OOP, Code Style, and Reviews

Overview

- When you’re a software engineer, you’re not just writing code for your own immediate needs - you’re building a system that others will work on, from coworkers and open-source collaborators to future-you.
- To do this, we must write code based on shared principles of design and style.
- In this module we will learn the basic principles of object-oriented programming, the most common paradigm for the design and building of large codebases.
- We will also learn about standard code style and tooling, and the code review process.

Objective

- Create a basic Python class, with a constructor, methods, and fields
- Write stylistic (PEP 8) Python code
- Give and receive a basic code review

OOP

What is OOP

- Object oriented programming refers to the use of objects in programming.
- It is a methodology or paradigm to design a program using classes and objects.
- Aims to implement/model real-world things and relations between them.
    - As software objects that have some data associated with them and can perform certain functions.

Terms & Concepts

- Terms
  - Class
  - Object
  - Method
  - Instance
- Concepts
  - Inheritance
  - Encapsulation
  - Abstraction
  - Polymorphism

Class

- Classes are the top level organizational structure in OOP.
- A class is a user defined prototype/blueprint from which objects are created.
- It represents the set of properties or methods that are common to all objects of one type.
- Constructors are used for initializing new objects.
- Fields are variables that provides the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.

Object

- The basic unit of OOP.
- Objects correspond to things found in the real world.
- They may represent a person, a place or any item that the program must handle.
- An object is an instance of a class.

Inheritance

- Inheritance is the ability of one object to acquire some or all properties of another object.
- It allows code reusability.

Encapsulation

- Encapsulation means that we want to hide unnecessary details from the user.
    - For example, when we call from our mobile phone, we select the number and press call button. But the entire process of calling or what happens from the moment we press or touch the call button to the moment we start having a phone conversation is hidden from us.

Abstraction

- Abstraction means, showcasing only the required things to the outside world while hiding the details.
- The concept of abstraction focuses on what an object does instead of how it works.
- Data abstraction is often used for managing large and complex programs.

Polymorphism

- Polymorphism is a concept, which allows us to redefine the way something works, by either changing how it is done or by changing the parts used to get it done.

OOP in Python

- Python is an object oriented programming language.
- Almost everything in Python is an object.

Classes in Python

- Classes contain
    - Fields - also known as attributes or properties. They hold state information about a specific instance of an object. Examples might include name, size, or image.
    - Methods - functions that belong to a specific class. They represent the behaviors this object should have.
    - Constructors - special methods that are used to instantiate an object of this class.
    - self - a keyword used to refer to class-level variables and methods. These have scope across the entire class.

__init__ & self

- Constructor is the method that’s called when the class name is called.
    - It is always named __init__( )
- self parameter
    - is required in the definition
    - it must come first, before all other parameters
    - it is a reference to the instance itself
    - every method call automatically passes self
    - every variable prefixed with self is available to every method in the class

Code Style

Code Style

- Code style refers to the “look” and formatting of the code.
- Code with “bad style” can still run and get results.
- So why worry about style?
    - Because, though it can run, code that is harder to read is more error prone, and harder to fix when errors occur.
    - It’s also harder to extend and build on, which is the real long term goal of a production codebase.

Code Review

Code Review

- Design
    - How is the overall design.
    - Does it fit well with existing codebase
- Functionality
    - Does it fulfill the requirements
    - Does it do what it’s supposed to do
- Complexity
    - Is the code more complex than it needs to be
    - Functions, Classes, Modules

Code Review

- Look & Feel
    - How does the UI look.
    - Is it user friendly.
    - Spelling and grammatical errors
- Naming
    - Are the names for variables, functions, classes, etc clear.
- Comments
    - Are the comments clear and useful.
    - Do they explain what and why

Code Review

- Style
    - Does the code conform to the style guide.
- Tests
    - Unit tests are appropriate and complete.
- Documentation
    - Code is appropriately documented.

Feedback

- Constructive
- Positive
- Respectful