

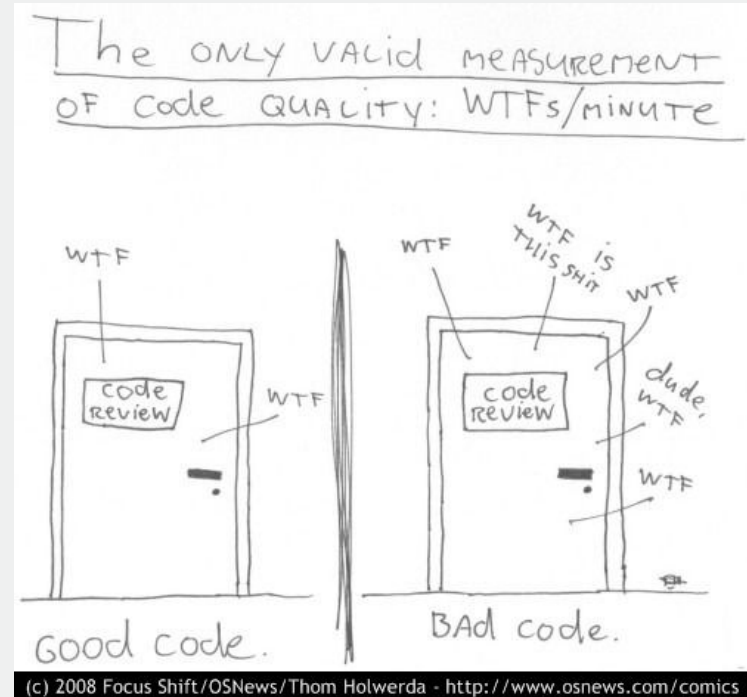


LAB #6

**Adv. CiC
Spring 2026**

Agenda

1. Housekeeping
2. Organization
3. Complexity
4. Continuous Integration
5. Lab 6 Exercise



Housekeeping

Due this past week

- Reading 6
- Lab 5

*Just added Project Groups to Courseworks! Apologies if that caused any weirdness. Need to do for **Lab Groups** too.*

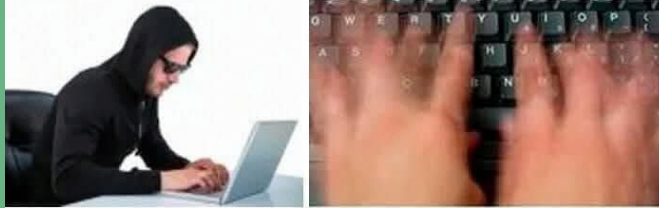
Upcoming

1. Reading 7 - **3/5 @ 3pm**
2. Lab 6 - **3/6 @ 1pm**

Additional Resources

- Online tools
 - Google, ChatGPT, etc.
 - ***Make sure to review AI policy!***
- **ED!!!**
- Office hours
 - 1 hr/week set time
 - Appts by email

What people think programming
is like



What programming is actually like



Questions
(on Lectures, Labs, or Readings)?

