DRM Development, Inc,
Located in the University of Central Florida Research Park

website: drmdev.net
email: philm@drmdev.net
phone: (954) 657-1018

12001 Research Pkwy. Suite 236
Orlando, Florida 32826 USA

Welcome to the comprehensive outline for our Python training program! This document serves as a roadmap for individuals looking to enhance their Python programming skills, whether you're a beginner eager to explore the basics or an experienced developer aiming to delve into more advanced concepts.

Our Python training program is meticulously crafted to cater to learners of all levels, from introductory to intermediate and beyond. Through a series of meticulously planned modules, participants will embark on a journey to master Python programming, explore its diverse applications, and gain practical experience through hands-on projects and real-world scenarios.

The outline covers a wide range of topics, starting with the fundamentals of Python syntax and data types, progressing to more advanced concepts such as object-oriented programming, automated testing, data analysis, and network programming. Participants will also have the opportunity to delve into specialized areas like graphical user interface (GUI) development, automated software testing, and data manipulation.

Each section of the outline is carefully structured to provide a comprehensive understanding of the topic at hand, accompanied by practical exercises, case studies, and real-world applications to reinforce learning. Additionally, our experienced instructors will be on hand to guide participants through each module, offering insights, tips, and best practices gleaned from their industry experience.

Furthermore, our certification guarantee ensures that participants have the opportunity to validate their newfound skills and expertise, giving them a competitive edge in today's job market.

Whether you're looking to kickstart your career in Python programming, enhance your existing skills, or explore new avenues of application, our Python training program offers a dynamic and immersive learning experience tailored to your needs.

Experience Live, Instructor-Led Remote Bootcamps with Cutting-Edge Technology!

Utilizing GoTo.com technology, our bootcamps bring the classroom directly to you, allowing you to interact with expert instructors and fellow participants in real-time. Whether you're located across the globe or just around the corner, you can join our bootcamps from anywhere with an internet connection.

Here's what sets our bootcamps apart:

1. Live Instruction: Our bootcamps feature live, interactive sessions led by experienced instructors who are passionate about empowering participants with the knowledge and skills they need to succeed. You'll have the opportunity to ask questions, participate in discussions, and receive personalized guidance throughout the program.

2. Remote Accessibility: Say goodbye to long commutes and rigid schedules! With our remote bootcamps, you can attend sessions from the comfort of your own home or any location of your choosing. All you need is a computer or mobile device and an internet connection to join the virtual classroom.

3. Cutting-Edge Technology: We leverage advanced technology provided by GoTo.com to deliver seamless and engaging learning experiences. From interactive whiteboards and screen sharing to chat functionality and breakout rooms, our platform is designed to enhance collaboration and learning efficiency.

4. Session Recordings: Can't make it to a live session? No problem! All bootcamp sessions are recorded and made available to participants in MP4 format for playback at their convenience. Whether you need to review a concept, catch up on a missed session, or revisit key insights, you'll have access to the recorded content whenever you need it.

At our bootcamps, we're committed to providing participants with the flexibility, accessibility, and support they need to excel in their learning journey. Join us and experience the future of education with our live, instructor-led remote bootcamps today!

Ready to take your skills to the next level? Browse our upcoming bootcamp offerings and reserve your spot today. We can't wait to see you in the virtual classroom!

Certification Guarantee:

We are committed to your success! Our certification guarantee ensures that you'll receive a $200 refund of your bootcamp price, if you do not pass a certification test, upon meeting the following criteria:

Completion of the intensive bootcamp and all related labs.

Certification test must be at a valid certification center and taken within 3 months of the completion of the bootcamp. Ex> Certiport, Pearson VUE, Prometric, etc.

Proof of passing any practice exam related to the Python certification.

Proof of the failed examination related to your Python bootcamp intensive.

Note: If you are planning a Python Certification path, email us your path and testing center and we can extend the time frame on our certification guarantee based on your personal educational path. We want you to excel in your professional life!

