SHREE SWAMINARAYAN COLLEGE OF COMPUTER SCIENCE

GURUKUL CAMPUS -SARDARNAGAR-BHAVNAGAR

INTERSHIP MIDTERM REPORT FOR REVIEW

[FROM: 01 / 01 /2026   TO: 01 / 03 /2026]

2324152

Chudasama Krishnadeepsinh S

STUDENT OF BCA SEMESTER-6

TITLE OF INTERNSHIP

Python Based Internship

SUPERVISOR NAME

AnkitPatel

ORGANISATION NAME

THE EASYLEARN

ORGANISATION ADDRESS

105, EVA SURBHI, OPP AKSHARWADI TEMPLE,
WAGHAWADI ROAD, BHAVNAGAR

Introduction to Python

- As a BCA Semester-6 student, I am undertaking a Python-based internship at The Easylearn, an organization focused on providing practical training in programming and software development. This internship aims to enhance my foundational skills in Python programming through hands-on coding exercises and real-world applications. The company profile emphasizes skill-building in core programming concepts, scripting, and basic web interactions, aligning with my academic background in computer science.

- During the first half of the internship period (January 1, 2026 to March 1, 2026), I have engaged in structured learning modules to build proficiency in Python fundamentals and introductory advanced topics.

- Python is a high-level, interpreted programming language widely recognized for its simplicity, readability, and versatility. It is designed to reduce programming complexity while enabling developers to build powerful applications efficiently. Python's syntax is clean and human readable, making it highly suitable for beginners and professionals alike.

- One of the most significant advantages of Python is its emphasis on code readability. The language structure encourages developers to write clear and concise programs, which improves maintainability and reduces the likelihood of errors. Python eliminates unnecessary syntactical elements, allowing programmers to focus more on logic rather than complex language rules. This design philosophy contributes greatly to faster development cycles and improved productivity.

- Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming. This flexibility allows developers to adopt various approaches depending on project requirements. The ability to combine different programming styles within the same application makes Python extremely adaptable. It enables efficient handling of small scripts as well as large-scale enterprise applications.

- Another major strength of Python lies in its vast ecosystem of libraries and frameworks. Python provides extensive built-in modules and third party packages that simplify complex tasks such as data analysis, machine learning, web development, automation, and system integration. These libraries significantly reduce development effort, as developers can reuse well-tested solutions instead of building functionalities from scratch.

Programming Style and Basic Syntax

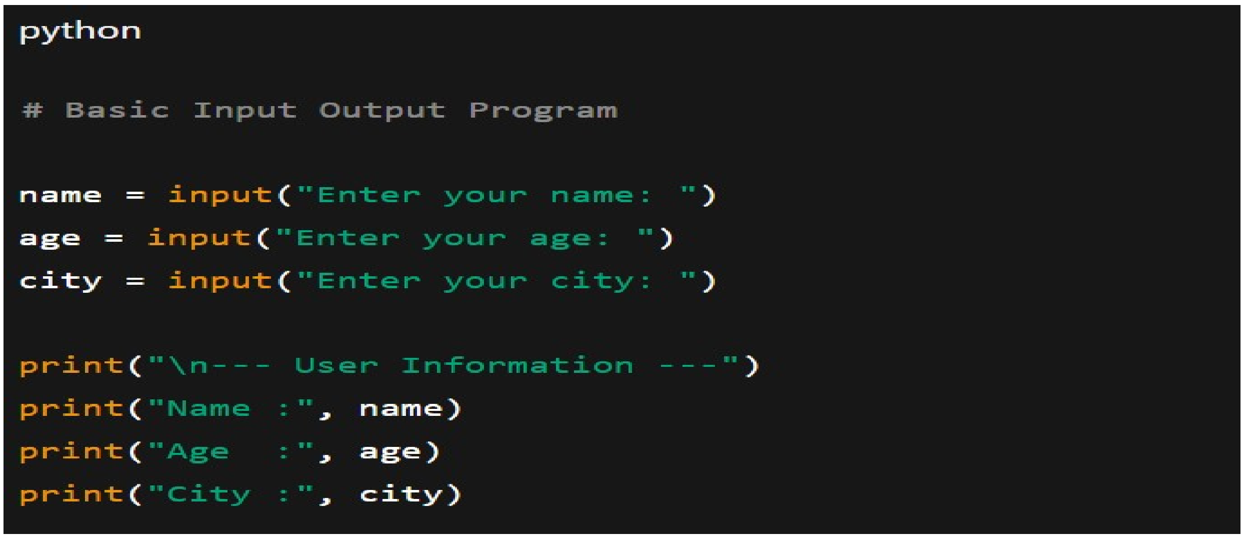
- Python emphasizes readability and structured programming practices. Unlike many programming languages, Python uses indentation to define code blocks, promoting visually clean and logically organized code. Variables are dynamically typed, allowing flexible assignment without explicit type declarations.

- Python provides fundamental data types including integers, floating-point numbers, strings, and Boolean values. Proper naming conventions play a critical role in writing maintainable programs. Meaningful variable names enhance readability, reduce errors, and improve collaboration.

Input and Output Operations

- Input and output mechanisms enable programs to interact dynamically with users. Python allows user-driven execution by accepting runtime inputs, making applications adaptable to various scenarios.

