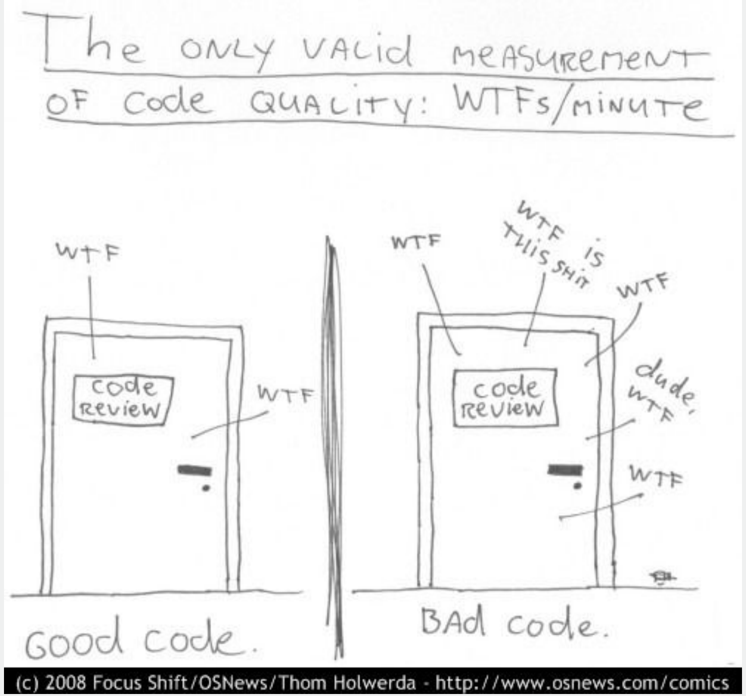
LAB #6

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

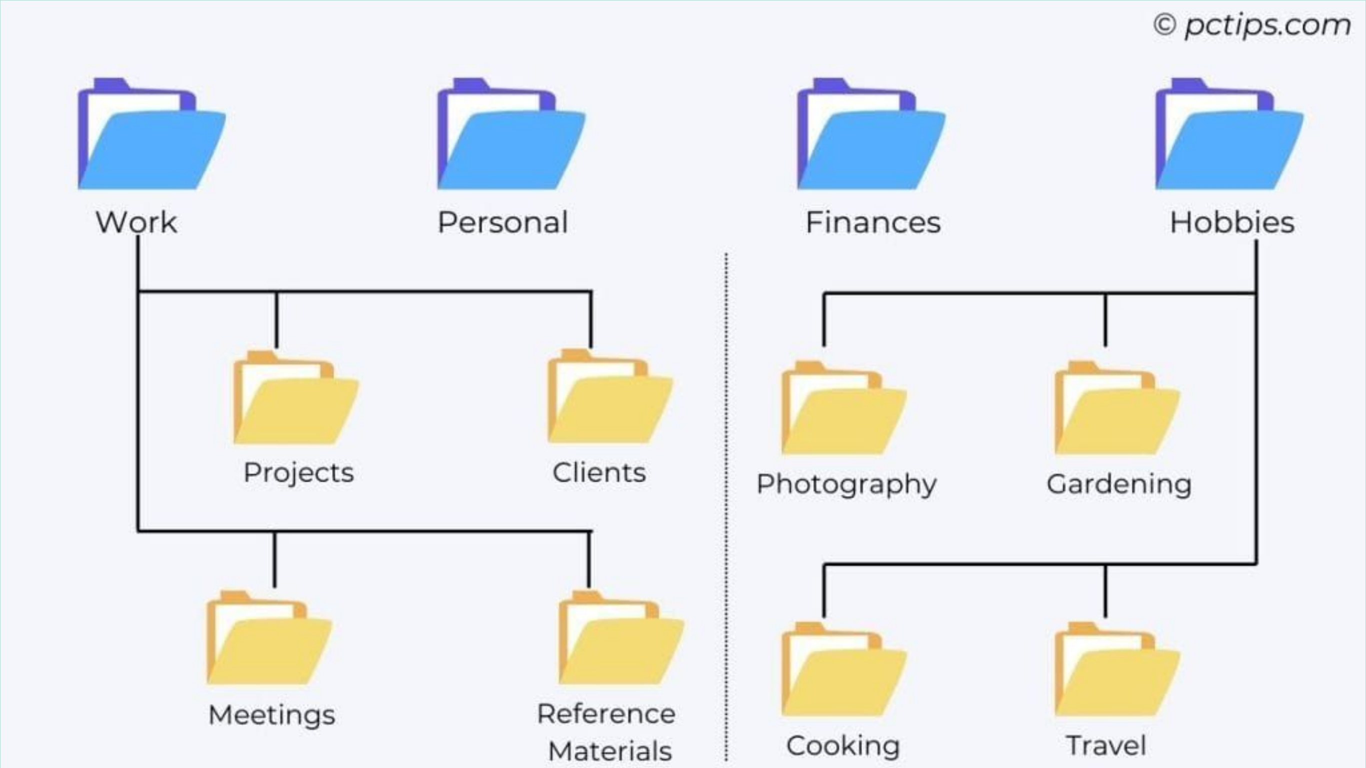
Questions
(on Lectures, Labs, or Readings)?