2024 Python Intensive Bootcamp Dates and Payment Plans
Live Classroom Time is 9:30 EST – 4:30 EST

Bootcamp Dates (Tuesday's thru Friday's)	Balance Due Date (Tuesday's)	Last Day to Register (Monday's)
January 9, 2024 thru January 12, 2024	December 19, 2023	January 1, 2024
January 23, 2024 thru January 26, 2024	January 2, 2024	January 15, 2024
February 20, 2024 thru February 23, 2024	January 30, 2024	February 12, 2024
March 12, 2024 thru March 8, 2024	February 20, 2024	March 4, 2024
April 9, 2024 thru April 12, 2024	March 19, 2024	April 1, 2024
April 30, 2024 thru May 3, 2024	April 9, 2024	April 22, 2024
May 28, 2024 thru May 31, 2024	May 7, 2024	May 20, 2024
June 4, 2024 thru June 7, 2024	May 14, 2024	May 27, 2024
June 18, 2024 thru June 21, 2024	May 28, 2024	June 10, 2024
July 16, 2024 thru July 19, 2024	June 25, 2024	July 8, 2024
August 6, 2024 thru August 9, 2024	July 16, 2024	July 29, 2024
September 24, 2024 thru September 27, 2024	September 3, 2024	September 16, 2024
October 22, 2024 thru October 25, 2024	October 1, 2024	October 14, 2024
November 5, 2024 thru November 8, 2024	October 15, 2024	October 28, 2024
December 17, 2024 thru December 20, 2024	November 26, 2024	December 9, 2024

4 Day Python Intensive Bootcamp $1695

Don't let finances hold you back from pursuing your Python programming dreams. Reserve your spot today with a small deposit and enjoy the flexibility of our payment plan.

Deposit to Hold Date: $500
Balance can be split monthly, with the total due 3 weeks before the class begins.
Payment due dates will be scheduled and automatically applied to the credit card on file.

Title: Introduction to Python Syntax

I. Introduction to Python
- A. Brief history and importance of Python
- B. Setting up Python environment (if necessary)
- C. Overview of Python syntax and structure

II. Data Types and Variables
- A. Numbers
  1. Integers
  2. Floating-point numbers
- B. Strings
  1. Definition and characteristics
  2. String manipulation (concatenation, slicing, methods)
- C. Variables
  1. Declaring variables
  2. Variable naming conventions
  3. Variable assignment and reassignment

III. Data Structures
- A. Lists
  1. Definition and characteristics
  2. List manipulation (slicing, indexing, methods)
- B. Tuples
  1. Definition and characteristics
  2. Tuple manipulation (packing, unpacking)
- C. Sets
  1. Definition and characteristics
  2. Set operations (union, intersection, difference)
- D. Dictionaries
  1. Definition and characteristics
  2. Dictionary manipulation (key-value pairs, methods)

IV. Control Structures
- A. Conditional Statements (if, elif, else)
  1. Syntax and usage
  2. Nested if statements
- B. Loops

1. For loops
    a. Syntax and usage
    b. Looping through lists, tuples, and dictionaries
2. While loops
    a. Syntax and usage
    b. Loop termination conditions

V. Functions
   A. Definition and purpose of functions
   B. Function syntax
   C. Parameters and arguments
   D. Return statement

Title: Deep Dive into Python Programming: Object-Oriented Problem Solving

I. Review of Python Basics
- A. Quick recap of fundamental Python syntax and data types
- B. Revisiting control structures and functions

II. Introduction to Object-Oriented Programming (OOP)
- A. Overview of OOP principles
- B. Importance of OOP in problem-solving and code organization
- C. Classes and objects
  - 1. Defining classes
  - 2. Creating objects from classes
- D. Encapsulation, inheritance, and polymorphism

III. Modeling Real-Life Problems with OOP
- A. Identifying objects and their behaviors
- B. Mapping real-world entities to classes
- C. Implementing class methods and attributes
- D. Example: Building a simple class hierarchy for a real-life scenario

IV. Best Practices in Python Programming
- A. Code readability and maintainability
  - 1. PEP 8 guidelines
  - 2. Naming conventions
- B. Modularity and code reuse
  - 1. Writing modular and reusable code
  - 2. Leveraging built-in and external libraries
- C. Error handling and debugging techniques
  - 1. Exception handling
  - 2. Debugging tools and strategies

V. Solutions Architecture and Design Patterns
- A. Design patterns overview
- B. Commonly used design patterns in Python
  - 1. Singleton
  - 2. Factory
  - 3. Observer
  - 4. Strategy

C. Applying design patterns to solve problems effectively

VI. Analytical Thinking Methods
A. Problem-solving strategies
1. Breaking down complex problems
2. Algorithmic thinking
B. Data modeling and analysis
1. Identifying data requirements
2. Structuring data for analysis and processing
C. Visualization techniques for data interpretation

VII. Case Studies and Practical Exercises
A. Solving real-life problems using OOP concepts and best practices
B. Group exercises to apply analytical thinking and design patterns
C. Code walkthroughs and discussions on solutions architecture

