**SIMPLE CHATBOT USING PYTHON**

**A PROJECT REPORT**

*Done by*

**SANGEETHA A**

**MITHA K**

***In partial fulfillment for the award of the degree***

***Of***

***In***

******

**COMPUTER SOFTWARE COLLEGE**

**MADURAVOYAL**

**CHENNAI-600095**

***MAY-2025***

**BONAFIED CERTIFICATE**

Certified that this project report titled as **“SIMPLE CHATBOT”** is the bonafied work of **SANGEETHA A , MITHA K** carried out the project work under my supervision.

**SIGNATURE SIGNATURE**

***Mr.A.RAMAKRISHNAN Mr.HARI PRASAD***

**HEAD OF THE CSC PROJECT -**

**GUIDE TRAINER MADURAVOYAL MADURAVOYAL**

**ACKNOWLEDGEMENT**

At the outset thank god for having brought this throughout as in completing thisproject.We convey our thanks to**A.RAMAKRISHNAN (MANAGER), CSC**computereducation,maduravoyal Branch for giving an opportunity to embark on this project successfully.

With almost pleasure we here by express our faculty

**Mr.HARIPRSAD**

By.

**SANGEETHA A**

**MITHA K**

**TABLE OF CONTENT**

* **ABOUT PYTHON PROGRAMMING**
* **VARIABLES AND DATA TYPES**
* **OPERATORS IN PYTHON**
* **TYPE CONVERSION**
* **ABOUT CHATBOT**
* **COMMON FEATURES IN PYTHON**
* **ADVANTAGES**
* **DISADVANTAGES**
* **FUTURES CHATBOT IN PYTHON**
* **KEY FEATURES IN FUTURE CHATBOTS**
* **SOURCE CODE**
* **OUTPUT**
* **CONCLUSION**

**PYTHON PROGRAMMING LANGUAGE**

Python is a versatile and powerful high-level programming language created by **Guido Van Rossum** and first released in **1991.**

Known for its readability, simplicity, and broad applicability, python has become one of the most popular programming languages in the world. Its design philosophy emphasizes code readability, which allows developers to write clean and maintainable code. This approach has made Python a preferred choice for both beginners and experienced programmers**.**



