1. Consistency

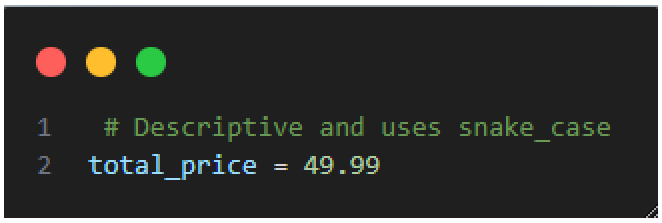
- Ensure a uniform coding style throughout the codebase for readability and maintainability.
- Use consistent naming conventions, indentation, and formatting.

2. Naming Conventions

2.1 Variables

- Use `snake_case` for variables.
- Choose descriptive, meaningful names.
- Avoid single-letter or ambiguous variable names.

Example:

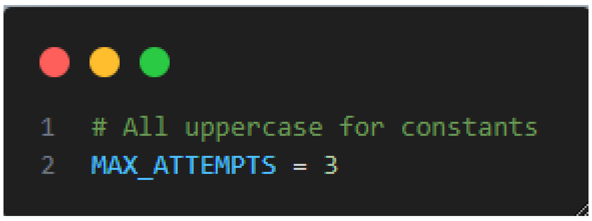


```python
# Descriptive and uses snake_case
total_price = 49.99
```

2.2 Constants

- Use `UPPERCASE_SNAKE_CASE` for constants, typically defined at the module level.

Example:

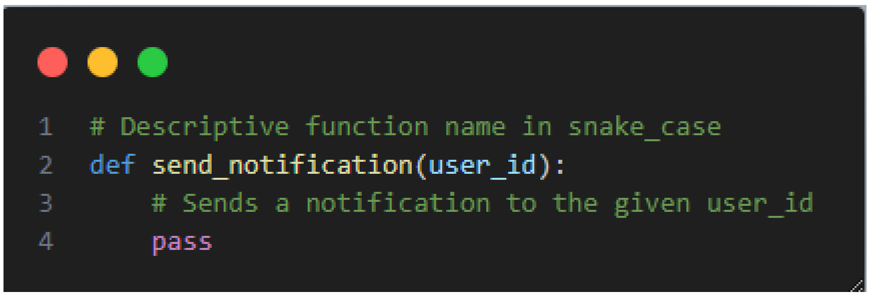


```python
# All uppercase for constants
MAX_ATTEMPTS = 3
```

2.3 Functions

- Use `snake_case` for function names.
- Function names should be action-oriented and descriptive.

Example:

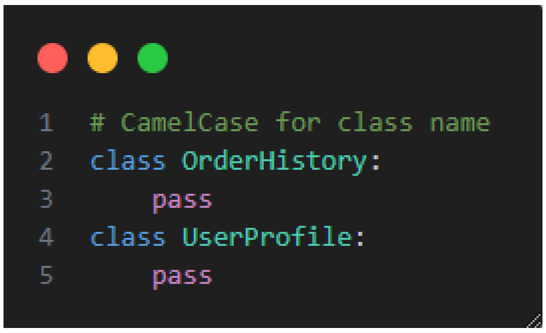


```python
# Descriptive function name in snake_case
def send_notification(user_id):
    # Sends a notification to the given user_id
    pass
```

2.4 Classes

- Use `CamelCase` for class names.
- Start class names with uppercase letters.

Example:

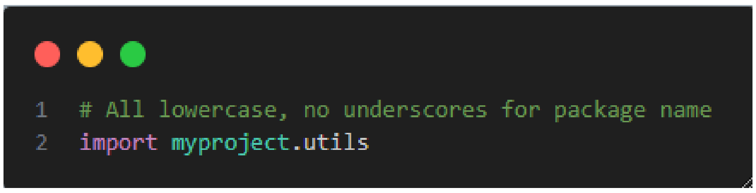


```python
# CamelCase for class name
class OrderHistory:
    pass
class UserProfile:
    pass
```

2.5 Packages & Modules

- Use short, all-lowercase names for packages and modules.
- Avoid underscores in package names for compatibility.