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

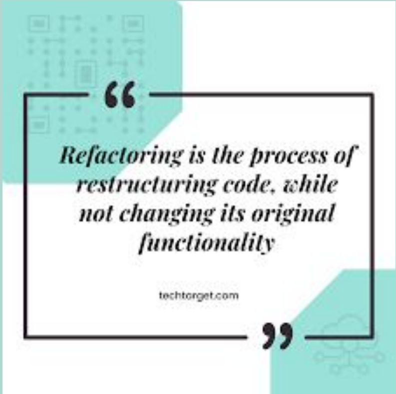
File Structure

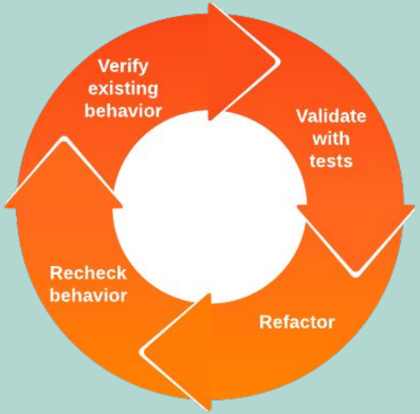


File Structure

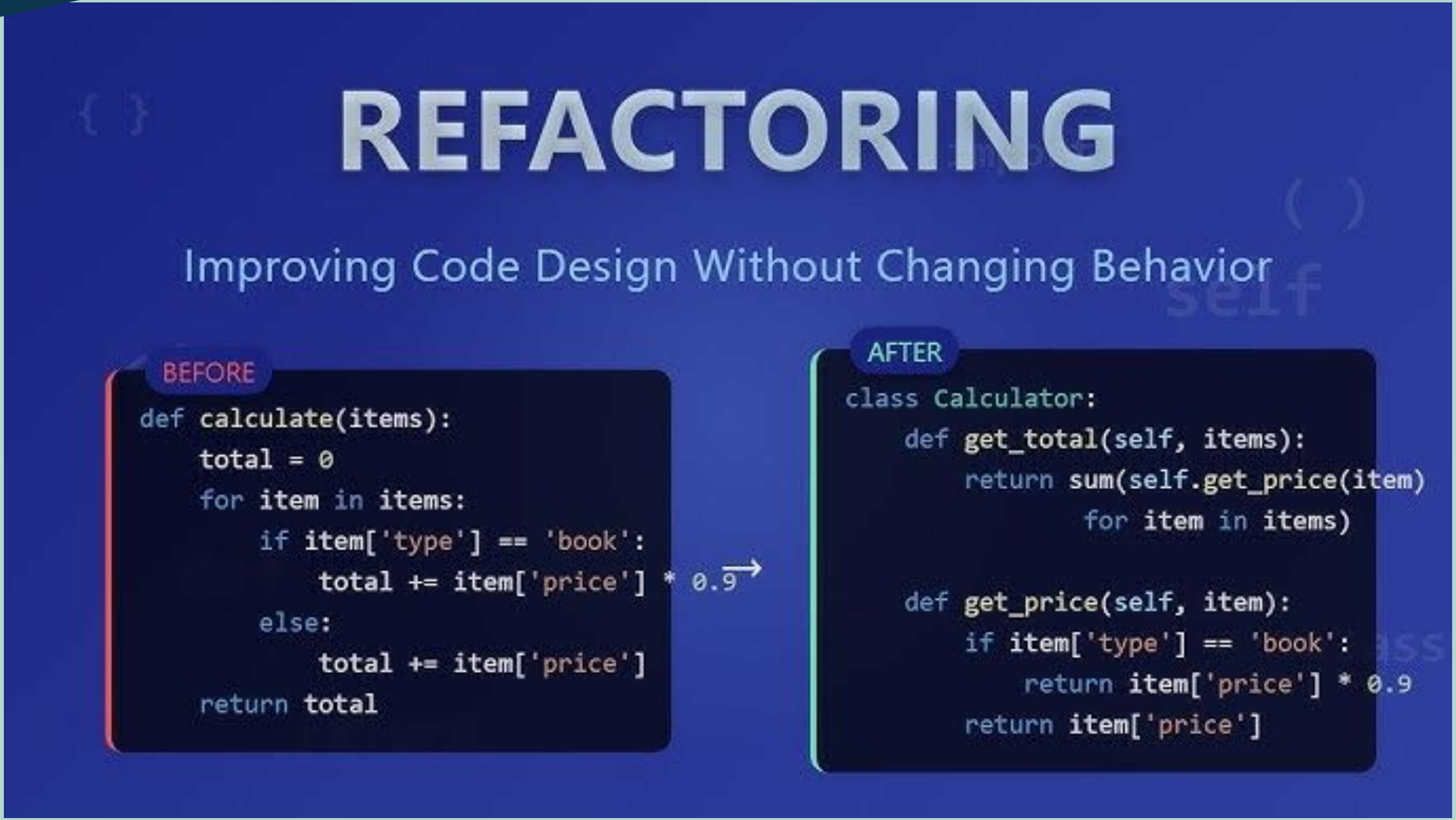
```
project_name/
│
├── src/                        # Main source code lives here
│   ├── __init__.py
│   ├── main.py                 # Entry point of your project
│   ├── config/                 # Configuration files
│   │   ├── __init__.py
│   │   └── settings.py
│   ├── utils/                  # Helper functions and utilities
│   │   ├── __init__.py
│   │   └── logger.py
│   ├── modules/                # Core app modules
│   │   ├── __init__.py
│   │   └── data_processor.py
│   └── services/               # External service integrations
│       ├── __init__.py
│       └── api_handler.py
│
├── tests/                      # Unit and integration tests
│   ├── __init__.py
│   └── test_main.py
│
├── requirements.txt            # Project dependencies
├── README.md                   # Documentation
├── .gitignore                  # Ignore unnecessary files
├── pyproject.toml              # Modern Python build configuration
└── setup.py                    # For packaging (optional)
```

