AWS Cloud Institute: Developer Fundamentals 2 Syllabus

Course Overview
In this course, you will take a deeper dive into the concepts presented in Developer Fundamentals 1, including Python data structures, classes, object-oriented programming, and functions, in addition to new concepts such as algorithms, dynamic programming, math modules, and RegEx. You will learn to describe the features and use cases for AWS Lambda, explain the structure of schemas and relational databases, analyze network connectivity, and protect the confidentiality and integrity of data in-transit.

Course Structure
Developer Fundamentals 2 is a combination of digital e-learning content ("modules") that you can read, watch, and engage with on your own time (asynchronous content) and instructor-led training sessions (ILTs) that focus on the topics presented in the asynchronous content.

Developer Fundamentals 2 modules include text, video, audio, hands-on learning activities, and knowledge checks. There are over a dozen ILT sessions offered each week that can be watched live or on-demand. Although ILT attendance is not required for course completion, each session is an opportunity to connect with and learn from your AWS instructor and other Developer Fundamentals 2 students.

As with all AWS Cloud Institute courses, the expectation is that you will complete your modules and associated assessments on a weekly basis. You also have the option to attend the ILTs live or watch the recorded version that will be posted online after each session. Taking advantage of this "flipped classroom" model is the best way to ensure that you develop the professional and technical skills required to be successful as a cloud application developer.

Course Sessions

Table 1. Module Overview by Content Type

	Content Type	Percentage of total course time (by hours)
1	e-learning (text, video, audio)	~50% (42 hours)
2	Ungraded assessments	~11% (9 hours)
3	Ungraded activities (games, case studies, simulations)	~12% (10 hours)
4	Hands-on learning activities (labs, activities and/or challenges)	~14% (12 hours)
5	Graded assessments	~13% (11 hours)
6	**Total Time**	**100% (84 hours)**

Instructor-Led Training (ILT) Sessions
Mondays: Weekly Stand-Up, 12:00PM, 2:00PM & 4:00PM EST

Tuesdays:	Weekly Spotlight, 12:00PM, 2:00PM & 4:00PM EST
Wednesdays:	Weekly Focus Session, 12:00PM, 2:00PM & 4:00PM EST
Thursdays:	Weekly Labology Session, 12:00PM, 2:00PM & 4:00PM EST
Fridays:	Weekly Fun Friday Event, 4:00PM EST
M, T, W, Th:	Office Hours, 9:00-11:30 AM, 05:30-8:00PM EST

All ILT sessions, with the exception of office hours, are recorded and will be available on-demand for you to watch when you have time. Multiple live ILT sessions are offered each day to accommodate a range of schedules. All ILT sessions, with the exception of Fun Friday events, are also offered on a PST schedule (i.e., Weekly Stand-Up 9:00AM, 11:00AM & 1:00PM PST).

Prerequisites
Developer Fundamentals 1

Assessment and Grading Policy
Courses are offered with no letter grades. Learners must achieve a passing score of 85% or higher on all weekly final assessments to pass each course. Each final assessment consists of a 20-question multiple-choice quiz presented at the end of each week's assigned module. Learners have unlimited opportunities to achieve a passing score on each final assessment.

Attendance at ILT sessions is strongly encouraged, but attendance does not count toward course completion. Learners can use the ILT sessions to explore concepts presented in the online learning content in greater depth, connect with AWS instructors and ACI learners, and prepare for the weekly final assessment.

Course Completion Requirements
All learners must complete eleven (11) weekly final assessments with a score of 85% or higher to receive credit for Developer Fundamentals 2. ILT attendance (or watching ILT sessions on demand) is strongly encouraged, but is not required for course completion.

Week 1: Python 2 – Object Oriented Programming

Goal:
Explore object-oriented programming concepts like classes, objects, inheritance, and polymorphism to build robust and modular Python applications.