Example

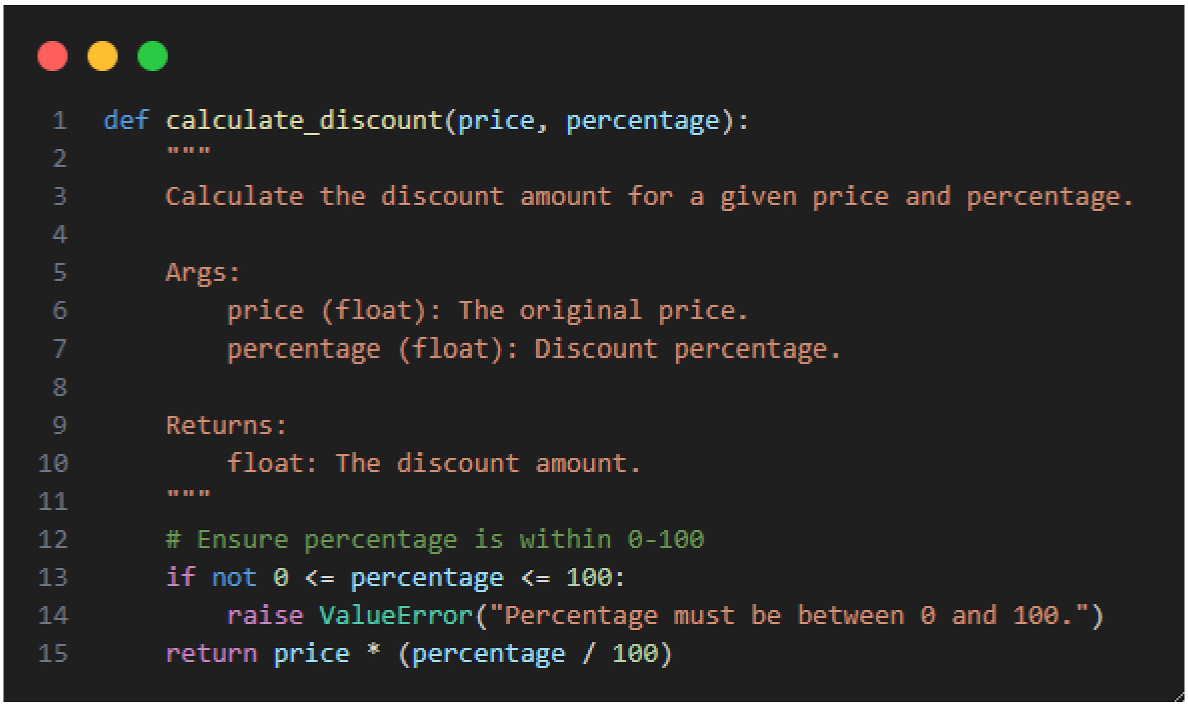


```python
# All lowercase, no underscores for package name
import myproject.utils
```

3. Comments & Documentation

- Use **docstrings** (" " ") to describe modules, classes, methods, and functions.
- Write inline comments sparingly, only for non-obvious logic.

Example:

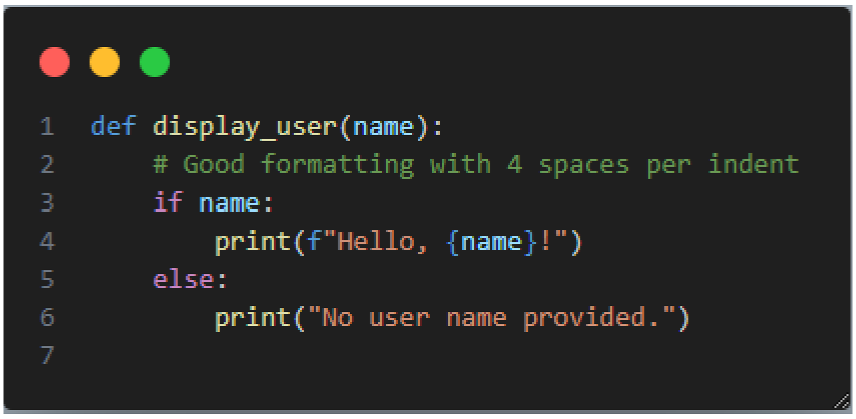


```python
def calculate_discount(price, percentage):
    """
    Calculate the discount amount for a given price and percentage.

    Args:
        price (float): The original price.
        percentage (float): Discount percentage.

    Returns:
        float: The discount amount.
    """
    # Ensure percentage is within 0-100
    if not 0 <= percentage <= 100:
        raise ValueError("Percentage must be between 0 and 100.")
    return price * (percentage / 100)
```

4. Formatting & Indentation

- Use **4 spaces** per indentation level.
- Limit lines to **79 characters** (as per PEP 8).
- Use blank lines to separate logical code blocks.

