Functions

- common set of codes that can be repititvely used.

In [3]:
```
def evenoddcheck(num):
    if num%2==0:
        print('your number is even')
    else:
        print('your number is odd')
```

In [6]:
```
result= evenoddcheck(47)
```

```
your number is even
```

In [7]:
```
result= evenoddcheck(42)
print('your number is {}'.format(result))
# because we are not returning anything from our evenoddcheck function we can see that t
he second output
#below shows result as 'None'
```

```
your number is even
your number is None
```

In [93]:
```
#example showing returning values from a function


def primecheck(num):
    for i in range(2,num):
        if num%i==0:
            return "non-prime"
    return "prime"
```

In [95]:
```
result1= primecheck(48)
print("your number is {}".format(result1))
```

```
your number is non-prime
```

Positional and Keyword arguments

- Keyword arguments are initiated in the function itself
- positional arguments receive their value from the location from which they are being called

In [17]:
```
def functio(positional, keyword="good"):
    print("{}, you are doing quite {}".format(positional, keyword))
```

In [18]:
```
functio("Mr. X")
```

```
Mr. X, you are doing quite good
```

Positional args as args and Keywordarguments as kwargs

In [20]:
```
def variety(*args, **kwargs):
    print(args)
    print(kwargs)
```

In [21]:
```
variety("Maruti", "Xcent", yearofpurchase= 1999, enginetype='geared')
```

```
('Maruti', 'Xcent')
{'yearofpurchase': 1999, 'enginetype': 'geared'}
```

Lets try reverse engineering on our data and function

lets say we have a list and a dictionary definition which we wish to pass to our variety function

In [23]:
```
list1= ['Maruti', 'Xcent']
dict_def={'yearofpurchase': 1999, 'enginetype': 'geared'}

#what if we wish to pass these 2 variables to our function which should receive these as
#positional and keyword arguments

#can we use the below call? let's try



variety(list1, dict_def)
```

```
(['Maruti', 'Xcent'], {'yearofpurchase': 1999, 'enginetype': 'geared'})
{}
```

In [24]:
```
#the above output shows that our variety function received both
#of our variables as positional argument while the keyword argument remained null as {}

#so, below is the code that should be used to specify which variable should be going as
 positional arg
#and which should be going as keyword argument

variety(*list1, **dict_def)
```

```
('Maruti', 'Xcent')
{'yearofpurchase': 1999, 'enginetype': 'geared'}
```

Returning multiple values using single return

In [36]:
```
def expon(num):
    square=num*num
    cube=num*num*num
    return  square, cube
```

In [37]:
```
print(expon(3))
```

```
(9, 27)
```

Lambda Functions

- function without a name or anonymous functions
- works faster
- used when we have a single line of code to be defined

In [34]:
```
square= lambda i:i*i
square(12)
```

Out[34]:
```
144
```

In [38]:
```
sum= lambda a,b,c,d:a+b+c+d
sum(5,10,3,2)
```

Out[38]:
```
20
```

In [40]:
```
evencheck= lambda num:(num%2==0)
evencheck(25)
```

Out[40]:
```
False
```

Map()

In [100]:
```
def evenoddcheck_(num):
    if num%2==0:
        print('{} is even'.format(num))
        return True
    else:
        print('{} is odd'.format(num))
```

In [101]:
```
list1=[2,3,4,5,6,7,7]
list(map(evenoddcheck_, list1)) #instantiating memory locs by using list() funcn
```

```
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is odd
7 is odd
```

Out[101]:
```
[True, None, True, None, True, None, None]
```

Filter()

In [91]:
```
def prime(num):
    for i in range(2,num):
        a=num%i
        if a==0:
            return False
    return True
```

In [92]:
```
list4=[3,37,6,31,8,9,79]
list(filter(prime, list4))
```

Out[92]: [3, 37, 31, 79]

In [99]:
```
list4=[3,37,6,31,8,9,79]
list(map(prime, list4))
```

Out[99]: [True, True, False, True, False, False, True]

In [98]:
```
#even check

list(filter(lambda num:num%2==0, list4))
```

Out[98]: [6, 8]

In [102]:
```
#even check

list(map(lambda num:num%2==0, list4))
```

Out[102]: [False, False, True, False, True, False, False]

List comprehension

- shortcut to creating lists
- inside brackets: an expression first, and then any number of for or if clauses

In [103]:
```
list10=[i*i for i in range(1,8) if i%2==0]
```

In [104]:
```
list10
```

Out[104]: [4, 16, 36]

In [106]:
```
list11=[i*i*i for i in list10]
```

In [107]:
```
list11
```

Out[107]: [64, 4096, 46656]