Python: how to use multiple decorators on one function

Jose Salvatierra
2 min read · Oct 5

Hopefully you've learned about decorators in Python already! This blog post will talk about what happens when you use more than one decorator on a function.

For example:

```
@user_has_permission
@user_name_starts_with_j
def double_decorator():
    return 'I ran.'
```

Decorators are most often used to extend a function, by adding something either before or after it.

What happens in a decorator internally is that it changes the function it decorates. It changes the function and converts it into a new function with the added functionality.

So if we had this:

```python
@user_name_starts_with_j
def decorated_function():
    return 'I ran.'
```

The `decorated_function` no longer does just `return 'I ran.'`. Now it has turned into whatever the decorator returns. Hopefully, the decorator

returns something that uses the `decorated_function` so that it still returns `'I ran.'`!
Let's say the `user_name_starts_with_j` decorator does two things:

1. It checks whether the username (whatever that might be) starts with the letter `'j'`.
2. If it does, then it calls `decorated_function`. Otherwise, it prints an error.

From the moment that `decorated_function` is defined, it has changed into a function that does the above. Now whenever we call `decorated_function()`, if the username does not start with `'j'`, we'll get an error. We don't have to use `user_name_starts_with_j` again.

Two decorators

When you have two decorators, the same thing applies. Let's take this code as an example:

```python
@user_has_permission
@user_name_starts_with_j
```

```
def double_decorator():
    return 'I ran.'
```

First, `@user_name_starts_with_j` modifies the `double_decorator` function.

Then, `@user_has_permission` modifies the result of the previous modification.

`double_decorator` has now become the function that `user_has_permission` returned. It's a function that:

1. Checks the user has correct permission.
2. If they do, call the original function. Otherwise, print an error.

So when we call `double_decorator()`, the above happens first of all. If the user has correct permissions, we call the original function—which is the result of the previous transformation.

Therefore, we then check that the username starts with j. If they do, we call the original function (and return `'I ran.'`). Otherwise, we print another error.

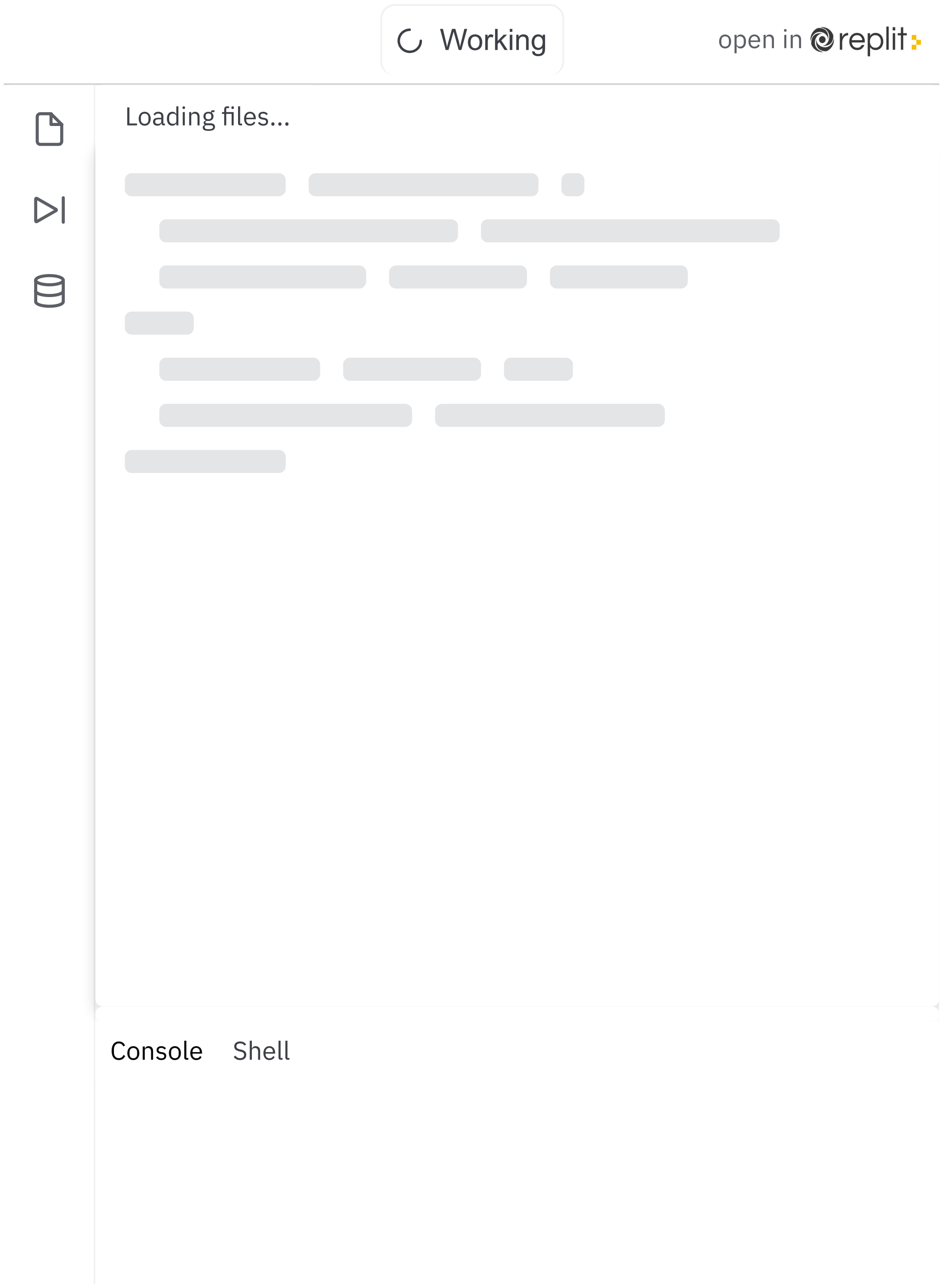
So the order of checks with two decorators has become:

1. Check permission.
2. Check username starts with j.
3. Run original function.

As you can see, the decorators run in the inverse order in which they are used to decorate the function—because of the way they work internally.

A code example

Below you can play around with the example code. I've defined a few `user` variables with different usernames and access levels that you can try. You can also run this code in your browser to see the output!



Thanks for reading! If you'd like to take your Python skills to the next level, please check out our Complete Python Course!

Jose Salvatierra
I'm a software engineer turned instructor! I founded Teclado to help me do this for everyone. I hope you enjoy the content!

Read More

Learn Python Programming

The @property decorator in Python →

How to simplify long if statements in Python →

Basics of Python a Beginner Should Know - Part 1 →

→ See all 149 posts

LEARN PYTHON PROGRAMMING

Monthly Python Talk

I've been collecting interesting Python talks, for students of all skill levels. Come back every month for a new one!

Jose Salvatierra
4 min read · Oct 15