**PYTHON**

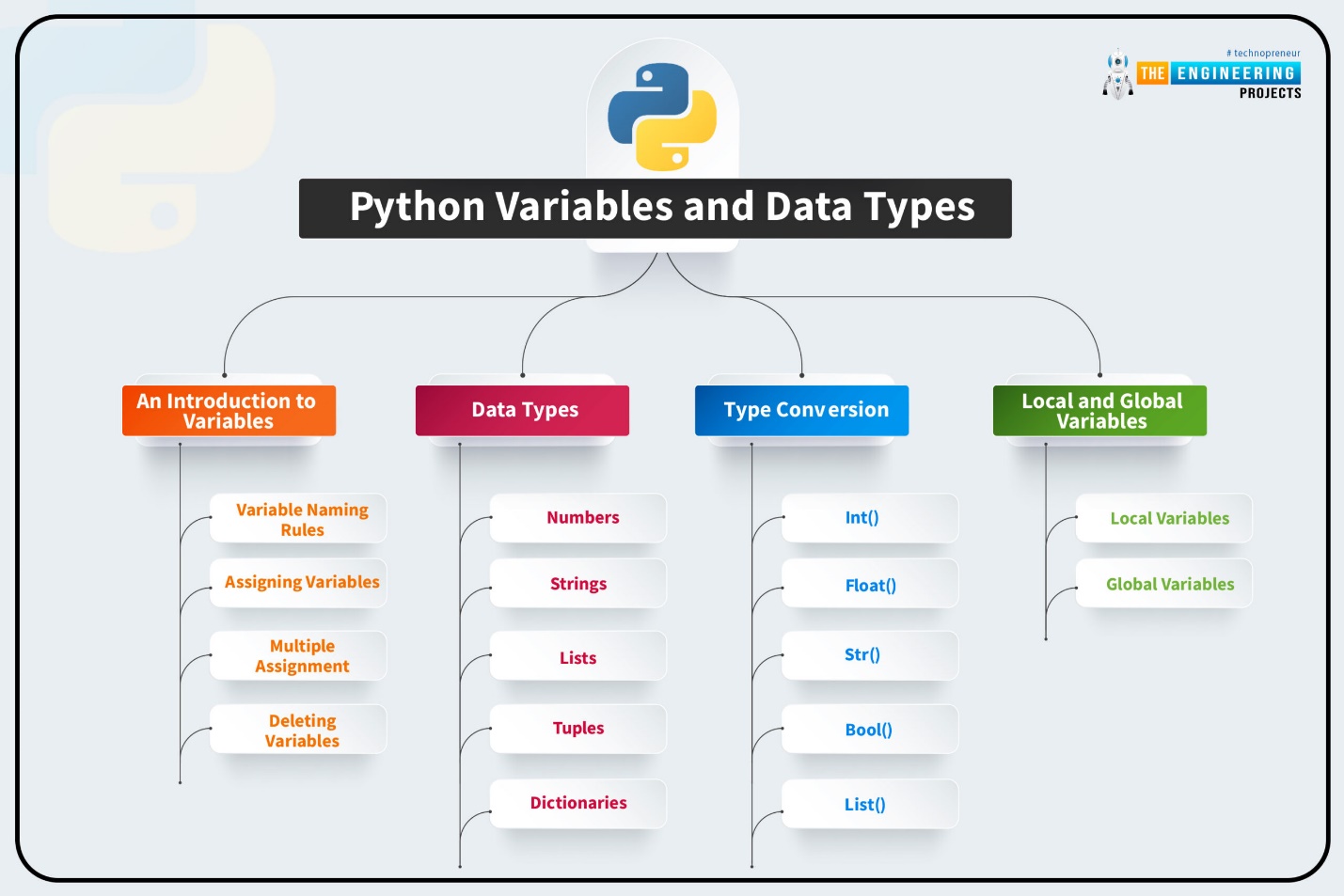
**KEY MILETONES IN PYTHON’S DEVELOPMENT:**

* 1990: Python 1.0 released ( Basic features like exception handling, core data types)
* 2000: Python 2.0 released (List comprehensions, garbage collections)
* 2008: Python 3.0 released (Improved Unicode support. Print as a function better integer division)
* 2020: Python 2 officially discontinued 2023: latest stable version is python 3.12

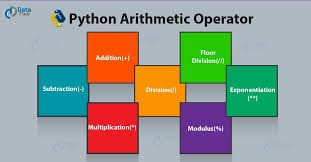
**PLATFORM DEPENDCIES:**

In python, platform dependencies refer to hoe some code or libraries might be specific to certain operating systems. Although python is designed to work across different systems like Windows, macOS, and Linux, there are still cases where code or libraries might rely on features unique to a particular platform. Developers often need to manage these dependencies by using cross-platform libraries, handling platform-specific details carefully ,and testing their applications on different operating systems to ensure compatiability.

**VARIABLES AND DATA TYPES:**



**OPERATORS IN PYTHON**



**EXAMPLE:**

def calculator():

n=int(input("enter a number:"))

n1=int(input("enter a number:"))

print("the sum",n+n1)

print("the sub",n-n1)

print("the multiple\*n1)

print("the divide",n/n1)

print("the modules",n%n1)

print("the floor division//n1)

calculator()

**OUTPUT:**

enter a number: 2

enter a number: 4

the sum 6

the sub -2

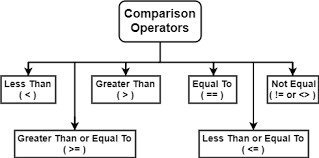
the multiple 8

the divide 0.5

the modules 2

the floor division 0

**COMPARISON OPERATOR:**



**EXAMPLE**

a = 10

b = 20

print("a == b:", a == b) # False

print("a != b:", a != b) # True

print("a > b:", a > b) # False

print("a < b:", a < b) # True

print("a >= b:", a >= b) # False

print("a <= b:", a <= b) # True

if a < b:

print("a is less than b")