Refactoring





Refactoring



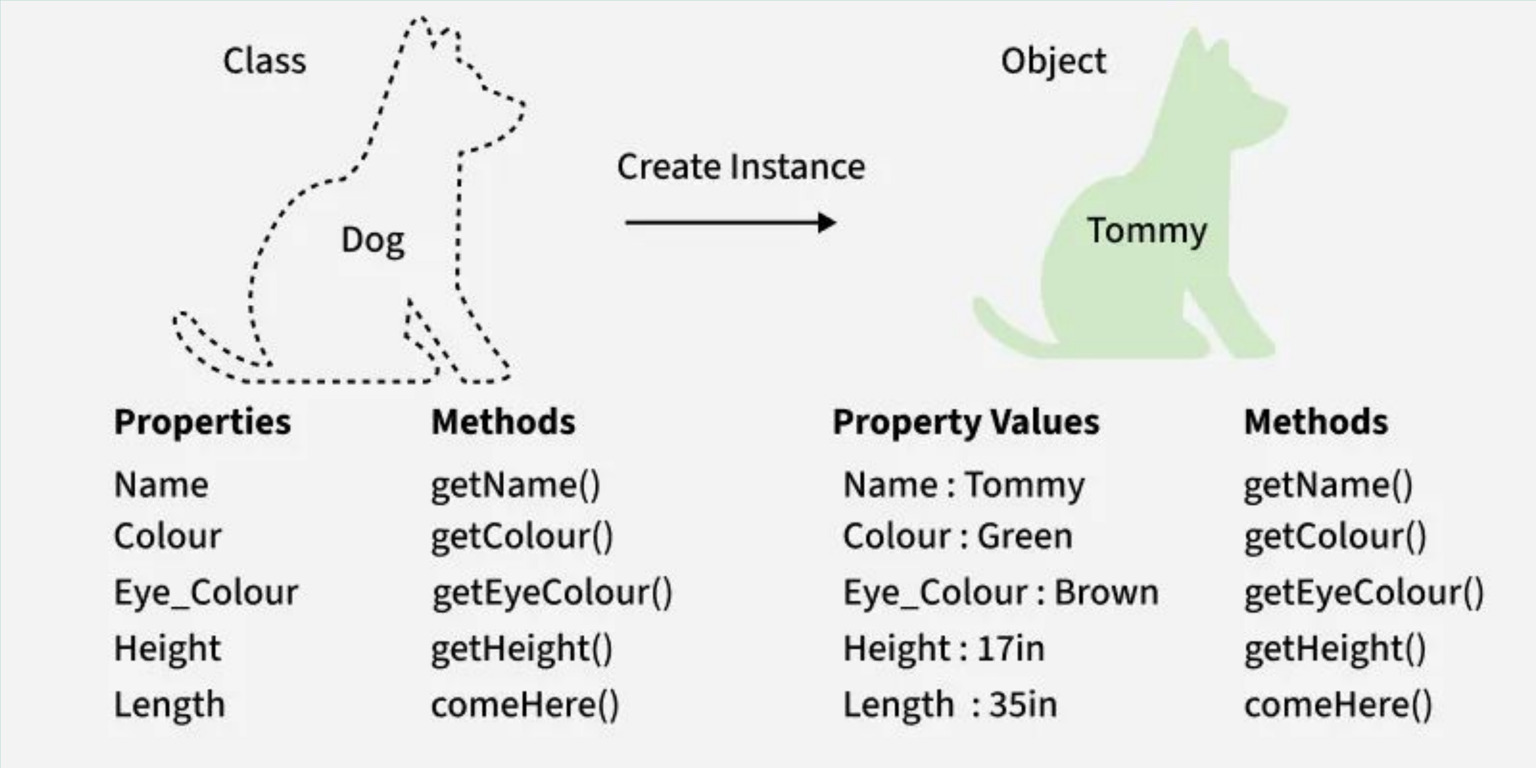
Classes

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

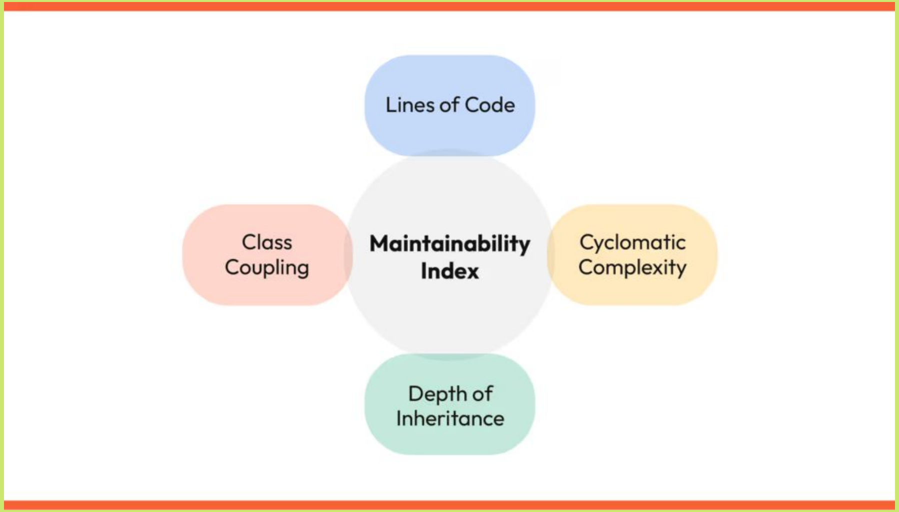


Class
Dog
Create Instance
Object
Tommy
Properties
Name
Colour
Eye_Colour
Height
Length
Methods
getName()
getColour()
getEyeColour()
getHeight()
comeHere()
Property Values
Name : Tommy
Colour : Green
Eye_Colour : Brown
Height : 17in
Length : 35in
Methods
getName()
getColour()
getEyeColour()
getHeight()
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

