Python Techniques That Will Make Your Professor Accuse You of Academic Dishonesty

Ethan Clark and Joshua Ganschow

April 8, 2025

Introduction

Together, we're going to learn some secret techniques in Python that *might* get you accused of academic dishonesty. We'll start off with some tame examples of what you can do, then get further into the weeds

Introduction

Together, we're going to learn some secret techniques in Python that *might* get you accused of academic dishonesty. We'll start off with some tame examples of what you can do, then get further into the weeds

Disclaimer: We assume no responsibility if you use these and get accused.

Reinventing the Plus Operator

```
class Strange:
    def __init__(self, value):
        self.value = value
    def __add__(self, other):
        # Instead of summing, multiply the values
        return Strange(self.value * other.value)

a = Strange(3)
b = Strange(4)
print((a + b).value)  # Outputs 12 instead of 7
```

One-Liner Examples

```
nums = [1, 2, 3, 4, 5]
squared_evens = [x*x for x in nums if x % 2 == 0]
```

- Creates a list of numbers.
- Uses a list comprehension to square each even number.

```
flat = [item for row in matrix for item in row]
```

- Flattens a 2D list (matrix) into a 1D list.

Using eval and exec

```
string_to_eval = "2*10"
eval(string_to_eval)
code = "print('Hello from exec')"
exec(code)
```

Recursive Lambda for Factorial Calculation

```
factorial = (
    lambda f: lambda x: 1 if x == 0 else x * f(f)(x -
        1))
    (
    lambda f: lambda x: 1 if x == 0 else x * f(f)(x -
        1)
)
print(factorial(5))
```

- The outer lambda accepts a function, and the inner lambda performs the recursion.
- The self-application trick (passing the lambda to itself) allows it to call itself recursively.

Function Decorator

```
def evil_decorator(func):
    def wrapper(*args, **kwargs):
        print("Before execution")
        result = func(*args, **kwargs)
        print("After execution")
        return result
    return wrapper

@evil_decorator
def greet(name):
    print(f"Hello, {name}!")

greet("World")
```

Metaclasses for Adding Magic

```
class Meta(type):
    def __new__(self, name, bases, dct):
        dct['added_attribute'] = 'Mystery'
        return super().__new__(self, name, bases, dct)

class sub(metaclass=Meta):
    pass

print(sub.added_attribute)
```

When new is called

- `name` is the name of the class being created
- `bases` is a tuple of base classes
- `dct` is a ditionary containing the class's attributes and methods

Changing Behavior at Runtime

```
import math

original_sqrt = math.sqrt

math.sqrt = lambda x: 'Cheated sqrt!'

print(math.sqrt(16))

math.sqrt = original_sqrt
```

- Temporarily replaces the `math.sqrt` function with a lambda that returns a fixed string.
- Restores the original function afterwards.

Inspecting Function Source Code

```
import inspect

def sample_function(a, b):
    return a + b

# Print the source code of sample_function
print(inspect.getsource(sample_function))
```

Closing Thoughts

- Python is powerful; apply it as such
- Maybe avoid some of these tricks if you can help it