Learning Objectives:
- Describe programming paradigms
- Discuss classes and objects
- Describe information hiding
- Describe inheritance
- Describe polymorphism

Module Outline:
1. **Review of Developer Fundamentals 1 and Preview of Developer Fundamentals 2**
    a. Introduction: Review of Developer Fundamentals 1 and Preview of Developer Fundamentals 2

   b. Review of Developer Fundamentals 1
   c. Coming Up: Preview of Developer Fundamentals 2
2. **Programming Paradigms**
   a. Introduction: Programming Paradigms
   b. The Four Main Programming Paradigms
   c. Knowledge Check
3. **Classes and Objects**
   a. Introduction: Classes and Objects
   b. Using Classes and Objects in Python
   c. Class Methods and Static Methods
   d. Declaring a Class in Python
   e. Initializing Objects
   f. Activity: Creating Classes and Objects
   g. Object Printing
   h. Access Modifiers
   i. Knowledge Check
4. **Information (Data) Hiding**
   a. Introduction: Information (Data) Hiding
   b. Core Concepts in OOP
   c. Information (Data) Hiding
   d. Encapsulation
   e. Getters and Setters
   f. Understanding Encapsulation Using Examples
   g. Data Abstraction
   h. Knowledge Check
5. **Inheritance**
   a. Introduction: Inheritance
   b. Exploring Inheritance
   c. Syntax and Terminologies
   d. Types of Inheritance
   e. Inheritance and the super() Function
   f. Advantages of Inheritance
   g. Method Overriding
   h. Knowledge Check
6. **Polymorphism**
   a. Introduction: Polymorphism
   b. Exploring Polymorphism
   c. Implementing Polymorphism Using Methods
   d. Implementing Polymorphism Using Inheritance
   e. Operator Overloading
   f. Knowledge Check
7. **Hands-On Lab Activity**
   a. Lab: Introduction to Object-Oriented Programming

Activities:

- **Challenge: Changing attributes**
    - Add new attributes to existing data structures.
- **Challenge: Instantiating an object**
    - Study the code snippet and add lines that will instantiate the object.
- **Lab: Introduction to Object-Oriented Programming (60 Mins)**
    - This lab provides you with an introduction to object-oriented programming concepts using Python. You start by learning common troubleshooting methods to ensure that the code runs properly. You then expand on an existing application by introducing new classes that inherit from and extend the original functionality. Through hands-on activities, you gain experience with core object-oriented principles, like inheritance, while becoming familiar with the Python class syntax.

Videos (Total Time 19:45):

- Review of Developer Fundamentals 1 (7:29)
- Coming Up: Preview of Developer Fundamentals 2 (1:57)
- Classes and Objects (3:41)
- Core Concepts in OOP (6:38)

Assessments:

For each module, you will complete a series of ungraded assessments to help measure your progress. At the end of each week, you will complete a graded final assessment that tests your knowledge of the learning objectives presented. A passing grade of 85% is required to move forward with your learning, and you can take final assessments as many times as needed.

Week 2: Python 2 – Data Structures

Goal:

Explore fundamental data structures like arrays, stacks, and queues, and analyze their use cases and time complexities to efficiently store and manipulate data in Python applications.

Learning Objectives:

- Explain the importance of data structures
- Use terminology to discuss memory and data-storage concepts
- Identify Python built-in data structures
- Implement arrays, stacks, and queues in Python
- Analyze a simple case study to determine which data structure to use

Module Outline:

1. **Data Structures Overview**
    a. Introduction: Data Structures Overview
    b. Why Data Structures Are Important
    c. Comparison of Built-in Python Data Structures
    d. Memory and Data Storage
    e. Knowledge Check
2. **Stacks and Queues**

   a. Introduction: Stacks and Queues
   b. About Stacks and Queues
   c. Implementing Stacks
   d. Implementing Queues
   e. Knowledge Check

3. **Arrays**
   a. Introduction: Arrays
   b. Array Overview
   c. Implementing Arrays
   d. Knowledge Check
4. **Case Study and Time Complexity**
   a. Introduction: Case Study and Time Complexity
   b. Case Study
   c. Overview of Time Complexity and Data Structures
   d. Knowledge Check
5. **Hands-On Lab Activity**
   a. Lab: Stacks and Queues

Activities:

- **Case study**
  - AnyCompany Software
- **Lab: Stacks and Queues (60 Mins)**
  - This lab teaches you how to implement and use queues and stacks in Python. Queues and stacks are widely used linear data structures for accessing data in first in, first out (FIFO) or last in, first out (LIFO) fashion, respectively. You will learn how to use different techniques for creating queues and stacks. You will also learn how to use their associated methods for adding and removing data.