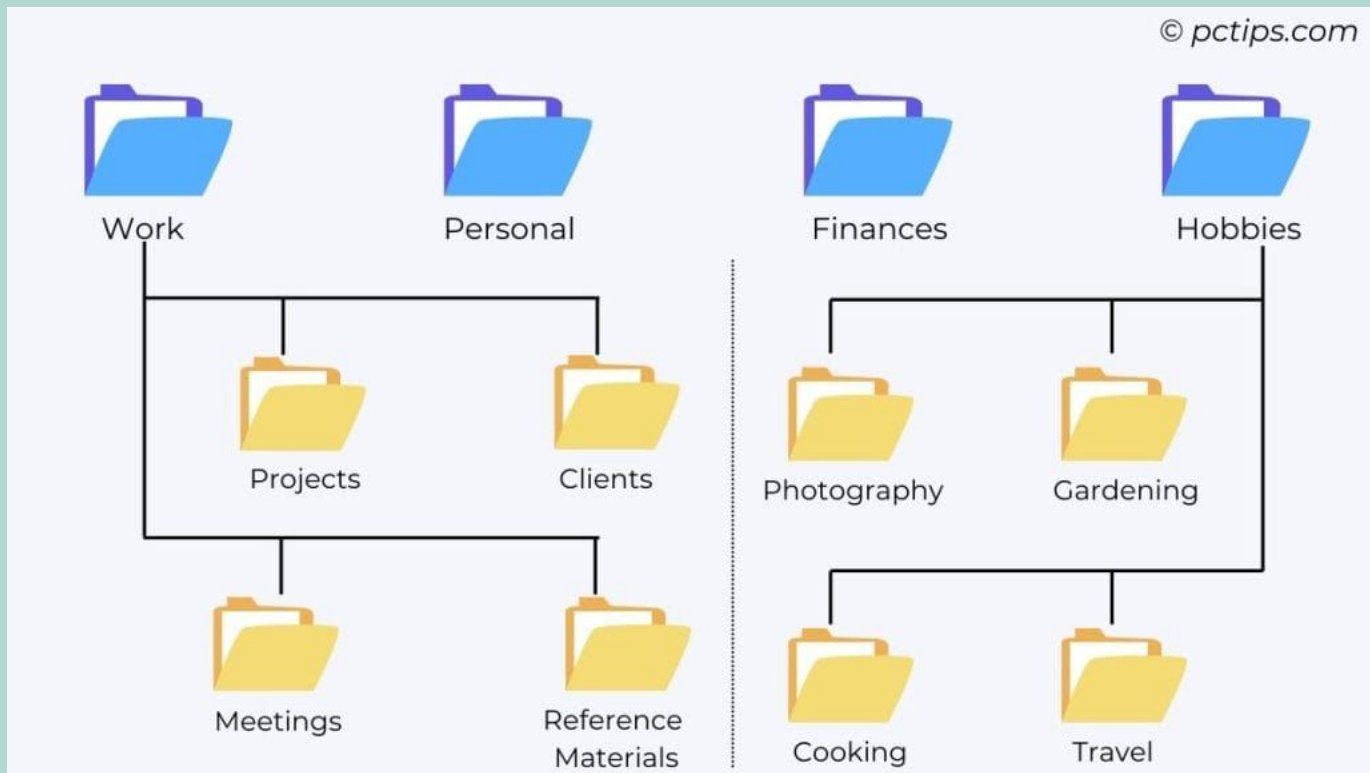
The background is a solid dark olive green. It is decorated with several geometric elements: a light blue rounded rectangle in the top left, a yellow rounded rectangle in the top center, a light green rounded rectangle in the top right, a light blue rounded rectangle in the bottom right, a yellow rounded rectangle in the bottom left, and three white-outlined rounded rectangles scattered in the center and bottom. A large, light green rounded rectangle is centered horizontally, containing the word "Organization" in a bold, dark blue font.

Organization

Why is it important to organize your code?

Organizing your code makes it easier to *read, maintain, debug, test, and collaborate on* - while reducing errors and duplication over time

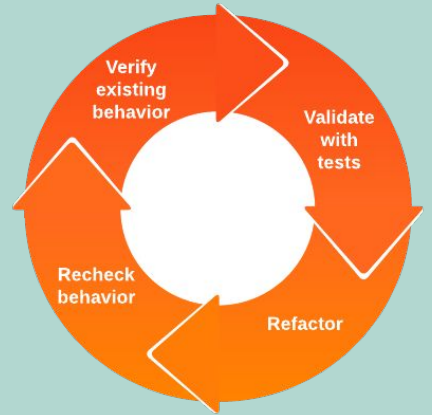
File Structure



Refactoring

“Refactoring is the process of restructuring code, while not changing its original functionality”

techtarget.com



Refactoring

REFACTORING

Improving Code Design Without Changing Behavior

BEFORE

```
def calculate(items):  
    total = 0  
    for item in items:  
        if item['type'] == 'book':  
            total += item['price'] * 0.9  
        else:  
            total += item['price']  
    return total
```

AFTER

```
class Calculator:  
    def get_total(self, items):  
        return sum(self.get_price(item)  
                    for item in items)  
  
    def get_price(self, item):  
        if item['type'] == 'book':  
            return item['price'] * 0.9  
        return item['price']
```

Classes

Syntax of a Python Class:

```
[ ]:
```

```
class ClassName:
    # Class variables
    class_variable = value

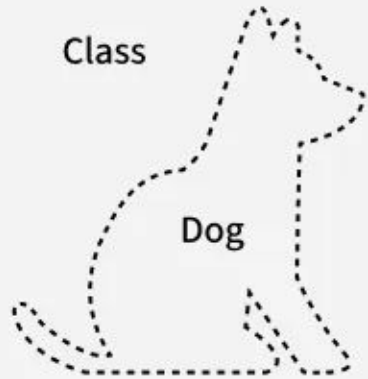
    # Constructor
    def __init__(self, parameters):
        self.instance_variable = parameters

    # Instance method
    def method_name(self, parameters):
        # method body

#clcoding.com
```