Example

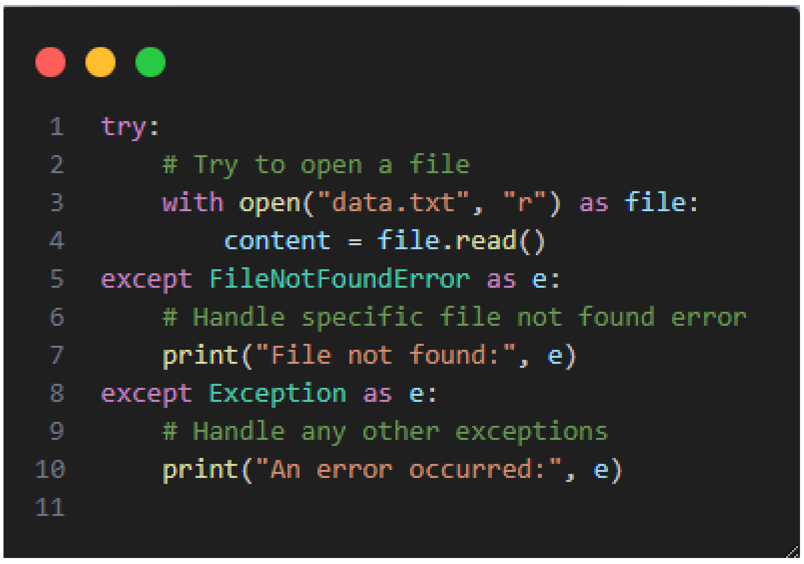


```python
def display_user(name):
    # Good formatting with 4 spaces per indent
    if name:
        print(f"Hello, {name}!")
    else:
        print("No user name provided.")
```

5. Error Handling

- Prefer specific exceptions over generic ones.
- Always use try-except blocks for error-prone code.

Example:

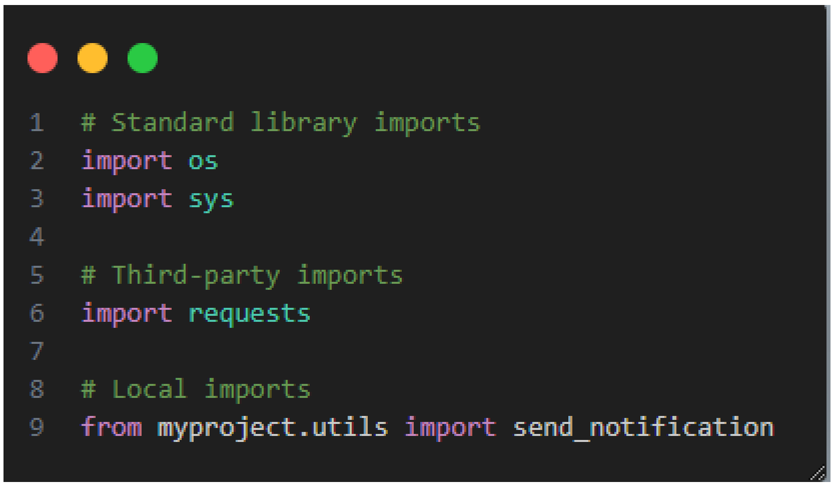


```python
try:
    # Try to open a file
    with open("data.txt", "r") as file:
        content = file.read()
except FileNotFoundError as e:
    # Handle specific file not found error
    print("File not found:", e)
except Exception as e:
    # Handle any other exceptions
    print("An error occurred:", e)
```

6. Import Formatting

- Each import should be on a separate line.
- **Import order:**
    1. Standard library imports
    2. Third-party imports
    3. Local application imports
- Use absolute imports for clarity.

