In [2]:

```
"""Function breakdown a program into smaller and moduler chunk"""

def messagePrinter(message):
    """This function just print a message. The parameter is the user message and return
value is nothing"""
    print('The custorm message is: ' + message)

messagePrinter('Beachtung! Türen werden automatisch geöffnet')
```

```
The custorm message is: Beachtung! Türen werden automatisch geöffnet
```

In [6]:

```
#function with return type

def getSum(data):
    """This function calculate the summation of a give list and returns it sum value"""
    _sum = 0

    for element in data:
        _sum += element
    return _sum

myData = [1, 3, 5, 2, 4]
summation = getSum(myData)

print(getSum.__doc__) #we can print the function desxription comment using __doc__
print(summation)
```

```
This function calculate the summation of a give list and returns it sum va
lue
15
```

In [7]:

```
"""There are two types of functions: Standrad library function and user define functio
n. Here are some STD Functions"""

#abs function return the absolute value of a function

n = -100
print(abs(n))
```

```
100
```

In [8]:

```
#dir function returns all the attributes and functions of a data object

arr = []
print(dir(arr))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__
dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reverse
d__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__',
'__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index',
'insert', 'pop', 'remove', 'reverse', 'sort']
```

In [9]:

```
#divmod function takes parameters(d1, d2) and return a tuple(q, m) where d1=divident, d
2=divisor, q=quotient & m=modules

dm = divmod(7, 2)
print(dm)
```

```
(3, 1)
```

In [18]:

```
#enumerate function

data = [10, 20, 30, 40, 50]

ind = [i for i,val in enumerate(data)]

print(ind)
```

```
[0, 1, 2, 3, 4]
```

In [24]:

```
#filter function take two paremeter one is the custom function we use and another one i
s the sequence

def positiveFilter(num):
    """This function remove all the negative numbers from a list"""

    if num > 0:
        return num

numbers = range(-10, 10)
print(list(numbers))

pos_num = list(filter(positiveFilter, numbers))
print(pos_num)
```

```
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [3]:

```
#Map function is used to do any action over the list

def squareMaker(num):
    return num ** 2

numbers = [1, 2, 3, 4, 5]

squares = list(map(squareMaker, numbers))

print(squares)
```

```
[1, 4, 9, 16, 25]
```

In [5]:

```
#Reduce function apply the action in consequence elements in a list

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

from functools import reduce

numbers = [1, 2, 3, 4, 5]

mulOutput = reduce(multiplicationFunction, numbers)

print(mulOutput)
```

```
120
```

In [7]:

```
#Default Arguments

def printer(name, message="No message found"):
    print('Hello ' + name + ' Your message: ' + message)
    return;

#call by using two arguments
printer('Jaber', 'How are you?')

#call by using one argument
printer('Arif')
```

```
Hello Jaber Your message: How are you?
Hello Arif Your message: No message found
```

In [9]:

```
#Keyword Arguments: Allow to pass variable length of arguments

def printer1(**kwargs):
    if kwargs:
        print('Hello ' + kwargs['name'] + '. Your message: ' + kwargs['message'])
    return;

printer1(name = 'Pream', message='How are you?')
```

```
Hello Pream. Your message: How are you?
```

In [10]:

```
#Arbitary Arguments

def printer3(*names):
    for name in names:
        print(name)
    return;

printer3('Mahfuz', 'Azhar', 'Foysol', 'Maha')
```

```
Mahfuz
Azhar
Foysol
Maha
```

In [15]:

```
#Recursion

def factorial(n):
    return 1 if n==1 else (n * factorial(n-1))

n = 5
print('factorial of {} is = {}'.format(n, factorial(n)))
```

```
factorial of 5 is = 120
```

In [22]:

```
#Lambda Function

square = lambda x: x**2
#here, square is the name of the function, x is input and x**2 is the return statement

numbers = [1, 2, 3, 4, 5]

squareNumber = list(map(square, numbers)) #here, I pass the name of the lambda function
print(squareNumber)

cubeNumber = list(map(lambda x: x**3, numbers)) #here, I write the lambda expression in
the map function
print(cubeNumber)
```

```
[1, 4, 9, 16, 25]
[1, 8, 27, 64, 125]
```