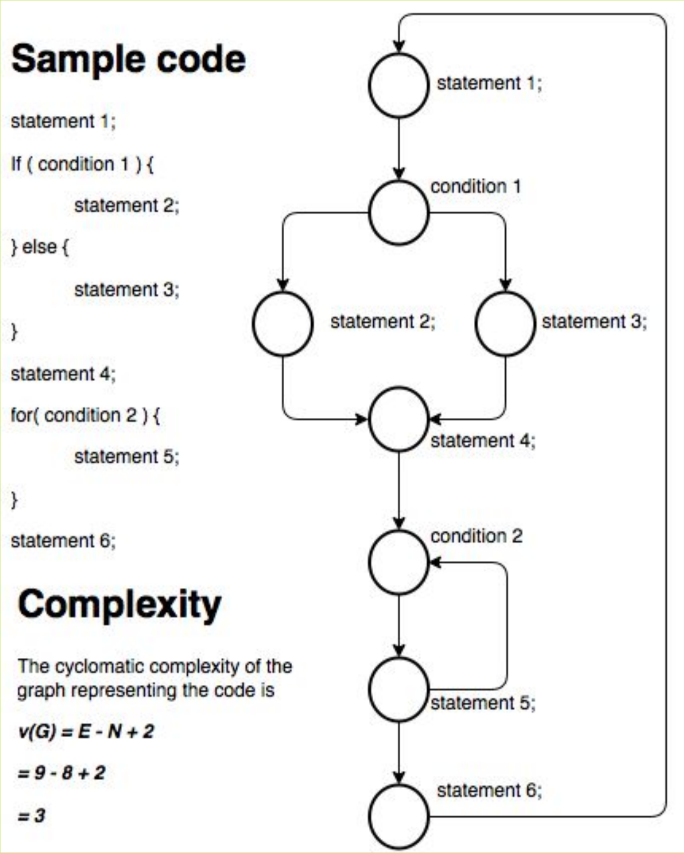


Low Complexity

```
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 += þ 1
      }
      return total
  }
```



