ISIA 2025/2026 - PL 1
Python Docstrings

---

1 Getting Started with Python Docstrings & Documentation

1.1 What Are Docstrings?

- **Docstrings** are special string literals used to document Python modules, classes, methods, and functions.
- Unlike comments (#), docstrings are:
  - Stored in the object's `__doc__` attribute
  - Accessible via `help()`
  - Extracted by tools like **pdoc** or **Sphinx** to generate documentation

---

1.2 Syntax Basics

[1]:
```python
def greet(name: str) -> str:
    """
    Return a greeting message.

    Args:
        name (str): The name of the person.

    Returns:
        str: A greeting message.
    """
    return f"Hello, {name}!"
```

- Triple quotes (`""" ... """`)
- One-line summary first
- Sections for **Args**, **Returns**, **Raises**, **Examples**

1.3 Docstring Conventions

- **PEP 257**: Official docstring guidelines: https://peps.python.org/pep-0257/
- **PEP 8**: Style guide for Python code: https://peps.python.org/pep-0008/
- Common styles:
  - **Google style** (clean, readable)
  - **NumPy style** (scientific, tabular)

- **reStructuredText (reST)** (native to Sphinx)

1.4 Documenting OOP Code

1.4.1 Classes

[2]:
```
class Circle:
    """
    A circle shape.

    Attributes:
        radius (float): Radius of the circle.
    """
```

1.4.2 Methods

[3]:
```
    def area(self) -> float:
        """
        Calculate the area of the circle.

        Returns:
            float: The area of the circle.
        """
        import math
        return math.pi * self.radius ** 2
```

1.4.3 Attributes

- Document in the class docstring (`Attributes:` section)
- Or use reST `:ivar:` and `:vartype:` directives

1.5 Subclasses & Overrides

- **If behavior is unchanged**: reference parent docstring
- **If extended/modified**: document only the differences

[4]:
```
class Shape:
    """Base class for geometric shapes."""

    def area(self):
        """Return the area of the shape (must be implemented)."""
        raise NotImplementedError


class Square(Shape):
    """
    Square shape, inherits from Shape.

    Attributes:
        side (float): Length of one side.
    """
```

```python
    def area(self) -> float:
        """
        Calculate the area of the square.

        Overrides:
            Shape.area
        """
        return self.side ** 2
```

1.6 Style Comparison

Google:

```
Args:
    x (int): First number
    y (int): Second number
Returns:
    int: Sum
```

NumPy:

```
Parameters
----------
x : int
    First number
y : int
    Second number
Returns
-------
int
    Sum
```

reST:

```
:param x: First number
:type x: int
:param y: Second number
:type y: int
:return: Sum
:rtype: int
```

1.7 Best Practices

✔ One-line summary first

✔ Document attributes & methods consistently

✔ Use `Raises:` for exceptions

✔ Keep style consistent across project

✖ Don't duplicate obvious code behavior

1.8 Quick Reference

- `help(obj)` → view docstring
- `obj.__doc__` → raw docstring
- Styles: Google, NumPy, reST
- Tools:
  - **pdoc** for HTML docs
  - **Sphinx** for HTML/PDF docs

👉 **Takeaway**: Good docstrings make your code self-explanatory, maintainable, and ready for professional documentation.

1.9 🖳 Vewing docstring

1.9.1 In Python Console:

[5]:
```
help(Square)
```

```
Help on class Square in module __main__:

class Square(Shape)
 |  Square shape, inherits from Shape.
 |
 |  Attributes:
 |      side (float): Length of one side.
 |
 |  Method resolution order:
 |      Square
 |      Shape
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  area(self) -> float
 |      Calculate the area of the square.
 |
 |      Overrides:
 |          Shape.area
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from Shape:
 |
 |  __dict__
 |      dictionary for instance variables
 |
 |  __weakref__
 |      list of weak references to the object
```

[6]:
```
print(Square.__doc__)
```

```
Square shape, inherits from Shape.

Attributes:
    side (float): Length of one side.
```

1.9.2 Generate HTML file

Use documentation generation tools: see Sections 2.2 and 2.4 below.

