PYTHON Complete Course

By:

PASS Education System

What is Python?

History

Python was created by Guido van Rossum, a Dutch programmer, and was first released in 1991. Here's a brief overview of the history of Python:

Origin and Early Development (1980s-1990s):

In the late 1980s, Guido van Rossum worked at the National Research Institute for Mathematics and Computer Science in the Netherlands.

In December 1989, Guido started working on a new programming language as a hobby project during his Christmas break. He named it Python, inspired by the British comedy series "Monty Python's Flying Circus."

Python's development was influenced by several programming languages, including ABC, Modula-3, C, and others.

Guido aimed to create a language that was easy to read, write, and understand, with a focus on code readability and simplicity.

Python 1.0 and Early Adoption (1994-2000):

Python 1.0 was released in January 1994, and it included features such as lambda functions, map/filter/reduce functions, and exception handling.

Python gained attention and started to be used in various academic and scientific communities.

In 2000, Python 2.0 was released, introducing list comprehensions, a garbage collector, and other enhancements.

Python 2.x and 3.x Split (2008):

As Python evolved, some decisions made in the early days became limitations, particularly related to Unicode and byte strings.

In 2008, Python 3.0 (also known as Python 3.x) was released, introducing significant changes and improvements to the language.

Python 3.x aimed to clean up the language, remove redundancy, and provide better support for Unicode and modern programming practices.

However, Python 3.x was not backward compatible with the earlier Python 2.x versions, causing a division in the Python community.

Python's Growing Popularity (2010s-Present):

Despite the initial challenges with the Python 2.x and 3.x split, Python's popularity continued to grow.

Python gained traction in web development, scientific computing, data analysis, machine learning, and artificial intelligence domains.

The Python Package Index (PyPI) became a central repository for third-party libraries, enabling easy installation and sharing of code.

Python became one of the most widely used programming languages, embraced by developers and adopted by organizations worldwide.

Recent Developments:

Guido van Rossum stepped down as Python's Benevolent Dictator for Life (BDFL) in July 2018 but remains involved in the Python community.

Python continues to evolve with regular releases. The most recent major version at the time of my knowledge cutoff is Python 3.9 (released in October 2020).

The Python Software Foundation (PSF) oversees Python's development and promotes its use.

What is Python?

Python is a high-level, interpreted programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python emphasizes code readability and a clean syntax, making it easier to understand and write compared to other programming languages.

Here are some key features of Python:

Readability: Python code is designed to be easily readable and understandable, using a clean and straightforward syntax.

Simplicity: Python aims to provide simplicity and ease of use, allowing developers to express concepts in fewer lines of code compared to other languages.

Versatility: Python is a versatile language that can be used for various applications, such as web development, data analysis, scientific computing, artificial intelligence, machine learning, and more.

Interpreted Language: Python code is executed line by line by the Python interpreter, which eliminates the need for compiling the code before running it. This makes development and debugging faster.

Large Standard Library: Python comes with a comprehensive standard library that provides numerous modules and functions to perform various tasks, ranging from file I/O to networking and web development.

Third-Party Libraries: Python has a vast ecosystem of third-party libraries and frameworks that extend its capabilities. These libraries offer ready-to-use solutions for specific domains, such as NumPy and Pandas for data analysis, Django and Flask for web development, and TensorFlow and PyTorch for machine learning.

Cross-Platform: Python is a cross-platform language, meaning it can run on different operating systems, including Windows, macOS, and various Linux distributions.

Python's popularity has been growing rapidly due to its simplicity, versatility, and the extensive support of its community. It has become one of the most widely used languages for a wide range of applications, from scripting small tasks to developing complex software systems.

Applications of Python?

Python is a versatile programming language that finds applications in various fields due to its simplicity, flexibility, and extensive ecosystem of libraries and frameworks. Here are some common fields where Python is used:

Web Development:

Python web frameworks like Django and Flask are widely used for building scalable and secure web applications.

Python's ease of use and rich libraries make it suitable for tasks such as server-side scripting, handling APIs, and web scraping.

Data Analysis and Visualization:

Python libraries like NumPy, Pandas, and Matplotlib provide powerful tools for data manipulation, analysis, and visualization.

Python is often used in data science workflows and can handle large datasets efficiently.

Machine Learning and Artificial Intelligence (AI):

Python offers popular libraries such as scikit-learn, TensorFlow, Keras, and PyTorch for implementing machine learning models and neural networks.

Python's simplicity and extensive libraries make it suitable for tasks like natural language processing, computer vision, and deep learning.

Scientific Computing:

Python, along with libraries like SciPy and NumPy, is widely used in scientific computing, numerical simulations, and mathematical modeling.