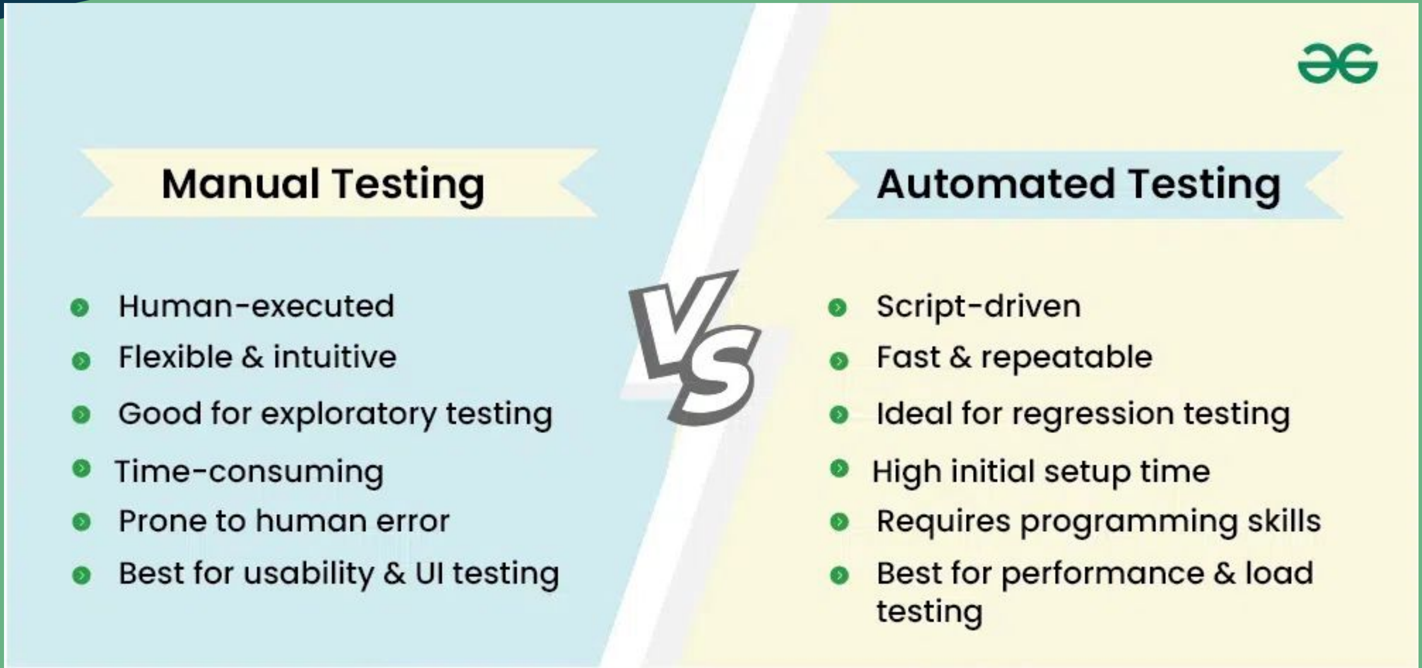
Continuous Integration

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

Automated Tests

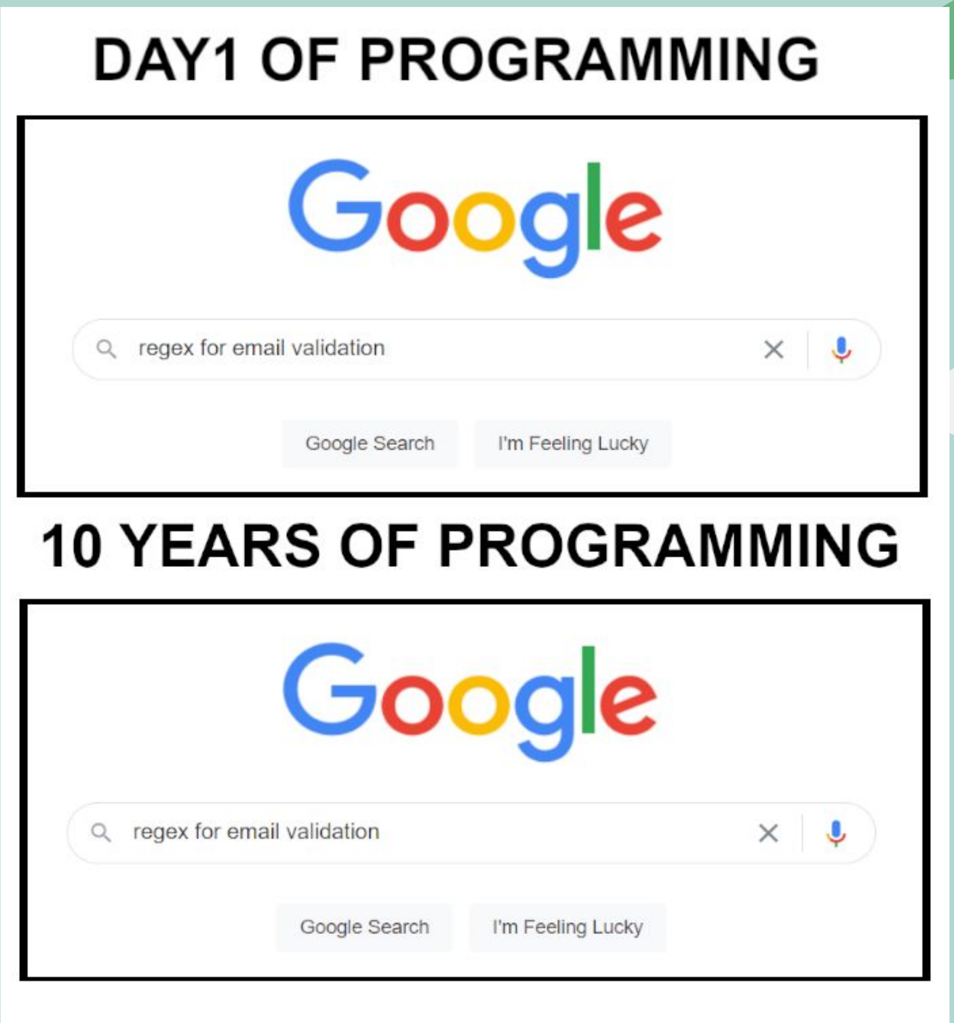


Data Validation

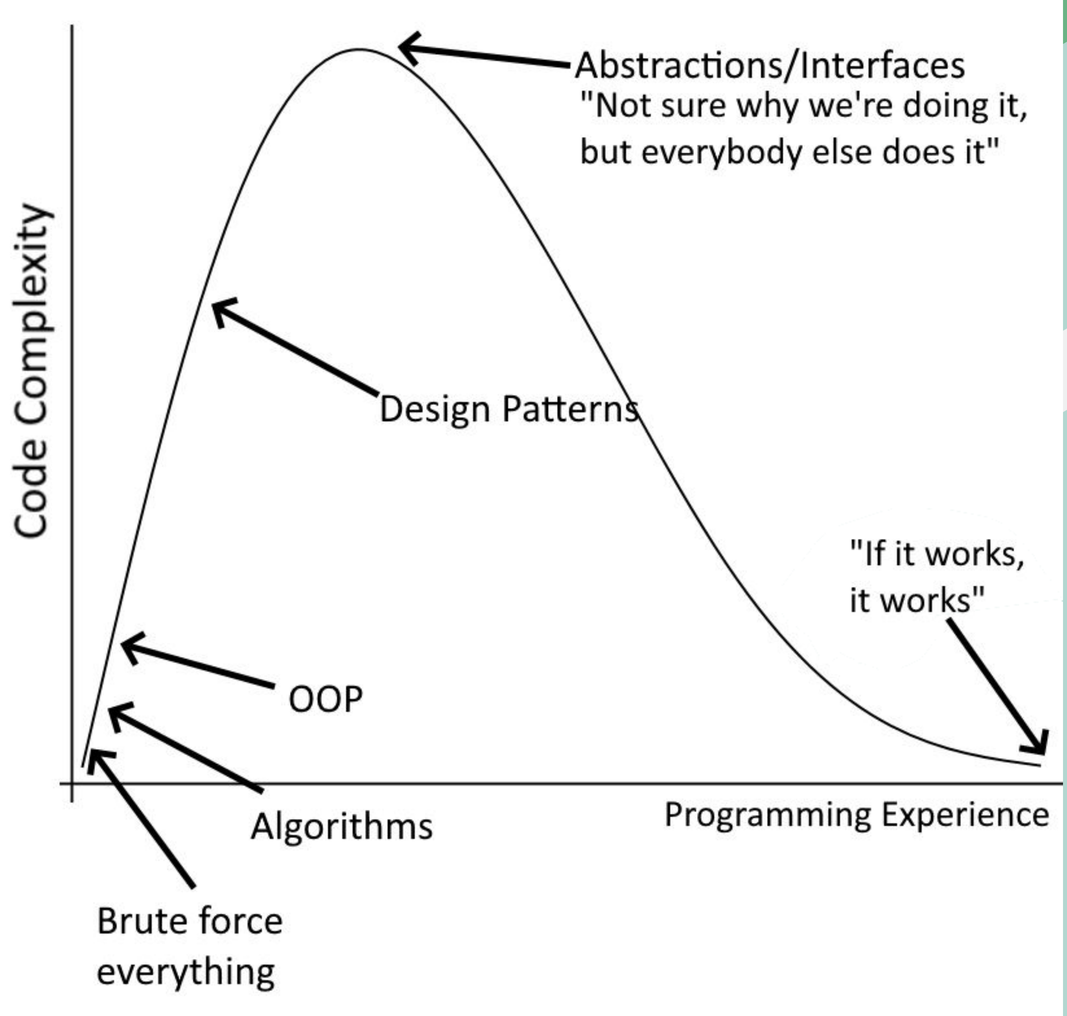
- Data validation is the process of **checking that inputs or datasets meet expected formats**, types, and constraints before being used in a program

- **Why it matters:** It prevents errors, improves reliability, and protects systems from bad or malicious data

```python
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)
"""
SchemarError: Column 'age' failed element-wise
validator number 0: in_range(0, 120) failure cases: 200
"""
```


DAY1 OF PROGRAMMING
Google
regex for email validation
Google Search
I'm Feeling Lucky
10 YEARS OF PROGRAMMING
Google
regex for email validation
Google Search
I'm Feeling Lucky


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


Abstractions/Interfaces
"Not sure why we're doing it,
but everybody else does it"
Code Complexity
Design Patterns
"If it works,
it works"
OOP
Algorithms
Programming Experience
Brute force
everything


EXERCIS
E
Lfg