Pitt INFORMS: Best Practices Workshop 1

Prepared by Jared Lawrence

University of Pittsburgh

Outline

1. Unit Testing
    a. pytest
2. Type Hinting
    a. mypy
3. Docstrings
4. Command-line arguments
    a. argparse
5. Shell Script (time permitting)

* Note: This workshop mostly follows existing demos.

Unit Testing

University of Pittsburgh

What is Unit Testing?

What is Unit Testing?

- Testing small, isolated units of code.
- Ensures functions work as expected.
- Helps catch bugs early in development.
- Improves code reliability and maintainability.

Popular Testing Frameworks in Python

- unittest (built-in)
- pytest (more flexible and widely used)

Demo

Doing the demo from
https://cs50.harvard.edu/python/2022/notes/5/

Assert

```
def square(n):
    return n * n

assert square(2) == 4
```

pytest

```
from calculator import square
def test_positive():
    assert square(2) == 4
    assert square(3) == 9
def test_negative():
    assert square(-2) == 4
    assert square(-3) == 9
def test_zero():
    assert square(0) == 0


>>> pytest test_calculator.py
```

Practice Problem

Create unit test for `is_valid_email`.

A valid email address:

- Contains exactly one `@` symbol.
- The part before the `@` symbol must not be empty.
- The part after the `@` symbol must contain at least one '`.`' and at least one character before and after the '`.`'.
- There must be no spaces in the email address.

i.e., Write example email addresses to test the required behavior.

```
def is_valid_email(email):
    if ' ' in email:
        return False
    if '@' not in email or email.count('@') != 1:
        return False
    local, domain = email.split('@')
    if not local or not domain or '.' not in domain:
        return False
    return True
```

Potential Solution

```
import pytest
from validate_emails import is_valid_email

def test_is_valid_email():
    assert is_valid_email("example@example.com") == True
    assert is_valid_email("user.name@domain.co") == True
    assert is_valid_email("test123@domain.com") == True

    assert is_valid_email("user@@domain.com") == False  # More than one '@'
    assert is_valid_email("exampledomain.com") == False  # Missing '@'
    assert is_valid_email("user name@domain.com") == False  # Space in email
    assert is_valid_email("@domain.com") == False  # Missing local part
    assert is_valid_email("user@.com") == False  # Missing domain before dot
    assert is_valid_email("user@domain.") == False  # Domain ends with dot
    assert is_valid_email(".user@domain.com") == False  # Local starts with dot
```

Type Hints

University of Pittsburgh

Functions

```
def calculate_area(radius):
    return 3.14159 * radius ** 2
```

Type Hints

Python is dynamically typed, but type hints allow you to indicate the expected data types of function arguments and return values.

Helps catch potential errors early.

```
from typing import Union

def calculate_area(radius: Union[int, float]) -> float:
    return 3.14159 * radius ** 2


# Or as of Python 3.10

def calculate_area(radius: int | float) -> float:
    return 3.14159 * radius ** 2
```

Demo

Using demo from
https://cs50.harvard.edu/python/2022/notes/9/#type-hints

Practice Problem

Add type hints to the function on the right.

```
def calculate_statistics(data):
    total = sum(data)
    average = total / len(data) if data else 0.0
    maximum = max(data) if data else None
    minimum = min(data) if data else None
    return (total, average, maximum, minimum)
```

Solution

```
from typing import List, Tuple, Optional

def calculate_statistics(data: List[int]) -> Tuple[int, float,
Optional[int], Optional[int]]:
    total: int = sum(data)
    average: float = total / len(data) if data else 0.0
    maximum: Optional[int] = max(data) if data else None
    minimum: Optional[int] = min(data) if data else None
    return (total, average, maximum, minimum)
```

Docstrings

What are docstrings?

- Docstrings are strings used to describe the functionality of a function, class, or module in Python.
- Written between triple quotes (""" docstring """) right after the function or class definition.
- Provides an easy way for other developers (and your future self) to understand what the code does without needing to dive into the implementation.
- Can be accessed using Python's built-in help() function for interactive documentation.