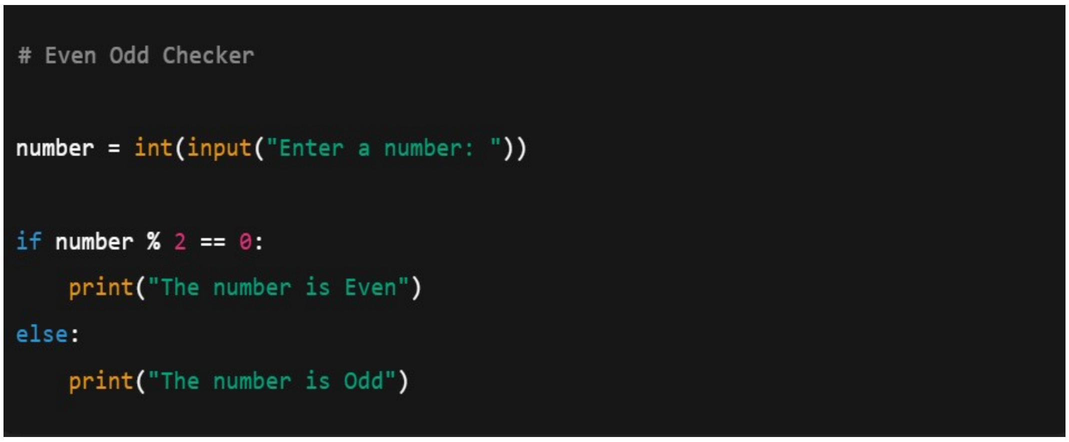
- Output operations communicate results, messages, and system feedback. Python supports simple printing as well as advanced formatting methods that improve clarity and presentation. Together, input and output operations form the basis of interactive software systems.



```
python

# Basic Input Output Program

name = input("Enter your name: ")
age = input("Enter your age: ")
city = input("Enter your city: ")

print("\n--- User Information ---")
print("Name :", name)
print("Age  :", age)
print("City :", city)
```

Conditional Statements

- Conditional statements enable decision-making within programs. They allow software to evaluate logical conditions and execute specific code paths base on outcomes.

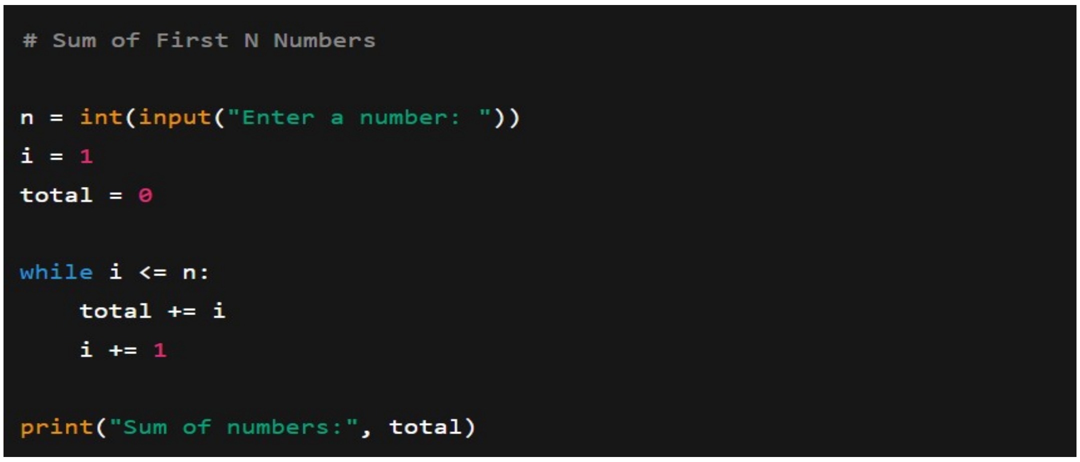


```
# Even Odd Checker

number = int(input("Enter a number: "))

if number % 2 == 0:
    print("The number is Even")
else:
    print("The number is Odd")
```

Looping Constructs

- Loops allow repetitive execution of code blocks, improving efficiency and reducing redundancy. Python provides flexible looping mechanisms suitable for iteration, counting, searching and pattern generation.

- Control statements such as break, continue, and pass refine loop behaviour. These constructs provide developers with precise control over execution flow, improving algorithmic design and problem-solving efficiency.

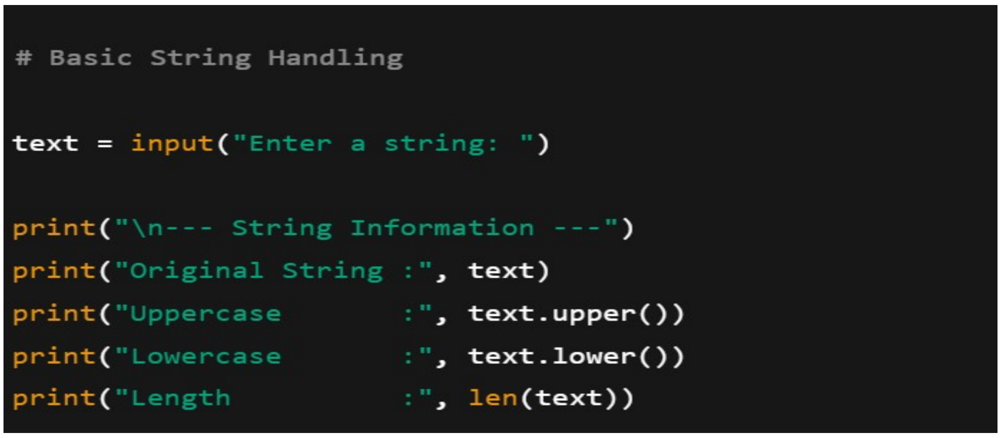


```python
# Sum of First N Numbers

n = int(input("Enter a number: "))
i = 1
total = 0

while i <= n:
    total += i
    i += 1

print("Sum of numbers:", total)
```

String Handling

- Strings represent sequences of characters and are essential in programming. Python treats strings as immutable objects, ensuring consistent behaviour. String operations include indexing, slicing, concatenation, and membership testing.

