Comments and Docstrings

Premanand S

Assistant Professor
School of Electronics Engineering
Vellore Institute of Technology
Chennnai Campus

premanand.s@vit.ac.in

November 5, 2024

What are Comments?

- Comments are annotations in the code, prefixed with a '#'.
- They explain non-obvious parts of the code to make it easier to understand.
- Used primarily for:
  - Clarifying complex logic.
  - Leaving reminders or notes for future reference.
- **Single-Line Comments:** Use # to add comments on one line.
- **Multi-Line Comments:** Use multiple # symbols or triple quotes (''' ... ''') for multi-line comments.

Example - Comments in Python

Example (Python Code)

```
# This is a single-line comment
x = 10  # Inline comment

'''
This is a multi-line comment
that spans multiple lines
'''
y = 20
```

Best Practices for Comments

- **Clarity and Conciseness:** Avoid overly verbose or redundant comments.
- **Focus on Why, Not What:** Explain why the code does something.
- **Updating Comments:** Keep comments up-to-date as code evolves.

Advanced Comments with TODOs

- TODO comments are used to mark tasks or parts of the code that need further work.
- Typically prefixed by `# TODO:` in Python.
- Helpful for project planning and organization, especially in larger codebases.

Syntax of TODO Comments

- Common format: `# TODO: [Description of the task]`.
- Placed in code to indicate improvements, additional features, or debugging needs.

Example (Python Code)

```
# TODO: Optimize this function for better performance
def calculate_factorial(n):
    if n == 0:
        return 1
    else:
        return n * calculate_factorial(n - 1)
```

Advanced Comments with TODOs

- **TODO Comments:** Mark areas for future work using `# TODO`.
- **FixMe and Bug Tracking:** Indicate parts of code that need attention.

Example (Python Code)

```
x = 10  # TODO: Implement error handling for x
```

Best Practices for TODO Comments

- Be specific: Clearly describe the task or issue.
- Keep TODO comments up-to-date: Remove or update them as tasks are completed.
- Use descriptive text to make TODOs easier to understand by others.

Example (Python Code)

```python
# TODO: Refactor the following loop to improve readability
for i in range(10):
    print("Number:", i)
```

Using TODOs for Collaborative Projects

- TODO comments act as reminders for all contributors.
- Helps distribute tasks efficiently within a team.
- Some IDEs and code editors allow quick navigation to TODO comments.

Benefits of Using TODO Comments

- Helps track incomplete tasks or known issues within the code.
- Provides guidance for future development or debugging.
- Improves collaboration in teams by indicating work-in-progress areas.

What are Docstrings?

- Docstrings are special string literals used to document modules, classes, and functions.
- Enclosed in triple quotes (‘"""‘).
- Serve as formal documentation:
  - Describe what a function/class does.
  - Specify parameters and return types.
  - Can be accessed via the __doc__ attribute or using the ‘help()‘ function.
- **Single-Line Docstrings:** Use for simple functions or methods.
- **Multi-Line Docstrings:** Use triple quotes (`"""`) for complex functions.

Example - Docstring in Python

Example (Python Code)

```
def add(a, b):
    """Adds two numbers and returns the result."""
    return a + b
```

What is __doc__?

- In Python, `__doc__` is a special attribute that stores the **docstring** for functions, classes, modules, and methods.
- Useful for accessing documentation programmatically, which helps in exploring code.
- This attribute allows for quick insight into what a function, class, or module does.

How __doc__ Works?

- Defined at the start of a function, class, or module.
- Accessible using `object.__doc__`, where `object` can be a function, class, or module.

Example (Example: Accessing Function Docstring)

```
def greet():
    """This function greets the user with 'Hello, World!'."""
    return "Hello, World!"

print(greet.__doc__)

Output:
        This function greets the user with 'Hello, World!'.
```

Class-Level Docstring Example

Example (Example: Accessing Function Docstring)

```
class Calculator:
    """
    A simple calculator class to perform basic operations.
    Methods:
        add(a, b): Returns the sum of a and b.
        subtract(a, b): Returns the difference of a and b"""
    def add(self, a, b):
        """Return the sum of a and b."""
        return a + b
    def subtract(self, a, b):
        """Return the difference of a and b."""
        return a - b
print(Calculator.__doc__)
print(Calculator.add.__doc__)
```

Example: Module-Level Docstring

Example (Python)

```python
# Content of my_module.py
"""
This module provides basic math operations.
"""

def multiply(a, b):
    """Return the product of a and b."""
    return a * b

import my_module
print(my_module.__doc__)
print(my_module.multiply.__doc__)
```

Practical Uses of `__doc__`

- **Interactive Help:** Use `help(object)` to display the docstring.
- **Documentation Generation:** Tools like Sphinx utilize `__doc__` to create structured documentation.
- **Code Analysis:** Accessing docstrings helps developers quickly check what functions or classes do.

Introduction to PEP 257

- **PEP 257** is a Python Enhancement Proposal that sets conventions for writing docstrings.
- Builds on PEP 8 guidelines, focusing specifically on documentation strings for functions, methods, classes, and modules.
- Promotes readability, consistency, and helps in automatic documentation generation.

Single-Line Docstrings

- Used for simple functions where a brief description suffices.
- Should be concise and fit on one line without a line break after opening quotes.
- Example:

Example (Python Code)

```
def add(a, b):
    """Return the sum of a and b."""
    return a + b
```

Multi-Line Docstrings

- Use when more explanation is needed, with a summary on the first line.
- First line should be followed by a blank line and more details, if necessary.
- Example:

Example (Python Code)

```
def calculate_area(radius):
    """
    Calculate the area of a circle.

    Uses the formula  * r^2 where r is the radius.
    """
    import math
    return math.pi * radius ** 2
```

Example - Standard Format for Docstrings

Example (Python Code)

```
def multiply(a, b):
    """
    Multiplies two numbers.
    Args:
        a (int): The first number.
        b (int): The second number.
    Returns:
        int: The product of a and b.
    Example:
        >>> multiply(2, 3)
        6
    """
    return a * b
```

Key Differences

- **Purpose:** Comments are for developers; docstrings provide documentation for users of the code.
- **Syntax:** Comments use #, while docstrings use triple quotes.
- **Accessibility:** Docstrings are stored in `__doc__` and can be accessed via `help()`.

When to Use Each?

- **Comments:** Use for explaining complex code logic, marking sections, or leaving notes.
- **Docstrings:** Use to document functions, classes, and modules with descriptions of their purpose and usage.

Conclusion

- Both comments and docstrings enhance code readability.
- Comments are intended for developers; docstrings are meant for documentation.
- Using both appropriately can significantly improve code quality and maintainability.