---

2 Starter files

2.1 File: shapes.py

[7]:
```
class Shape:
    """Base class for geometric shapes."""

    def area(self):
        """
        Calculate the area of the shape.

        Raises:
            NotImplementedError: Must be implemented by subclasses.
        """
        raise NotImplementedError


class Circle(Shape):
    """
    A circle shape.

    Attributes:
        radius (float): Radius of the circle.
    """

    def __init__(self, radius: float):
        """
        Initialize a Circle.

        Args:
            radius (float): Radius of the circle.
        """
        self.radius = radius

    def area(self) -> float:
        """
        Calculate the area of the circle.

        Returns:
```

```
            float: The area of the circle.
        """
        import math
        return math.pi * self.radius ** 2


class Rectangle(Shape):
    """
    A rectangle shape.

    Attributes:
        width (float): Width of the rectangle.
        height (float): Height of the rectangle.
    """

    def __init__(self, width: float, height: float):
        """
        Initialize a Rectangle.

        Args:
            width (float): Width of the rectangle.
            height (float): Height of the rectangle.
        """
        self.width = width
        self.height = height

    def area(self) -> float:
        """
        Calculate the area of the rectangle.

        Returns:
            float: The area of the rectangle.
        """
        return self.width * self.height


class Square(Rectangle):
    """
    A square shape, subclass of Rectangle.

    Attributes:
        side (float): Length of one side.
    """

    def __init__(self, side: float):
        """
        Initialize a Square.

        Args:
            side (float): Length of one side.
        """
        super().__init__(side, side)
```

2.2 Using pdoc

- **Zero configuration**: just point it at your `.py` file or package
- **Instant HTML output**: generates a self-contained HTML site
- **Supports type hints** and modern docstring styles
- **Built-in web server**: preview docs locally with live reload
- **Markdown support**: docstrings can include Markdown formatting

2.2.1 Installation

```
pip install pdoc
```

2.2.2 Usage Examples

Generate HTML docs for a single file

```
pdoc shapes.py -o docs
```

- Creates a `docs/` folder with `shapes.html`
- Open it in your browser: `docs/shapes.html`

Serve docs locally (no files written)

```
pdoc shapes.py
```

- Starts a local server (default: http://localhost:8080)
- Live reloads when you edit code/docstrings

Generate docs for a whole package

```
pdoc mypackage -o docs
```

2.3 Example Workflow

1. Students write docstrings in `shapes.py`
2. Run: `pdoc shapes.py -o docs`
3. Open `docs/shapes.html` in a browser

2.4 Sphinx Documentation Setup

2.4.1 Install Sphinx

```
pip install sphinx
```

2.4.2 Initialize project

```
sphinx-quickstart
```

- Answer prompts (project name, author, etc.)
- This creates `conf.py`, `index.rst`, and build folders.

2.4.3 Enable autodoc in `conf.py`

```
extensions = ['sphinx.ext.autodoc']
import os, sys
sys.path.insert(0, os.path.abspath('..'))  # ensure shapes.py is found
```

2.4.4 Edit `index.rst` to include your module

```
Welcome to Shapes Documentation
==============================

.. automodule:: shapes
   :members:
   :undoc-members:
   :show-inheritance:
```

2.4.5 Build HTML docs

```
make html
```

- Output will be in `build/html/index.html`.

2.4.6 Open in browser

Navigate to `build/html/index.html` → you'll see a professional documentation site generated from your docstrings.

2.5 ⚖ Comparison: Sphinx vs pdoc

Feature	**Sphinx**	**pdoc**
Setup	Needs `conf.py`, `.rst` files	No setup, runs immediately
Output	HTML, PDF, LaTeX, EPUB	HTML only
Customization	Very high (themes, extensions)	Minimal, but clean
Best for	Large projects, official docs	Quick API docs, teaching, small/medium projects