- Built-in string methods simplify formatting, cleaning, searching, and transformation. Effective string handling is crucial for applications involving text processing, validation, and structured reporting.

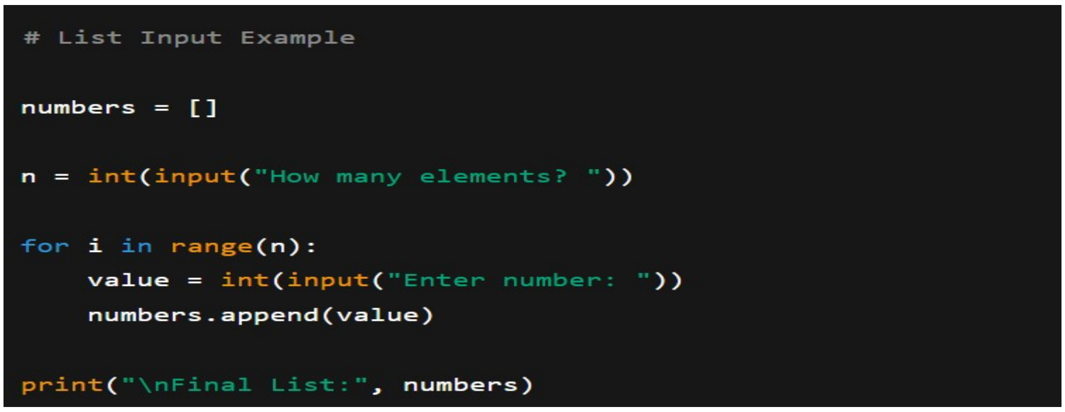


```python
# Basic String Handling

text = input("Enter a string: ")

print("\n--- String Information ---")
print("Original String :", text)
print("Uppercase       :", text.upper())
print("Lowercase       :", text.lower())
print("Length          :", len(text))
```

List Data Structure

- Lists are dynamic, mutable collections used to store grouped data. Python lists support indexing, slicing, and various modification methods. They are widely used for managing datasets, records, and structured information.

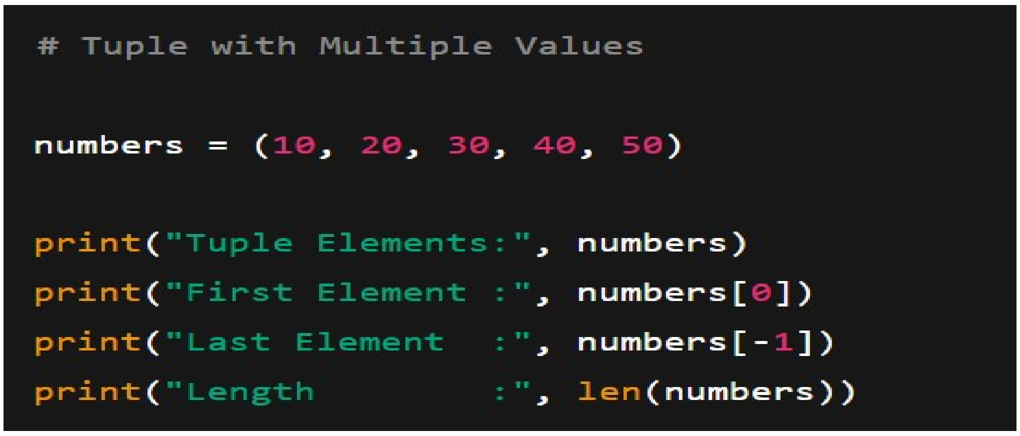
- . List operations enhance data organization, while list comprehension provides an efficient mechanism for generating and transforming lists. Lists are fundamental to advanced programming techniques



```
# List Input Example

numbers = []

n = int(input("How many elements? "))

for i in range(n):
    value = int(input("Enter number: "))
    numbers.append(value)

print("\nFinal List:", numbers)
```

Tuple Data Structure

- A tuple is an ordered, immutable collection of elements used to store multiple values in a single variable. Unlike lists, tuples cannot be modified after creation, making them suitable for storing fixed data. Tuples are defined using parentheses and can contain elements of different data types.
- One of the key advantages of tuples is data integrity. Since tuples are immutable, their values remain constant throughout program execution. This characteristic makes tuples ideal for representing data that should not change, such as coordinates, configuration settings, and constant records.
- Tuples are memory-efficient compared to lists, as their immutability allows Python to optimize storage. Additionally, tuples provide faster performance in certain scenarios due to their fixed structure. They also support indexing, slicing, and iteration, similar to other sequence data types.
- Tuples are commonly used when multiple values need to be grouped logically without the requirement of modification. They are particularly useful when returning multiple values from functions or representing structured datasets.
- Overall, tuples provide a reliable and efficient mechanism for managing fixed collections of data within Python applications.



```
# Tuple with Multiple Values

numbers = (10, 20, 30, 40, 50)

print("Tuple Elements:", numbers)
print("First Element :", numbers[0])
print("Last Element  :", numbers[-1])
print("Length        :", len(numbers))
```