Assessments:

For each module, you will complete a series of ungraded assessments to help measure your progress. At the end of each week, you will complete a graded final assessment that tests your knowledge of the learning objectives presented. A passing grade of 85% is required to move forward with your learning, and you can take final assessments as many times as needed.

Week 3: Python 2 – Advanced Data Structures Part 1

Goal:

Explore advanced data structures like linked lists, hash tables, and heaps. Analyze their advantages and disadvantages, and implement basic operations to efficiently store and retrieve data in Python applications.

Learning Objectives:

- Describe advanced data structures, such as linked lists, hash tables, and heaps
- Analyze the advantages and disadvantages of linked lists, hash tables, and heaps
- Write basic code to perform tasks with linked lists, hash tables, and heaps

Module Outline:

1. **Linked Lists**
    a. Introduction: Linked Lists
    b. Linked Lists Overview
    c. Activity: Singly Linked Lists
    d. Activity: Doubly Linked Lists
    e. Activity: Programming Subway Stops
    f. Knowledge Check
2. **Hash Tables**
    a. Introduction: Hash Tables
    b. Hash Tables Overview
    c. How Hash Tables Work
    d. Handling Collisions
    e. Activity: Hash Tables and Equality
    f. Knowledge Check
3. **Heaps**
    a. Introduction: Heaps
    b. Heaps Overview
    c. Terminology Review
    d. Heap Data Structure
    e. Activity: Implementing a Heap
    f. Knowledge Check

Activities:

- **Activity: Singly linked lists**
    - Examine the code sample for a singly linked list and build one of your own.
- **Activity: Doubly linked lists**
    - Create a doubly linked list and traverse the list forward and back.
- **Activity: Programming subway stops**
    - Assist in the creation of a subway transit system, and determine which lists to use and how.
- **Activity: Hash tables and equality**
    - Check for duplicates and test for equality using hash tables.
- **Challenge: Hash tables**
    - Apply what you have learned about hash tables and hash functions to a real-life scenario.
- **Activity: Implementing a heap**
    - Use the heapify() function to sort a list of numbers so that element 0 becomes the minimum value.
- **Challenge: Determine maximum value in a heap**
    - Find the maximum value in a heap using heapify process.

Videos (Total Time 4:59):

- Linked Lists Overview (4:59)

Assessments:
For each module, you will complete a series of ungraded assessments to help measure your progress. At the end of each week, you will complete a graded final assessment that tests your knowledge of the learning objectives presented. A passing grade of 85% is required to move forward with your learning, and you can take final assessments as many times as needed.

Week 4: Python 2 – Advanced Data Structures Part 2

Goal:
Explore advanced tree data structures like binary trees (AVL, Red-Black), trie, and graphs; understand their properties and learn how to implement algorithms to efficiently store, search, and traverse data in Python applications.

Learning Objectives:
- Review highlights from the previous week's material
- Understand trees as a data structure
- Define the trie data structure
- Define the graph data structure
- Discuss the graphing algorithm

