Functional Programming in Python

Topics:

- Functions as arguments
- Nested Functions
- Functions as return values
- Operators
- Decorators
- Decorators & arguments

Higher order Functions

- Any function that operates on other functions
- In python, almost everything is an object — even functions

```
l_factorial = lambda n: 1 if n == 0 else n*l_factorial(n-1)
```

Gauging how long our function takes

```
import time
t0 = time.time()
l_factorial(900)
t1 = time.time()
print('took: %.5f s' % (t1-t0))
```

- This works, but it’s time consuming and goes line by line. Let’s make it functional

Functional Way

```
def timer(fnc, arg):
  t0 = time.time()
  fnc(arg
  t1 = time.time()
  return t1-t0

print('took: %.5f s' % timer(l_factorial, 900))
```

- With this solution, we’re able to make it so **any** function can be timed. This is an example of a higher order function in Python

Implementing it with only lambda function

```
l_timestamp = lambda fnc, arg: (time.time(), fnc(arg), time.time())
l_diff = lambda t0, retval, t1: t1-t0
l_timer = lambda fnc, arg: l_diff(*l_timestamp(fnc, arg))
```

- I hate lambdas
- They seem very confusing to me
- Functional programming is not my jam whatsoever.