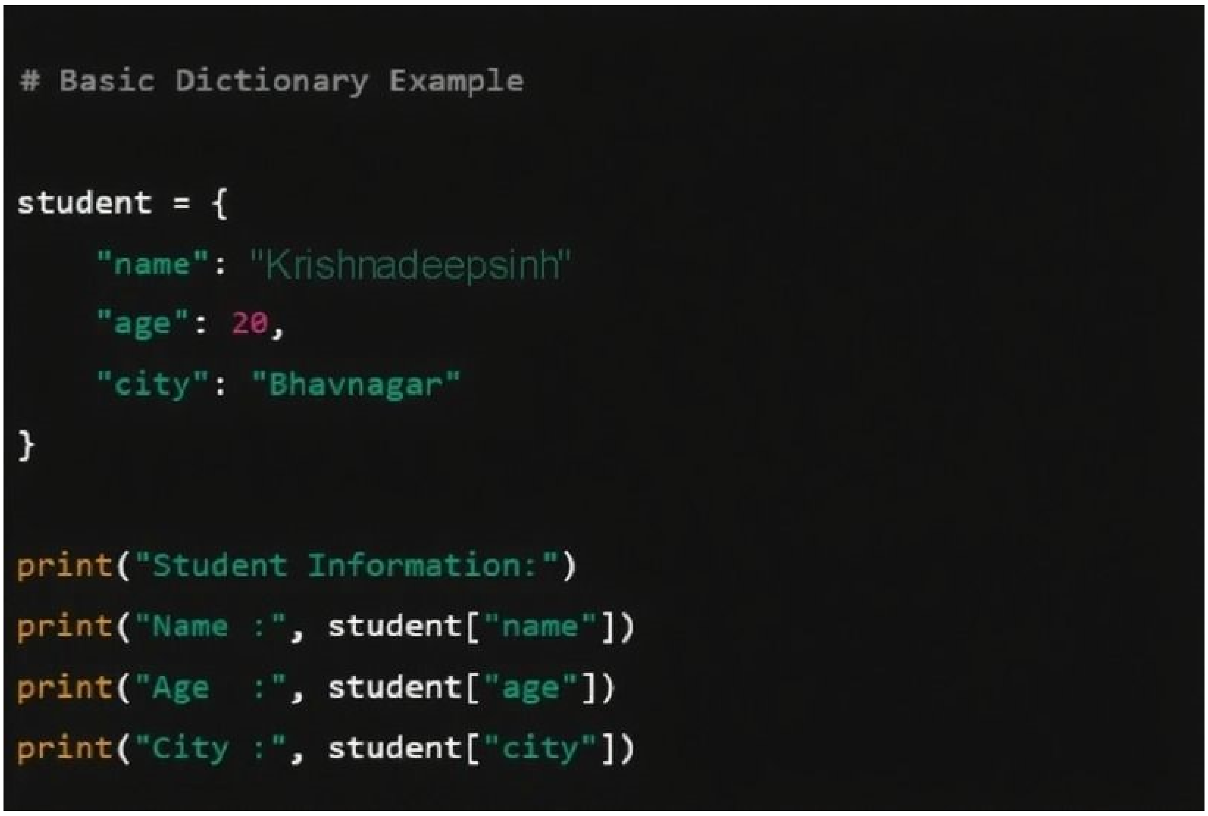
Dictionary Data Structure

- Dictionaries organize data using key-value pairs, enabling efficient retrieval. They are essential for structured data storage, mappings, and database-like operations.

- Dictionaries are widely used when data needs to be organized logically rather than sequentially. Each key acts as an identifier that maps to a specific value. This structure enables efficient data management and quick retrieval operations.

- One of the major advantages of dictionaries is their flexibility. Developers can easily add, update, or remove elements without affecting other data. Dictionary methods such as keys(), values(), items(), and get() provide convenient mechanisms for accessing and manipulating stored information.

- Dictionaries are particularly useful in real-world applications involving records, databases, mappings, configurations, and data processing systems. Their ability to represent complex relationships through nested dictionaries makes them essential in modern programming.