Module Outline:
1. **Trees as a Data Structure**
    a. Introduction: Tree Data Structure
    b. Quick Review of Data Structures in Python
    c. Exploring Trees as a Data Structure
    d. Knowledge Check
2. **Binary Trees: AVL & Red-Black Trees**
    a. Introduction: Binary Trees: AVL & Red-Black Trees
    b. Binary Search Trees
    c. AVL Trees
    d. Red-Black Trees
    e. Knowledge Check
3. **Trie Data Structures**
    a. Introduction: Trie Data Structures
    b. Trie Data Structure and Purpose
    c. Searching for a Value in a Trie
    d. Using Autocomplete with the Trie Data Structure
    e. Knowledge Check
4. **Graph Data Structure**
    a. Introduction: Graph Data Structure
    b. Graph Data Structure and Uses
    c. Graphs in the Real World
    d. Graphs in Python Code
    e. Knowledge Check
5. **Graphing Algorithms**

   a. Introduction: Graphing Algorithms
   b. Shortest Distance: Dijkstra Algorithm
   c. Shortest Distance: Bellman-Ford Algorithm
   d. Minimum Spanning Tree Algorithms
   e. Traveling Salesperson Algorithms
   f. Knowledge Check

6. **Hands-On Lab Activity**
   a. Lab: Finding the Shortest Path With Directed and Undirected Graphs

Activities:

- **Challenge: Adding generations to a tree**
  - Add additional children, grandchildren, and great-grandchildren to the tree. Specify a new root and print the associated group.
- **Challenge: Editing binary search trees**
  - Change parts of the code to create different binary search trees.
- **Challenge: Working with a trie**
  - Change the strings entered into code and create a work counter to print the total number of words in the trie.
- **Lab: Finding the Shortest Path With Directed and Undirected Graphs (60 Mins)**
  - In this lab, you will work with graph structures, build a graph structure and fill it with data, and implement Dijkstra's algorithm to find the shortest route.

Videos (Total Time 63:30):

- Exploring Trees as a Data Structure (7:50)
- Binary Search Trees (4:42)
- AVL Trees (9:49)
- Trie Data Structure and Purpose (6:02)
- Inserting values into a trie data structure (5:10)
- Graph Data Structure and Uses (8:24)
- Shortest Distance: Dijkstra Algorithm (6:50)
- Shortest Distance: Bellman-Ford Algorithm (6:50)
- Traveling Salesperson Algorithms (7:53)

Assessments:

For each module, you will complete a series of ungraded assessments to help measure your progress. At the end of each week, you will complete a graded final assessment that tests your knowledge of the learning objectives presented. A passing grade of 85% is required to move forward with your learning, and you can take final assessments as many times as needed.

Week 5: Python 2 – Algorithm Part 1

Goal:

Explore importing data using NumPy and visualizing data with Matplotlib in Python2. Reinforce your understanding of data structures, iterables, and basic programming concepts through hands-on activities and assessments.

Learning Objectives:

- Describe algorithms
- Describe recursion and backtracking
- Describe complexity analysis
- Describe search algorithm

Module Outline:

1. **Algorithms**
    a. Introduction: Algorithms Part 1
    b. Algorithms Overview
    c. Knowledge Check
2. **Recursion and Backtracking**
    a. Recursion Overview
    b. Classic Recursive Examples
    c. Example: The Tower of Hanoi
    d. Backtracking
    e. Knowledge Check
3. **Complexity Analysis**
    a. Complexity Problems
    b. Analyzing Basic Python Code
    c. Common Complexity Scenarios
    d. Basic For Loop
    e. For Loop With Increments
    f. Basic Nested For Loop
    g. Nested For Loop With Index Modification
    h. Nested Loops with Math Operations
    i. Knowledge Check
4. **Search Algorithms**
    a. Introduction to Search Algorithms
    b. Linear Search Algorithm
    c. Binary Search Algorithm
    d. Comparing Linear and Binary Search Algorithms
    e. Other Search Algorithms
    f. Knowledge Check

Activities:

- **Challenge: Calculate the sum**
    - Use what you have learned about factorials to write a recursive function that calculates the sum of the positive integers from n to 1.

- **Challenge: Solve the 4-Queens problem**
    - Use backtracking to solve the 4-Queens problem in three steps.

Videos (Total Time 8:56):

- Algorithms Overview (4:59)
- Recursion Overview (3:57)

Assessments:

For each module, you will complete a series of ungraded assessments to help measure your progress. At the end of each week, you will complete a graded final assessment that tests your knowledge of the learning objectives presented. A passing grade of 85% is required to move forward with your learning, and you can take final assessments as many times as needed.