Classes



Create Instance



Properties

Name

Colour

Eye_Colour

Height

Length

Methods

getName()

getColour()

getEyeColour()

getHeight()

comeHere()

Property Values

Name : Tommy

Colour : Green

Eye_Colour : Brown

Height : 17in

Length : 35in

Methods

getName()

getColour()

getEyeColour()

getHeight()

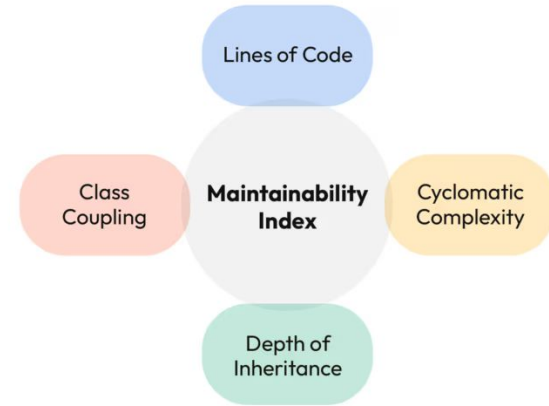
comeHere()



Complexity

Types of Code Complexity

- Code complexity refers to **how difficult code is to understand, reason about, test, and modify** without introducing bugs
- **Different types:** Includes cyclomatic complexity (number of decision paths), cognitive complexity (how hard it is for humans to read), and structural complexity (how tightly components are coupled)
- **Good practices:** Keep functions small and focused, reduce nested logic, follow clear naming conventions, and refactor regularly to keep code simple and modular



Low Complexity

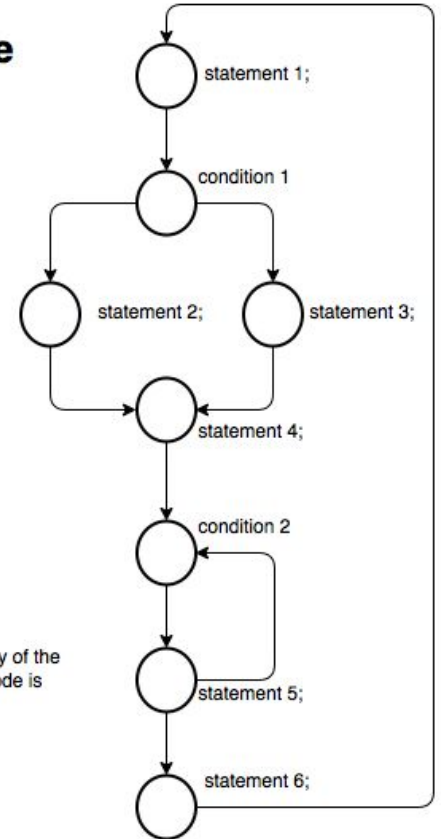
```
def calculate_total(prices)
    return sum(prices)
```

High Complexity

```
def calculate_total(prices)
    total = 0
    for price as prices {
        if (price >= 0) {
            while price >= 0 {
                total += p 1
            }
        }
        return total
    }
```

Sample code

```
statement 1;
If ( condition 1 ) {
    statement 2;
} else {
    statement 3;
}
statement 4;
for( condition 2 ) {
    statement 5;
}
statement 6;
```