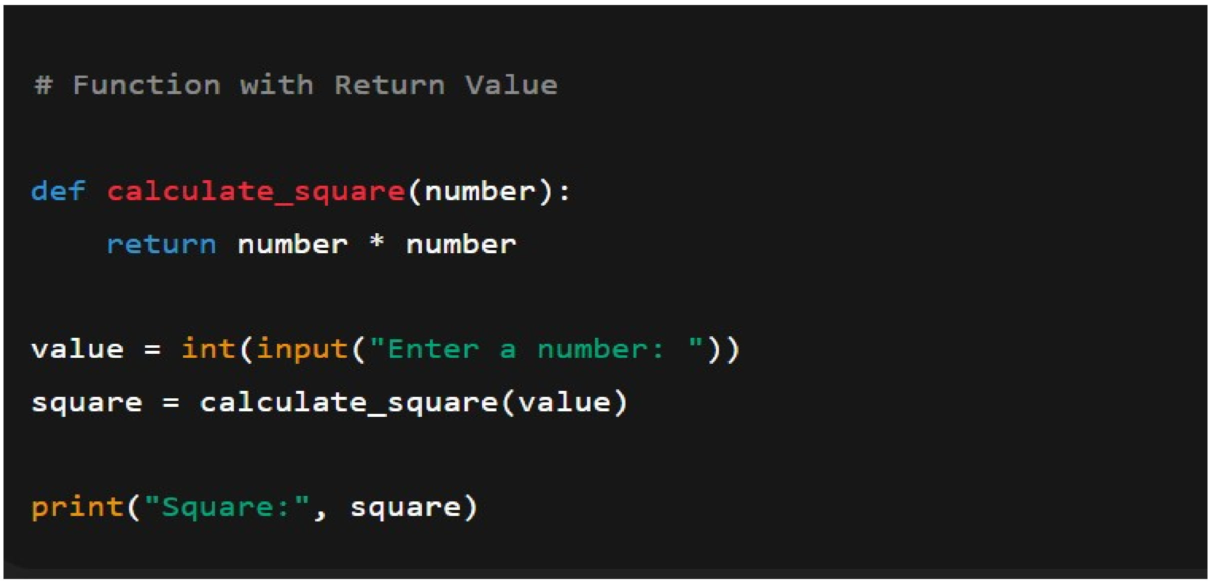
```python
# Basic Dictionary Example

student = {
    "name": "Krishnadeepsinh"
    "age": 20,
    "city": "Bhavnagar"
}

print("Student Information:")
print("Name :", student["name"])
print("Age  :", student["age"])
print("City :", student["city"])
```

Functions in Python

- A function is a reusable block of code designed to perform a specific task. Functions help organize programs into manageable components, improving readability, maintainability, and efficiency. Instead of rewriting the same logic multiple times, developers can define a function once and call it whenever needed.
- Functions play a crucial role in modular programming. They enable the separation of complex problems into smaller, structured units. This approach simplifies debugging, enhances code clarity, and promotes

reusability. Python functions are defined using the def keyword followed by the function name and parameters.

- One of the major advantages of functions is code optimization. By encapsulating logic within functions, programs become more structured and easier to understand. Functions also support parameter passing, allowing dynamic input values, which increases flexibility.
- Functions improve collaboration in software development, as clearly defined functional blocks make code easier to read and modify. They are widely used in real-world applications for calculations, validations, data processing, automation tasks, and system operations.
- Overall, functions contribute significantly to writing efficient, scalable, and professional-quality programs.



```python
# Function with Return Value

def calculate_square(number):
    return number * number

value = int(input("Enter a number: "))
square = calculate_square(value)

print("Square:", square)
```

OBJECTIVES & GOALS (Achieved vs. Pending)

Primary objectives:

- Master Python basics (variables, data structures, control statements, loops, functions)
- Develop simple scripts for problem-solving
- Explore libraries for text- to-speech and web content fetching

Achieved so far:

- Conditional logic (profit/loss, leap year, shape classification)
- Loop-based programs (series, patterns, prime number checks)
- Functions (default, arbitrary, keyword, recursive)
- Web scraping + TTS integration (Rashi/Horoscope reader project)

Pending:

- Object-oriented programming
- File handling in complex scenarios
- Advanced library integration & automation scripts

TECHNICAL STACK

- Core Python Basics
- Built-in data structures: list, tuple, dict, set
- Libraries used: - requests → HTTP requests - BeautifulSoup → HTML parsing - gTTS → Google Text-to-Speech (Gujarati support) - pygame → Audio playback - datetime → Date handling
- No advanced frameworks (Django/Flask) or data science libraries (Pandas/NumPy) were used in this phase.

COMPLETED TASKS / MILESTONES

- Basic arithmetic & decision-making scripts (profit/loss, cheaper product per gram)
- Series generation & pattern printing using loops
- Number-to-words conversion (up to 3 digits)
- Prime number check (optimized with early exit)
- Recursion (reverse number printing)
- Data structure operations (vowel count, positive filter, dictionary queries)
- Advanced project: **Rashi Bhavishya Reader** - Fetches daily Gujarati horoscope from website - Parses content using BeautifulSoup - Converts text to Gujarati speech using gTTS -

Plays audio automatically using pygame - Includes simplified Chandra Rashi calculation based on DOB

CHALLENGES & LEARNINGS

Challenges faced:

- Inconsistent HTML structure on target website → solved with flexible selectors & error handling
- Audio playback timing issues → resolved using pygame mixer loop & sleep
- Understanding library documentation quickly

Learnings:

- Importance of try-except blocks for robust code
- Modular design improves debugging
- Real-world application of web scraping + multimedia in Python

CODE QUALITY & VERSION CONTROL

- Followed basic PEP 8 style (indentation, naming, comments)
- Used docstrings in major programs
- Plan to use Git in next phase

FUTURE PLAN

Remaining goals:

- Learn & implement OOP concepts in mini-projects
- Create automation scripts (file batch processing, etc.)
- Optimize existing code (e.g., prime check efficiency)
- Add file/database storage to Rashi reader
- Prepare final project documentation & presentation