Week 6: Python 2 – Algorithms Part 2

Goal:

Gain a comprehensive understanding of sorting algorithms, algorithm analysis, and time and space complexities, which will help you to analyze and efficiently implement various sorting algorithms in Python.

Learning Objectives:

- Describe the sorting process
- Describe the built-in sorting process of algorithms
- Describe the algorithm analysis process
- Describe time and space complexities in algorithms

Module Outline:

1. **Getting Started**
    a. Review of Algorithms Part 1
2. **Understanding Sorting**
    a. Introduction: Sorting Algorithms
    b. What is Sorting?
    c. Sorting Algorithms and Their Benefits
    d. Knowledge Check
3. **Python's Built-in Sorting Algorithms**
    a. Built-in sorting algorithms
    b. The sort() method
    c. The sorted() method
    d. Knowledge Check
4. **Analyzing Algorithms**
    a. Importance of algorithm analysis
    b. Complexity analysis
    c. Knowledge Check
5. **Time and Space Complexities**
    a. Big O Notation

   b. Constant Time Complexity
   c. Logarithmic Time Complexity
   d. Linear Time Complexity
   e. Quasilinear Time Complexity
   f. Quadratic Time Complexity
   g. Exponential Time Complexity
   h. Factorial Time Complexity
   i. Comparing Time Complexities
   j. Space Complexity
   k. Knowledge Check

6. **Sorting Algorithms**
   a. Types of Sorting Algorithms
   b. Bubble Sort Algorithm
   c. Selection Sort Algorithm
   d. Insertion Sort Algorithm
   e. Merge Sort Algorithm
   f. Quicksort Algorithm
   g. Comparing Sorting Algorithms
   h. Knowledge Check
7. **Hands-On Lab Activity**
   a. Lab: Benchmarking Your Algorithms

Activities:

- **Challenge: Sort in reverse alphabetical order**
  - Change the code so that the output will be sorted in reverse alphabetical order.
- **Challenge: Sort by length**
  - Change the code so that the output will be sorted by length, in ascending order.
- **Challenge: Sort based on the second element of each tuple in reverse order**
  - Add code to print the sorted list based on the second element of each tuple in reverse order.
- **Lab: Benchmarking Your Algorithms (120 Mins)**
  - This lab gives you hands-on experience in benchmarking the performance of sort algorithms. You learn how to create a benchmark function, generate input data, and interpret the benchmark results.

Videos (Total Time 16:27):

- Review of Algorithms Part 1 (4:18)
- Complexity analysis (1:20)
- Bubble Sort Algorithm (2:06)
- Selection Sort Algorithm (1:30)
- Insertion Sort Algorithm (1:57)
- Merge Sort Algorithm (1:35)
- Quicksort Algorithm (3:41)

Assessments:

For each module, you will complete a series of ungraded assessments to help measure your progress. At the end of each week, you will complete a graded final assessment that tests your knowledge of the learning objectives presented. A passing grade of 85% is required to move forward with your learning, and you can take final assessments as many times as needed.

Week 7: Python 2 – Dynamic Programming

Goal:

Learn about the concept of dynamic programming, including memoization and tabulation techniques, and their applications in solving complex problems. Develop an understanding of dynamic programming principles and common use cases. Gain hands-on experience in implementing dynamic programming solutions, particularly for problems like the Knapsack problem, the Coin Change problem, and the Fibonacci sequence.

