

Higher Order Functions

```
In [1]: x = 24
```

```
In [2]: def square(x):  
        return x*x
```

```
In [3]: square(6)
```

```
Out[3]: 36
```

```
In [4]: square
```

```
Out[4]: <function __main__.square>
```

```
In [5]: f = square
```

```
In [6]: f
```

```
Out[6]: <function __main__.square>
```

```
In [7]: f(6)
```

```
Out[7]: 36
```

So, `f` is just another name for `square`.

Simple Example

Let's say we want to write a function for calculating sums:

$$\sum_{i=1}^{10} i^2$$

```
In [8]: def summation(low, high):  
        total = 0  
        for i in range(low, high+1):  
            val = i ** 2  
            total += val  
  
        return total
```

```
In [9]: summation(1, 3)
```

```
Out[9]: 14
```

Now, if I ask you to write another function that does this: $\sum_{i=1}^{10} i^3$

```
In [10]: def summation(low, high):
          total = 0
          for i in range(low, high+1):
              val = i ** 3
              total += val

          return total
```

But what's the difference? Only the `val` has changed. Can't we just write one function and have you decide how val needs to be calculated?

```
In [11]: def square(x):
          return x ** 2
          def cube(x):
              return x ** 3
```

```
In [12]: def summation(low, high, fn):
          total = 0
          for i in range(low, high+1):
              val = fn(i)
              total += val

          return total
```

Now, we can call summation and just change the function that calculates `val`.

```
In [13]: summation(1, 2, square)
```

```
Out[13]: 5
```

```
In [14]: summation(1, 2, cube)
```

```
Out[14]: 9
```

We don't even have to name functions!

```
In [15]: summation(1, 2, lambda i: i**2 )    # This is an anonymous function.
```

```
Out[15]: 5
```

Notice that this statement now looks very similar to the actual math notation we had earlier:

$$\sum_{i=1}^{10} i^2$$

So, if you have to write another one for this:

$$\sum_{i=1}^{10} 2i^2$$

```
In [16]: summation(1, 10, lambda i: 2*(i**2) )    # no need to define a new summation
```

```
Out[16]: 770
```

Summation is a "higher order function" because it takes another function as its input.

Case Study: Square Roots

```
In [20]: def sqrt(x, guess=0.1):
          print("Trying:", guess, "-- Value:", guess*guess)
          if good_enough(guess, x):
              return guess

          else:
              guess = improve_guess(guess, x)
              return sqrt(x, guess)
```

```
In [17]: # Defined by the weather people
          def good_enough(guess, x):
              if abs(guess * guess - x) < 1:
                  return True
              else:
                  return False
```

```
In [18]: def avg(a, b):
          return (a + b) / 2.0

          def improve_guess(guess, x):
              return avg(guess, float(x)/guess)
```

```
In [21]: sqrt(36)
```

```
Trying: 0.1 -- Value: 0.010000000000000002
Trying: 180.05 -- Value: 32418.002500000002
Trying: 90.12497222993613 -- Value: 8122.510619446759
Trying: 45.26220878399787 -- Value: 2048.667544006214
Trying: 23.028787149504215 -- Value: 530.3250375771704
Trying: 12.296023969924157 -- Value: 151.19220546894942
Trying: 7.611899827408357 -- Value: 57.94101898249938
Trying: 6.170668368771986 -- Value: 38.07714811736313
Trying: 6.002360173190208 -- Value: 36.028327648699985
```

```
Out[21]: 6.002360173190208
```

```
In [22]: def sqrt(x, is_ge=good_enough, guess=0.1):  
        print("Trying:", guess, "-- Value:", guess*guess)  
        if is_ge(guess, x):  
            return guess  
  
        else:  
            guess = improve_guess(guess, x)  
            return sqrt(x, is_ge, guess)
```

```
In [23]: # By the nuclear reactor people  
def very_accurate_good_enough(guess, x):  
    return abs(guess * guess - x) < 0.000000001
```

```
In [24]: sqrt(36, is_ge=very_accurate_good_enough)
```

```
Trying: 0.1 -- Value: 0.010000000000000002  
Trying: 180.05 -- Value: 32418.002500000002  
Trying: 90.12497222993613 -- Value: 8122.510619446759  
Trying: 45.26220878399787 -- Value: 2048.667544006214  
Trying: 23.028787149504215 -- Value: 530.3250375771704  
Trying: 12.296023969924157 -- Value: 151.19220546894942  
Trying: 7.611899827408357 -- Value: 57.94101898249938  
Trying: 6.170668368771986 -- Value: 38.07714811736313  
Trying: 6.002360173190208 -- Value: 36.028327648699985  
Trying: 6.00000046401893 -- Value: 36.00000556822737  
Trying: 6.000000000000018 -- Value: 36.00000000000021
```

```
Out[24]: 6.000000000000018
```

The variable `is_ge` is what is termed as a "*Callback*" -- some function that you send to another piece of code. That piece of code calls this function back at a later point in time.

This is the concept around which half of modern javascript is built!