**OUTPUT**

a == b: False

a != b: True

a > b: False

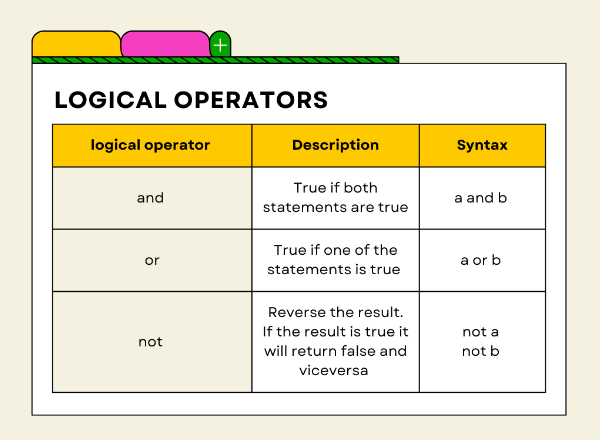
a < b: True

a >= b: False

a <= b: True

a is less than b

**LOGICAL OPERATOR:**



**EXAMPLE:**

x = 7

if x > 5 and x < 10:

print("x is between 5 and 10")

if x < 5 or x > 6:

print("x is either less than 5 or greater than 6")

if not(x == 10):

print("x is not equal to 10")

**OUTTPUT:**

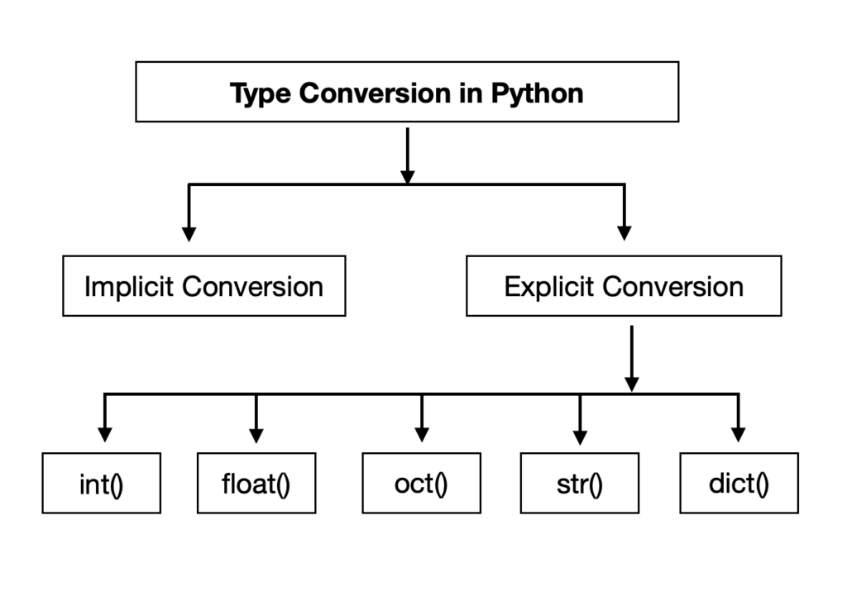
x is between 5 and 10

x is either less than 5 or greater than 6

x is not equal to 10

**TYPE CONVERSION:**

**Type conversion in Python refers to converting one data type into another. It can be**



**1. Implicit Type Conversion (Automatic)**

**Python automatically converts smaller data types to larger ones to prevent data loss.**

**EXAMPLE:**

x = 5

y = 2.5

z = x + y

print(z)

print(type(z))

**OUTPUT:**

7.5

<class 'float'>

**2. Explicit Type Conversion (Manual / Type Casting)**

**You manually convert types using built-in functions**

| **Function** | **Converts To** | **Example** |
| --- | --- | --- |
| **int()** | **Integer** | **int("5") → 5** |
| **float()** | **Floating point** | **float("3.14") → 3.14** |
| **str()** | **String** | **str(10) → "10"** |
| **bool()** | **Boolean** | **bool(0) → False** |
| **list(), tuple(), set()** | **Convert between collections** | **list("abc") → ['a','b','c']** |

**EXAMPLE:**

a = "10"

b = "3.5"

a\_int = int(a)

b\_float = float(b)

result = a\_int + b\_float

print("Result:", result)

print("Type of result:", type(result))

num = 42

print("String version:", str(num))

**OUTPUT:**

Result: 13.5

Type of result: <class 'float'>

String version: 42

**PYTHON CHATBOT (Simple Definition):**

A chatbot in Python is a script or program that uses input/output functions, conditional logic, or machine learning libraries (like NLTK, transformers, or ChatterBot) to mimic a conversation with users.

