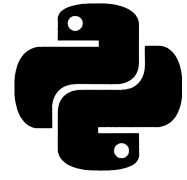
In28 Minutes

Python Bootcamp

Getting Started

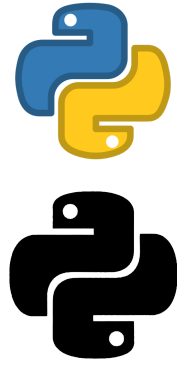
In28 Minutes



- Python is one of the **Top Programming Languages** in the world today
  - Known for its simplicity, readability, and a rich ecosystem of libraries and frameworks
- **Python** is used to build:
  - Web Applications
  - Data Science Applications
  - Machine Learning Applications
  - And a lot more...
- Learning Python can help you become a **versatile** and **in-demand** developer

Getting Started - Challenges

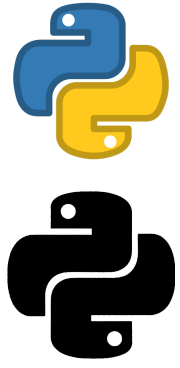
In28 Minutes



- Beginners **find the first steps very difficult**:
  - **Steep Learning Curve**: Programming involves understanding new concepts, syntax, and logic
  - **Difficulty in Installing Software**: Getting Python installed along with installing IDE can be tricky!
  - **Solving Syntax Errors**: Making mistakes in code syntax is common. These errors can be frustrating.
  - **Difficulty in Getting Help**: Getting help when you are stuck might be difficult
  - **Staying Motivated**: Maintaining enthusiasm and motivation can be tough

A Very Simple Path to Learn Python

In28 Minutes

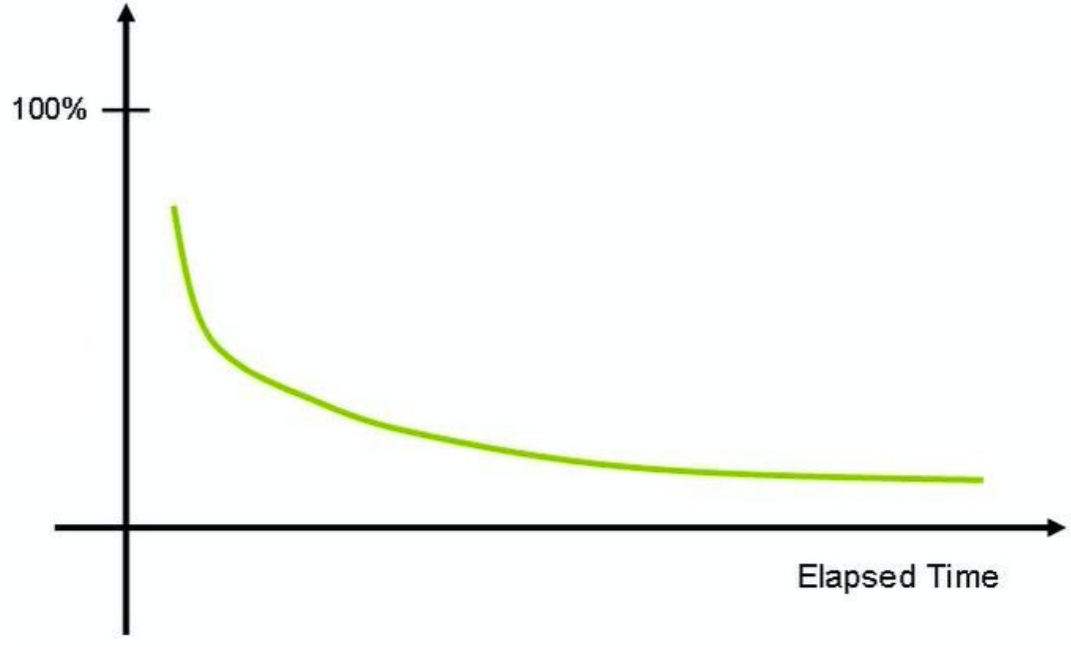


- We've created a **simple path** using a **hands-on approach**
- We've made the entire course **a fun ride**
- You will solve ~200 awesome coding exercises!
- We will make use of **Udemy's coding exercise** platform!
  - You do NOT need to install Python
  - You do NOT need to install IDE
- You will have excellent support in **Q&A**
  - Post all your questions in the course Q&A
- **Our Goal** : Make it really easy for you to start your Python journey!

How do you put your best foot forward?

In28 Minutes

- **Learning Python can be tricky**:
  - Lots of new terminology, concepts and syntax
- As time passes, we forget things
- How do you **improve your chances** of remembering things?
  - **Active learning** - think & make notes
  - **Solve Exercises** on your own
  - **Review the presentation** once in a while