Learning Objectives:

- Describe dynamic programming
- Describe memoization and tabulation
- Describe common dynamic programming use cases

Module Outline:

1. **Dynamic Programming**
    a. Introduction: Dynamic Programming
    b. Dynamic Programming Overview
    c. Dynamic Programming and Recursion
    d. Time and Space Complexity Review
    e. Knowledge Check
2. **Memoization and Tabulation**
    a. Memoization and Tabulation Overview
    b. Memoization with Factorials
    c. Tabulation with Factorials
    d. Knowledge Check
3. **Use Cases**
    a. Use Cases Overview
    b. Knapsack Problem: Memoization
    c. Coin Change Problem: Tabulation
    d. Knowledge Check
4. **Fibonacci Examples**
    a. Fibonacci Overview
    b. Naive Recursive Approach
    c. Top-Down Approach: Memoization
    d. Bottom-Up Approach: Tabulation
    e. Use Cases Summary
    f. Knowledge Check
5. **Hands-On Lab Activity**
    a. Lab: Writing Code to Solve Subset Sum Problems

Activities:

- **Challenge: Print the factorials**
    - Update the tabulation code to create a table of the numbers 0–100. Then, instead of printing a range of numbers, print the sum of several factorials.
- **Challenge: Code speed check**
    - Experiment with changing the range of your naive recursive code.
- **Challenge: Code speed bumps**
    - Explore using a memoization approach to solving for the Fibonacci sequence.
- **Challenge: Bottom-up Fibonacci sequencing**
    - Explore the bottom-up approach for storing the results of subproblems in the Fibonacci sequence using tabulation.
- **Lab: Writing Code to Solve Subset Sum Problems (120 Mins)**
    - Learn how to solve problems using dynamic programming. Focus on the subset sum problem and solve it using methods of recursion and memoization.

Videos (Total Time 9:53):

- Dynamic Programming Overview (2:07)
- Memoization and Tabulation Overview (4:14)
- Use Cases Overview (3:32)

Assessments:

For each module, you will complete a series of ungraded assessments to help measure your progress. At the end of each week, you will complete a graded final assessment that tests your knowledge of the learning objectives presented. A passing grade of 85% is required to move forward with your learning, and you can take final assessments as many times as needed.

Week 8: Python 2 – Technical Interviews

Goal:

Prepare for technical coding interviews with an overview of the interview process, essential topics, and strategies. Review fundamental concepts that are commonly tested in coding interviews, such as object-oriented programming, data structures, algorithms, and dynamic programming. Review sample interview question and complete hands-on activities and weekly assessments to improve your readiness. Get additional guidance on effective interview strategies to help you approach coding interviews with confidence.

Learning Objectives:

- Describe the purpose of coding interviews
- Discuss interview strategies

Module Outline:

1. **Coding Interviews**
    a. Introduction: Coding Interviews
    b. The Essentials

   c. Knowledge Check
2. **Developer Fundamentals & Interviews**
   a. Introduction: Developer Fundamentals and Interviews
   b. Object-Oriented Programming and Other Programming Paradigms
   c. Data Structures
   d. Linked Lists and Heaps
   e. Trees, Trie, and Graphs
   f. Algorithms
   g. Dynamic Programming
   h. Knowledge Check
3. **Additional Coding Questions**
   a. Introduction: Additional Coding Questions
   b. Sample Interview Questions
   c. Knowledge Check
4. **Interview Strategies**
   a. Introduction: Interview Strategies
   b. Knowledge Check

Activities:

- **Scenario Activity: Example interview questions**
  - Test your Python knowledge and preparedness by answering a series of technical interview questions.

Videos (Total Time 10:45):

- The Essentials (3:30)
- Trees, Trie, and Graphs (3:45)
- Algorithms (3:30)

Assessments:

For each module, you will complete a series of ungraded assessments to help measure your progress. At the end of each week, you will complete a graded final assessment that tests your knowledge of the learning objectives presented. A passing grade of 85% is required to move forward with your learning, and you can take final assessments as many times as needed.

Week 9: Python 2 – Defining NumPy and Pandas

Goal:

Learn about NumPy and pandas, and explore the common use cases for each. You will also learn about arrays, the pandas Series, and DataFrames.

