TOPS Technology

Python Fundamentals

Presented By:

Nandni Vala

Programming Style:

1.Understanding Python's PEP 8 guidelines.

- PEP 8 (Python Enhancement Proposal 8) is the style guide for Python code that provides conventions and best practices for writing clean, readable, and consistent code.
- 1. **Code Layout**
- **Indentation**: Use **4 spaces per indentation level**, not tabs.
- **Maximum Line Length**: Limit all lines to **79 characters** (72 characters for docstrings).
- **Blank Lines**:
  - 1 blank line between functions and class definitions.
  - 2 blank lines between top-level functions and class definitions.
- **Imports**: Imports should be on separate lines, and in this order:
  - Standard library imports.
  - Third-party library imports.
  - Local application/library imports.

- **Naming Conventions**
- **Variables/Functions**: Use lowercase with underscores (e.g., my_variable, my_function).
- **Classes**: Use CapitalizedWords (e.g., MyClass, MyDerivedClass).
- **Constants**: Use uppercase with underscores (e.g., MAX_VALUE, PI).
- **Modules and Packages**: Use short, all-lowercase names (e.g., my_module, mypackage).
- **Whitespace in Expressions and Statements**
- Avoid unnecessary spaces:
  - Correct:
- If x == 4:
-     print(x)
- Incorrect:
- if x == 4 :
-     print(x)

- **Docstrings**
- Use docstrings to describe all public classes and methods.
- Docstrings should be enclosed in triple quotes ("""Docstring""").
- **One-liner docstrings**: Place the docstring on one line (if simple).
  - Example:
- def square(x):
-     """Return the square of x."""
-     return x * x
- **Multi-line docstrings**: Use triple quotes for multi-line docstrings, with the description and explanation formatted clearly.
- Example:
- def my_function(x, y):
-     """
-     This function adds x and y.
-     Parameters:

- ➢ x (int): The first number.
- ➢ y (int): The second number.
- ➢ Returns:
- ➢int: The sum of x and y.
- ➢"""
- ➢return x + y
- ➢**Programming Recommendations**
- ➢**Avoid using global variables**: Instead, pass variables to functions where possible.
- ➢**Use exception handling properly**: Use try, except, finally blocks as needed to handle errors gracefully.
- ➢**Use comprehensions**: Prefer list, dictionary, and set comprehensions over loops when appropriate for readability and efficiency.

- **Versioning**
- When writing Python 2/3 compatible code, always be explicit with the versions and use `six` or `future` libraries to bridge the compatibility gaps.

5.Indentation, comments, and naming conventions in Python.

- **Indentation in Python**
- **Use 4 spaces** per indentation level. Do not use tabs.
- Indentation is important in Python because it indicates blocks of code. For example, in control structures (if, for, etc.), indentation shows the scope of the block.
- Example:
- if x > 10:
-     print("x is greater than 10")
-     x -= 1 # Decrease x by 1
- else:
-     print("x is less than or equal to 10")
- **Comments in Python**
- **Inline comments**: Start with # and are placed on the same line as code.
- x = 5 # This is an inline comment.

- **Naming Conventions in Python**
- **Variables/Functions**: Use **lowercase letters** with **underscores** to separate words (my_variable, my_function).
- my_variable = 10
- def calculate_sum(a, b):
-     return a + b
- **Modules and Packages**: Use **short, lowercase names** (e.g., my_module, utils).
- import math

3.Writing readable and maintainable code.

- **Use Meaningful and Consistent Naming**
- **Variables, Functions, and Classes**: Use clear and descriptive names that convey purpose. For example:
- int numOfItems; // Instead of "n"
- double calculateTax(double amount); // Instead of "calcT"

- **Comment Smartly**
- Explain *why* a piece of code exists, not *what* it does (if the code is self-explanatory).
- // Calculate the total cost, including a 10% service charge.
- double totalCost = baseCost * 1.1;
- **Structure Your Code**
- **Indentation**: Use consistent indentation (e.g., 4 spaces or a tab).
- **Logical Grouping**: Group related pieces of code together, such as declarations, loops, or helper functions.
- // Declare variables
- int a = 10, b = 20;
- // Perform calculations
- int sum = a + b;
- // Output the result
- printf("Sum: %d\n", sum);