Complexity

The cyclomatic complexity of the graph representing the code is

$$v(G) = E - N + 2$$

$$= 9 - 8 + 2$$

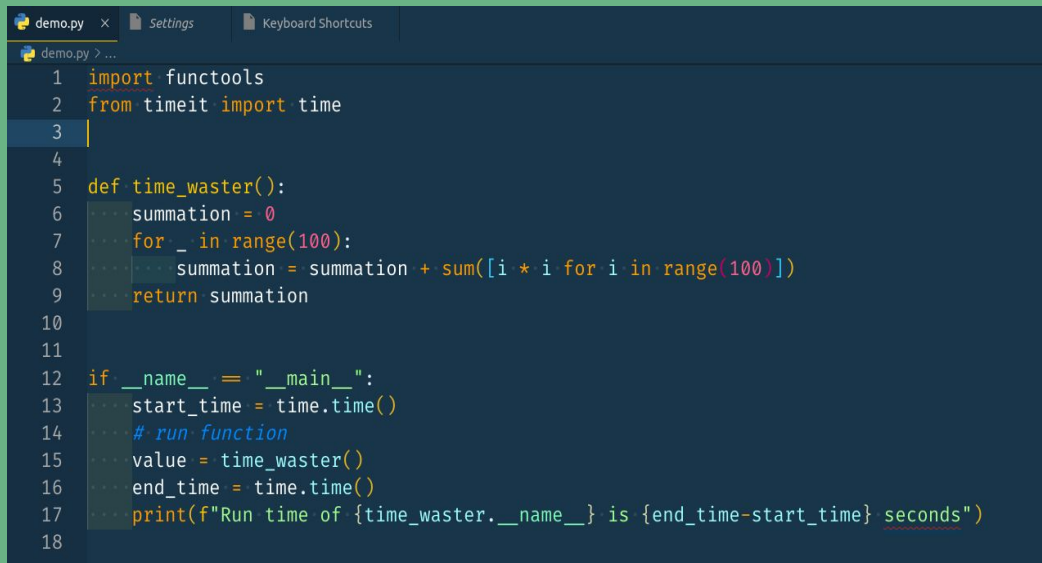
$$= 3$$

The background is a solid green color. It features several decorative elements: a light blue rounded rectangle in the top left, a yellow rounded rectangle in the top right, a light blue rounded rectangle in the bottom right, and a yellow rounded rectangle in the bottom left. There are also several thin, rounded rectangular outlines in yellow and dark blue scattered across the background.

Continuous Integration

Linting

- **What it is:** Linting is the process of **automatically checking code** for syntax errors, stylistic issues, and potential bugs
- **Why it matters:** It enforces **consistent formatting and best practices**, helping teams catch problems early and maintain clean, readable code



```
demo.py x  Settings  Keyboard Shortcuts
demo.py > ...
1  import functools
2  from timeit import time
3  |
4
5  def time_waster():
6      summation = 0
7      for _ in range(100):
8          summation = summation + sum([i * i for i in range(100)])
9      return summation
10
11
12 if __name__ == "__main__":
13     start_time = time.time()
14     # run function
15     value = time_waster()
16     end_time = time.time()
17     print(f"Run time of {time_waster.__name__} is {end_time-start_time} seconds")
18
```

Automated Tests



Manual Testing

- Human-executed
- Flexible & intuitive
- Good for exploratory testing
- Time-consuming
- Prone to human error
- Best for usability & UI testing

VS

Automated Testing

- Script-driven
- Fast & repeatable
- Ideal for regression testing
- High initial setup time
- Requires programming skills
- Best for performance & load testing

Data Validation

- Data validation is the process of **checking that inputs or datasets meet expected formats**, types, and constraints before being used in a program
- **Why it matters:** It prevents errors, improves reliability, and protects systems from bad or malicious data

```
from pandera import check_input
from pandera import DataFrameSchema, Column, Check

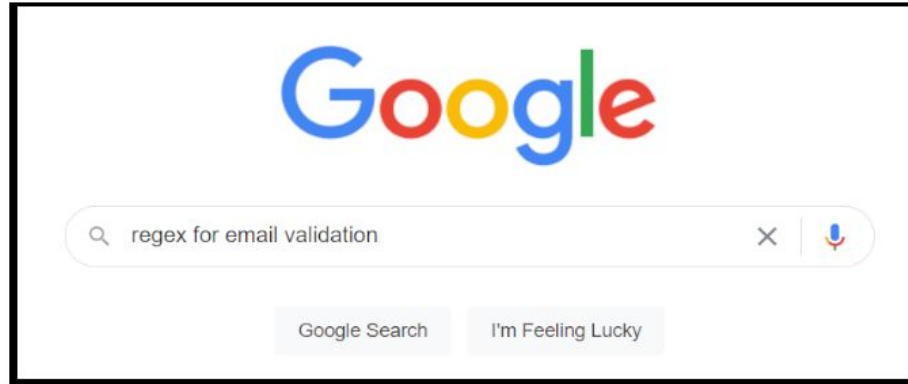
student_schema = DataFrameSchema(
    {
        "name": Column(str),
        "age": Column(int, Check.between(0, 120)),
        "score": Column(float, Check.between(0, 100)),
    }
)

@check_input(student_schema)
def calculate_grade(data: pd.DataFrame):
    data["grade"] = pd.cut(
        data["score"], bins=[0, 70, 80, 90, 100],
        labels=["F", "C", "B", "A"],
    )
    return data

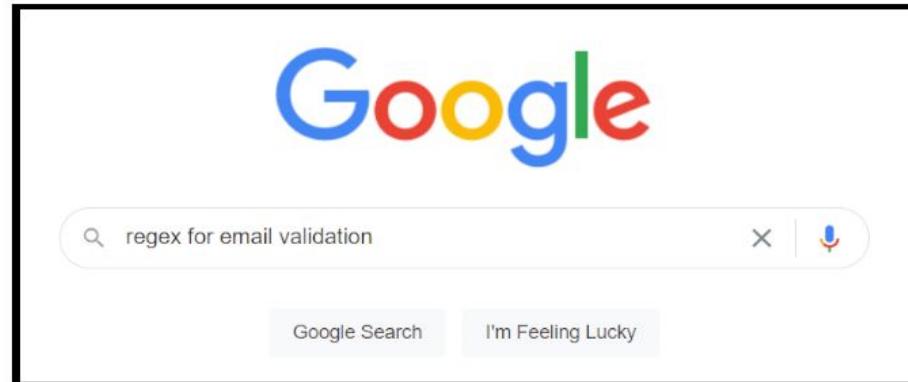
student_df = pd.DataFrame(
    {
        "name": ["John", "Jane", "Bob"],
        "age": [25, 30, 200], # invalid input
        "score": [95.5, 88.3, 92.7],
    }
)

student_schema.validate(student_df)
"""
SchemaError: Column 'age' failed element-wise
validator number 0: in_range(0, 120) failure cases: 200
"""
```

