Introduction to Decorators: Takeaways

Syntax

NESTED FUNCTIONS

- Create a nested function:

```
def parent():
    def child():
        pass
    return child
```

CLOSURES

- Check the number of variables in a function's closure:

```
len(func.__closure__)
```

- View the value of the variables in a function's closure by accessing the "cell_contents" of the item:

```
func.__closure__[0].cell_contents
```

DECORATORS

- To use a decorator, type the "@" symbol followed by the decorator's name on the line directly above the function:

```
@double_args
def multiply(a, b):
    return a * b
```

Concepts

- **Decorators** are functions that take a function as an argument and return a modified version of that function. In order to work, decorators have to make use of the following concepts:
    - Functions as objects
    - Nested functions
    - Nonlocal scope
    - Closures
- Since a function is just an object like anything else in Python, we can pass one as an argument to another function.
- A **nested function** is a function defined inside another function. We sometimes refer to the outer function as the parent and the nested function as the child.
- **Scope** determines which variables can be accessed at different points in your code.
- The **local scope** is made up of the arguments and any variables defined inside the function. If the interpreter can't find the variable in the local scope, it expands its search to the **global scope**. These are the things defined outside the function. Finally, if it can't find the thing it is looking for in the global scope, the interpreter checks the **builtin** scope. These are things that are always available in Python. In the case of nested functions, where one function is defined inside another function, Python will check the scope of the parent function before checking the global scope. This is called the **nonlocal scope**.
- A **nonlocal variable** is any variable that gets defined in the parent function's scope and that gets used by the child function.
- A **closure** is Python's way of attaching nonlocal variables to a returned function so that the function can operate even when it is called outside of its parent's scope.

Resources

- Python documentation for decorators