Example:

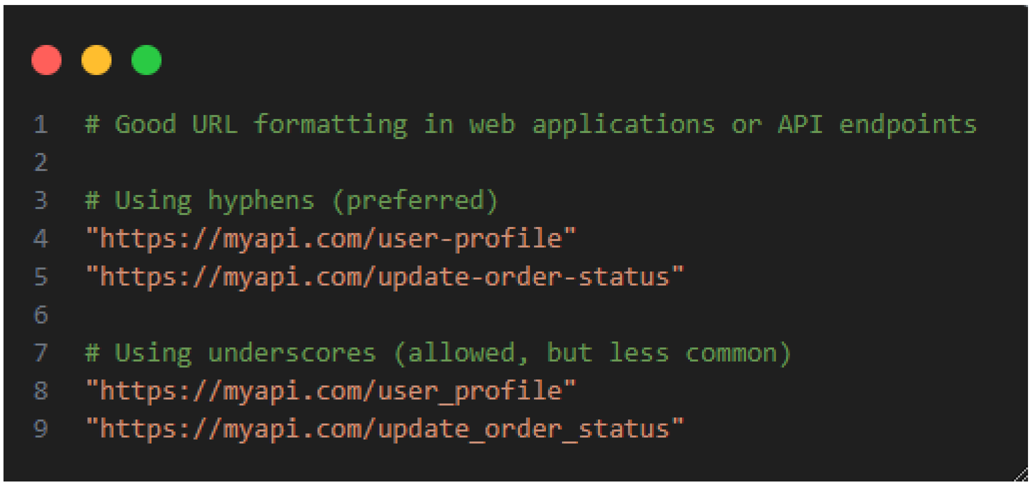


```python
# Standard library imports
import os
import sys

# Third-party imports
import requests

# Local imports
from myproject.utils import send_notification
```

7. URL Formatting

- In web/API projects, use lowercase and separate words with hyphens or underscores.

Example:

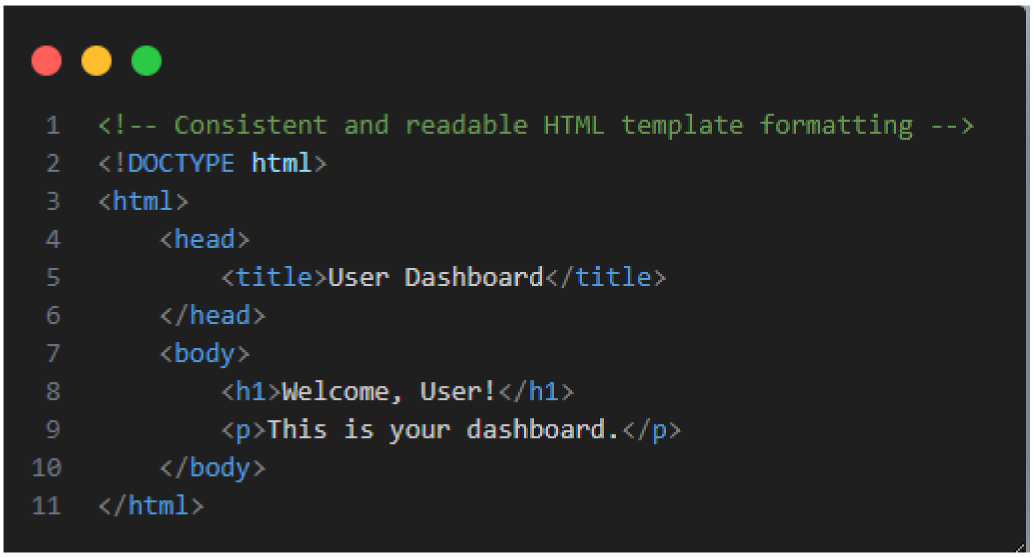


```
# Good URL formatting in web applications or API endpoints

# Using hyphens (preferred)
"https://myapi.com/user-profile"
"https://myapi.com/update-order-status"

# Using underscores (allowed, but less common)
"https://myapi.com/user_profile"
"https://myapi.com/update_order_status"
```

8. Template Style

- For HTML templates, use consistent indentation.
- Write clean, semantic markup.

Example:

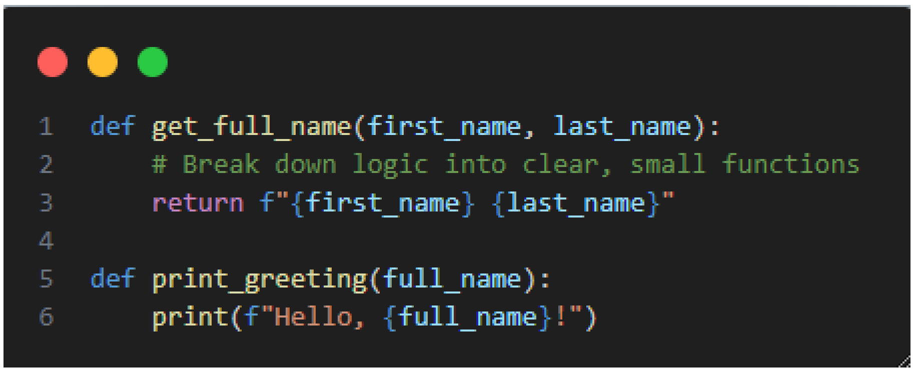


```
<!-- Consistent and readable HTML template formatting -->
<!DOCTYPE html>
<html>
    <head>
        <title>User Dashboard</title>
    </head>
    <body>
        <h1>Welcome, User!</h1>
        <p>This is your dashboard.</p>
    </body>
</html>
```

