Python Explanation - Super Simple Guide

Table of Contents

What is Python?

Python is a programming language used for:

- Data Science
- Web applications
- Automation
- Data analysis

Why Python for Data Science?

- Easy to learn and read
- Powerful libraries (Pandas, NumPy)
- Great for data analysis
- Large community support

Basic Python Concepts

1. Variables

What: Store data with a name

```
# Simple variable
name = "Virat Kohli"
age = 35
runs = 25000

# Using variables
print(name)  # Output: Virat Kohli
```

Real-life analogy: Like a labeled box where you store things

---

2. Data Types

```
# String (text)
player_name = "Rohit Sharma"

# Integer (whole number)
matches = 250

# Float (decimal number)
average = 48.5

# Boolean (True/False)
is_captain = True

# List (multiple items)
players = ["Virat", "Rohit", "Bumrah"]

# Dictionary (key-value pairs)
player = {
    "name": "Virat Kohli",
    "age": 35,
    "role": "Batsman"
}
```

Real-life analogy: Different types of containers for different things

---

3. Functions

What: Reusable block of code

```
# Define a function
def calculate_average(runs, matches):
    average = runs / matches
    return average

# Use the function
player_avg = calculate_average(10000, 200)
print(player_avg)  # Output: 50.0
```

Real-life analogy: Like a recipe - write once, use many times

---

4. If-Else (Conditions)

What: Make decisions in code

```
runs = 85

if runs >= 100:
    print("Century!")
```

```
elif runs >= 50:
    print("Half-century!")
else:
    print("Good innings")
```

Real-life analogy: Like traffic lights - if red, stop; if green, go

---

5. Loops

What: Repeat actions

```
# For loop - repeat for each item
players = ["Virat", "Rohit", "Bumrah"]
for player in players:
    print(player)

# Output:
# Virat
# Rohit
# Bumrah
```

Real-life analogy: Like checking each student's attendance one by one

---

6. Lists

What: Store multiple items in order

```
# Create a list
scores = [45, 67, 89, 102, 34]

# Access items
first_score = scores[0]  # 45
last_score = scores[-1]   # 34

# Add item
scores.append(78)

# Length
total_innings = len(scores)  # 6
```

Real-life analogy: Like a shopping list

---

7. Dictionaries

What: Store key-value pairs

```
# Create dictionary
player = {
    "name": "Virat Kohli",
    "runs": 25000,
    "average": 58.2,
    "format": "ODI"
}

# Access values
player_name = player["name"]  # "Virat Kohli"
player_runs = player["runs"]  # 25000

# Add new key-value
player["centuries"] = 48
```

Real-life analogy: Like a phone book - name (key) → phone number (value)

---

8. Classes (Object-Oriented Programming)

What: Blueprint for creating objects

```
# Define a class
class Player:
    def __init__(self, name, runs):
        self.name = name
        self.runs = runs

    def display_info(self):
        print(f"{self.name} scored {self.runs} runs")

# Create object
virat = Player("Virat Kohli", 25000)
virat.display_info()  # Output: Virat Kohli scored 25000 runs
```

Real-life analogy: Like a car blueprint - one design, many cars

---

9. Try-Except (Error Handling)

What: Handle errors gracefully

```
try:
    result = 100 / 0  # This will cause error
except:
    print("Error: Cannot divide by zero")
```

Real-life analogy: Like having a backup plan if something goes wrong

---

10. Importing Libraries

What: Use pre-written code

```
# Import entire library
import pandas

# Import with alias
import pandas as pd

# Import specific function
from datetime import datetime
```

Real-life analogy: Like borrowing tools from a friend instead of making them

---

Your Project's Python Code Explained

File 1: app.py (Main File)

Import Section

```
import streamlit as st
from utils.db_connection import initialize_database
import os
from dotenv import load_dotenv
```

What it does:

- `streamlit` → Tool to create web interface
- `db_connection` → Your custom code to connect to database
- `os` → Work with operating system (files, environment)
- `dotenv` → Load passwords from .env file

Simple explanation: "Bring all the tools I need"

Load Environment Variables

```
load_dotenv()
```

What it does: Reads the `.env` file and loads passwords/settings

Simple explanation: "Load my secret passwords from the .env file"

Page Configuration

```
st.set_page_config(
    page_title="Cricket Data Analytics",
    page_icon="🏏",
    layout="wide"
)
```

What it does: Sets up the webpage title, icon, and layout

Simple explanation: "Set up how my webpage looks"

Main Function

```
def main():
    if 'db_initialized' not in st.session_state:
        with st.spinner("Initializing database..."):
            if initialize_database():
                st.session_state.db_initialized = True
```

What it does:

1. Check if database is already set up
2. If not, set it up
3. Show loading spinner while setting up
4. Remember that it's done

Simple explanation: "Check if database is ready. If not, set it up once."