**COMMON FEATURES IN PYTHON:**

| **Feature** | **Description** |
| --- | --- |
| **🗣️ Text-based Interaction** | **Accepts user input via keyboard and responds with text.** |
| **🧠 Rule-Based Logic** | **Uses if-else statements or dictionaries to match keywords or phrases.** |
| **🔄 Continuous Conversation** | **Runs in a loop to simulate back-and-forth chatting.** |
| **🧾 Predefined Responses** | **Provides fixed answers for certain inputs (e.g., “hello”, “bye”).** |
| **🔡 Input Normalization** | **Converts user input to lowercase or removes punctuation for matching.** |
| **✅ Exit Option** | **Can end the conversation on command (e.g., “bye”).** |

**💡 ADVANCED FEATURES WITH LIBRARIES/APIs:**

| **Feature** | **Description** |
| --- | --- |
| **🧠 Natural Language Understanding (NLU)** | **Understands user intent using NLP libraries like spaCy or NLTK.** |
| **🤖 Machine Learning / AI** | **Learns from data using ChatterBot, transformers, or OpenAI API.** |
| **🌐 API Integration** | **Can fetch weather, news, or other data from web APIs.** |
| **🖥️ GUI Interface** | **Uses Tkinter or Flask for user-friendly interfaces.** |
| **🔊 Speech Input/Output** | **Converts voice to text and vice versa using speech\_recognition or pyttsx3.** |
| **📦 Memory / Context Handling** | **Remembers previous conversation using lists, dicts, or state tracking.** |

**SOURCE CODE:**

def chatbot():

print("🤖 Hello! I'm ChatBot. Type 'bye' to exit.\n")

while True:

user\_input = input("You: ").lower()

if user\_input == 'hello':

print("Bot: Hi there! How can I help you?")

elif user\_input == 'how are you':

print("Bot: I'm just a bunch of code, but I'm doing great!")

elif user\_input == 'what is your name':

print("Bot: I'm ChatBot, your friendly Python assistant.")

elif user\_input == 'bye':

print("Bot: Goodbye! Have a great day! 👋")

break

else:

print("Bot: Sorry, I didn't understand that.”)

chatbot()

**OUTPUT:**

🤖 Hello! I'm ChatBot. Type 'bye' to exit.

You: HELLO

Bot: Hi there! How can I help you?

You: HOW ARE YOU

Bot: I'm just a bunch of code, but I'm doing great!

You: WHAT IS YOUR NAME

Bot: I'm ChatBot, your friendly Python assistant.

You: BYE

Bot: Goodbye! Have a great day! 👋

**DEFINATION: while Loop in a Python Chatbot Program**

In a Python chatbot program, a while loop is used to continuously run the chatbot, allowing it to keep interacting with the user until a specific condition is met (like typing "bye" to exit).

**SAMPLE CODE:**

while True:

user\_input = input("You: ")

if user\_input == "bye":

print("Bot: Goodbye!")

break

else:

print("Bot: You said:", user\_input)

**OUTPUT:**

You: HII

Bot: You said: HII

You: bye

Bot: Goodbye!

**DEFINATION: if in a Python Chatbot**

In a Python chatbot program, the if statement is used to check if a specific condition is true. If the condition is true, the program executes the corresponding block of code (the chatbot's response). If the condition is false, it skips that block and moves to the next one.

**DEFINATION: elif in a Python Chatbot**

In a Python chatbot program, the elif (short for "else if") statement is used to check additional conditions after the initial if statement. It allows the program to evaluate more conditions if the previous if (or elif) conditions are false.