Python's ability to integrate with lower-level languages like C and Fortran allows for high-performance computing.

Automation and Scripting:

Python's easy-to-read syntax and extensive standard library make it ideal for automating repetitive tasks and scripting system administration tasks.

Python can interact with operating system APIs and perform tasks like file manipulation, process automation, and network automation.

Internet of Things (IoT):

Python is used in IoT applications to control and manage connected devices.

Its simplicity and support for microcontrollers make it suitable for prototyping, sensor data analysis, and device communication.

Game Development:

Python frameworks like Pygame provide a platform for game development, prototyping, and scripting.

Python's simplicity and libraries for graphics and multimedia make it a popular choice for building 2D games.

Desktop Applications:

Python, along with frameworks like PyQt and Tkinter, can be used for developing cross-platform desktop applications with graphical user interfaces (GUIs).

DevOps and System Administration:

Python is used for tasks related to system administration, automation, and DevOps workflows.

Python scripts can automate server configurations, deployment, and management

tasks.

Education and Research:

Python's simplicity and readability make it an excellent choice for teaching programming to beginners.

Python is widely used in academic research and scientific communities due to its extensive libraries and ease of use.

These are just a few examples of how Python is applied across different fields. Python's versatility and large community ensure that it continues to find new applications and use cases in various domains.

Basics of Python? / Requirements for Python / How to Start Python?

Installation and Setup:

Install Python: Download and install the latest version of Python from the official Python website **(python.org).**

Set up the Development Environment: Choose an Integrated Development Environment (IDE) like **PyCharm**, Visual Studio Code, or IDLE.

Python Syntax:

Print Statements: Use the print() function to display output on the console.

Variables: Understand how to declare variables and assign values to them.

Comments: Add comments to your code using the # symbol to make it more readable.

Data Types:

Numeric Types: Learn about integers, floating-point numbers, and basic mathematical operations.

Strings: Understand string data type, string manipulation, and string formatting.

Lists: Explore lists, which are ordered collections of items.

Tuples: Learn about immutable sequences of objects.

Dictionaries: Understand key-value pairs and how dictionaries are used to store and retrieve data.

Control Flow and Loops:

Conditional Statements: Use if, elif, and else to make decisions based on conditions.

Loops: Learn about for and while loops for iterative tasks.

Functions:

Defining Functions: Create your own functions to modularize code and reuse it.

Function Parameters: Understand how to define functions with parameters.

Return Statements: Learn how to return values from functions.

Input and Output:

User Input: Use the input() function to receive user input during program execution.

File Input and Output: Learn how to read from and write to files using Python.

Error Handling:

Exceptions: Understand how to handle errors and exceptions using try, except, and finally blocks.

Modules and Libraries:

Importing Modules: Learn how to import and use pre-built Python modules.

Standard Library: Explore commonly used modules such as math, random, and datetime.

Basic Concepts:

Operators: Understand arithmetic, comparison, logical, and assignment operators.

String Manipulation: Learn about string concatenation, slicing, and formatting.

Type Conversion: Convert data between different types using type casting.

Practice and Projects:

Apply the concepts learned through small coding exercises and projects.

Challenge yourself with simple tasks like calculating a sum, reversing a string, or building a basic calculator.

Basics Operators for Python?

Here are some basics operators commonly used in Python Syntax:

Arithmetic Operators:

Addition: + (e.g., a + b)

Subtraction: - (e.g., a - b)

Multiplication: * (e.g., a * b)

Division: / (e.g., a / b)

Floor Division: // (e.g., a // b returns the integer quotient)

Modulo (Remainder): % (e.g., a % b returns the remainder)

Exponentiation: ** (e.g., a ** b raises a to the power of b)

Assignment Operators:

Assignment: = (e.g., x = 5)

Addition Assignment: += (e.g., x += 3 is equivalent to x = x + 3)

Subtraction Assignment: -= (e.g., x -= 2 is equivalent to x = x - 2)

Multiplication Assignment: *= (e.g., x *= 4 is equivalent to x = x * 4)

Division Assignment: /= (e.g., x /= 2 is equivalent to x = x / 2)

Modulo Assignment: %= (e.g., x %= 3 is equivalent to x = x % 3)

Exponentiation Assignment: **= (e.g., x **= 2 is equivalent to x = x ** 2)

Comparison Operators:

Equal to: == (e.g., a == b)

Not equal to: != (e.g., a != b)

Greater than: > (e.g., a > b)

Less than: < (e.g., a < b)

Greater than or equal to: >= (e.g., a >= b)

Less than or equal to: <= (e.g., a <= b)

Logical Operators:

Logical AND: and (e.g., a and b)

Logical OR: or (e.g., a or b)

Logical NOT: not (e.g., not a)

Membership Operators:

in: Returns True if a value is found in a sequence (e.g., x in myList)

not in: Returns True if a value is not found in a sequence (e.g., x not in myList)

Identity Operators:

is: Returns True if two variables refer to the same object (e.g., x is y)

is not: Returns True if two variables do not refer to the same object (e.g., x is not y)

Bitwise Operators:

Bitwise AND: & (e.g., a & b)

Bitwise OR: | (e.g., a | b)

Bitwise XOR: ^ (e.g., a ^ b)

Bitwise NOT: ~ (e.g., ~a)

Left Shift: << (e.g., a << b)

Right Shift: >> (e.g., a >> b)

Basics Variables for Python?

In Python, you can create variables to store and manipulate data. Here are some examples of basic variable types in Python:

Integer: Variables that store whole numbers without decimal points.

```
age = 25
quantity = 10
```

Float: Variables that store decimal numbers.

```
pi = 3.14
price = 9.99
```

String: Variables that store a sequence of characters.

```
name = "PASS"
message = "Hello, world!"
```

Boolean: Variables that can have a value of either 'True' or 'False'.

```
is_valid = True
has_permission = False
```

List: Variables that store an ordered collection of elements.

```
numbers = [1, 2, 3, 4, 5]
names = ["Alice", "Bob", "Charlie"]
```

Tuple: Similar to lists, but they are immutable (cannot be modified after creation).

```
coordinates = (10, 20)
colors = ("red", "green", "blue")
```

Dictionary: Variables that store key-value pairs.

```
person = {"name": "PASS", "age": 25, "country": "Pakistan"}
```

None: A special variable that represents the absence of a value.

```
result = None
```

Basics Functions for Python?

In Python, there are several built-in functions that provide useful functionality for various tasks. Here are some of the basic functions frequently used in Python:

print(): Outputs text or values to the console.

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

input(): Reads input from the user through the console.

```
name = input("Enter your name: ")
```

len(): Returns the length of an object (e.g., string, list, tuple).

```
text = "Hello"
length = len(text)  # length will be 5
```

type(): Returns the type of an object.

```
value = 5
data_type = type(value)  # data_type will be <class 'int'>
```

int(), float(), str(), bool(): Convert values to integer, float, string, or boolean types, respectively.

```
num_str = "10"
num_int = int(num_str)  # num_int will be 10
```

range(): Generates a sequence of numbers.

```
numbers = range(1, 5)   # numbers will be [1, 2, 3, 4]
```

sum(): Returns the sum of all elements in an iterable.

```
numbers = [1, 2, 3, 4, 5]
total = sum(numbers)   # total will be 15
```

abs(): Returns the absolute value of a number.

```
value = -10
absolute = abs(value)   # absolute will be 10
```

round(): Rounds a number to a specified number of decimal places.

```
value = 3.14159
rounded = round(value, 2)   # rounded will be 3.14
```

sorted(): Returns a sorted version of a list or iterable.

```
numbers = [5, 3, 1, 4, 2]
sorted_numbers = sorted(numbers)   # sorted_numbers will be [1, 2, 3, 4, 5]
```

These are just a few examples of the basic built-in functions available in Python. Python provides a rich standard library with numerous functions for performing a wide range of tasks. Additionally, there are many external libraries and modules that offer additional functions to further extend Python's capabilities.

How to Run Python Code?

To run Python syntax, you have a few options:

Using a Python Interactive Shell:

Open a terminal or command prompt on your computer.

Type python or python3 (depending on your Python version) to start the Python interpreter.

Once you see the Python prompt (>>>), you can enter Python code line by line and press Enter to execute each line.

Running Python Scripts:

Create a new file with a .py extension using a text editor or an Integrated Development Environment (IDE).

Write your Python code in the file.

Save the file.

Open a terminal or command prompt and navigate to the directory where the Python file is located.

Type python or python3 (depending on your Python version) followed by the name of the Python file, and press Enter.

Python will execute the code in the file, and any output or results will be displayed in the terminal or command prompt.

Using an Integrated Development Environment (IDE):

Open your preferred Python IDE, such as PyCharm, Visual Studio Code, or IDLE.

Create a new Python file.

Write your Python code in the file.

Use the IDE's built-in run or execute command, often indicated by a play button icon or a keyboard shortcut, to run the code.

The IDE will execute the code and display the output or results in the IDE's output window or console.

Regardless of the method you choose, make sure you have Python installed on your computer. You can check if Python is properly installed by running python --version or python3 --version in the terminal or command prompt.