Learning Objectives:

- Getting started with NumPy
- Creating NumPy arrays
- Defining linear algebra terms

- Pandas Series and DataFrames

Module Outline:

1. **Defining NumPy and Pandas**
    a. Introduction: Defining NumPy and Pandas
    b. NumPy and Pandas
    c. Installing NumPy
    d. Identifying NumPy Data Types
    e. Knowledge Check
2. **Multidimensional NumPy Arrays**
    a. Introduction: Multidimensional NumPy Arrays
    b. Creating Multidimensional Arrays
    c. Manipulations on Arrays
    d. Creating an Array from Tabular Data
    e. Knowledge Check
3. **Defining Linear Algebra Terms**
    a. Introduction: Defining Linear Algebra Terms
    b. Multidimensional Arrays and Linear Algebra
    c. Knowledge Check
4. **Pandas Series and DataFrames**
    a. Introduction: Pandas Series and DataFrames
    b. Installing Pandas
    c. Defining Pandas Series
    d. Convert a NumPy Array to a Pandas Series
    e. Pandas DataFrames
    f. Knowledge Check

Activities:

- **Challenge: Convert the following data type to integers**
    - Complete the code for the activities and observe the output.
- **Challenge: What are the start, stop, and step values for breaks of 2.5 hour?**
    - Complete the code with the correct start, stop, and step values where the question marks (?) are shown in the code, and then run the code and observe the output.
- **Challenge: What is the temperature at opening and closing time?**
    - Index the array for temperatures at opening and closing times.
- **Challenge: Slice an array for specific elements**
    - Complete the code for the activities and observe the output.
- **Challenge: Find the index values for odd numbers in the array**
    - Use the code to find the index values of the odd numbers in the array.
- **Challenge: Searching an array**
    - Print the array, filter the array, calculate the results.
- **Challenge: Filter the array for values less than 6**
    - Filter the array and print the new filtered array.
- **Challenge: Is there a variation in temperature between the first to second Saturday?**
    - Create two new arrays, find the variation, confirm the output.
- **Challenge: Create a pandas Series about the price of olive oil from January to September 2023**

  - Create a pandas Series of the price of olive oil and display it on the screen.
- **Challenge: Index a pandas Series with city names**
  - Created a pandas Series, print and label the pandas Series with index values.
- **Lab: Data Processing with NumPy and Pandas (60 mins)**
  - This lab provides hands-on practice with core data analysis skills using Python. You go through the key steps involved in working with data programmatically. First, you import a dataset into Python. Then, you use numerical analysis techniques from NumPy to explore the data. Next, you use Pandas for additional data exploration tasks, like filtering and sorting. Finally, you create data visualizations to present the insights from your analysis.

Videos (Total Time 22:37):
- Python Lists and NumPy arrays: Similarities and Differences (3:29)
- Install and Import NumPy (1:20)
- Slicing an Array (5:04)
- Creating an Array from Tabular Data (4:20)
- Define Linear Algebra Terms: Scalar, Vector, Matrix, and Tensor (1:42)
- Install and Test Pandas (1:31)
- NumPy and Pandas: Similarities and Differences (2:18)
- Create Pandas Series (2:27)
- Creating a DataFrame (1:26)

Assessments:
For each module, you will complete a series of ungraded assessments to help measure your progress. At the end of each week, you will complete a graded final assessment that tests your knowledge of the learning objectives presented. A passing grade of 85% is required to move forward with your learning, and you can take final assessments as many times as needed.

Week 10: Python 2 – Regular Expressions

Goal:
Learn about Regular Expressions, a powerful method for searching, manipulating, and identifying patterns in text, whether in strings, files, or larger text blocks. You will also learn how to employ regular expressions within Python for sophisticated text searches and explore various applications of regular expressions.