Sidebar Navigation

```
with st.sidebar:
    st.title("🏏 Cricket Data Analytics")

    page = st.radio(
        "Navigation",
        ["🏠 Home", "🔴 Live Matches", "🔍 SQL Queries"]
    )
```

What it does:

1. Create a sidebar (menu on left)
2. Show title
3. Show radio buttons for navigation
4. Store selected page in `page` variable

Simple explanation: "Create a menu where users can choose which page to see"

Show Different Pages

```
if page == "🏠 Home":
    from pages import home
    home.show()
elif page == "🔍 SQL Queries":
    from pages import sql_queries
    sql_queries.show()
```

What it does:

1. Check which page user clicked
2. Import that page's code
3. Show that page

Simple explanation: "If user clicked Home, show Home page. If clicked SQL Queries, show SQL page."

Check Database Connection

```
def check_database_connection():
    try:
        db = get_db_connection()
        result = db.execute_query("SELECT 1", fetch=True)
        return result is not None
    except:
        return False
```

What it does:

1. Try to connect to database
2. Run a simple query
3. If it works, return True
4. If error, return False

Simple explanation: "Test if database is working. Return Yes or No."

File 2: db_connection.py (Database Connection)

Class Definition

```
class DatabaseConnection:
    def __init__(self):
        self.host = os.getenv('DB_HOST', 'localhost')
        self.user = os.getenv('DB_USER', 'root')
        self.password = os.getenv('DB_PASSWORD', '')
        self.database = os.getenv('DB_NAME', 'cricbuzz_db')
```

What it does:

1. Create a class (blueprint) for database connections
2. `__init__` runs when you create a new connection
3. Get database details from .env file
4. Store them in variables

Simple explanation: "When creating a database connection, get the username, password, and database name from the .env file"

Connect Function

```
def connect(self):
    try:
        self.connection = mysql.connector.connect(
            host=self.host,
            user=self.user,
            password=self.password,
            database=self.database
        )
        return self.connection
    except Error as e:
        st.error(f"Database connection error: {e}")
        return None
```

What it does:

1. Try to connect to MySQL database
2. Use the host, user, password from **init**
3. If successful, return the connection
4. If error, show error message and return None

Simple explanation: "Try to connect to the database. If it works, great! If not, show error."

Execute Query Function

```
def execute_query(self, query, params=None, fetch=True):
    try:
        connection = self.connect()
        cursor = connection.cursor(dictionary=True)
        cursor.execute(query, params or ())

        if fetch:
            result = cursor.fetchall()
            return result
        else:
            connection.commit()
            return True
    except Error as e:
        st.error(f"Query execution error: {e}")
        return None
```

What it does:

1. Connect to database
2. Create a cursor (pointer to read data)
3. Execute the SQL query
4. If `fetch=True`, get the results
5. If `fetch=False`, save changes (for INSERT/UPDATE)
6. If error, show error message

Simple explanation:

- "Connect to database"
- "Run the SQL query"
- "Get the results"
- "If error, show message"

File 3: sql_queries.py (SQL Queries Page)

Show Function

```
def show():
    st.title("🔍 SQL Analytics & Practice Queries")

    tab1, tab2, tab3 = st.tabs([
        "📚 Beginner",
        "📊 Intermediate",
        "🚀 Advanced"
    ])

    with tab1:
        display_beginner_queries()
```

What it does:

1. Show page title
2. Create 3 tabs (like folders)
3. In first tab, show beginner queries

Simple explanation: "Create a page with 3 tabs for different difficulty levels"

Execute Query Function

```
def execute_query(query, query_name):
    db = get_db_connection()

    with st.spinner(f"Executing {query_name}..."):
        results = db.execute_query(query, fetch=True)

        if results and len(results) > 0:
            df = pd.DataFrame(results)
            st.success(f"✅ Found {len(results)} rows")
            st.dataframe(df)
        else:
            st.info("No results found")
```

What it does:

1. Get database connection
2. Show loading spinner
3. Execute the SQL query
4. If results found:
    - Convert to Pandas DataFrame
    - Show success message
    - Display table
5. If no results, show message

Simple explanation:

- "Connect to database"
- "Run the query"
- "Show results in a table"
- "If no results, say so"

Python Libraries Used

1. **Streamlit**

Purpose: Create web interface without HTML/CSS

```
import streamlit as st

# Display text
st.title("My Title")
st.write("Some text")

# Create button
if st.button("Click me"):
    st.success("Button clicked!")

# Show table
st.dataframe(my_data)

# Create columns
col1, col2 = st.columns(2)
with col1:
    st.metric("Players", 100)
```

Why use it: Easy way to create data dashboards in Python

---

2. **Pandas**

Purpose: Work with data in tables (DataFrames)

```
import pandas as pd

# Create DataFrame from dictionary
data = {
    'name': ['Virat', 'Rohit'],
    'runs': [25000, 18000]
}
df = pd.DataFrame(data)

# Read from database results
df = pd.DataFrame(sql_results)

# Basic operations
df['average'] = df['runs'] / df['matches']
df_sorted = df.sort_values('runs', ascending=False)
top_10 = df.head(10)
```