Docstrings

A special type of comment used to describe the purpose, usage, and behavior of a function, class, or module.

What to include:

1. Concise summary.
2. Parameters and return values.
3. Examples.

```
from typing import Union


def calculate_area(radius: Union[int,
float]) -> float:
    """
    Calculate the area of a circle.


    Parameters:
    radius (Union[int, float]): The radius
of the circle, can be an int or float.


    Returns:
    float: The area of the circle.
    """
    return 3.14159 * radius ** 2
```

Another Example

```
def my_add(x: float | int, y: float | int) -> float | int:
    """
    Adds two numbers together. Only accepts integers or floats.

    Parameters:
    x (int or float): The first number to be added.
    y (int or float): The second number to be added.

    Returns:
    int or float: The sum of x and y.

    Raises:
    TypeError: If either x or y is not an int or float.

    Example:
    >>> my_add(3, 5)
    8
    >>> my_add(2.5, 4.1)
    6.6
    >>> my_add("3", 5)
    TypeError: Both arguments must be int or float
    """
    if not isinstance(x, (int, float)) or not isinstance(y, (int, float)):
        raise TypeError("Both arguments must be int or float")

    return x + y
```

Demo

https://cs50.harvard.edu/python/2022/notes/9/#docstrings

Practice Problem

To the right is our function from the previous practice problem.

Add an informative docstring.

```
from typing import List, Tuple, Optional

def calculate_statistics(data: List[int])
-> Tuple[int, float, Optional[int],
Optional[int]]:
    total: int = sum(data)
    average: float = total / len(data) if
data else 0.0
    maximum: Optional[int] = max(data) if
data else None
    minimum: Optional[int] = min(data) if
data else None
    return (total, average, maximum,
minimum)
```

```
def calculate_statistics(data: List[int]) -> Tuple[int, float,
Optional[int], Optional[int]]:
    """
    Calculate sum, average, max, and min for a list of integers.

    :param data: List of integers to calculate statistics for
    :type data: List[int]
    :return: A tuple containing sum, average, max, and min of the list
    :rtype: Tuple[int, float, Optional[int], Optional[int]]

    :example:
    >>> calculate_statistics([1, 2, 3, 4, 5])
    (15, 3.0, 5, 1)
    >>> calculate_statistics([10, 20, 30])
    (60, 20.0, 30, 10)
    """
```

Command-line Arguments

sys.argv

It is possible to pass in arguments when running a file, rather than hard-coding them.

```
>>> python hello_name.py world
Hello, world
>>> python hello_name.py Jared
Hello, Jared
```

Demo

https://cs50.harvard.edu/python/2022/notes/4/#command-line-arguments

argparse

Passing in arguments can get complicated.

The argparse package exists to help simplify things by naming the arguments.

```
>>> python hello_name.py --name Jared
Hello, Jared
```

Demo

https://cs50.harvard.edu/python/2022/notes/9/#argparse

Practice Problem

Write a Python script that calculates the area of a rectangle, triangle, or circle based on the user's input from the command line.

Shape Type (--shape): Specifies the shape (rectangle, triangle, circle) for which to calculate the area.

For rectangle, you need to specify the --length and --width.

For triangle, you need to specify the --base and --height.

For circle, you need to specify the --radius.

Output: The script should calculate and print the area based on the shape type and the provided dimensions.

E.g.

>>> python calculate_area.py --shape rectangle --length 5 --width 3

Area of rectangle: 15

>>> python calculate_area.py --shape pentagon --length 5 --width 3

Error: Invalid shape type. Choose from rectangle, triangle, or circle.