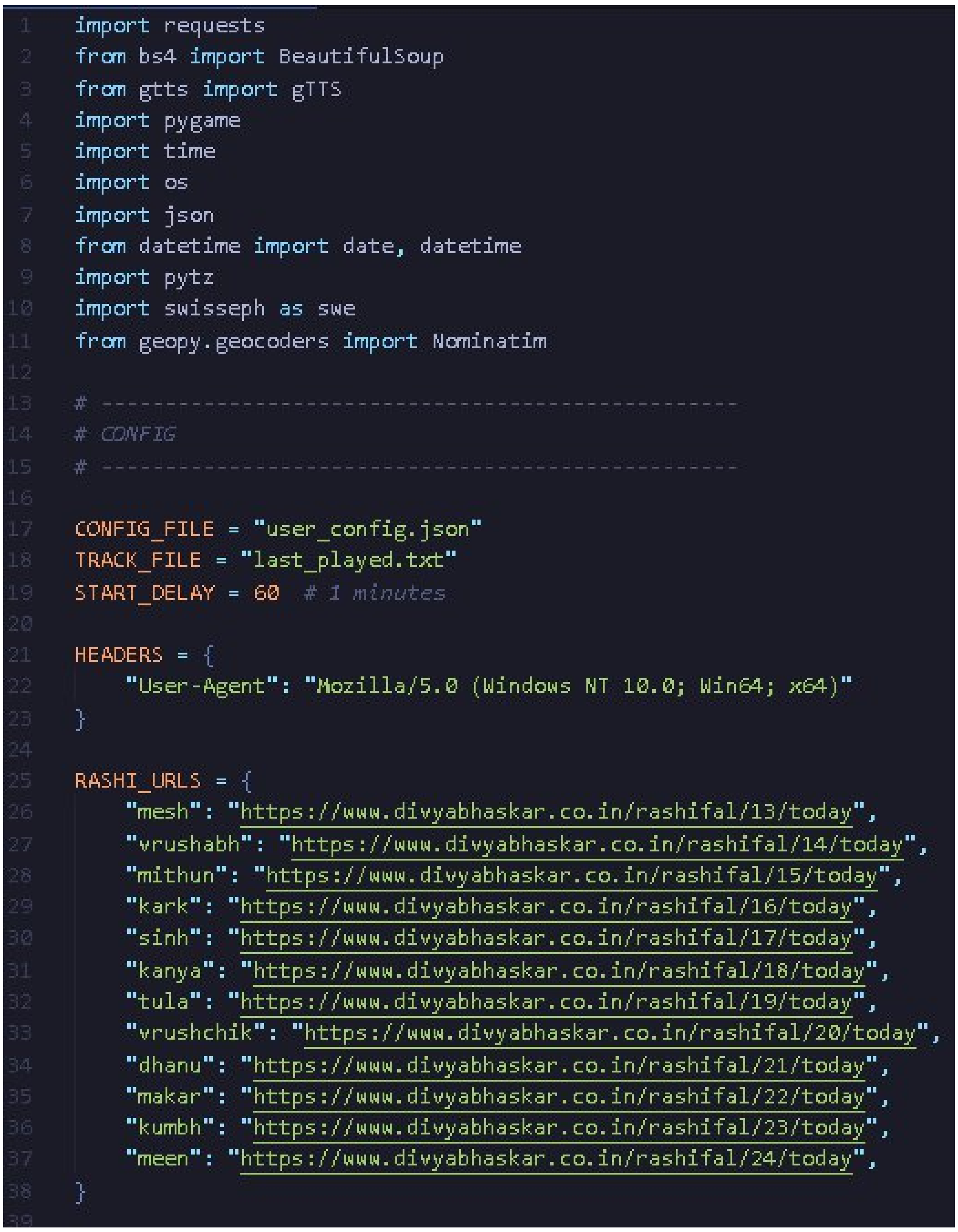
Project Main Points

Objective:

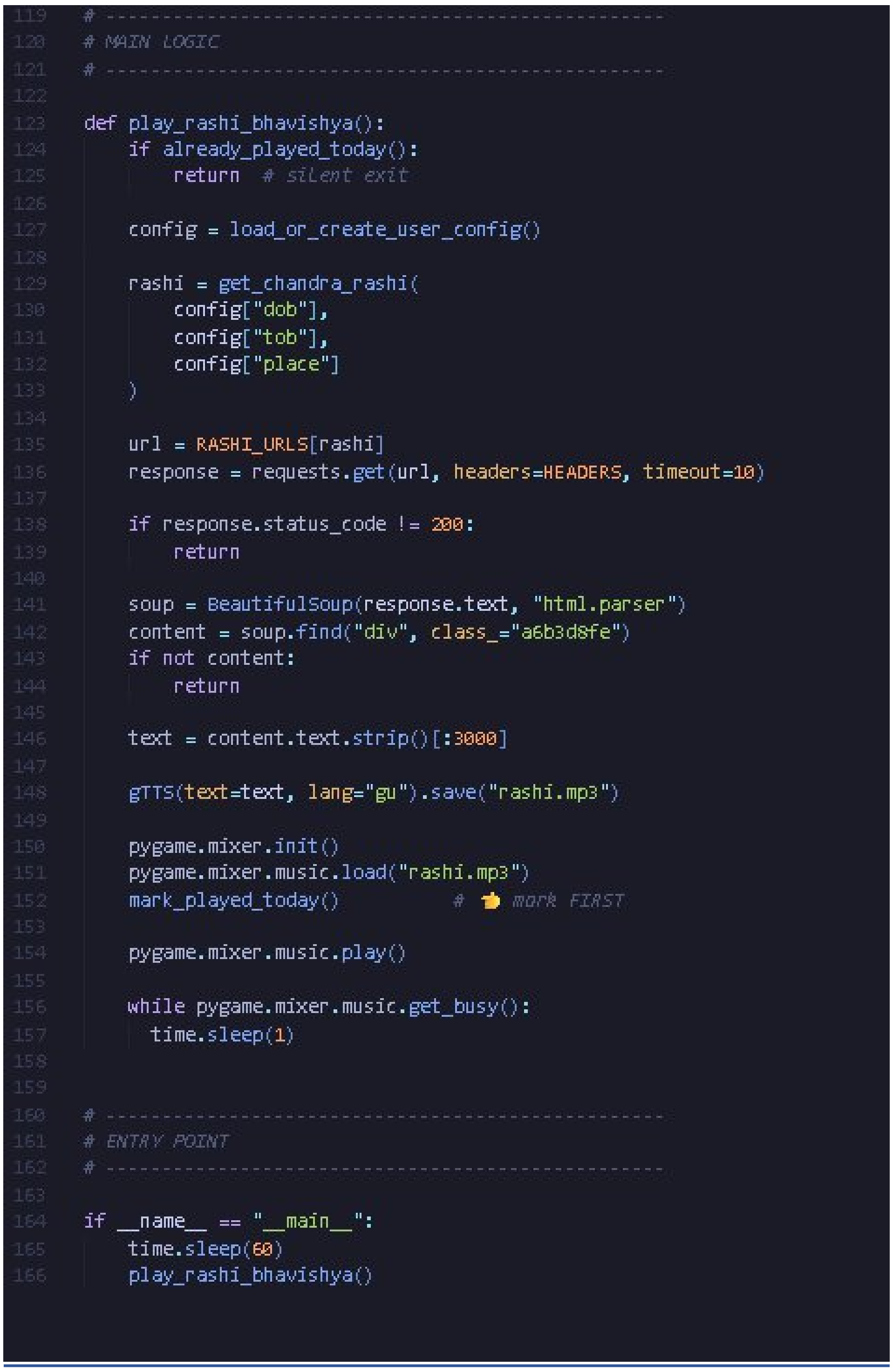
- To develop an automated Python-based system that fetches daily *Rashi Bhavishya* (horoscope) data from the Divya Bhaskar website, processes the content, and presents it in a structured, readable format for end users.

