Building a Pipeline Class: Takeaways

Syntax

- Adding an arbitrary number of arguments:

```
def add(*args):
    parent_args = args
    def inner(*inner_args):
        return sum(parent_args + inner_args)
    return inner
add_nine = add(1, 3, 5)
print(add_nine(2, 4, 6))
 # prints 21
```

- Using a decorator:

```
def logger(func):
    def inner(*args):
        print('Calling function: {}'.format(func.__name__))
        print('With args: {}'.format(args))
        return func(*args)
    return inner
@logger
def add(a, b):
    return a + b
print(add(1, 2))
 # 'Calling function: add'
 # 'With args: (1, 2)'
 #  3
```

Concepts

- An inner function is a function within a function. The benefit of these inner functions is that they are encapsulated in the scope of the parent function.
- A closure is defined by an inner function that has access to its parent's variables. We can pass any number of arguments from the parent function down to the inner function using the `*` character.
- A decorator is a Python callable object that modifies the behavior of any function, method, or class.
- The `StringIO` object mimics a file-like object that keeps a file-like object in memory.

Resources

- io module
- A guide to Python's function decorators