Solution Pt 1

```
import argparse
import math

def calculate_area(shape: str, dimensions: dict) -> float:
    if shape == "rectangle":
        return dimensions['base'] * dimensions['height']
    elif shape == "triangle":
        return 0.5 * dimensions['base'] * dimensions['height']
    elif shape == "circle":
        return math.pi * dimensions['radius'] ** 2
    else:
        raise ValueError("Invalid shape type. Choose from rectangle, triangle, or circle.")

… continued on next slide
```

Solution Pt 2

```
def main():
    parser = argparse.ArgumentParser(description="Calculate area of a shape")
    parser.add_argument('--shape', choices=['rectangle', 'triangle', 'circle'], required=True, help="Shape type (rectangle,
triangle, circle)")
    parser.add_argument('--base', type=float, help="Base of the rectangle or triangle")
    parser.add_argument('--height', type=float, help="Height of the rectangle or triangle")
    parser.add_argument('--radius', type=float, help="Radius of the circle")
    args = parser.parse_args()

    if args.shape == "rectangle":
        if args.base is None or args.height is None:
            print("Error: You must provide both --base and --height for a rectangle.")
            return
        dimensions = {'base': args.base, 'height': args.height}

    elif args.shape == "triangle":
        if args.base is None or args.height is None:
            print("Error: You must provide both --base and --height for a triangle.")
            return
        dimensions = {'base': args.base, 'height': args.height}

    elif args.shape == "circle":
        if args.radius is None:
            print("Error: You must provide --radius for a circle.")
            return
        dimensions = {'radius': args.radius}

    try:
        area = calculate_area(args.shape, dimensions)
        print(f"Area of {args.shape}: {area}")
    except ValueError as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    main()
```

Shell Script

What is a shell script?

Shell Script:

- A text file containing a sequence of commands for Unix/Linux-based systems (e.g., Bash). It automates tasks by running multiple commands in a specific order.

Batch File:

- A script file for automating tasks in Windows Command Prompt. It includes a series of commands to run programs, manage files, or configure settings, all executed in a sequence.

Why use a shell script?

Run Multiple Commands at Once:

- Automate and execute entire workflows in a single step.

Consistency in Experiments:

- Ensure all experiments run with the same setup every time.

In a shell script

```
#!/bin/bash

# Run the script with a rectangle
echo "Running rectangle example:"
python3 calculate_area.py --shape rectangle --base 5 --height 3

# Run the script with a triangle
echo "Running triangle example:"
python3 calculate_area.py --shape triangle --base 6 --height 4

# Run the script with a circle
echo "Running circle example:"
python3 calculate_area.py --shape circle --radius 7

# Run with missing arguments for a rectangle (will show an error)
echo "Running rectangle with missing arguments (error case):"
python3 calculate_area.py --shape rectangle --base 5

# Run with invalid shape type (will show an error)
echo "Running with invalid shape type (error case):"
python3 calculate_area.py --shape pentagon --base 5 --height 5
```

In a Batch File

```
@echo off

echo Running rectangle example:
python3 calculate_area.py --shape rectangle --base 5 --height 3

echo Running triangle example:
python3 calculate_area.py --shape triangle --base 6 --height 4

echo Running circle example:
python3 calculate_area.py --shape circle --radius 7

echo Running rectangle with missing arguments (error case):
python3 calculate_area.py --shape rectangle --base 5

echo Running with invalid shape type (error case):
python3 calculate_area.py --shape pentagon --base 5 --height 5
```

Other Programming Languages

Everything today (besides type hints) generalizes to other languages and are important to do in any language.

E.g. Modules in Julia:

- Unit Testing ⇒ Test
- Command-line Arguments ⇒ ArgParse

References

Unit Tests: https://docs.pytest.org/

Type Hints: https://mypy.readthedocs.io/en/stable/cheat_sheet_py3.html

Docstrings: https://peps.python.org/pep-0257/

argparse: https://docs.python.org/3/library/argparse.html

https://cs50.harvard.edu/python/2022/notes/5/

https://cs50.harvard.edu/python/2022/notes/9/

https://cs50.harvard.edu/python/2022/shorts/pytest/