Remember to save your Python code with the .py extension and ensure that the file is located in the correct directory or specify the file path when executing it.

As you become more comfortable with Python, you can explore additional features of your chosen IDE or text editor to enhance your development experience, such as debugging, code completion, and integrated terminal support.

Python Projects :

- Simple Calculator

- To-Do List Manager

- Make Your Own for Practice

Project: Simple Calculator

Description:

Create a simple calculator program that performs basic arithmetic operations such as addition, subtraction, multiplication, and division. The program should prompt

the user to enter two numbers and the operation they want to perform. After performing the calculation, the program should display the result to the user.

Example Output:

```
=== Simple Calculator ===

Enter the first number: 10
Enter the second number: 5

Select an operation:
1. Addition
2. Subtraction
3. Multiplication
4. Division

Enter your choice (1-4): 1

The result is: 15
```

Implementation Steps:

- Display a welcome message and instructions to the user.
- Prompt the user to enter the first and second numbers.
- Prompt the user to select an operation by entering a number corresponding to the operation (e.g., 1 for addition, 2 for subtraction, etc.).

- Based on the user's choice, perform the corresponding arithmetic operation using appropriate Python operators.
- Display the result to the user.
- Optionally, allow the user to perform more calculations or exit the program.

Here's a sample code implementation to get you started:

```
print("=== Simple Calculator ===")
print()

num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))

print()
print("Select an operation:")
print("1. Addition")
print("2. Subtraction")
print("3. Multiplication")
print("4. Division")
print()

choice = int(input("Enter your choice (1-4): "))

result = 0

if choice == 1:
    result = num1 + num2
```

```
elif choice == 2:
    result = num1 - num2
elif choice == 3:
    result = num1 * num2
elif choice == 4:
    result = num1 / num2

print()
print("The result is:", result)
```

Project: To-Do List Manager

Description:

Create a simple to-do list manager that allows users to add, view, and delete tasks. The program should provide a menu-based interface where users can choose various options to interact with their to-do list. The tasks can be stored in memory or saved to a file for persistence.

Example Output:

```
==== To-Do List Manager ====

1. Add a task
2. View all tasks
3. Delete a task
4. Quit
```

```
Enter your choice (1-4): 1
Enter the task description: Buy groceries
Task added successfully!

==== To-Do List Manager ====

1. Add a task
2. View all tasks
3. Delete a task
4. Quit

Enter your choice (1-4): 2
Tasks:
1. Buy groceries

==== To-Do List Manager ====

1. Add a task
2. View all tasks
3. Delete a task
4. Quit

Enter your choice (1-4): 3
Enter the task number to delete: 1
Task deleted successfully!
```

```
==== To-Do List Manager ====

1. Add a task
2. View all tasks
3. Delete a task
4. Quit

Enter your choice (1-4): 4
Goodbye!
```

Implementation Steps:

- Display a welcome message and create an empty list to store tasks.
- Continuously prompt the user for their choice until they choose to quit.
- Depending on the user's choice, implement the following functionality:
- Option 1: Add a task
- Prompt the user to enter the task description.
- Add the task to the list and display a success message.
- Option 2: View all tasks
- Iterate over the list of tasks and display them with their corresponding numbers.
- Option 3: Delete a task
- Prompt the user to enter the number of the task they want to delete.

- Remove the task from the list and display a success message.
- Option 4: Quit
- Display a goodbye message and exit the program.
- Validate user inputs, handle errors, and provide appropriate feedback for invalid inputs.

Here's a sample code implementation to get you started:

```
# To-Do List Manager

print("==== To-Do List Manager ====\n")

tasks = []

while True:
    print("1. Add a task")
    print("2. View all tasks")
    print("3. Delete a task")
    print("4. Quit\n")

    choice = input("Enter your choice (1-4): ")

    if choice == "1":
        task_description = input("Enter the task description: ")
        tasks.append(task_description)
        print("Task added successfully!\n")
```

```
    elif choice == "2":
        if len(tasks) == 0:
            print("No tasks found.\n")
        else:
            print("Tasks:")
            for index, task in enumerate(tasks, start=1):
                print(f"{index}. {task}")
            print()
    elif choice == "3":
        if len(tasks) == 0:
            print("No tasks to delete.\n")
        else:
            task_number = int(input("Enter the task number to delete: "))
            if task_number > 0 and task_number <= len(tasks):
                deleted_task = tasks.pop(task_number - 1)
                print("Task deleted successfully!\n")
            else:
                print("Invalid task number.\n")
    elif choice == "4":
        print("Goodbye!")
        break
    else:
        print("Invalid choice. Please enter a number between 1 and 4.\n")
```