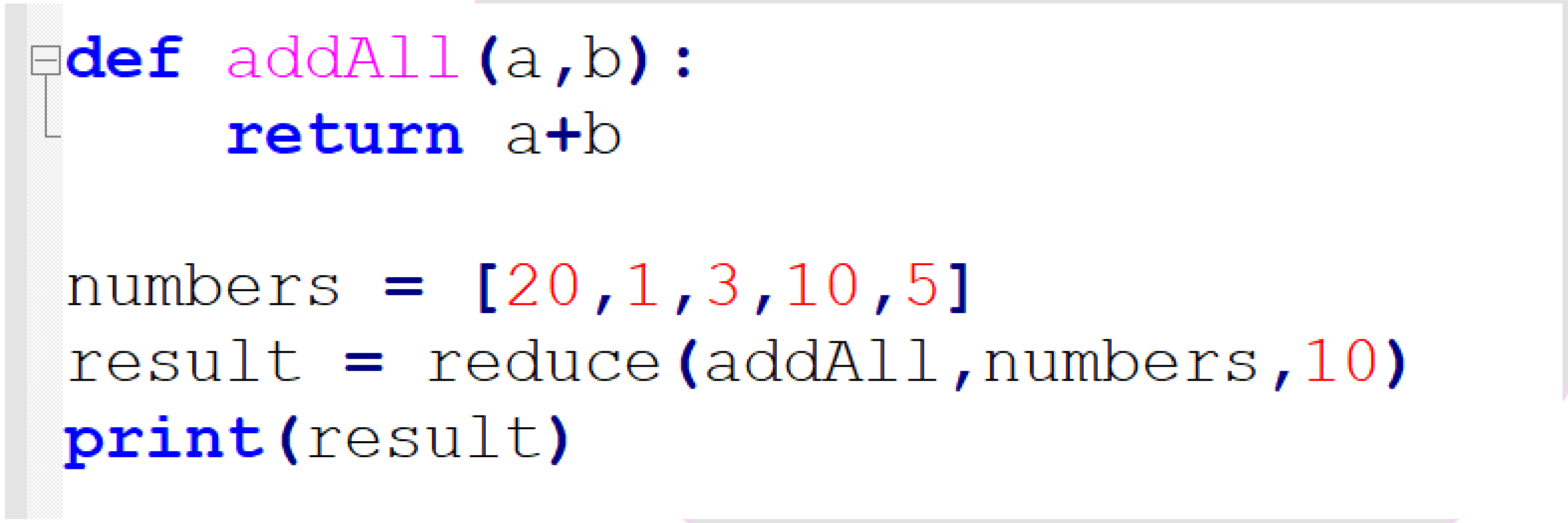
- Returns single value
- reduce(**function,sequence**,initial)



```
def addAll(a,b):
    return a+b

numbers = [20,1,3,10,5]
result = reduce(addAll,numbers)
print(result)
```

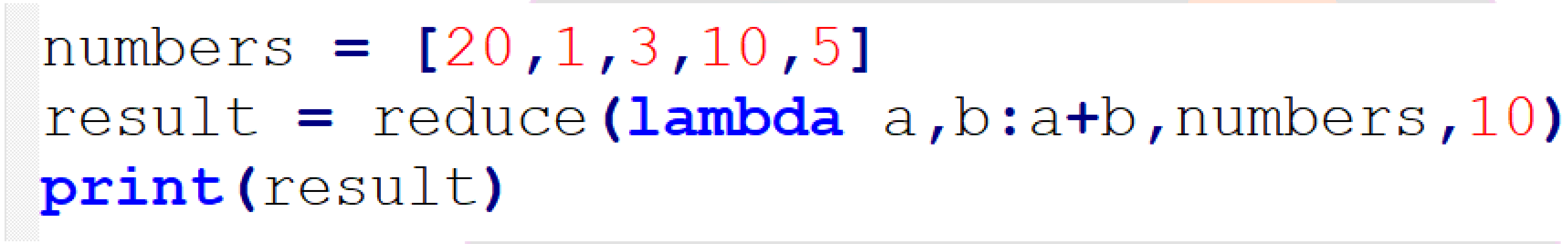
Should import reduce from functools from python 3

- from functools import reduce
- Initial value usage



```
def addAll(a,b):
    return a+b

numbers = [20,1,3,10,5]
result = reduce(addAll,numbers,10)
print(result)
```

Let’s convert to lambda



```
numbers = [20,1,3,10,5]
result = reduce(lambda a,b:a+b,numbers,10)
print(result)
```

Later

- Closures, decorators and first class functions will be discussed later.

PYTHON

చిన్న బ్రేక్

చిటికలో వచ్చేస్తా

#sureshtechs