Title: Advanced Python Programming and Object-Oriented Techniques

I. Advanced Concepts in Object-Oriented Programming (OOP)
- A. Advanced Class Concepts
  - 1. Class attributes vs. instance attributes
  - 2. Class methods vs. instance methods
  - 3. Private and protected attributes/methods
- B. Shallow vs. Deep Operations
  - 1. Understanding shallow copy and deep copy
  - 2. Implementing deepcopy with deepcopy() function
- C. Polymorphism and Duck Typing
  - 1. Dynamic typing in Python
  - 2. Implementing polymorphic behavior
- D. Special Methods (Magic Methods)
  - 1. Introduction to special methods
  - 2. Commonly used special methods (e.g., __init__, __str__, __repr__)

II. Advanced Techniques in Object-Oriented Programming
- A. Static and Class Methods
  - 1. Defining static and class methods
  - 2. Use cases and benefits
- B. Abstract Classes and Interfaces
  - 1. Using abstract base classes (ABCs)
  - 2. Implementing interfaces with abstract methods
- C. Method Overriding and Method Resolution Order (MRO)
  - 1. Understanding method overriding
  - 2. Dealing with method resolution order conflicts

III. Composition, Inheritance, and Subclassing
- A. Composition over inheritance
  - 1. Designing classes with composition
  - 2. Advantages and disadvantages
- B. Inheritance and subclassing
  - 1. Extending base classes
  - 2. Method overriding and super() function usage
- C. Multiple inheritance and method resolution order (MRO)

IV. Advanced Exception Handling Techniques

A. Custom Exception Classes
  1. Creating custom exception classes
  2. Handling custom exceptions in code
B. Handling Exceptions Gracefully
  1. Using try-except-else-finally blocks effectively
  2. Reraising exceptions and traceback manipulation

V. Serialization and Persistence
  A. Pickle Module
    1. Serializing Python objects with pickle
    2. Deserializing pickled objects
  B. Shelve Module
    1. Storing Python objects in persistent storage
    2. Retrieving and updating objects from shelves

VI. Decorators and Metaprogramming
  A. Decorators
    1. Introduction to decorators
    2. Creating and using decorators in Python
  B. Metaprogramming
    1. Dynamically creating classes and functions
    2. Metaclasses and their usage

VII. Case Studies and Practical Exercises
  A. Hands-on exercises to implement advanced OOP concepts
  B. Exploring real-world scenarios and applying advanced techniques
  C. Code reviews and discussions on best practices

Title: Python Best Practices, Standardization, and Coding Conventions

I. Introduction to Python Coding Standards
- A. Importance of coding standards in Python development
- B. Overview of PEP (Python Enhancement Proposal) and its role in defining standards

II. Understanding PEP 8
- A. Overview of PEP 8
- B. Key principles and guidelines
  - 1. Indentation and whitespace
  - 2. Naming conventions
  - 3. Import formatting
  - 4. Line length and wrapping
  - 5. Comments and documentation
- C. Tools for enforcing PEP 8 compliance (e.g., linters, formatters)

III. Implementing PEP 8 Conventions
- A. Setting up code editors and IDEs for PEP 8 compliance
- B. Using tools like pylint, flake8, or black to automate adherence to PEP 8
- C. Refactoring existing code to comply with PEP 8 guidelines

IV. The Zen of Python (PEP 20)
- A. Introduction to the Zen of Python
- B. Exploring the guiding principles
  - 1. Readability counts
  - 2. Simple is better than complex
  - 3. Explicit is better than implicit
  - 4. There should be one-- and preferably only one --obvious way to do it
- C. Applying the Zen of Python in code design and development

V. PEP 257: Docstring Conventions
- A. Overview of PEP 257
- B. Importance of docstrings for code documentation
- C. Docstring conventions and formatting guidelines
- D. Writing meaningful and descriptive docstrings for functions, modules, and classes

VI. Applying Best Practices in Real-world Scenarios
- A. Reviewing code examples and identifying deviations from PEP 8 and Zen of Python

   B. Refactoring code to adhere to coding standards
   C. Discussing the impact of following best practices on code quality and maintainability

VII. Avoiding Common Errors and Mistakes
   A. Understanding common pitfalls in Python development
   B. Best practices for error handling, debugging, and troubleshooting
   C. Strategies for writing clean, efficient, and Pythonic code

VIII. Case Studies and Practical Exercises
   A. Hands-on exercises to apply PEP 8, Zen of Python, and docstring conventions
   B. Analyzing code snippets and discussing improvements based on coding standards
   C. Group discussions on best practices and coding conventions in Python development