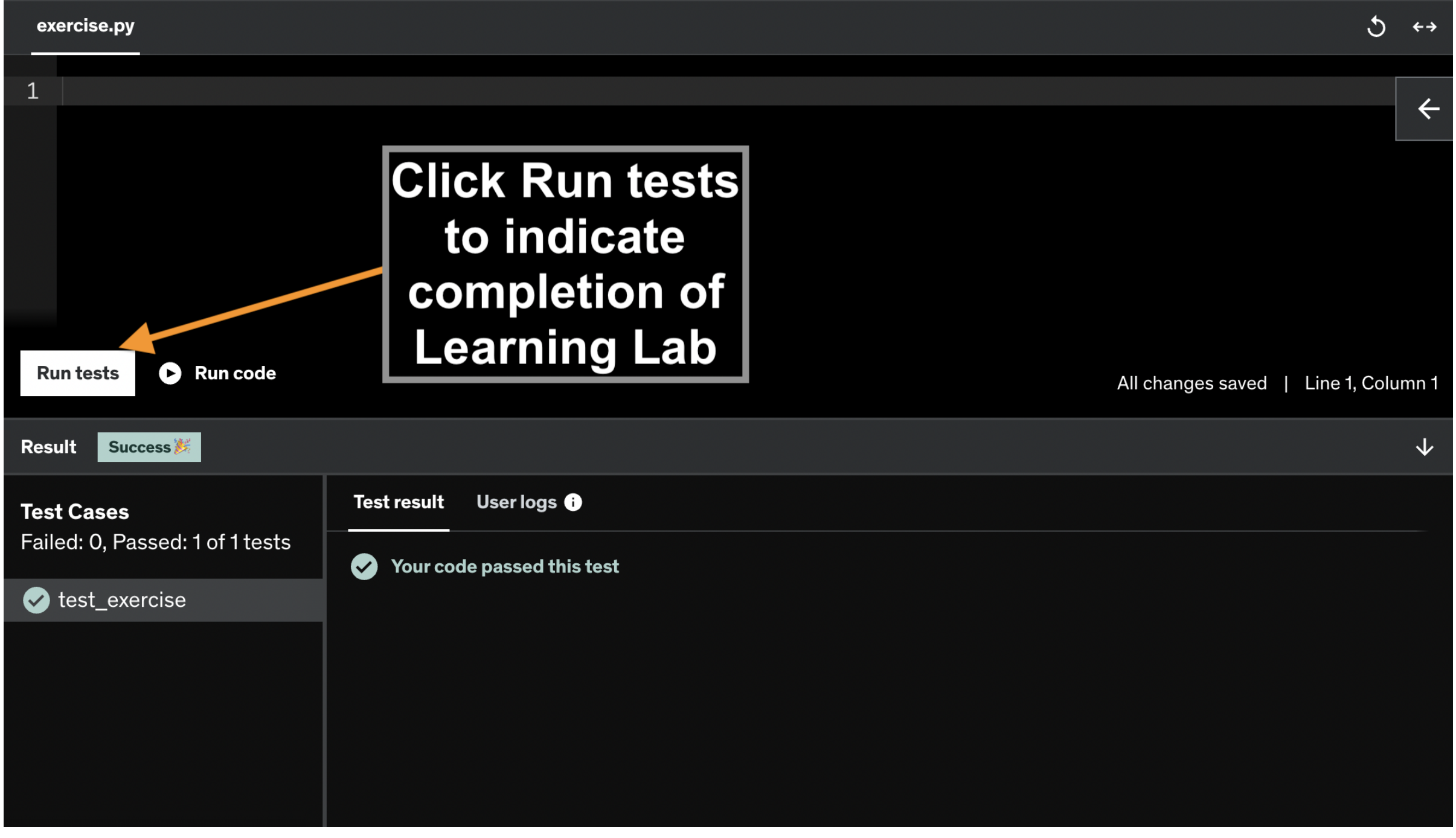
Our Approach

In28 Minutes

- **Four Pronged Approach:**
  - **Learning Lab Videos**: Learn Concepts
  - **~200 Coding Exercises**: Solve Problems
  - **Coding Exercise Solution Videos**: Watch me solve problems
  - **Quizzes**: To Reinforce Concepts
- (Recommended) Take your time. Do not hesitate to replay videos!
- (Recommended) Have Fun!

Learning Lab - Completion

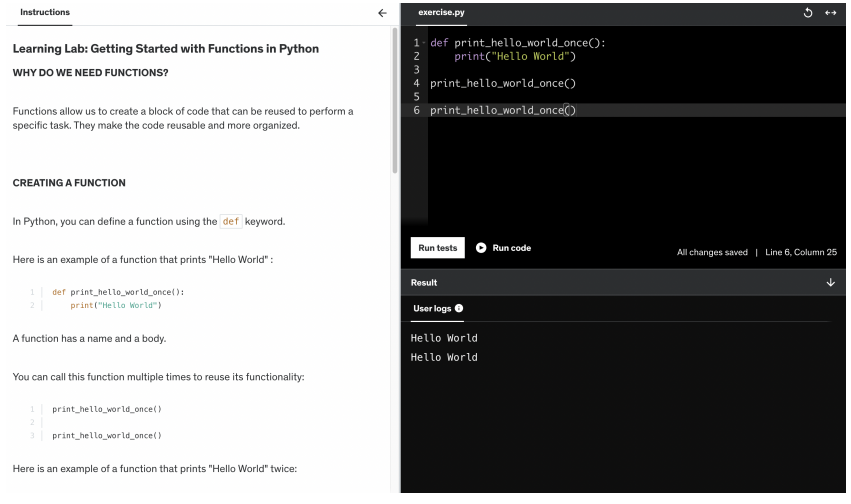
In28 Minutes


exercise.py
1
Click Run tests to indicate completion of Learning Lab
Run tests
Run code
All changes saved | Line 1, Column 1
Result
Success
Test Cases
Failed: 0, Passed: 1 of 1 tests
test_exercise
Test result
User logs
Your code passed this test

Getting Started with Learning Labs

In 28 Minutes

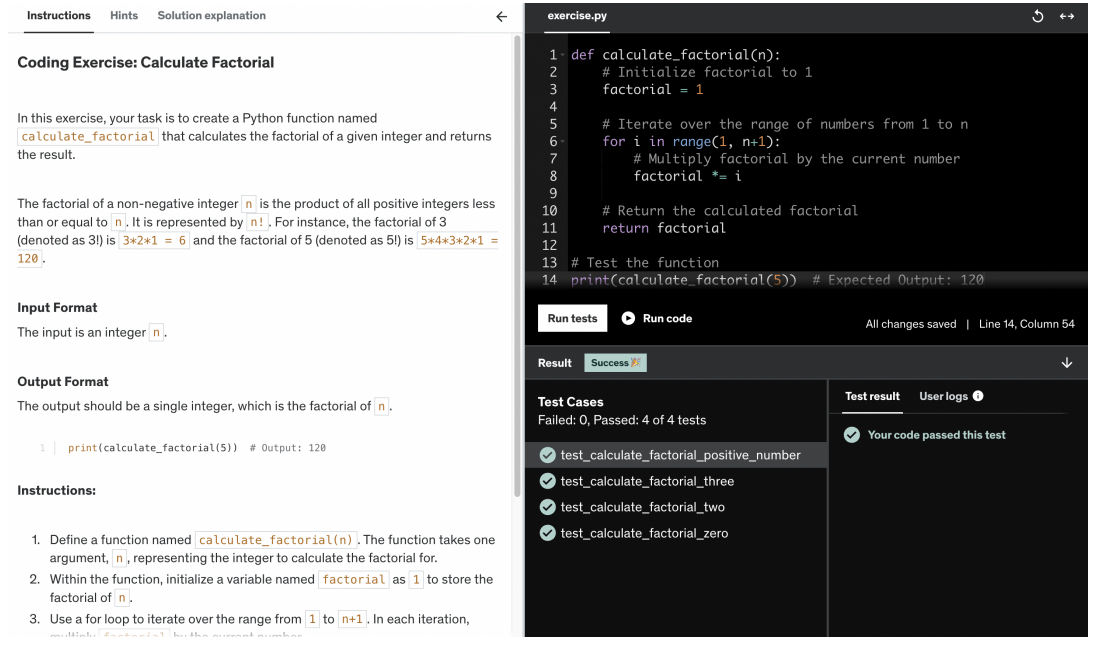
- **HURRAH**! You can do the entire course directly on Udemy
  - Without needing an IDE!
- Let's get started with **Learning Labs**
  - Each Learning Lab has:
    - **Instructions**:
      - Learn all concepts with a lot of code examples
    - **Practice Window**:
      - Write Code
      - See Output (by clicking ***"Run Code"***)
  - Advantages:
    - Read Notes and Write Code in side by side windows
    - You do NOT need an IDE
- **Tip**: Copy notes for revision
- **Tip**: Don't worry about **Run tests** for now



Getting Started with Coding Exercises

In 28 Minutes

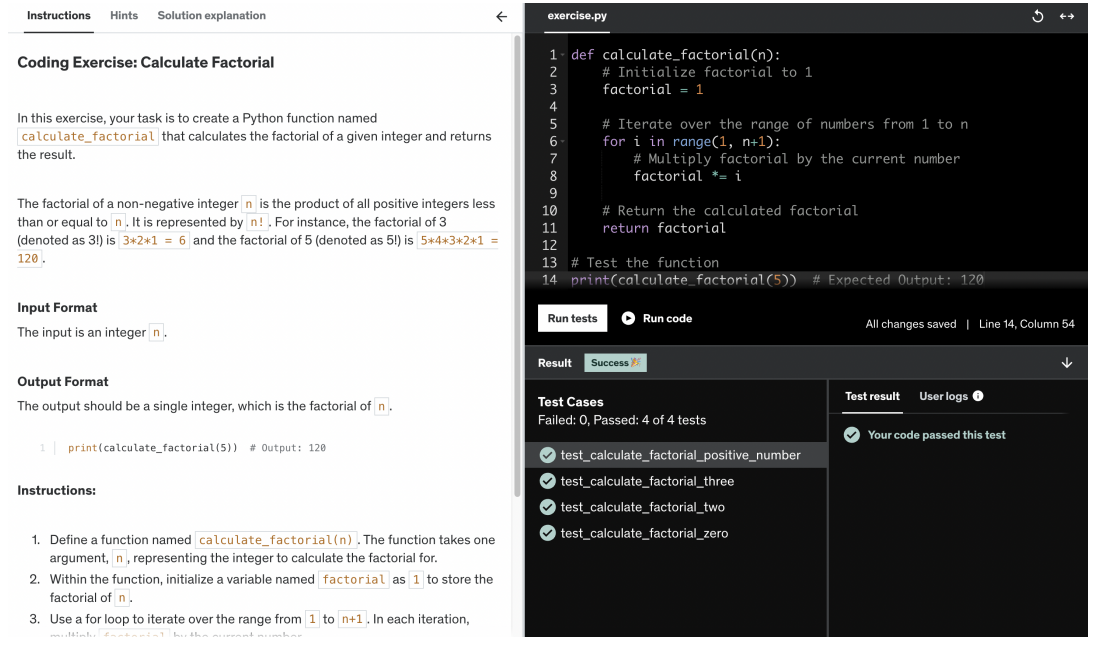
- How to become **a great programmer**?
  - **PRACTICE, PRACTICE and PRACTICE**
  - We have lots of **coding exercises** for you!
- Each **coding exercise** has:
  - Instructions (problem statement) & Hints
  - Solution Explanation
  - Solution video (watch me solve it!)
  - **Advantages**:
    - You get **a lot of practice**
    - Your solution is **automatically checked**
    - **Additional skills** you'll improve: Reading and Documentation
- **NEXT**: Test Exercise to help you get familiar with interface



Our Recommendations

In 28 Minutes

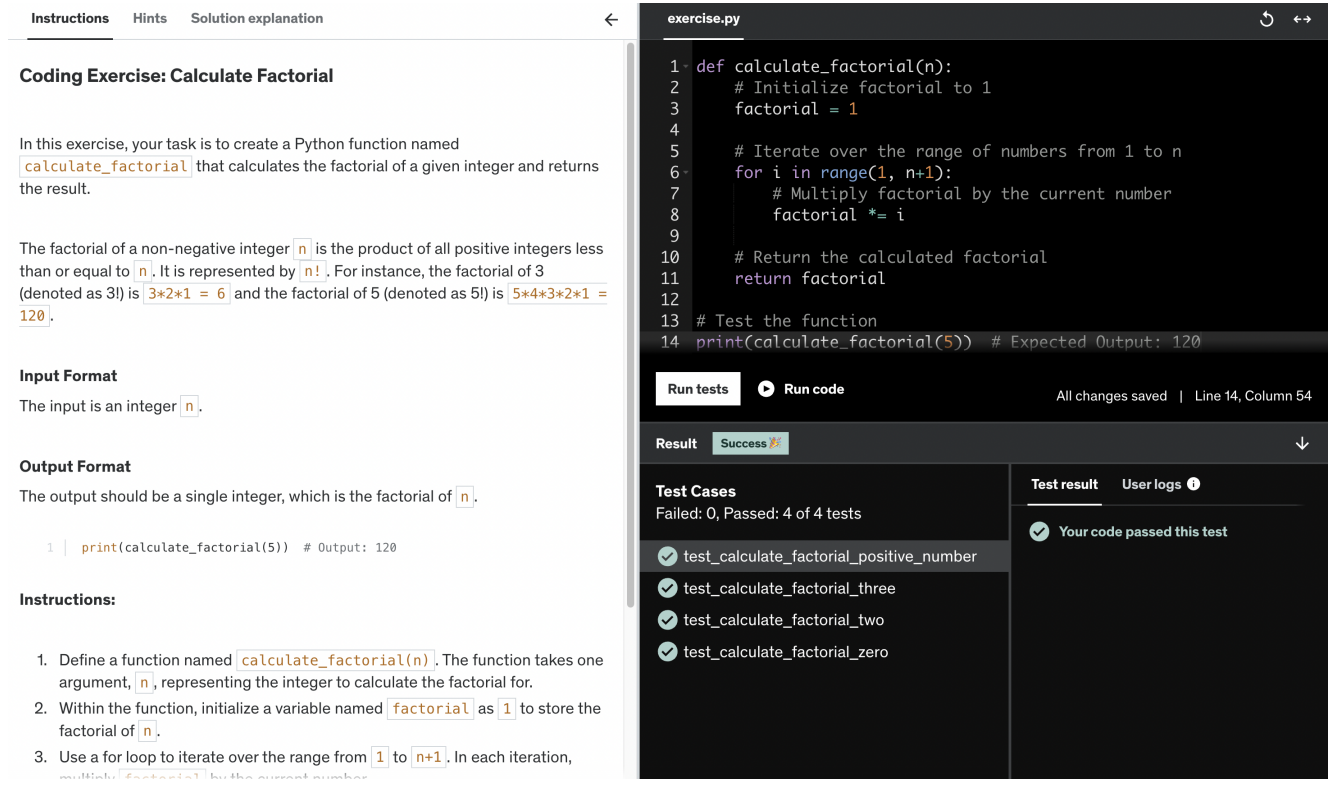
- **Take your time** to do the exercises:
  - Do NOT skip them!
  - Try to solve them **on your own** first
  - If you need help, use the **hints**
  - If you still need help, look at the **solution**
    - BUT type it in on your own
    - Do not copy and paste the solution!
- **Making the most** of these exercises:
  - Have a little bit of patience!
  - If you have questions or success stories, post them in the Q&A forum
- **Remember**: Next lecture is NOT a real exercise
  - Helps you get familiar with the interface



Udemy Coding Interface: Run code vs Run tests

In 28 Minutes

- **Run code**: Run Python Code As Is!
  - **Learning Labs**: Run code, see output and learn concepts
  - **Coding Exercises**: See the output of your code before running a test
- **Run tests**: Run tests that we wrote for the exercise!
  - **Learning Labs**: Mark completion of learning lab
  - **Coding Exercises**: See if you've completed the exercise correctly by running all the tests!



Getting Started with Python Programming - Objectives

In28 Minutes

- Run **Your First Python** Program!
- Learn **Printing** in Python:
  - Using the `print()` function for output
  - Understanding the importance of quotes for text data - `'Hello World'`
- Learn **Basic Math** Operations:
  - Multiplication (*), addition (+), subtraction (-), division (/), exponentiation (**), and remainder (%)
- Learn about **Built-in** Functions:
  - `print()`, `abs()`, `pow()`, `max()`, and `min()`
- Apply these in **practical** exercises

Getting Started with Python Programming - Summary

In 28 Minutes

Concept	Description
Built-in Functions	Explored essential built-in functions: `print()`, `abs()`, `pow()`, `max()`, and `min()`
Basic Math Operations	multiplication (*), addition (+), subtraction (-), division (/), exponentiation(**), and remainder(%)
Text	Should be within double or single quotes - `'Hello World'` or `"Hello World"`
Expression - `5 * 4 * 50`	* is an operator, and 5, 4 and 50 are operands. 5, 4 and 50 are also called literals because these are constant values. Their values don't really change.
Statement	A unit of code that the Python interpreter can execute. For example, `print(5)`.
Calling a Function	Invoking or initiating a function using its name and parentheses. For example, `print(5)`.
Argument	A value provided to a function when called. Ex: `print(5)`, `5` is an argument.

Introduction to Variables & Assignment - Objectives

In28 Minutes

- Learn the fundamentals of **variables**
- Understand the **importance of variable names**
  - Grasp the concept of **case sensitivity** variable names.
- Explore **basic assignments** and how to use expressions for assignment
- Practice **updating variable values** based on their current values
- Gain hands-on experience with **coding exercises** related to variables

Introduction to Variables & Assignment - Summary

In28 Minutes

Concept	Description
Variable	Fundamental data storage units in Python that hold values. Value of variables will change over the runtime of a program.
Defining, Declaring or Creating a Variable	Giving a variable a name and an initial value
Assignment	The process of assigning a value to a variable using the = operator (In programming, the = symbol doesn't denote equality as in mathematics but represents assignment. The value of the expression on the right-hand side is assigned to the variable on the left-hand side.)
Choosing Good Names	Choose meaningful and descriptive names for variables based on their purpose
Case Sensitivity	Variable names are sensitive to uppercase and lowercase. 'Count' and 'count' would be two different variables.

Introduction to For Loop - Objectives

In28 Minutes

- Understand the concept of **loops** in programming.
- Learn about the **for loop** and **its syntax** in Python.
- Explore how to use a for loop to iterate over a sequence of elements.
- **Practice writing simple for loops** to perform repetitive tasks.
- Gain hands-on experience with **coding exercises** related to for loop

Introduction to For Loop - Summary

In28 Minutes

Concept	Description
For Loop	A for loop in Python is used to iterate over a sequence of values.
Syntax	`for val in sequence:` `  #Body`
Example	`for i in range(1,10):` `  print(i)`
`range` function	Produces a sequence of integers from start (inclusive) to stop (exclusive) by step. `range(i, j)` produces `i, i+1, i+2, ..., j-1` When step is given, it specifies the increment (or decrement).
Indentation is crucial	If you skip indentation, you'll encounter an IndentationError.
Nested loop	A for loop can be nested inside another for loop, and this is known as a nested loop. The inner loop executes completely for each iteration of the outer loop.

Getting Started with Functions - Objectives

In28 Minutes

- Let's explore the fundamentals of functions in Python
  - Learn about **functions** and the **need for functions**
  - Gain **hands-on experience** with **coding exercises** related to functions
- By the **end of this section**, you will be able to:
  - Define and call functions
  - Understand parameters and arguments
  - Utilize return values

Getting Started with Functions - Summary

In28 Minutes

Concept	Description
Function	A reusable piece of code. Example: `def product_of_two_numbers(a,b):`
Parameter	A named entity in a function definition that specifies an argument the function can accept. Example: In `def product_of_two_numbers(a,b):`, `a` and `b` are parameters.
Argument	The actual value that is passed to a function when it is invoked/called. Example: In the call `product_of_two_numbers(1,2)`, `1` and `2` are arguments.
Function Declaration	The act of defining a function's name, parameters, and body. It doesn't run the function but sets it up to be called later
Function Invocation	The act of running a function by using its name followed by arguments in parentheses. Example: `product_of_two_numbers(2,3)`
Return Value	The value a function sends to calling function. `return a * b`
Built-in Function	Functions that are always available in Python. Example: `max(5,6)`

Introducing Data Types - Objectives

In28 Minutes

- Understand **more data types** in Python
  - Integer
  - Floating Point Numbers
  - Boolean
- Understand what would happen if you combine data types in operations
- Understand how you can **convert one data type to another**
- Gain hands-on experience with **coding exercises** related to Data Types

Introducing Data Types - Summary

In28 Minutes

Concept	Description
Integer	An integer is a whole number, without a fraction. Examples include 1, 2, 6, -1, and -2. `int`
Floating Point Numbers	Floating-point numbers, or floats, represent real numbers and are written with a decimal point dividing the integer and fractional parts. Examples include 2.5 and 2.55.
Boolean Values	In Python, `True` and `False` are the Boolean values.
+= OPERATOR	Shorthand to increment a variable. `i += 1`. Similar operators `-=`, `/=`, `*=`
Integer division	The double slash operator (//) performs integer (or floor) division. `5//2` results in `2`.
Dynamic Typing	In Python, the type of a variable can change during the execution of a program
Remember	Operations can also be performed between int and float. The result of an operation between an int and a float is always a float. Use == for comparison and = for assignment.

Exploring Conditionals in Python - Objectives

In 28 Minutes

- Understand the concept of **conditionals** in programming.
- Learn about **if statements** and how they control program flow.
- Explore **logical operators** (`and`, `or`, `not`) and their application in conditionals.
- Dive into more advanced logical operations like **logical XOR**, **NOT**, and **NOT EQUAL TO**.
- Master the usage of **if, else, and elif** statements for complex decision-making.
- Engage with hands-on exercises and puzzles to solidify your understanding.

Exploring Conditionals in Python - Summary

In28 Minutes

Concept	Description
`if` statement	An if statement is used to check a condition. If the condition is True, the indented block of code following the if statement is executed.
`else`	Provides an alternative code block that will execute if the `if` statement's condition is not met
`elif`	elif stands for "else if". If the `if` condition is false, the program checks the `elif` condition. If the `elif` condition is true, it executes the code block underneath it.
Logical operators	The `and` operator returns True only when both operands are True. The `or` operator returns True if at least one of the operands is True. The not operator returns the negation of the bool value. The `^` (xor) operator returns True when the operands have different boolean values.
!= operator	Check if value is not equal to something. Example: `if x != 6:`
Remember	Forgetting to indent the block of code following the if statement gives `IndentationError`. In Python: any non-zero value is considered True.

Exploring While Loop in Python - Objectives

In28 Minutes

- Understand the basic structure and functionality of a **while loop**.
- Learn how to use a **while loop** to perform iterative tasks.
- Explore the concept of **loop control** with `break` and `continue` statements.
- Apply your knowledge through **hands-on exercises** and engaging **puzzles**.
- Gain confidence in using **while loops** for various scenarios in Python programming.

Exploring While Loop in Python - Summary

In28 Minutes

Concept	Description

`while` loop	In a while loop, we specify a logical condition. While the condition is true, the loop continues running.
	`while i < 5:`
	`  print(i)`
	`  i = i + 1`

`break` statement	The break statement is used to exit a loop when a specific condition is met.
`continue` statement	The continue statement is used to skip the current iteration of a loop and proceed to the next iteration.
Remember	Always remember to increment the variable used in the while loop condition. Forgetting to do this can result in an infinite loop.
`for` vs `while`	Use a for loop when you know the number of iterations in advance. Use a while loop when the number of iterations is determined by a condition that may change during execution.

Let's Play with String - Objectives

In28 Minutes

- Understand Python Type for Text: **str**
- Explore Why Python Has No Separate Character Type
- Learn About the **string** Module in Python
- Master the Art of **Comparing Strings** in Python
- Discover How to **Repeat Strings**
- Engage with Hands-On Exercises and Puzzles

Let's Play with String - Summary

In28 Minutes

Concept	Description
`str` type	Strings in Python are represented with str type. You can use either single quotes or double quotes to define a string.
`str` methods	The str class provides various methods to manipulate and inquire about strings - `capitalize`, `islower`, `isupper`,`isdigit`,`isalpha`,`endswith`,`startswith`,`find`
`in` keyword	You can use the in keyword to check whether a character or sequence of characters exists within a specific set. `print('Hello' in 'Hello World')`
`string` module	The string module in Python provides a collection of utilities that can be used for common string operations. To use this module, you'll need to import it first. Examples - `string.ascii_letters`, `string.ascii_lowercase`,`string.digits`
Remember	In Python, there is no distinct data type for single characters. Both strings and single characters are represented by the `str` class.

Introduction to Object Oriented Programming - Objectives

88 Minutes

- Understand the basics of Object-Oriented Programming (OOP) in Python.
- Learn about **Classes**, **Objects**, and **Constructors**.
- Gain **hands-on experience** with `MotorBike` and `Book` classes.
- Explore **instance methods** and **encapsulation**.
- Comprehend the role of `self` in Python classes.
- Realize that **everything in Python is an object**.

Object Oriented Programming (OOP)

In28 Minutes

```
class Planet
    name, location, distanceFromSun // data / state / fields
    rotate(), revolve() // actions / behavior / methods

earth = Planet()

venus = Planet()
```

- A **class** is a template.
  - In above example, `Planet` is a class
- An **object** is an instance of a class.
  - `earth` and `venus` are objects.
  - `name`, `location` and `distanceFromSun` compose object state.
  - `rotate()` and `revolve()` define object's behavior.
- **Fields** are the elements that make up the object state. Object behavior is implemented through **Methods**.

Object Oriented Programming (OOP) - 2

In28 Minutes

```
class Planet
    name, location, distanceFromSun // data / state / fields
    rotate(), revolve() // actions / behavior / methods

earth = Planet()

venus = Planet()
```

- Each Planet has **its own state**:
  - `name`: "Earth", "Venus"
  - `location` : Each has its own orbit
  - `distanceFromSun` : They are at unique, different distances from the sun
- Each Planet has its **own unique behavior**:
  - `rotate()` : They rotate at different rates (and in fact, different directions!)
  - `revolve()` : They revolve round the sun in different orbits, at different speeds

Introduction to Object Oriented Programming - Summary

28 Minutes

Concept	Description
Structured vs OOP	In structured programming, code is organized around functions. In Object Oriented Programming, code is organized around classes and objects.
Class	A blueprint for creating objects. Use CamelCase to name classes (e.g., `MotorBike`).
Object	An instance of a class
Method	A function defined within a class
Attribute	Variables belonging to an object
State	Values assigned to attributes of an object

Constructor	Used to create an object. A constructor is defined using the `__init__` method.
	`def __init__(self, speed):`
	`  self.speed = speed`

Encapsulation	Bundling of data (attributes) and the methods

that operate on data into a single unit

Getting Started with Data Structures - List - Objectives

In28 Minutes

- Understand the importance of **data structures** in programming.
- Learn about the `list` data structure and its operations.
- Apply list operations in **practical exercises**.
- Solve **puzzles** involving lists of strings.
- Explore **sorting, looping, and reversing** techniques for lists.

Getting Started with Data Structures - List - Summary

In 28 Minutes

Concept	Description
`list`	Versatile data structure that allows you to store and manipulate a collection of items, which can be of different types and can be accessed by their position or index within the list.
Edit a 'list'	Append elements to the end of a list using `append()` Insert elements at a specific position using `insert()` Remove elements from a list using `remove()`
Example operations	`sum()`: Computes the sum of elements `max()`: Finds the maximum value `min()`: Determines the minimum value `len()`: Calculates the length (number of elements)
Accessing Elements	Access elements by referring to index number inside square brackets. `print(animals[2])`
Sorting and Reversing	Reverse - `reverse()` method modifies original list and `reversed()` yields an iterator. Soring - `sort()` modifies the original list while sorted() function returns an iterator

Data Structures: 2D Lists in Python - Objectives

In28 Minutes

- Understand the concept of a **2D list** and its applications.
- Learn to **implement a 2D list** in Python for structured data storage.
- Practice **searching for elements** within a 2D list and retrieving their indices.
- Explore the process of **adding two matrices** of the same size using 2D lists.
- Engage in **hands-on exercises and puzzles** to understand 2D lists.

Data Structures: 2D Lists in Python - Summary

In 28 Minutes

Concept	Description
2D list	A list of list ex: A matrix
Visualizing 2D list	[[00, 01, 02, 03], [10, 11, 12, 13], [20, 21, 22, 23]]
Accessing an element	`two_d_list[1][2]`
Setting value	`two_d_list[2][3] = 4`
Looping around 2D list	`for i in range(rows):` `  for j in range(cols):`

Playing With a List Of Strings in Python - Objectives

In28 Minutes

- Learn to **store multiple pieces of text values** in a List
- Solve **Hands-on Exercises and Puzzles** with a List Of Strings
  - Rotate a List of Strings 'n' Times
  - Encode List of Strings
  - Perform Alternate Merging of Two Lists
  - Implement an Anagram Checker using Lists
- Understand **ASCII Values** and how to use them in Python

Playing With a List Of Strings in Python - Summary

In28 Minutes

- Playing with a List of Strings is the same as playing with a List of numbers
- **ASCII** is a character encoding standard representing text in computers.
  - Each character corresponds to a **unique number** in the ASCII table.
- The `ord(char)` function returns the **ASCII value** of a character.
- `chr(ascii_val)` returns the **character representation** of an ASCII value.
- **Unicode**, a superset of ASCII, aims to represent text in all languages with a larger character set.

Advanced Object Oriented Programming - Objectives

In28 Minutes

- Revise **Object Oriented Programming** Fundamentals
- Explore **Object Oriented Programming** Advanced Concepts with Examples
  - Explore **Object Composition**
  - Dive into **Inheritance**
  - Understand the **object Class** in Python 3.
  - Explore **Multiple Inheritance** in Python.
  - Grasp the concept of **Abstract Classes** in Python.
  - Learn to use the **Template Method Pattern**
  - Understand and apply **Polymorphism** in Python.

Advanced Object Oriented Programming - Summary

In 28 Minutes

Concept	Description
Object composition	Allows you to combine simple types or classes to create more complex ones. For instance, a Book class can contain multiple Review objects.
Inheritance	Allows one class to inherit properties and behaviors (methods) from another class. The class that is inherited from is known as the “superclass” or “parent class,” and the class that inherits is called the “subclass” or “child class.” `class Pet(Animal):`
`object` class	Starting Python 3, every class implicitly inherits from `object` class unless you override it
Multiple inheritance	Allows a class to inherit from multiple classes `class Amphibian(WaterAnimal, LandAnimal)`
An abstract class	Serves as a blueprint for sub-classes. Cannot be instantiated on its own. Contains abstract methods (declared but not implemented). Derived classes provide implementation.
Polymorphism	"Poly" means "many," and "morph" means "forms." So, polymorphism means "many forms." Same code - Different Results.

Data Structures - Implementing Stack & Queue - Objectives

- Understand the concept of a **Stack**
  - Understand **Operations** on a Stack:
    - Push
    - Pop
    - Top
    - IsEmpty
- Implement **Stack** in Python using a `list`
- Understand the concept of a **Queue**
  - Understand **Operations** on a Queue:
    - Enqueue
    - Dequeue
    - Front
    - IsEmpty
- Implement **Queue** in Python using a `list`

Data Structures - Implementing Stack & Queue - Summary

Concept	Description
Stack	A stack is a LIFO (Last In, First Out) data structure. This means the last element you insert is the first one you take out.
Operations on Stack	**Push (Add to top of stack)** - `self.items.append(item)` **Pop (Remove top of stack)** - `return self.items.pop()` **Top (Inspect top of stack)** - `return self.items[-1]` **IsEmpty (Check if stack is empty)** - `return len(self.items) == 0`
Queue	A queue follows a FIFO (First In, First Out) principle.
Operations on Queue	**Enqueue (Add to rear of queue)** - `self.items.append(item)` **Dequeue (Remove front of queue)** - `return self.items.pop(0)` **Front (Inspect front of queue)** - `return self.items[0]` **IsEmpty (Check if queue is empty)** - `len(self.items) == 0`

Exploring Time Complexity & Recursion - Objectives

In 28 Minutes

- Learn the concept of **time complexity** and its importance in algorithm analysis
- Compare and evaluate the efficiency of different algorithms using **Big O notation**
- Gain a fundamental understanding of **recursion** as a programming technique.
  - Learn to calculate sum of a list using Recursion
- Understand and implement **Linear Search**
- Understand and implement **Binary Search** using:
  - Iterative Approach
  - Recursive Approach

Exploring Time Complexity & Recursion - Summary

In28 Minutes

Concept	Description
Time Complexity	Measures how an algorithm's runtime grows with input size
O(1)	Constant Time Complexity - Accessing the first element of an array.
O(n)	Linear Time Complexity - Finding the maximum element in an array.
O(n^2)	Quadratic Time Complexity - Nested loop iterating over an array.
Recursion	Technique where a function calls itself. Simplifies implementation is some scenarios.
Linear Search	Search for elements in the list one by one
Binary Search	Highly efficient algorithm for finding a target value within a sorted array. Works by repeatedly dividing the search space in half until the target element is found.

Introduction To Exception Handling - Objectives

In28 Minutes

- Learn the significance of **error handling** in Python programming
- Master the `try` and `except` blocks for handling exceptions
- Learn to handle **different types of errors** using multiple `except` blocks
- Understand the role of `finally` and `else` blocks in error handling
- Learn how to raise **custom exceptions** to handle specific scenarios
- Understand the process of creating a **custom exception class**

Introduction To Exception Handling - Summary

In28 Minutes

Concept	Description
`try` block	Encloses the code that might raise an exception.
`except` block	Specifies how to handle specific exceptions that occur within the `try` block.
`finally` block	Contains code that will be executed regardless of whether an exception occurs or not.
`else` block	Executes if no exceptions are raised in the `try` block.
Raising an exception	`raise Exception("Currencies Do Not Match")` - Manually triggers an exception with a custom message.

Creating Custom Exception	`class CurrenciesDoNotMatchError(Exception):`
	`  def __init__(self, message):`
	`  super().__init__(message)`

Getting Started with Sets - Objectives

In28 Minutes

- Understand the concept of a **set** in Python.
- Learn how **sets differ from other data structures** like lists
- Understand **Set operations**:
  - Add and remove elements from a set
  - Perform aggregate operations - `min`, `max`, `sum`, & `len`
  - Perform Union, Intersection, and Difference of Sets
- Solve **hands-on coding exercises** to understand Sets
  - Find Intersection Between Multiples of Two Numbers
  - Identify Unique Colors
  - Merge Multiple Shopping Lists

Getting Started with Sets - Summary

In28 Minutes

Concept	Description
Set	Set cannot have duplicates. Sets do not support access using index. `numbers_set[0]` gives error
Creating a Set	You can directly create a set: `numbers_set = {1, 2, 3, 4}`. You can also create a set from a list of numbers by using `set(numbers)`.
Adding to a Set	You can add a number to a set - `numbers_set.add(3)`
Removing an Element	When you remove an element, it gets deleted from the set. `numbers_set.remove(100)`
`in` operator	You can check if an element is in a set or not using the `in` operator. For example: `print(1 in numbers_set)` will output `True`.
Aggregate Operations	You can perform aggregate operations like `min`, `max`, `sum`, and `len`.
Union, Intersection, and Difference of Sets	In a set, you can perform operations like union, intersection (& operator), and difference (– operator).

Getting Started with Dictionary - Objectives

In 28 Minutes

- Get an introduction To `dict` in Python
- Learn how **dict differs from other data structures** like lists and sets
- Learn about **key-value pairs**, and basic operations.
- **Practice using dictionaries** to count occurrences of characters in a string.
- Utilize dictionaries to count the occurrences of words in a text.
- Explore **dictionary comprehension** to create a mapping of numbers to their squares.

Getting Started with Dictionary - Summary

In28 Minutes

Concept	Description
Dictionary	A dictionary in Python is a collection of key-value pairs.
Creating	Dictionary Creation : `occurances = {'a': 5, 'b': 6, 'c': 8}`.
Accessing Values	You can access and modify values using keys, like `occurances['d'] = 15`.
Dictionary Methods	`keys()`, `values()`, and `items()` methods provide views of the dictionary's keys, values, and key-value pairs respectively.
Deleting	Use the `del` keyword to delete a specific key-value pair.
Dict Comprehension	`squared_numbers = {x: x**2 for x in [1,2,3,4,5]}`
Summary	A dictionary in Python represents key-value pairs, providing methods to access, modify, iterate through, and delete keys and values. It offers flexibility and efficiency in managing collections of data where the index can be anything, not just a number.

Getting Started with Tuples - Objectives

In28 Minutes

- Understand the concept of **tuple** - An immutable sequence type in Python
- Learn **how to create** tuples
- Understand **tuple operations**
- **Compare and contrast tuples with lists** to understand their key differences and use cases
- Solve variety of **hands-on exercises** with tuples

Getting Started with Tuples - Summary

In28 Minutes

Concept	Description
Tuple	Allows you to store a sequence of values. `my_tuple = (1, 2, 3, 'hello')`
Immutable Nature	Tuples are immutable, meaning their elements cannot be changed after creation.
Creating and Returning	Tuples can be defined by separating values with a comma - `my_tuple = (1, 2, 3, 'hello')` Functions can return multiple values as a tuple. `return 'Ranga', 1981, 'India'`
Destructuring a Tuple	Values in a tuple can be assigned to individual variables. This is known as destructuring. `name, year, country = my_tuple #'Ranga', 1981, 'India'`
Tuple Operations	Length of a tuple can be found using the `len` function Elements can be accessed by index `print(my_tuple[0])`
Summary	Tuples provide an immutable way to group data. They can be created, returned, and destructured. Tuple operations include finding length and indexing.

High Level vs Low Level Programming Languages

- **Low-Level**: Assembly language, Machine code
- **High-Level**: Python, Java, JavaScript, Go
- What is the **difference**?
  - Computers understand only **0** and **1** (transistor is on or off)
  - Writing programs in 0 and 1 is really tough
  - **Low Level Languages**: Write programs using syntax very near to 0s and 1s
    - **Example**: `MOV AX, 5, ADD AX, 3`
    - Very difficult to write
    - Very difficult to maintain
  - **High Level Languages**: Write programs using human readable syntax
    - Example: `x = 5, y = 3, z = x + y`
    - Easier to write
    - Easier to maintain
    - Portable (Write programs in Windows and run them in Mac and Linux)

Make the Best Use of Coding Exercises

In 28 Minutes

- **We have to:**
  - Design a Problem
  - Create Instructions
  - Write Solution Explanation
  - Write Tests
  - Create Solution Video
- This takes a lot of time
  - BUT we invested this time to help YOU
- **One Request**: Make the best use of exercises
  - Designed to reinforce learning, strengthen problem-solving skills, and prepare YOU for real-world applications
  - **PRACTICE, PRACTICE and PRACTICE**

Instructions | Related lectures | Hints | Solution explanation

Provide instructions so learners know what they're solving. Use accurate, grammatically correct language and avoid biases.

B *I* | ≡ | ≡ | 🔗 | 🖼 | <>

In this exercise, your task is to create a Python function named `sum_of_squares` that calculates the sum of squares of the first `n` even numbers and returns the result.

Input Format

The input is an integer `n`.

Output Format

The output should be a single integer, which is the sum of the squares of the first `n` even numbers.

```
print(sum_of_squares(5))  # Output: 220
```

```
import sys
from unittest import TestCase
from exercise import sum_of_squares
import importlib

class Evaluate(TestCase):

    def test_sum_of_squares_1(self):
        self.assertEqual(sum_of_squares(1), 4, "Sum of squares of
            first 1 even numbers should be 4")

    def test_sum_of_squares_2(self):
        self.assertEqual(sum_of_squares(2), 20, "Sum of squares of
            first 2 even numbers should be 20")

    def test_sum_of_squares_3(self):
        self.assertEqual(sum_of_squares(3), 56, "Sum of squares of
            first 3 even numbers should be 56")

    def test_sum_of_squares_4(self):
        self.assertEqual(sum_of_squares(4), 120, "Sum of squares of
            first 4 even numbers should be 120")
```

My 10 Rules for Happy Programmers

In28 Minutes

- **Embrace the challenge**: Each problem is an opportunity to learn
- **It's okay to fail**: Failure is a part of the learning process
- **Practice makes perfect**: The more you code, the better you'll get
- **Be patient**: Learning to code takes time and effort
- **Have fun**: Coding can be a lot of fun, enjoy the process
- **Don't give up.**: If you're struggling, keep at it
- **Break it down**: Break a complex problem into smaller parts
- **Be persistent.**: Don't give up on a problem just because it's difficult
- **Celebrate progress**: Acknowledge your achievements, no matter how small
- **Stay curious**: Keep exploring new technologies, programming languages, and concepts

You are all set!

Let's clap for you!

In28 Minutes

- You have a lot of patience! **Congratulations**
- You have put your best foot forward **to be a great developer!**
- Don't stop your learning journey!
  - Keep Learning Every Day!
- Good Luck!

Python is an Ocean!

In28 Minutes

- Python is an Ocean
- **Our Goal**: Help you start learning Python with a hands-on approach!
  - AND help you develop a love for programming
- I'm sure we are **successful** in that endeavor!

Do Not Forget!

In28 Minutes

- **Recommend** the course to your friends!
  - Do not forget to **review**!
- **Your Success = My Success**
  - Share your success story with me on LinkedIn (Ranga Karanam)
  - Share your success story and lessons learnt in Q&A with other learners!