Python Implementation (Basic Automated Scraper)

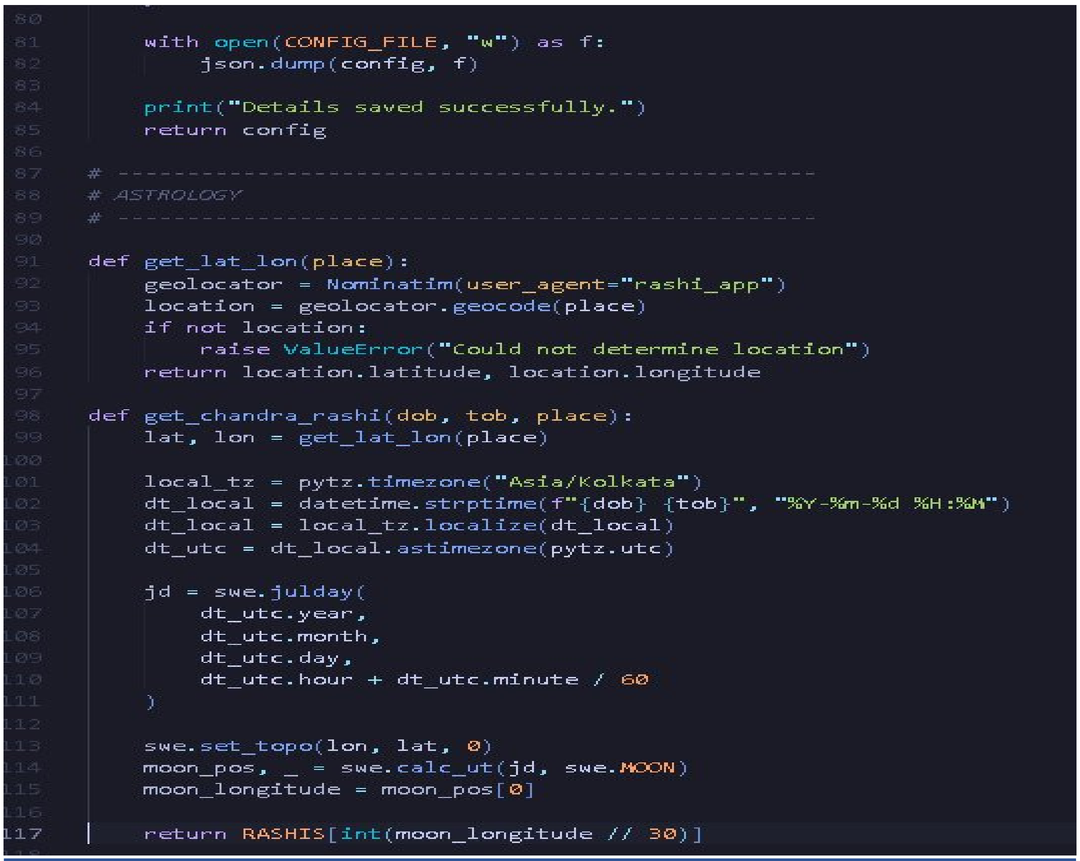
- This Python-based automation project dynamically delivers daily *Rashi Bhavishya* predictions by determining the user's *Chandra Rashi* through astronomical calculations and web data extraction. On first execution, the system securely stores the user's birth date, time, and place, then computes geographic coordinates using geolocation services and calculates the Moon's longitude via the Swiss Ephemeris library to identify the correct zodiac sign. Based on this computed rashi, the program automatically fetches the corresponding daily horoscope from the Divya Bhaskar website using web scraping techniques. The extracted Gujarati text is then converted into speech using Google Text-to-Speech and played through an audio module, ensuring a hands-free experience. A date-tracking mechanism prevents repeated playback within the same day, making the system efficient, automated, and user-personalized..
- Project code :



```python
import requests
from bs4 import BeautifulSoup
from gtts import gTTS
import pygame
import time
import os
import json
from datetime import date, datetime
import pytz
import swisseph as swe
from geopy.geocoders import Nominatim

# ------------------------------------------------
# CONFIG
# ------------------------------------------------

CONFIG_FILE = "user_config.json"
TRACK_FILE = "last_played.txt"
START_DELAY = 60  # 1 minutes

HEADERS = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
}

RASHI_URLS = {
    "mesh": "https://www.divyabhaskar.co.in/rashifal/13/today",
    "vrushabh": "https://www.divyabhaskar.co.in/rashifal/14/today",
    "mithun": "https://www.divyabhaskar.co.in/rashifal/15/today",
    "kark": "https://www.divyabhaskar.co.in/rashifal/16/today",
    "sinh": "https://www.divyabhaskar.co.in/rashifal/17/today",
    "kanya": "https://www.divyabhaskar.co.in/rashifal/18/today",
    "tula": "https://www.divyabhaskar.co.in/rashifal/19/today",
    "vrushchik": "https://www.divyabhaskar.co.in/rashifal/20/today",
    "dhanu": "https://www.divyabhaskar.co.in/rashifal/21/today",
    "makar": "https://www.divyabhaskar.co.in/rashifal/22/today",
    "kumbh": "https://www.divyabhaskar.co.in/rashifal/23/today",
    "meen": "https://www.divyabhaskar.co.in/rashifal/24/today",
}
```



```python
# --------------------------------------------------
# MAIN LOGIC
# --------------------------------------------------

def play_rashi_bhavishya():
    if already_played_today():
        return  # silent exit

    config = load_or_create_user_config()

    rashi = get_chandra_rashi(
        config["dob"],
        config["tob"],
        config["place"]
    )

    url = RASHI_URLS[rashi]
    response = requests.get(url, headers=HEADERS, timeout=10)

    if response.status_code != 200:
        return

    soup = BeautifulSoup(response.text, "html.parser")
    content = soup.find("div", class_="a6b3d8fe")
    if not content:
        return

    text = content.text.strip()[:3000]

    gTTS(text=text, lang="gu").save("rashi.mp3")

