Decorators

Decorators

- Let’s now discuss a more advanced Python topic: Decorators.
- Decorators allow you to “decorate” a function, let’s discuss what that word means in this context.

Decorators

- Imagine you created a function:

```
def simple_func():
    # Do simple stuff
    return something
```

Decorators

- Now you want to add some new capabilities to the function:

```
def simple_func():
    # Want to do more stuff!
    # Do simple stuff
    return something
```

Decorators

- You now have two options:
  - Add that extra code (functionality) to your old function.
  - Create a brand new function that contains the old code, and then add new code to that.

Decorators

- But what if you then want to remove that extra “functionality”.
- You would need to delete it manually, or make sure to have the old function.
- Is there a better way? Maybe an on/off switch to quickly add this functionality?

Decorators

- Python has **decorators** that allow you to tack on extra functionality to an already existing function.
- They use the **@** operator and are then placed on top of the original function.

Decorators

- Now you can easily add on extra functionality with a decorator:

```
@some_decorator
def simple_func():
    # Do simple stuff
    return something
```

Decorators

- This idea is pretty abstract in practice with Python syntax, so we will go through the steps of manually building out a decorator ourselves, to show what the @ operator is doing behind the scenes.

Decorators

- Keep in mind you won’t encounter decorators in basic Python code.
- They are common to encounter when working with Web Frameworks, such as Django or Flask.

GUI Example

Decorators