Learning Objectives:
- Defining regular expressions
- Understanding the fundamental components of Regular Expressions
- Diving deeper into advanced Regular Expressions
- Discussing RegEx functions and practical applications

Module Outline:
1. **Introduction to Regular Expressions**
   a. Regular Expressions in AWS Services
   b. Getting started with Regular Expressions

   c. Knowledge Check
2. **Fundamental Components of Regular Expressions**
   a. Understanding Basic RegEx Syntax
   b. Anchors and Word Boundaries
   c. Knowledge Check
3. **Diving Deeper: Advanced Regular Expressions**
   a. Quantifiers
   b. Character Sets and Ranges
   c. Groupings, Backreferences, and Alternation
   d. Knowledge Check
4. **RegEx Functions and Practical Applications**
   a. RegEx Functions In-Depth
   b. Practical Applications of Regular Expressions
   c. Knowledge Check

Activities:

- **Use case: Identifying error responses in web server logs**
  - Monitoring and analyzing web server logs to identify error responses, specifically client errors.
- **Use case: Validating email address formats with anchors in RegEx**
  - Explore how anchors are used in RegEx to validate email address formats.
- **Use case: Matching words using word boundaries**
  - Practice with word boundaries by exploring how they influence the matching of the word 'the' in various contexts.
- **Use case: The ? quantifier when used with re.findall()**
  - Explore how the question mark (?) quantifier behaves when used with the re.findall() function.
- **Challenge: Special characters in sets**
  - Experiment with several special characters to see how they behave when placed in sets.
- **Use case: Searching for dog breeds using re.compile()**
  - Search for different dog breeds using re.compile().
- **Use case: Using RegEx to parse a URL to extract components**
  - Using RegEx, parse a URL to extract components like domain names, port numbers, or paths.
- **Challenge: Validating user input with RegEx**
  - Validate user email addresses using the RegEx module in Python to streamline this process.
- **Lab: Using Regular Expressions in Real Life (120 mins)**
  - This lab reinforces your regular expression (RegEx) knowledge by using real-world examples that simulate practical scenarios that a developer might encounter. For these tasks, you act as a developer who extracts useful information from server logs. This might be information such as the most visited websites or the time it takes to send emails. You also anonymize a sensitive data file that contains personnel names and email addresses.

Videos (Total Time 14:45):

- What are Regular Expressions (2:24)
- Regular Expressions in AWS Services (2:04)
- Non-Greedy or Lazy quantifiers (2:45)

- Flags (4:00)
- Best Practices for writing Regular Expressions (3:30)

Assessments:

For each module, you will complete a series of ungraded assessments to help measure your progress. At the end of each week, you will complete a graded final assessment that tests your knowledge of the learning objectives presented. A passing grade of 85% is required to move forward with your learning, and you can take final assessments as many times as needed.

Week 11: Python 2 – Web Requests and REST APIs

Goal:

Explore using Python to make a web request and process the resulting response. You will also learn about performing operations on a web resource using a RESTful API.

Learning Objectives:

- Understanding the fundamentals of web requests
- Using the Python requests library
- Defining APIs and how RESTful APIs work
- Identifying RESTful API best practices

Module Outline:

1. **Web Requests**
    a. Introduction: Web Requests
    b. HTTP Protocol Overview
    c. Making HTTP Requests Using the Python Requests Library
    d. Knowledge Check
2. **REST APIs**
    a. Introduction: REST APIs
    b. API Overview
    c. Introduction to REST APIs
    d. Making RESTful API Requests
    e. API Design Best Practices
    f. Knowledge Check
3. **Hands-On Lab Activity**
    a. Lab: Forecasting the weather with an API

Activities:

- **Challenge: Sending HTTP requests using curl**
    - Practice sending a request using curl, a powerful command-line tool for transferring data with URLs.
- **Lab: Forecasting the weather with an API (60 mins)**

  - This lab gives hands on experience of using Python to request data from an API. You learn how to use the Python Requests library to query a remote API. Specifically, you use the appropriate library method to retrieve weather information from an API endpoint in a JSON format.

Videos (Total Time 3:19):

- HTTP Request and Response Example (1:01)
- Understanding APIs (1:00)
- Benefits of a REST API (1:01)
- REST Resource Representation (0:17)

Assessments:

For each module, you will complete a series of ungraded assessments to help measure your progress. At the end of each week, you will complete a graded final assessment that tests your knowledge of the learning objectives presented. A passing grade of 85% is required to move forward with your learning, and you can take final assessments as many times as needed.