Title: Building Graphical User Interfaces (GUIs) with Tkinter in Python

I. Introduction to Tkinter and GUI Development
- A. Overview of Tkinter and its integration with Python
- B. Importance of GUI development in Python applications
- C. Introduction to basic GUI components (widgets)

II. Getting Started with Tkinter
- A. Setting up Tkinter environment
- B. Creating a simple Tkinter window
- C. Understanding the mainloop() function

III. Tkinter Widgets and Layout Management
- A. Common Tkinter widgets
  - 1. Labels, Buttons, Entry fields
  - 2. Checkboxes, Radio buttons, Dropdown menus
  - 3. Text and Canvas widgets
- B. Organizing widgets with layout managers (pack, grid, place)
- C. Styling and customizing widgets

IV. Handling Events in Tkinter
- A. Introduction to event-driven programming
- B. Binding events to widgets
- C. Responding to user interactions (e.g., button clicks, mouse events)

V. Creating Simple GUI Applications
- A. Building a basic calculator application
- B. Developing a Tic-Tac-Toe game
- C. Designing a simple text editor or form

VI. Advanced Tkinter Concepts
- A. Using frames for organizing complex layouts
- B. Creating custom dialogs and message boxes
- C. Adding images and icons to GUI applications

VII. Real-World GUI Projects
- A. Exploring examples of real-world GUI applications built with Tkinter
- B. Discussing design considerations and best practices

C. Brainstorming ideas for potential GUI projects

VIII. Best Practices and Tips for GUI Development
- A. Writing clean and maintainable GUI code
- B. Handling user input validation and error handling
- C. Optimizing performance and responsiveness

IX. Testing and Debugging GUI Applications
- A. Strategies for testing GUI functionality
- B. Using debugging tools and techniques
- C. Addressing common bugs and issues in GUI development

Title: Introduction to Network Programming in Python

I. Understanding Network Programming Basics
   A. Overview of network programming and its importance
   B. Introduction to sockets and their role in network communication
   C. Overview of HTTP protocol and RESTful APIs

II. Working with Sockets in Python
   A. Introduction to socket programming
   B. Creating sockets in Python
   C. Establishing connections and handling data transfer
   D. Handling errors and exceptions in socket programming

III. Introduction to RESTful APIs
   A. Understanding RESTful architecture and principles
   B. Overview of CRUD operations (Create, Read, Update, Delete)
   C. Using HTTP methods: GET, POST, PUT, DELETE

IV. Communicating with RESTful APIs in Python
   A. Sending HTTP requests using the requests library
   B. Handling responses and parsing JSON data
   C. Performing CRUD operations with RESTful APIs

V. Working with JSON and XML Files
   A. Overview of JSON and XML formats
   B. Reading and writing JSON data in Python
   C. Parsing XML files using libraries like xml.etree.ElementTree

VI. Implementing HTTP Methods
   A. GET Method: Retrieving data from external resources
   B. POST Method: Sending data to servers
   C. PUT and DELETE Methods: Updating and deleting resources

VII. Practical Examples and Projects
   A. Building a simple client-server application using sockets
   B. Creating a Python script to interact with a RESTful API
   C. Integrating network communication into a larger project

VIII. Best Practices and Tips for Network Programming
   A. Handling authentication and security in network communication
   B. Error handling and graceful degradation
   C. Performance optimization techniques

IX. Real-World Applications and Use Cases
   A. Exploring examples of network programming in web development, IoT, and more
   B. Discussing real-world challenges and solutions
   C. Brainstorming ideas for potential network programming projects

Title: Data Processing and File Manipulation in Python

I. Introduction to File Processing in Python
   A. Importance of file processing in data analysis and software development
   B. Overview of different types of files and formats
   C. Introduction to file handling in Python

II. Interacting with SQLite Databases
   A. Understanding SQLite databases and their advantages
   B. Connecting to SQLite databases using the sqlite3 module
   C. Executing SQL queries and transactions
   D. Retrieving and manipulating data from SQLite databases

III. Processing XML Files
   A. Introduction to XML (eXtensible Markup Language)
   B. Parsing XML files using the xml module
   C. Navigating and extracting data from XML tree structures
   D. Creating and modifying XML files programmatically

IV. Reading and Writing CSV Files
   A. Understanding the CSV (Comma-Separated Values) format
   B. Using the csv module for reading and writing CSV files
   C. Handling different CSV formats and options
   D. Manipulating data in CSV files (e.g., sorting, filtering)

