Closures

Wikipedia says, "A closure is a record storing a function together with an environment: a mapping associating each variable of the function with the value or storage location to which the name was bound when the closure was created. A closure, unlike a plain function, allows the function to access those captured variables through the closure's reference to them, even when the function is invoked outside their scope

Example

```
def outer_func():
    message = 'Hi'

    def inner_func():
        print(message)

    return inner_func()

outer_func()
```

② Assigned message variable to value 'Hi'

③ We created this inner function, which just prints the message variable

④ returning and executing the inner_func()
- will print the message

① Executed outer_fun()

Return --> Hi ✓

```
def outer_func():
    message = 'Hi'

    def inner_func():
        print(message)

    return inner_func

my_func = outer_func()

print(my_func)
print(my_func.__name__)
```

no "()", return function w/o executing it.

my_func is a function equal to inner_func

Return --> <function outer_func.<locals>.inner_func at 0x11b4d1c80>
inner_func

```
def outer_func():
    message = 'Hi'

    def inner_func():
        print(message)

    return inner_func

my_func = outer_func()

my_func()
my_func()
my_func()
```

} Execute it a few times, prints Hi

Return --> Hi
Hi
Hi

A closure is an inner function that remembers and has access to variables in the local scope in which is was created even after the outer function has finished executing.

Parameters

A closure closes over the free variables from its environment
~e.g. the free variable in this example would be message.

```
def outer_func(msg):
    message = msg

    def inner_func():
        print(message)

    return inner_func

hi_func = outer_func('Hi')

hello_func = outer_func('Hello')

hi_func()
hello_func()
```

passing in a parameter

running

Return --> Hi
Hello

JavaScript Example:

```
function html_tag(tag) {
  function wrap_text(msg){
    console.log('<' + tag + '>' + msg + '</' + tag + '>')
  }
  return wrap_text

}

print_h1 = html_tag('h1')

print_h1('Test Headline!')

print_h1('Another Headline!')

print_p = html_tag('p')
print_p('Test Paragraph!')
```

Setting print_h1 to outer html_tag function

"Inner function does take one parameter"

Results:

<hi>Test Headline!</hi>
<hi>Another Headline!</hi>
Test Paragraph!

Complex Example

```
import logging

logging.basicConfig(filename ='example.log', level = logging.INFO)

def logger(func):
    def log_func(*args):
        logging.info('Running "{}" with arguments {}'.format(func.__name__, args))
        print(func(*args))

    return log_func

def add(x,y):
    return x+y

def sub(x, y):
    return x-y

add_logger = logger(add)
sub_logger = logger(sub)

add_logger(3,3)
add_logger(4,5)

sub_logger(10, 5)
sub_logger(20, 10)
```

Returned: 6
9
5
10

Created on example.log

tracked