* elif helps you to test multiple conditions sequentially.
* It's an alternative to using multiple if statements when you want to check mutually exclusive conditions (only one condition should be true).

**DEFINATION: else in a Python Chatbot**

In a Python chatbot program, the else statement is used as the default case when none of the if or elif conditions are met. It is a fallback option that handles situations where the user's input doesn't match any of the predefined conditions.

* else ensures that your program always has a response, even if the user provides unexpected or unrecognized input.
* It prevents the chatbot from failing or becoming unresponsive when an input doesn't match any specific rule.

**How it Works in a Chatbot:**

1. if: If the user's input matches a specific pattern, the chatbot executes the corresponding response.
2. elif: If the first if condition isn't met, it checks additional conditions.
3. else: If none of the conditions are met, it provides a default or fallback response.

**BREAK:**

The break statement in Python is used to exit a loop prematurely, that is, before the loop has naturally finished iterating over all items or conditions. It is commonly used inside for and while loops to terminate the loop when a specific condition is met.

**EXAMPLE: Using break in a for loop**

for number in range(1, 10):

if number == 5:

print("Breaking the loop at number", number)

break

print("Current number:", number)

**OUTPUT:**

Current number: 1

Current number: 2

Current number: 3

Current number: 4

Breaking the loop at number 5

**Advantages of Using Python for Chatbots:**

1. **Easy to Learn and Use:**

* Python’s syntax is simple and clean, making it easy for developers (especially beginners) to build chatbots quickly.
* Its readable code ensures that developers can focus on chatbot logic rather than dealing with complex syntax.

1. **Rich Libraries and Frameworks:**

* Python offers a variety of libraries and frameworks for building chatbots, such as:
* Natural Language Processing (NLP): Libraries like NLTK, spaCy, and transformers are widely used for understanding and processing human language.
* Machine Learning: Libraries like TensorFlow and scikit-learn help in training models for intelligent responses.
* Chatbot Frameworks: Libraries like ChatterBot simplify chatbot development with machine learning.

1. **Wide Community Support:**

* Python has a huge community of developers and researchers, which means you can easily find resources, tutorials, and documentation for building chatbots.
* There’s a large pool of open-source projects to refer to or build upon.

1. **Integration with Other Tools and APIs:**

* Python integrates easily with external APIs, databases, and web services, enabling chatbots to fetch real-time data (e.g., weather, news, product info).
* It allows easy integration with messaging platforms like Slack, Facebook Messenger, Telegram, and more.

1. **Platform Independence:**

* Python is cross-platform, meaning chatbots built with Python can be run on various operating systems like Windows, macOS, and Linux.

1. **Flexibility for Both Simple and Complex Bots:**

* Python allows you to build both simple rule-based chatbots (using if-else conditions) and AI-poweredchatbots (using machine learning models).
* You can scale the complexity of the bot based on the needs, making Python suitable for a wide range of chatbot applications.

**❌ Disadvantages of Using Python for Chatbots**

1. **Performance Limitations:**

* Python can be slower than languages like C++ or Java because it is an interpreted language, which may affect the performance of chatbots in high-load scenarios or when real-time response is crucial.
* While Python is versatile, it might not be the best choice for applications requiring extremely fast response times or handling large amounts of data in real-time.

1. **Requires External Tools for Deployment:**

* While Python is excellent for development, deploying a Python-based chatbot to production often requires integration with other technologies (e.g., Flask, Django, or cloud platforms like Heroku or AWS Lambda).
* Running chatbots on mobile devices or as a web service may involve additional overhead for setting up the backend infrastructure.

1. **Complexity in NLP for Advanced Understanding:**

* For chatbots that need to handle advanced language understanding (like open-ended questions or sarcasm), Python requires advanced NLP techniques and machine learning models. These can be complex to implement and train.
* Fine-tuning large models (like GPT-3 or BERT) requires significant computational resources and can be time-consuming.

1. **Not Ideal for Voice-Based Bots:**

* While Python can support text-based chatbots very well, it is not the most suitable for voice-based bots compared to other languages or specialized platforms.
* Integrating speech recognition and speech synthesis can add complexity and may require additional libraries (e.g., speech\_recognition, pyttsx3).