Why use it: Makes data manipulation easy

---

3. **mysql-connector-python**

Purpose: Connect Python to MySQL database

```
import mysql.connector

# Connect
connection = mysql.connector.connect(
    host='localhost',
    user='root',
    password='your_password',
    database='cricbuzz_db'
)

# Execute query
cursor = connection.cursor()
cursor.execute("SELECT * FROM players")
results = cursor.fetchall()
```

Why use it: Bridge between Python and MySQL

---

4. **python-dotenv**

Purpose: Load environment variables from .env file

```
from dotenv import load_dotenv
import os

# Load .env file
load_dotenv()

# Get values
db_password = os.getenv('DB_PASSWORD')
api_key = os.getenv('RAPIDAPI_KEY')
```

Why use it: Keep passwords secure, not in code

---

5. **requests**

Purpose: Make HTTP requests to APIs

```
import requests

# Make GET request
response = requests.get(
    url="https://api.example.com/data",
    headers={"Authorization": "Bearer token"}
)

# Get JSON data
data = response.json()
```

Why use it: Get data from external APIs

Code Examples with Explanations

Example 1: Simple Function

```
def calculate_strike_rate(runs, balls):
    """Calculate cricket strike rate"""
    if balls == 0:
        return 0
    strike_rate = (runs / balls) * 100
    return strike_rate

# Use it
sr = calculate_strike_rate(50, 40)
print(sr)  # Output: 125.0
```

Explanation:

1. Define function with 2 parameters
2. Check if balls is 0 (avoid division by zero)
3. Calculate strike rate formula
4. Return the result

Example 2: Working with Lists

```
# List of player names
players = ["Virat", "Rohit", "Bumrah", "Jadeja"]

# Loop through list
for player in players:
    print(f"Player: {player}")

# Add new player
players.append("Shami")

# Get first player
first = players[0]  # "Virat"

# Get last player
last = players[-1]  # "Shami"

# Count players
total = len(players)  # 5
```

Example 3: Dictionary Operations

```
# Player information
player = {
    "name": "Virat Kohli",
    "age": 35,
    "runs": 25000,
    "format": "ODI"
}

# Access values
name = player["name"]
runs = player["runs"]

# Add new key
player["centuries"] = 48

# Check if key exists
if "average" in player:
    print(player["average"])
else:
    print("Average not found")

# Loop through dictionary
for key, value in player.items():
    print(f"{key}: {value}")
```

Example 4: Class Example

```
class CricketPlayer:
    def __init__(self, name, runs, matches):
        self.name = name
        self.runs = runs
        self.matches = matches

    def calculate_average(self):
        if self.matches == 0:
            return 0
        return self.runs / self.matches

    def display_stats(self):
        avg = self.calculate_average()
        print(f"{self.name}: {self.runs} runs in {self.matches} matches")
        print(f"Average: {avg:.2f}")

# Create player object
virat = CricketPlayer("Virat Kohli", 25000, 500)
virat.display_stats()
```

```
# Output:
# Virat Kohli: 25000 runs in 500 matches
# Average: 50.00
```

Example 5: Error Handling

```python
def safe_divide(a, b):
    try:
        result = a / b
        return result
    except ZeroDivisionError:
        print("Error: Cannot divide by zero")
        return None
    except Exception as e:
        print(f"Error: {e}")
        return None

# Use it
result1 = safe_divide(10, 2)   # 5.0
result2 = safe_divide(10, 0)   # Error message, returns None
```

Summary: Key Python Concepts in Your Project

Concept	Used For	Example in Project
Variables	Store data	`page = "Home"`
Functions	Reusable code	`def show():`
Classes	Database connection	`class DatabaseConnection:`
If-Else	Decisions	`if page == "Home":`
Loops	Repeat actions	`for player in players:`
Try-Except	Error handling	Database connection errors
Dictionaries	Store key-value data	Query results
Lists	Store multiple items	List of queries
Imports	Use libraries	`import streamlit as st`

🎓 For Your Presentation

When asked "Explain your Python code":

"My project uses Python with several key concepts:

1. ***Functions*** *- I created reusable functions like* `execute_query()` *to run SQL queries*
2. ***Classes*** *- I used Object-Oriented Programming for the database connection*
3. ***Error Handling*** *- I used try-except blocks to handle database errors gracefully*
4. ***Libraries*** *- I used Streamlit for the interface, Pandas for data manipulation, and mysql-connector for database access*
5. ***Conditionals*** *- I used if-else statements for navigation and decision-making*

All the code is in Python - no other programming languages needed!"

Remember: You don't need to memorize code. Just understand:

- What each part does
- Why it's needed
- How it fits together

Good luck! 🚀