Python Training

Python Course Structure

1. Getting Started with Python
    - Overview of Python programming language and its history
    - Installing and setting up the development environment
    - Creating Virtual Environments
    - Running Python Programs
    - Basic syntax and programming concepts
    - Data types, variables, and operators
    - Practice & Exercises
2. Control Flow and Functions
    - Control flow statements (if/else, loops)
    - Functions and modules
    - Error handling and exceptions
    - Debugging techniques
    - Practice & Exercises
3. Data Structures and Algorithms
    - Lists, tuples, sets, and dictionaries
    - String manipulation
    - Searching and sorting algorithms
    - Recursion
    - Practice & Exercises
4. Object-Oriented Programming
    - Classes and objects
    - Inheritance and polymorphism
    - Abstraction and encapsulation
    - Exception handling
    - Practice & Exercises
5. File Handling and Working with Data
    - Opening, closing, and manipulating files
    - Reading and writing files
    - Working with CSV and JSON files
    - Introduction to data analysis and manipulation using pandas library
    - Practice & Exercises
6. Web Development with Python
    - HTML, CSS, and JavaScript basics
    - Introduction to Flask or Django frameworks

- Building web applications using Flask or Django
- Practice & Exercises

7. Databases and Data Science with Python

- SQL basics
- Introduction to SQLite or MySQL
- Using Python to interact with databases
- Data analysis and visualization using NumPy, Pandas, Matplotlib, and Seaborn
- Practice & Exercises

8. Advanced Topics in Python

- Regular expressions
- Working with APIs using Python
- Testing and debugging
- Practice & Exercises

Python Best Practices for application development:

1. Follow PEP 8: As mentioned earlier, following PEP 8 coding conventions can make your code more readable and consistent.
2. Write modular and reusable code: Break your code into smaller functions and classes, and reuse them as much as possible. This can make your code more efficient and easier to maintain.
3. Use virtual environments: Use virtual environments to manage dependencies and avoid conflicts between different projects.
4. Document your code: Write clear and concise documentation for your code, including docstrings for functions and modules.
5. Write tests: Write automated tests to ensure that your code is working as expected and catch errors before they make it to production.
6. Use version control: Use a version control system, such as Git, to keep track of changes to your code and collaborate with other developers.
7. Optimize your code: Optimize your code for performance, but don't sacrifice readability and maintainability. Use profiling tools to identify bottlenecks and optimize critical sections of code.
8. Use logging: Use a logging framework to track errors and debug your code.
9. Secure your code: Ensure that your code is secure by following security best practices, such as using secure passwords, encrypting sensitive data, and validating user input.
10. Use good design patterns: Follow good design patterns, such as MVC, to make your code more modular and maintainable.

PEP-8 Rules

PEP 8 is a set of coding conventions and style guidelines for Python code. It is intended to improve code readability and consistency, making it easier to understand and maintain. Here are some of the key rules in PEP 8:

1. Indentation: Use four spaces for indentation, not tabs.
2. Line length: Limit lines to 79 characters or fewer. If a line needs to be longer, break it into multiple lines using parentheses, backslashes, or a continuation character.
3. Naming conventions: Use lowercase words separated by underscores for variable, function, and module names. Class names should be in CamelCase. Constants should be in all caps.
4. Function and method arguments: Put a space after commas, but no space before the comma.
5. Comments: Use complete sentences for comments, starting with a capital letter and ending with a period. Use inline comments sparingly.
6. Blank lines: Use blank lines to separate functions, classes, and other blocks of code.
7. Imports: Import modules on separate lines. Put standard library imports first, followed by third-party library imports, and then local application imports.
8. Whitespaces: Use whitespaces around operators and after commas, but avoid using them excessively.
9. Function and class definitions: Put a single blank line before and after function and class definitions.
10. Coding style: Use consistent coding style throughout the code. Follow the conventions and guidelines presented in PEP 8 as closely as possible.

Following these guidelines can make your code more readable and maintainable, and it can also make it easier for other Python developers to understand your code.