1. **Scalability Challenges:**

* While Python works well for small to medium-scale projects, building and scaling a chatbot for a very large user base might require additional optimization (e.g., multi-threading, caching).
* Python doesn't offer built-in scalability mechanisms, so developers often need to rely on other technologies for handling scalability.

1. **Limited GUI Options:**

* While you can use libraries like Tkinter or PyQt for creating a graphical user interface (GUI), Python doesn’t have as rich or as polished GUI options for chatbot interfaces as some other languages or frameworks.

**FUTURES CHATBOT IN PYTHON**

* Here's a simple prototype of a "future chatbot" in Python using basic AI features like keyword recognition, emotion detection (very basic), and context memory. This example doesn't use external libraries like transformers or GPT models to keep it beginner-friendly, but I can expand it later if you'd like.
* The **future of chatbots** is evolving rapidly with the integration of **artificial intelligence (AI)**, **natural language processing (NLP)**, and **machine learning (ML)**. Future chatbots will be more **intelligent**, **context-aware**, and **emotionally responsive**, making interactions more human-like.

**Key Features of Future Chatbots**

| **Feature** | **Description** |
| --- | --- |
| **Human-like Conversations** | Understand context, slang, emotions, and tone. |
| **Multimodal Communication** | Use text, voice, images, and video in conversations. |
| **Emotional Intelligence** | Detect and respond to user emotions appropriately. |
| **Personalization** | Tailor responses based on individual user behavior, preferences, and history. |
| **Autonomous Task Execution** | Perform complex tasks (e.g., booking, purchases, scheduling) without human help. |
| **Multilingual Support** | Seamlessly converse in multiple languages. |
| **Integration with IoT & Smart Devices** | Control smart homes, vehicles, and other IoT ecosystems. |
| **Enhanced Security & Privacy** | Use of blockchain or secure AI models to protect user data. |

**Python Code: Future Chatbot Prototype**

**user\_name = ""**

**emotion\_state = ""**

**emotions = {**

**"happy": "I'm glad to hear that!",**

**"sad": "I'm here for you. Want to talk about it?",**

**"angry": "Take a deep breath. Would you like some music?",**

**"bored": "How about a fun fact or a joke?"**

**}**

**print("🤖 Hello! I'm your smart chatbot assistant.")**

**user\_name = input("Before we start, what's your name? ")**

**print(f"Nice to meet you, {user\_name}! How can I help you today?")**

**while True:**

**user\_input = input(f"{user\_name}: ").lower()**

**if "bye" in user\_input or "exit" in user\_input:**

**print("🤖 Goodbye! Have a great day!")**

**break**

**elif any(word in user\_input for word in emotions.keys()):**

**for emotion in emotions:**

**if emotion in user\_input:**

**emotion\_state = emotion**

**print(f"🤖 {emotions[emotion]}")**

**break**

**elif "name" in user\_input:**

**print(f"🤖 Your name is {user\_name}, and I never forget!")**

**elif "joke" in user\_input:**

**print("🤖 Why don’t scientists trust atoms? Because they make up everything!")**

**elif "fact" in user\_input:**

**print("🤖 Did you know? Honey never spoils. Archaeologists have found edible honey in ancient Egyptian tombs!")**

**else:**

**print("🤖 I'm still learning. Can you ask that in a different way?")**

**OUTOUT**

**vbnet**

**CopyEdit**

**🤖 Hello! I'm your smart chatbot assistant.**

**Before we start, what's your name? John**

**Nice to meet you, John! How can I help you today?**

**John: I'm feeling sad**

**🤖 I'm here for you. Want to talk about it?**

**John: Tell me a joke**

**🤖 Why don’t scientists trust atoms? Because they make up everything!**

**John: bye**

**🤖 Goodbye! Have a great day!**

**CONCLUSION:**

This Python chatbot program demonstrates a basic rule-based chatbot using simple conditional logic and a while loop. The chatbot interacts with the user by recognizing specific input phrases like "hello", "how are you", "what is your name", and "bye”.