Map functions

It allows you to "map" a function to an iterable object. i.e. you can call same function to every item in an iterable.

```
In: def square(num):
        return num**2

In: mynums = [1,2,3,4]

In: map(square, mynums)

o/p: <map at 0x205....>
```

To get results, iterate through map or call to a list

```
In: list(map(square, mynums))

o/p: [1,4,9,16]
```

Filter function

It returns an iterator yielding those items of iterable for which func^n(item) is true. i.e. you need to filter by a func^n that returns either true or false.

Then passing that into filter along with your iterable) & you'll get back only the results that would return true when passed to the func^n.

In: def check_even(num):
    return num%2==0

In: nums = [0, 1, 2, 3, 4, 5, 6]

In: list(filter(check_even, nums))

O/P: [0, 2, 4, 6]

Lambda Expression

They allow us to create anonymous func^n. without proper definition using def.
They work same as those created & assigned using def

Note: Lambda's body is similar to what we put

Note: Lambda is a single expression not a block of statements

qn: def square(num): return num**2

qn: square(2)

o/p: 4

qn: lambda n : n**2

o/p: <function main.<lambda>>

qn: # You wouldn't ~~assign~~ usually assign a name to a lamda expression, this is for demonstration

qn: square(2)

o/p: 4

Passing to MAP & FILTER

~~qn: mynums = [~~

qn: mynums = [1, 2, 3, 4, 5]

qn: list(map(lambda n: n**2, mynums))

o/p: [1, 4, 9, 16, 25]

qn: list(filter(lambda n: n%2==0, mynums))

o/p: [2, 4]

You can also pass multiple arg.

for eg:

lambda x,y: x+y