Functions and Parameters

Now we're going to work with **functions and parameters**. So to start off I'm going to set us up with some example data:

```
result = 1 + 3
print(result)
```

Let's create a function that can perform this kind of addition for us. To start off we going to do `def` (which means "definition"):

```
def add(n1, n2):
```

`add` is a function's name; `n1` and `n2` are arguments that it takes. Now the body of the function:

```
def add(n1, n2):
    result = (n1 + n2)
    return result
```

Using `return` we return the result. Now use the function:

```
add(1,3)
```

Now we're going to work with a **lambda**. And that is actually going to assign a function into a variable which we're going to then execute:

```
myfuncresult = lambda n1, n2: n1 + n2
print(myfuncresult(1,3))
```

Once again `n1` and `n2` are arguments and `n1 + n2` is the body of this function.

This lambda will implicitly return a value, and we don't have to worry about actually doing a return.

'myfuncresult'

Now create another lambda:

```
myfunc2result = map(lambda x: x*2, [1,4,20])
print(list(myfunc2result))
```

`x * 2` is multiplication.

Map an Object

Our `map` is working with a list `[1,4,20]`. It takes the number from the list multiplying it by 2, then takes the next number and so on.

Now prepare some sample data for the next example:

```
customer1 = "John Doe"
pickupLocation1 = "350 5th Ave"
package1 = customer1 + pickupLocation1
```

```
customer2 = "Jane Doe"
pickupLocation2 = "100 7th Ave"
package2 = customer2 + pickupLocation2

customer3 = "Joe Daniels"
pickupLocation3 = "11 1st Ave"
package4 = customer3 + pickupLocation3

customerList = [customer1, customer2, customer3]
```

We'll code a function `calculateFee` that'll take in miles as the parameter and calculate fee based on it.

Create Function 'calculateFee'

```
def calculateFee(miles):
    result = miles * .5 + 2.50
    return result

print(calculateFee(10))
```

Now suppose we want to be able to set the rate. However, if the rate is not passed, it should use the default value:

```
def calculateFee(miles, initialRate = 2.50):
    result = miles * .5 + initialRate
    return result

print(calculateFee(10, 3))
```

`initialRate = 2.50` means that this is an argument with a default argument. If this argument is not set, `2.50` will be used.

'calculateMultiCustomerRate'

```
def calculateMultiCustomerRate(*miles):
    rates = []
    for m in miles:
        rates.append(m * 2.50)
    return rates
```

`*miles` means that the function may accept any number of arguments and all of them will be stored inside the `miles` list. Inside the function we traverse this list and simply calculate the rates.

This function will return a list as well, so use it like this:

```
rates = calculateMultiCustomerRate(10,20,30)
i = 1
for r in rates:
    print("customer %d = $%d" % (i, r))
    i = i + 1
```

`i` is our indexer that's being incremented on every loop.

Note how we format the output: `%d` marks the place to interpolate a value. Values are being stored inside the `(i, r)` tuple. Of course, you may use string concatenation, but using this kind of formatting is simpler and more clear.