DAY1 OF PROGRAMMING

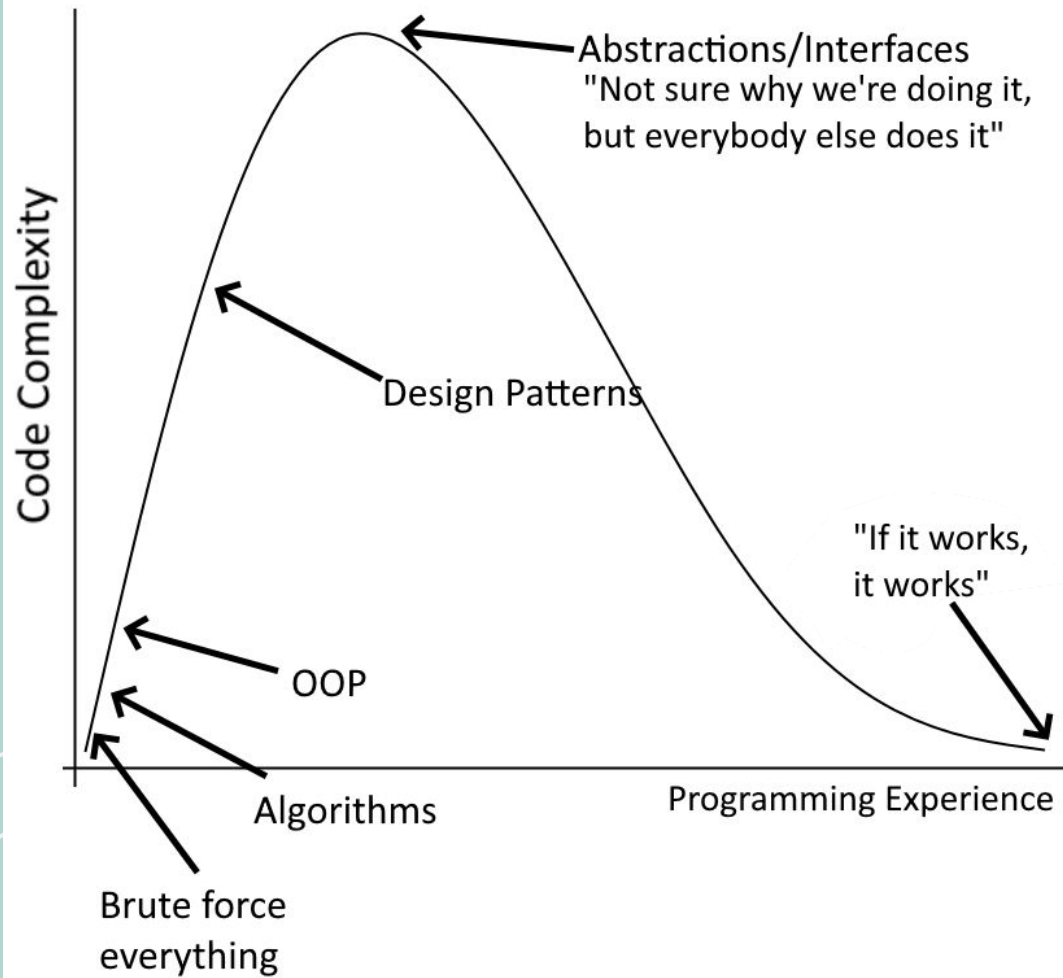


10 YEARS OF PROGRAMMING





**"Just keep coding.
We can always fix it later."**





**EXERCIS
E**

Lfg