    pygame.mixer.init()
    pygame.mixer.music.load("rashi.mp3")
    mark_played_today()          # 👈 mark FIRST

    pygame.mixer.music.play()

    while pygame.mixer.music.get_busy():
        time.sleep(1)


# --------------------------------------------------
# ENTRY POINT
# --------------------------------------------------

if __name__ == "__main__":
    time.sleep(60)
    play_rashi_bhavishya()
```



```python

    with open(CONFIG_FILE, "w") as f:
        json.dump(config, f)

    print("Details saved successfully.")
    return config

# ------------------------------------------------
# ASTROLOGY
# ------------------------------------------------

def get_lat_lon(place):
    geolocator = Nominatim(user_agent="rashi_app")
    location = geolocator.geocode(place)
    if not location:
        raise ValueError("Could not determine location")
    return location.latitude, location.longitude

def get_chandra_rashi(dob, tob, place):
    lat, lon = get_lat_lon(place)

    local_tz = pytz.timezone("Asia/Kolkata")
    dt_local = datetime.strptime(f"{dob} {tob}", "%Y-%m-%d %H:%M")
    dt_local = local_tz.localize(dt_local)
    dt_utc = dt_local.astimezone(pytz.utc)

    jd = swe.julday(
        dt_utc.year,
        dt_utc.month,
        dt_utc.day,
        dt_utc.hour + dt_utc.minute / 60
    )

    swe.set_topo(lon, lat, 0)
    moon_pos, _ = swe.calc_ut(jd, swe.MOON)
    moon_longitude = moon_pos[0]

    return RASHIS[int(moon_longitude // 30)]
```

Conclusion

- The successful completion of the Python-based internship marks a significant milestone in the development of programming proficiency and computational thinking. The internship offered a comprehensive understanding of Python’s core principles, emphasizing clarity, efficiency, and structured problem-solving techniques.
- The learning journey involved mastering essential programming constructs, including decision-making logic, iterative execution, data manipulation, and functional abstraction. Exposure to Python’s flexible syntax and powerful built-in capabilities fostered a deeper appreciation for modern software development methodologies.
- This experience not only strengthened technical competence but also cultivated analytical reasoning, debugging skills, and

systematic program design approaches. The knowledge and skills acquired during the internship establish a solid platform for exploring advanced technological domains such as software engineering, automation, data science, and artificial intelligence.

- In conclusion, the internship proved to be an enriching academic and practical experience, enhancing confidence, technical readiness, and professional growth.

- This mid-term phase has significantly improved my Python proficiency and confidence in applying concepts to practical problems. The Rashi reader project stands out as a meaningful integration of multiple skills. I look forward to building more advanced solutions and contributing effectively in the remaining internship period.

Submitted by: Chudasama Krishnadeepsinh S
BCA Semester-6
Roll No. 2324152