Introduction to Decorators in Python

Storing function objects in other variables an invoking them

Given a simple function...

In [4]:
```
def hello():
    """Print 'hello' to the console."""
    print "hello"
```

In [5]:
```
hello()
```

```
hello
```

In [6]:
```
print hello
```

```
<function hello at 0x27da938>
```

We can store the actual function object in a variable.

In [7]:
```
my_function = hello
print my_function
print hello
```

```
<function hello at 0x27da938>
<function hello at 0x27da938>
```

And we can invoke our function object.

In [8]:
```
my_function()
```

```
hello
```

Passing a function to another function

In [9]:
```
def wrapper(function):
    """Higher order function that prints its arguments.

    Args:
        function (function): Function to print to the console.

    """
    print "My argument is: ", function
```

In [10]:
```
wrapper(hello)
```

```
My argument is:  <function hello at 0x27da938>
```

Executing functions inside functions

In [13]:
```
def outer():
    """Dummy outer function."""
    print "outer"

    def inner():
        """Dummy inner function."""
        print "inner"

    inner()
```

In [12]:
```
outer()
```

```
outer
inner
```

Our first simple decorator

In [14]:
```
def print_info(function):
    """Execute some print statements before and after invoking function."""

    def wrapper():
        """Wrap function execution by printing to the console."""
        print "Before function."
        function()
        print "After function."

    return wrapper
```

In [18]:
```
@print_info
def world():
    """Print 'world' to the console."""
    print "world"
```

In [16]:
```
my_func = print_info(world)
my_func()
```

```
Before function.
world
After function.
```

In [19]:
```
world()
```

```
Before function.
world
After function.
```

In [ ]: