Python Refresher for AI

Introduction to Python: Fundamentals and Setup

Engage and Think

You have joined a startup tech company as a junior developer. Your team has been using multiple tools and programming languages for AI development, but your manager is considering transitioning to a more efficient and scalable option. The goal is to streamline data handling, automate repetitive AI-related tasks, and enhance overall productivity. As you explore different options, you come across Python, a widely used language in AI, data science, and automation.

Why do you think Python is commonly chosen for AI and machine learning tasks in tech companies? What challenges do you think Python can help you solve when working on AI projects?

Learning Objectives

By the end of this lesson, you will be able to:

- Compare and classify different types of programming languages, program flow, and key programming principles to build a strong foundation for programming
- Demonstrate how to install and configure the Python programming environment in an IDE to write and execute code effectively
- Explain Python’s benefits and its role in AI and data science to recognize its impact on modern technology
- Demonstrate the correct use of Python identifiers, indentation, and comments to write clean, structured, and readable code
- Construct and manipulate different Python data types to effectively store and manage data and apply various operators to perform computations and make logical decisions in Python programs

Programming Foundations

What Is Programming?

It is the process of writing instructions for a computer to perform specific tasks. It involves designing, coding, and testing to solve real-world problems.

Uses

- Develops applications, websites, and AI systems
- Automates repetitive tasks

Examples

- Web applications (Amazon, Netflix)
- AI-powered chatbots (ChatGPT, Siri)

Categories of Programming Languages

They are broadly classified into interpreted and compiled languages based on their method of execution.

Interpreted languages

- Code is executed line by line at runtime.
- They run slower because it’s translated while running.
- Errors are found while running, making debugging easier.
- Examples: Python, JavaScript, Ruby

Compiled languages

- Code is converted into machine language before running.
- They run faster because it’s already translated.
- Errors are found after compilation, making debugging harder.
- Examples: C, C++, Java

Types of Programming Languages

Procedural languages

Based on step-by-step instructions
Examples: C, Pascal

Object-oriented languages

Based on classes and objects
Examples: Java, Python

Functional languages

Based on mathematical functions
Examples: Haskell, Scala

Scripting languages

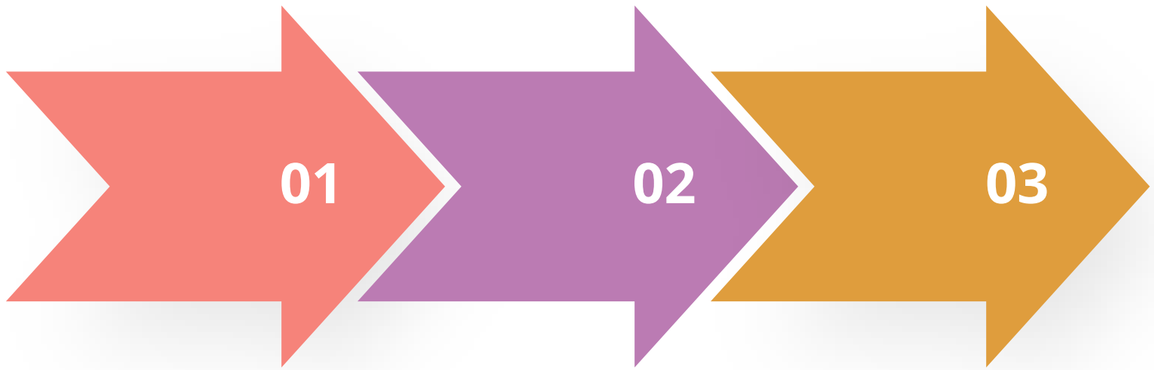
Used for automation and web development
Example: JavaScript, Python

Understanding Program Flow

It describes how a computer processes instructions in a logical sequence to produce the desired outcome.
The basic flow of a program is as follows:

Processing

The program validates and processes user input.
Example: Checking password validity



Input

A user enters data.
Example: Typing in a login form

Output

The program generates results.
Example: Receiving a successful login message

Key Programming Principles

The following programming principles ensure that code is efficient, readable, and maintainable:

Avoid repetition
Using functions and loops to avoid redundant code

Keep it simple
Write clean, simple, and understandable code

Ensure modularity
Divide a program into functions for better organization

Improve readability
Use meaningful variable names, comments, and proper indentation

Focus on debugging
Anticipate and handle errors in your code

Quick Check

How do interpreted languages handle execution?

A. They execute the entire code at once.

B. They convert code into machine language before execution.

C. They execute code line by line at runtime.

D. They do not require an interpreter to run.

Introduction to Python

What Is Python?

It is a high-level, interpreter-based, object-oriented programming language with dynamic semantics used for various applications, such as data science and automation.

Python's simple and easy-to-learn syntax emphasizes readability and reduces the cost of program maintenance. It also supports modules and packages, encouraging program modularity and code reuse.

Python: History

It was conceived in the late 1980s.

It was invented by Guido van Rossum (Centrum Wiskunde and Informatica, Amsterdam).

It is named after the BBC comedy series **Monty Python's Flying Circus**.

It is now maintained by the Python Software Foundation (PSF).

It is derived from the ABC, Modula-3, Lisp, and C languages.

Benefits of Python

Flexible

It aids in the cross-platform compatibility and scripting of web pages and applications.

Easy to learn and use

It uses a minimal amount of code to complete tasks.

Readability

It places a strong emphasis on readable code and permits the use of English keywords in place of punctuation.

Robust standard library

It allows selecting a module from a large selection based on the requirement.

Why Python for AI?

Python has become the dominant language for artificial intelligence (AI) and machine learning (ML) due to its:

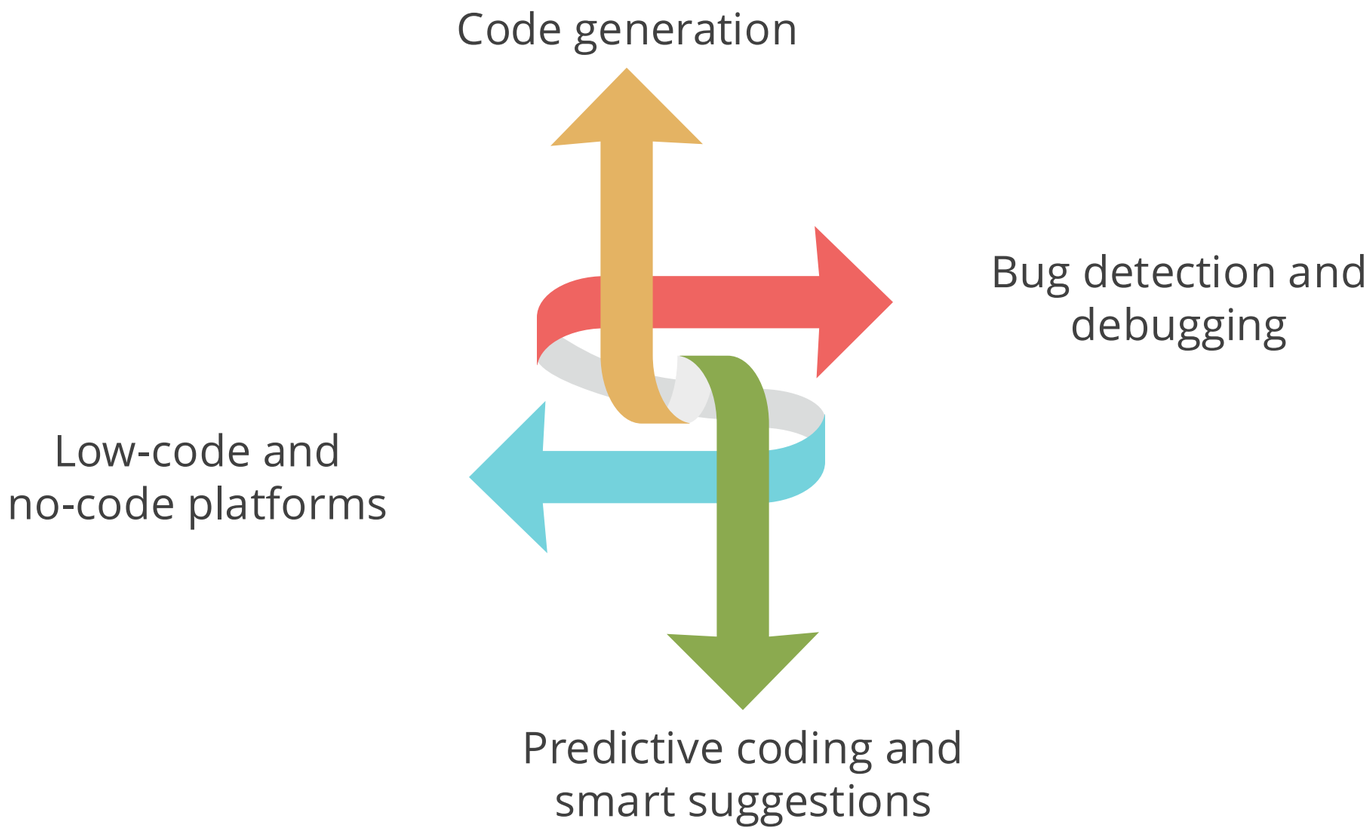
- Powerful AI libraries and frameworks
- Seamless data handling and visualization
- High-level syntax and rapid prototyping
- Strong industry adoption and community support
- Integration with other technologies

Example:

Companies like Netflix and Spotify use Python-based AI models to recommend personalized content, enhancing user experience through predictive analytics.

How AI Is Changing Coding

The rise of AI in software development has transformed how developers write, debug, and optimize code. The following are key areas where AI is reshaping the coding landscape:



Quick Check

What role does Python play in AI-driven recommendation systems like Netflix and Spotify?

A. It automates video playback

B. It helps train machine learning models for personalized content recommendations

C. It is used for user authentication

D. It improves internet speed for better streaming

Exploring Python Development Environments

Python IDE

An integrated development environment (IDE) is a software suite that consolidates the basic tools required to write and test software. A Python program can run on the following IDEs:

Eric

Wing

Atom

PyDev

PyCharm

Jupyter Notebook

Rodeo

Thonny

Spyder

Microsoft VS Code

Recommended Python Development Environments

Some commonly used and recommended Python workspaces are:

Visual Studio Code

Jupyter Notebook

Google Colab

Visual Studio Code: Features

Python extension

Multi-language support

Built-in terminal

AI-powered assistance

Live collaboration

Demo: Installing and Setting Up VS Code

Duration: 10 minutes

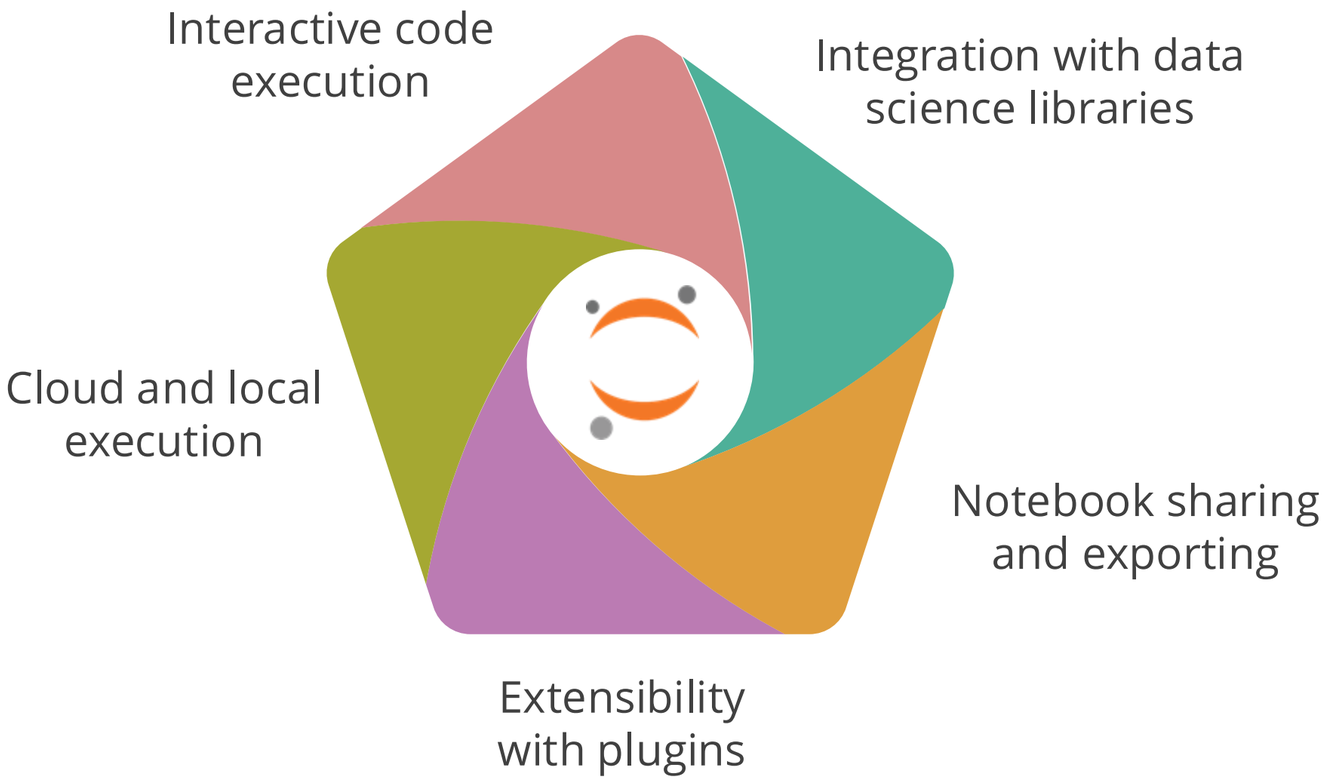
Overview: This demonstration covers installing and setting up VS Code for Python development. You will learn how to download the installer, configure essential settings, and launch VS Code without any errors.

Note:

Please download the demo document from the Reference Material section for step-by-step guidance.

INSTRUCTOR-LED ACTIVITY

Jupyter Notebook: Features



Demo: Installing Jupyter Notebook Using Anaconda

Duration: 10 minutes

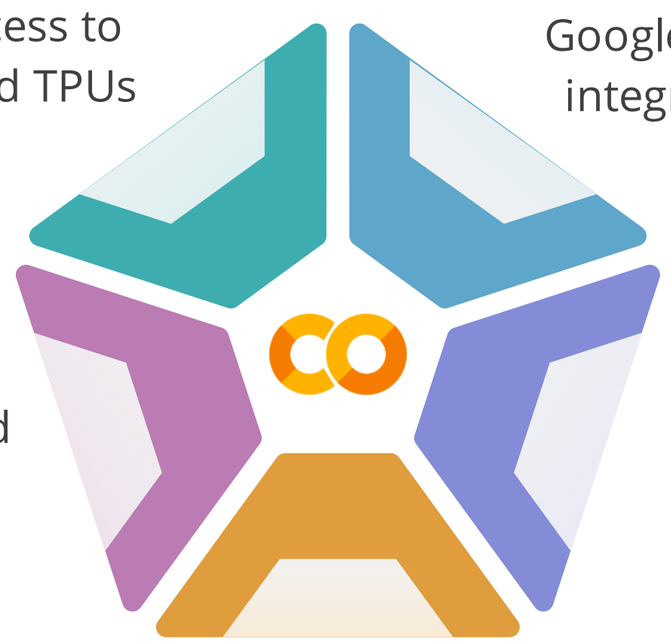
Overview: This demonstration covers installing Jupyter Notebook using the Anaconda distribution for a seamless Python development environment. You will learn how to install Anaconda to launch Jupyter Notebook.

Note:

Please download the demo document from the Reference Material section for step-by-step guidance.

INSTRUCTOR-LED ACTIVITY

Google Colab: Features



Demo: Setting Up a Google Colab Notebook

Duration: 10 minutes

Overview: This demonstration covers setting up a Google Colab notebook for writing and executing Python code in a cloud-based environment. You will learn how to create a new notebook in Google Colab and start coding without requiring any local installation.

Note:

Please download the demo document from the Reference Material section for step-by-step guidance.

INSTRUCTOR-LED ACTIVITY

Quick Check

Which of the following IDEs is specifically designed for scientific computing and includes an interactive environment for data analysis?

A. PyCharm

B. Jupyter Notebook

C. VS Code

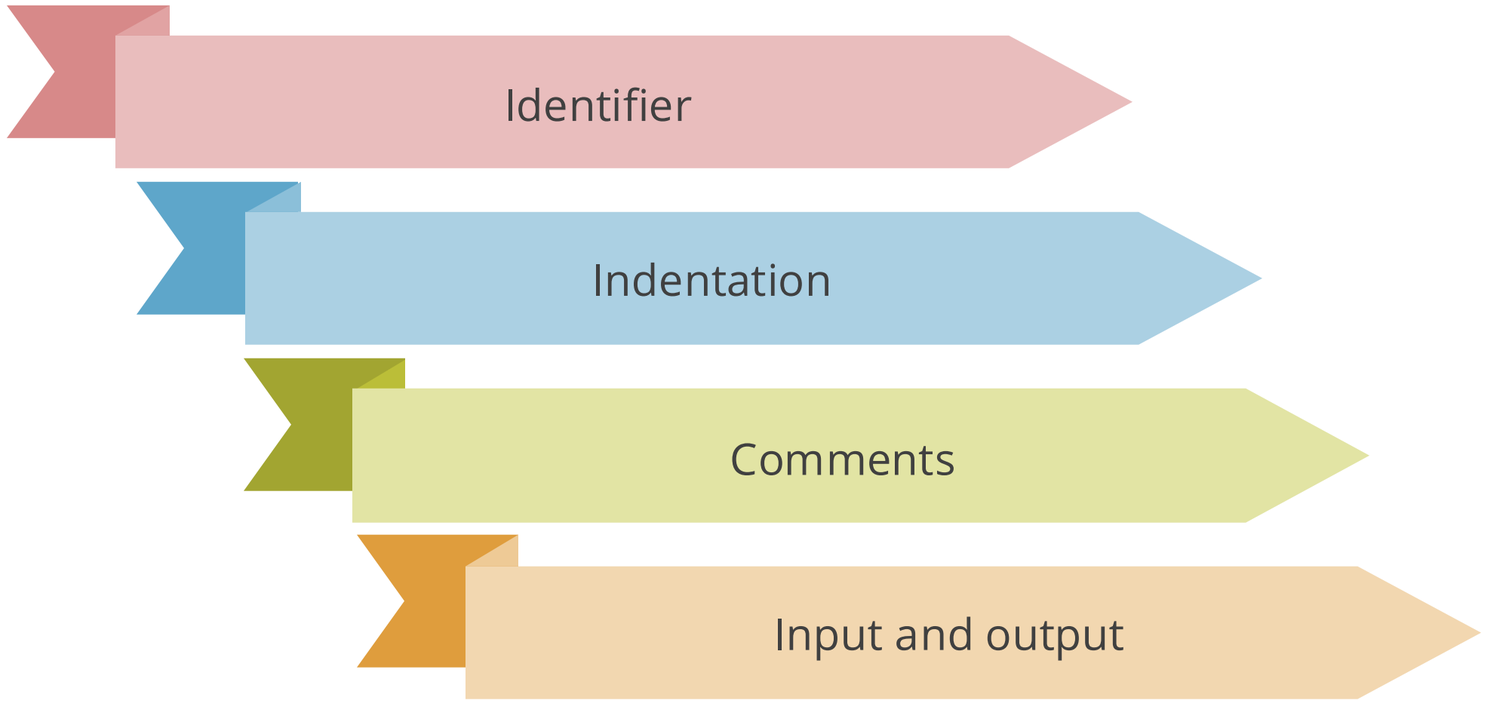
D. Atom

Python Syntax: Structure and Execution

Python Syntax

It refers to a set of rules that define how code should be written and structured so that the Python interpreter can understand and execute it correctly.

Some basic Python syntaxes are:

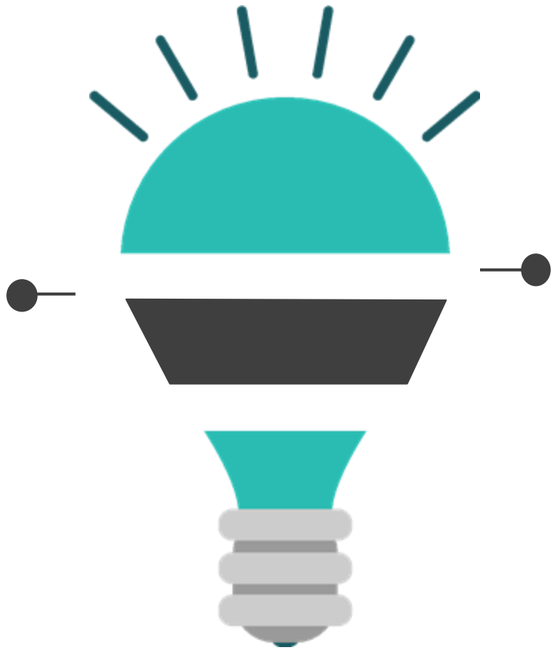


Python: Identifier

It is a name used to identify a variable, function, class, module, or another object.
The following are a few identifier naming rules:

Identifiers can be a combination of:

- Letters in lowercase (a to z) or uppercase (A to Z)
- Digits (0-9)
- Underscore (_)



Special symbols like !, @, #, $, and % cannot be used in an identifier.

An identifier can be of any length, but cannot start with a digit.

Keywords like *global* and *class* cannot be used as identifiers.

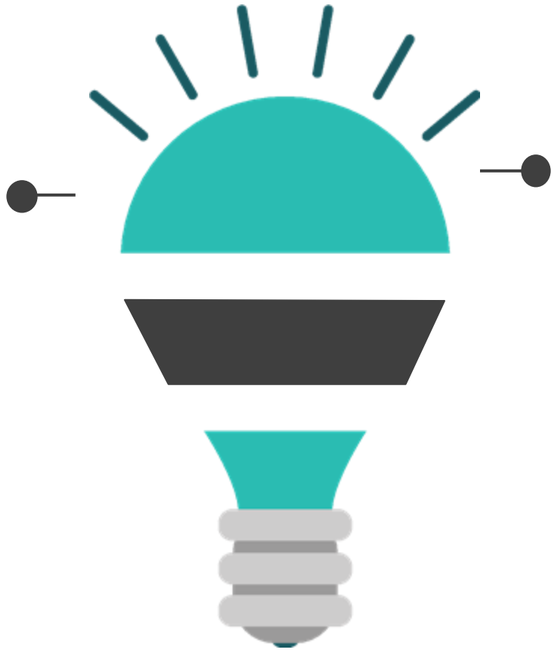
Python is case-sensitive, so *a* is not equal to *A*.

Python: Identifier

Here are a few examples of identifiers:

Valid identifiers:

- myClass,
- var_1,
- count



Invalid identifiers:

- 1variable,
- class@new,
- global

Python: Indentation

It refers to the space at the beginning of a code line.

Correct syntax

```
[1]: if 5 > 2:
         print("5 is greater than 2")

     5 is greater than 2
```

Incorrect syntax

```
[2]: if 5 > 2:
     print("5 is greater than 2")

       Input In [2]
         print("5 is greater than 2")
         ^
     IndentationError: expected an indented block
```

- An indented block of code begins with a colon (**:**).
- Python's indentation is crucial as it makes code easier to understand, unlike other programming languages.
- Python uses indentation to indicate a block of code. For example, indentation is required for conditional statements and loops such as *if … else*, *for loop*, and *while loop*, or else an error will occur.

Python: Comments

They are annotations or programmer-readable explanations in a computer program's source code that make it easier for humans to understand the code.

Example:

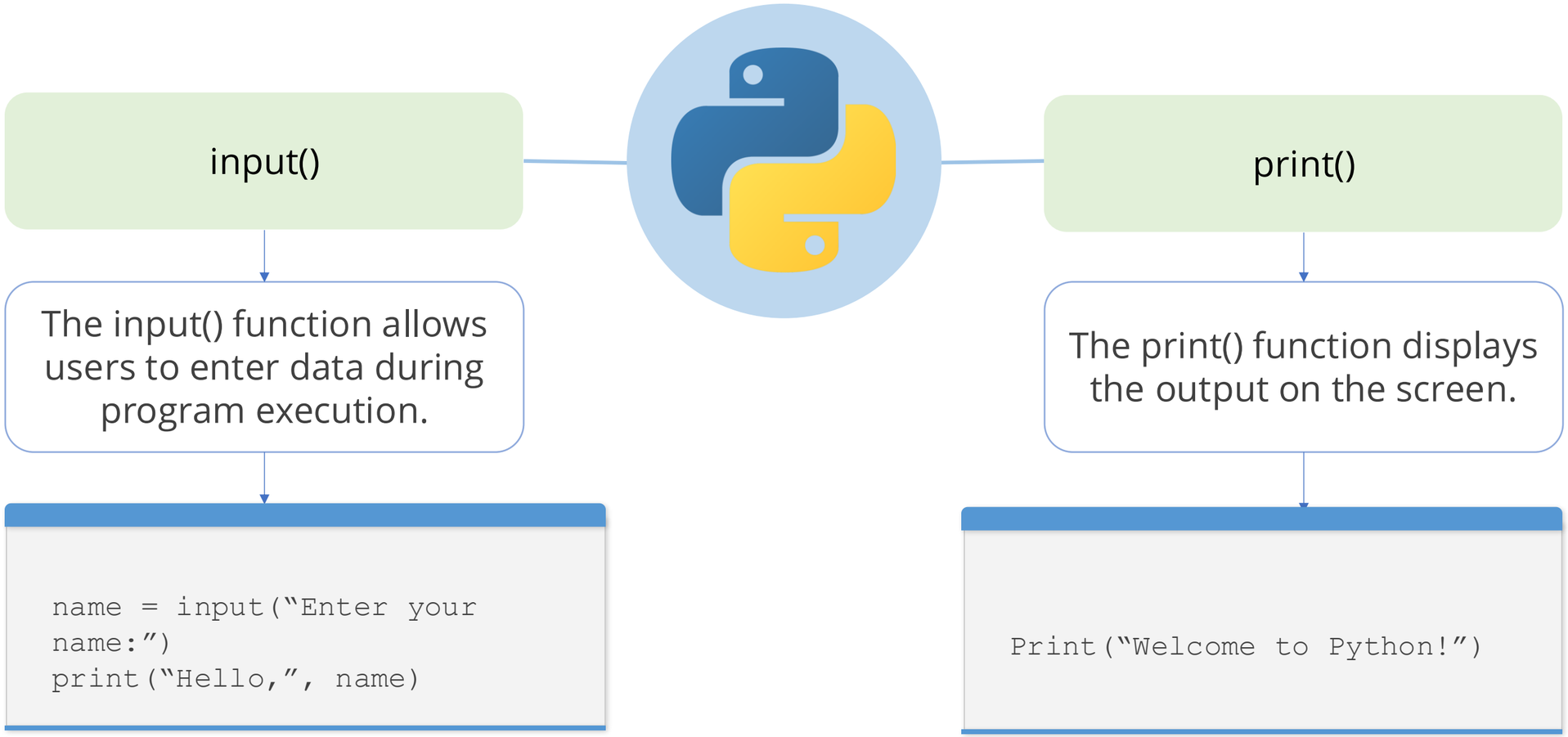
```
[1]: # This is a comment
     print("Hello World!")

     Hello World!
```

A comment in Python starts with #, and the rest of the line is considered a comment.

Python: Input and Output

Python provides simple yet powerful functions to handle user input and output display.
The two primary functions used for this are:



Quick Check

Which of the following identifier names follows Python's rules and best practices for readability?

A. _var123

B. _321CustomerDetailsDBTable

C. 1st_name

D. final_result_value

Demo: Writing and Running a Python Program

Duration: 10 minutes

Overview: This demonstration covers writing and running a Python program while exploring its basic syntax, indentation rules, and comments. You will learn how to incorporate user input and the print() function to create an interactive script.

Note:

Please download the demo document from the Reference Material section for step-by-step guidance.

INSTRUCTOR-LED ACTIVITY

Python Variables and Data Types

Python Variables: A Fundamental Building Block

Variables in Python are used to store data values, making it easier to manipulate and manage data in a program.

The following are examples of variables:

```
x = 10  # Integer variable
name = "Alice"  # String variable
price = 99.99  # Float variable
is_active = True  # Boolean variable


# Assigning multiple variables in a single line

a, b, c = 1, "Hello", 3.14
```

Data Types

Variables can store various kinds of data, and each kind has a specific function.

Scalar data

- Integer: 2323, 10
- Float: 32.3, 31.2e5
- Boolean: True , False
- Complex: 2 + 6j

Types of data

Aggregated data

- String: “abc” , ‘xyz’
- Set: {2, 4, 6, ‘abc’}
- List: [23, 9.8, ‘a’, ‘b’, ‘a’]
- Tuple: (34.5, ‘a’ , 45, ‘w’)
- Dictionary: {‘a’ : 1 , ‘b’ : 3}

Note

The data type of any object can be checked using the built-in function **type()**.

Scalar Data

This type represents a single value. The syntax and examples of scalar data are as follows:

```
int = 42
print(int)
```

```
42
```

```
float = 3.14
print(float)
```

```
3.14
```

```
bool = True
print(bool)
```

```
True
```

- **Integer** represents whole numbers without a fractional part.
- **Float** represents real numbers with a fractional part.
- **Boolean** represents two values, either *True* or *False*.

Scalar Data

This type represents a single value. The syntax and examples of scalar data are as follows:

```
complex = 3 + 4j
real = complex.real
imaginary = complex.imag
print("Complex Number:",complex)
print("Real Part:", real)
print("Imaginary Part:", imaginary)
```

```
Complex Number: (3+4j)
Real Part: 3.0
Imaginary Part: 4.0
```

- The **complex** data type in Python represents complex numbers.
- Complex numbers are expressed as **a + bj**, where **a** is the real part, and **b** is the imaginary part.

Aggregated Data

This type represents collections of values. The syntax and examples of aggregated data are as follows:

```
string = "Hello, World!"
print(string)
```

```
Hello, World!
```

```
set = {1, 2, 3, "a", "b", "c"}
print(set)
```

```
{'a', 1, 2, 3, 'c', 'b'}
```

```
list = [1, 2, 3, "a", "b", "c"]
print(list)
```

```
[1, 2, 3, 'a', 'b', 'c']
```

- **String** represents sequences of characters enclosed in single, double, or triple quotes.
- **Set** is an unordered collection of unique items in Python.
- **List** is an ordered collection of items, which are mutable.

Aggregated Data

This type represents collections of values. The syntax and examples of aggregated data are as follows:

```
tuple = (1, 2, 3, "a", "b", "c")
print(tuple)
```

```
(1, 2, 3, 'a', 'b', 'c')
```

```
dict = {"name": "Alice", "age": 35, "city": "New York"}
print(dict)
```

```
{'name': 'Alice', 'age': 35, 'city': 'New York'}
```

- **Tuple** is an ordered collection of items that are immutable.
- **Dictionaries** are an unordered collection of key-value pairs.

Data Assignment

Python variables are references to objects, but the actual data is contained in the objects.

```
x = 34
y = x
print('x = ', x, ' ; id of x: ', id(x))
print('y = ', y, ' ; id of x: ', id(y))
```

```
x =  34  ; id of x:  140210482584912
y =  34  ; id of x:  140210482584912
```

Here, *x* and *y* point to the same memory location where 34 (an integer object) is stored.

```
y = 78
print('x = ', x, ' ; id of x: ', id(x))
print('y = ', y, ' ; id of x: ', id(y))
```

```
x =  34  ; id of x:  140210482584912
y =  78  ; id of x:  140210482774800
```

Here, *y* points to a new integer object with 78 as a value, and *x* points to the previous object with a value of 34.

These references can be verified using the **id()** function.

Quick Check

What is the default data type of a variable assigned using the input() function in Python?

A. int

B. float

C. str

D. list

Demo: Storing User Information Using Different Data Types

Duration: 10 minutes

Overview: This demonstration covers storing and processing user input in Python using different data types. You will learn how Python assigns values such as strings, integers, and floats to variables, processes them for display using the print() function, and manages user input efficiently.

Note:

Please download the demo document from the Reference Material section for step-by-step guidance.

INSTRUCTOR-LED ACTIVITY

Python Operators

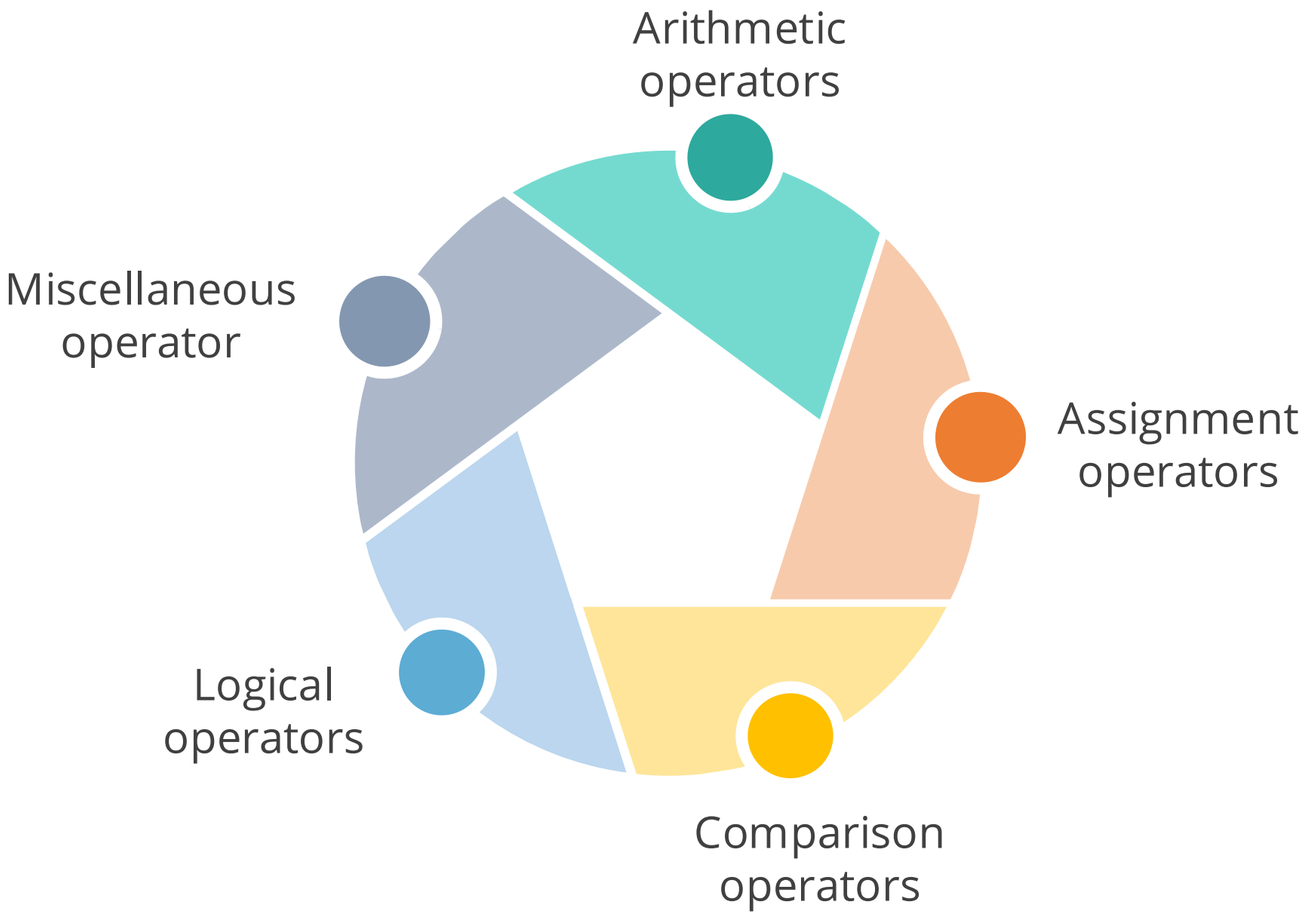
Operators

In Python, operators are special symbols or keywords that perform computations.

Its purposes include:

- Telling the interpreter to perform specific mathematical, relational, or logical operations to produce a result
- Operating on operands, which are the values or variables on which the operation is performed

Types of Operators



Arithmetic Operators

Different arithmetic operators are given below with examples:

Operator	Description	Example
+	Addition	>> x = 12 ; y = 50 >> x + y 62
-	Subtraction	>> x = 45 ; y = 24 >> x - y 21
*	Multiplication	>> x = 50 ; y = 4 >> x * y 200
/	Division	>> x = 50 ; y = 4 >> x / y 12.5

Arithmetic Operators

Different arithmetic operators are given below with examples:

Operator	Description	Example
%	Modulo	>> x = 50 ; y = 4 >> x % y 2
//	Integer divide	>> x = 50 ; y = 4 >> x // y 12
**	Exponent	>> x = 5 ; y = 4 >> x ** y 625

Assignment Operators

The equal to (=) operator is used for data assignment in Python.

Example:

```
a = 10
print(a)
a += 5
print(a)
```

```
10
15
```

- Assignment operators can be combined with arithmetic operators.
- Here, a += 5 is the same as a = a + 5.
- Similar operators are: -= , *=, /=, %=, //=, and **=

Comparison Operators

They are used to compare two values.

Operator	Description	Example
==	Returns **True** when the two values are equal	>> x = 20 ; y = 20 >> x == y True
!=	Returns **True** when the two values are not equal	>> x = 45 ; y = 24 >> x != y True

Note

These operators return true or false based on the comparison.

Comparison Operators

They are used to compare two values.

Operator	Description	Example
<	Returns **True** when the first value is less than the second	>> x = 20 ; y = 30 >> x < y True
>	Returns **True** when the first value is greater than the second	>> x = 40 ; y = 30 >> x > y True

Note

These operators return true or false based on the comparison.

Comparison Operators

They are used to compare two values.

Operator	Description	Example
<=	Returns **True** when the first value is less than or equal to the second value	>> x = 10 ; y = 30 >> x <= y True
>=	Returns **True** when the first value is greater than or equal to the second value	>> x = 30 ; y = 30 >> x >= y True

Note

These operators return true or false based on the comparison.

Logical Operators

They are used for combining conditional statements.

```
a = 1
b = 2
a == 1 and b == 2
```

```
True
```

```
a = 1
b = 4
a == 1 or b == 2
```

```
True
```

```
b = 0
not b
```

```
True
```

Operator	Description
and	Returns **True** if both statements are true
or	Returns **True** if one of the statements is true
not	Reverses the results, returns **True** if the result is false

Miscellaneous Operators

They are used for more specific operations than arithmetic, logical, and comparison operators. There are two types of miscellaneous operators: identity and membership.

```
a=['a','b','c']
b=['a','b','c']
a is b
```

```
False
```

```
a=['a','b','c']
b=['a','b','c']
a is not b
```

```
True
```

```
a = [1, 2, 3]
b = a
a is b
```

```
True
```

1. Identity operators

They compare variables to see whether they are the same object at the same memory address.

- **is:** Returns **True** if both variables are the same object
- **is not:** Returns **True** if both variables are not the same object

Miscellaneous Operators

```
a=[20, 45, 10]
10 in a
```

```
True
```

```
a=[20, 45, 10]
10 not in a
```

```
False
```

```
a=[20, 45, 10]
30 not in a
```

```
True
```

2. Membership operators

They test if a sequence is present in an object.

- **in:** Returns **True** if a value is present in the object
- **not in:** Returns **True** if a value is not present in the object

Quick Check

What is the main difference between the == and is operators in Python?

A. == checks object identity, whereas **is** compares values

B. == compares values, whereas **is** checks object identity

C. Both == and **is** perform the same function

D. == is used for arithmetic operations, while **is** is used for assignments

Demo: Comparing Numbers Using Python Operators

Duration: 10 minutes

Overview: This demonstration covers applying Python's comparison operators to evaluate relationships between numbers. You will learn how to take user input, implement comparison operations, and understand how conditions and comparisons support decision-making in Python programming.

Note:

Please download the demo document from the Reference Material section for step-by-step guidance.

INSTRUCTOR-LED ACTIVITY

Guided Practice

Overview **Duration: 20 minutes**

In this guided practice, you will take on the role of a software engineering intern at a fast-growing startup. Your first assignment is to develop a Comprehensive Employee Financial Insights System that collects and analyzes employee salary data. You will work hands-on with Python variables, data types, operators, indentation, and user input and output.

GUIDED PRACTICE

Key Takeaways

- Python is an interpreted language with powerful features like complex data structures and reusable modules.
- Multiple IDEs support Python development, including Jupyter Notebook, VS Code, Spyder, and PyCharm.
- Simple syntax makes Python easy to read, write, and maintain.
- Variables in Python store different data types, such as integers, floats, strings, booleans, lists, tuples, and dictionaries.
- Operators enable computations, including arithmetic, assignment, comparison, logical, and miscellaneous operations.

Q&A