V. Managing Log Messages
   A. Importance of logging in software development and troubleshooting
   B. Introduction to the logging module for creating log messages
   C. Configuring logging levels, formats, and handlers
   D. Writing log messages to files and other destinations

VI. Working with Configuration Files
   A. Overview of configuration files and their role in software configuration
   B. Introduction to the configparser module for parsing configuration files
   C. Reading and writing configuration data
   D. Handling configuration options and sections

VII. Practical Examples and Projects

A. Building a data processing pipeline using SQLite databases and CSV files
B. Parsing and analyzing XML data from external sources
C. Implementing logging in a Python application and managing log messages
D. Reading and updating configuration settings for a Python program

VIII. Best Practices and Tips for File Processing
A. Error handling and exception management
B. Data validation and sanitization techniques
C. Performance optimization strategies

IX. Real-World Applications and Use Cases
A. Exploring examples of file processing in data analysis, software development, and troubleshooting
B. Discussing real-world challenges and solutions
C. Brainstorming ideas for potential file processing projects

Title: Intermediate Python: Automated Software Testing

I. Introduction to Automated Software Testing
- A. Importance of automated testing in software development
- B. Overview of software testing theory and terminology
- C. Understanding the test pyramid and its layers

II. Understanding Code Coverage
- A. Definition and importance of code coverage
- B. Types of code coverage (line, branch, path coverage)
- C. Tools for measuring code coverage in Python

III. Test Automation in Python
- A. Introduction to test automation frameworks (e.g., unittest, pytest)
- B. Writing and running automated tests in Python
- C. Organizing test suites and test cases

IV. Code Refactoring and Testing
- A. Importance of refactoring for testability
- B. Strategies for refactoring code for better test coverage
- C. Writing tests for refactored code

V. Assertions, Context Managers, and Decorators
- A. Using assertions for automated testing
- B. Introduction to context managers and their role in testing
- C. Implementing decorators for test setup and teardown

VI. Types of Automated Tests
- A. End-to-End (E2E) tests
- B. Integration tests
- C. Unit tests
- D. Writing and running each type of test in Python

VII. Test Documentation and Reporting
- A. Importance of documenting tests
- B. Writing clear and descriptive test documentation
- C. Generating test reports and interpreting results

VIII. Test-Driven Development (TDD) and Behavior-Driven Development (BDD)
  A. Overview of TDD and BDD approaches
  B. Writing tests before implementing code (TDD)
  C. Using BDD frameworks like Behave for test automation

IX. Practical Exercises and Examples
  A. Hands-on exercises to write and run automated tests
  B. Building test suites for sample Python projects
  C. Demonstrating the benefits of automated testing through real-world examples

X. Best Practices in Automated Testing
  A. Writing maintainable and reusable tests
  B. Test isolation and test data management
  C. Continuous integration and continuous testing practices

Title: Intermediate Python for Data Analysis

I. Introduction to Data Analysis with Python
   A. Importance of Python in data analysis
   B. Overview of data analysis process
   C. Introduction to main toolkits and methodologies

II. Obtaining Data
   A. Accessing data from various sources (files, databases, APIs)
   B. Using libraries like pandas to import and load data
   C. Web scraping techniques for data extraction

III. Data Cleaning and Preprocessing
   A. Identifying and handling missing data
   B. Removing duplicates and outliers
   C. Data normalization and standardization

IV. Exploratory Data Analysis (EDA)
   A. Descriptive statistics and data summarization
   B. Visualization techniques (plots, charts, graphs)
   C. Correlation analysis and feature selection

V. Data Analysis and Manipulation with pandas
   A. Introduction to pandas library for data manipulation
   B. Filtering, sorting, and grouping data
   C. Applying functions and transformations to data

VI. Statistical Analysis with Python
   A. Introduction to statistical methods and hypothesis testing
   B. Performing statistical analysis using libraries like scipy and statsmodels
   C. Interpreting results and drawing conclusions

VII. Presenting Data
   A. Creating informative and visually appealing data visualizations
   B. Using libraries like Matplotlib and Seaborn for plotting
   C. Designing dashboards and interactive visualizations

VIII. Real-World Data Analysis Projects

A. Hands-on projects to apply data analysis concepts and techniques
B. Analyzing real-world datasets and drawing insights
C. Collaborative data analysis and peer review

IX. Best Practices in Data Analysis with Python
A. Writing clean and efficient code for data analysis tasks
B. Documentation and reproducibility of analysis steps
C. Version control and collaboration in data analysis projects