9. Code Readability & Reusability

- Break down complex tasks into smaller, reusable functions or methods.
- Avoid cryptic abbreviations—choose clarity over brevity.

Example:

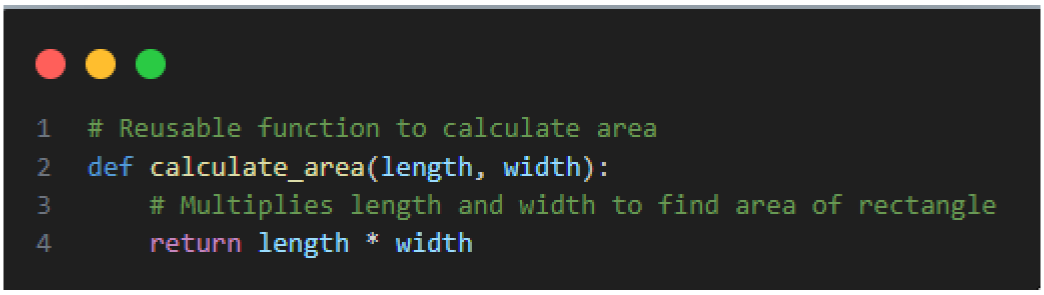


```python
def get_full_name(first_name, last_name):
    # Break down logic into clear, small functions
    return f"{first_name} {last_name}"

def print_greeting(full_name):
    print(f"Hello, {full_name}!")
```

10. Code Reusability

- Encapsulate reusable code into functions, classes, or modules to avoid duplication..

Example:

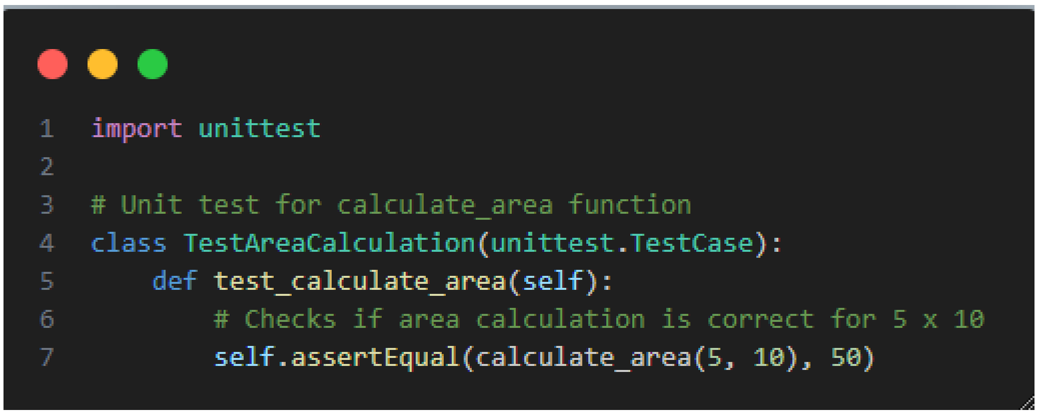


```python
# Reusable function to calculate area
def calculate_area(length, width):
    # Multiplies length and width to find area of rectangle
    return length * width
```

11. Testing and Quality Assurance

- Write unit tests using Python testing frameworks (unittest, pytest, etc.) to ensure code quality.

Example:



```python
import unittest

# Unit test for calculate_area function
class TestAreaCalculation(unittest.TestCase):
    def test_calculate_area(self):
        # Checks if area calculation is correct for 5 x 10
        self.assertEqual(calculate_area(5, 10), 50)
```

12. References & Resources

- PEP 8 – Python Style Guide
- Google Python Style Guide
- CKAN Python Contribution Guide
- Python Best Practices (Zenesys)