Bi-Lingual Programming:
C and Python for Modern Coders

Dr. B. Swapna
Mrs. A. Maheswari
Mrs. Chinchu Nair
Mr. R. Balaji

www.jpc.in.net

Bi-Lingual Programming: C and Python for Modern Coders

Author:
Dr. B. Swapna
Mrs. A. Maheswari
Mrs. Chinchu Nair
Mr. R. Balaji

First Published: June 2023

ISBN: 978-93-91303-63-1
DOI: https://doi.org/10.47715/JPC.B.978-93-91303-63-1
Pages: 122 (Front pages 16 & Inner pages 106)
Price: 300/-

Publisher:
Jupiter Publications Consortium
22/102, Second Street, Virugambakkam
Chennai, Tamil Nadu, India.
Website: www.jpc.in.net
Email: director@jpc.in.net

Imprint:
Magestic Technology Solutions (P) Ltd.
Chennai, Tamil Nadu, India.

TITLE VERSO

Title of the Book:
Bi-Lingual Programming: C and Python for Modern Coders

Author's Name:
Dr. B. Swapna
Mrs. A. Maheswari
Mrs. Chinchu Nair
Mr. R. Balaji

Published By:
Jupiter Publications Consortium
Publisher's Address:
22/102, Second Street, Venkatesa Nagar, Virugambakkam
Chennai, Tamil Nadu – 600 092

Printer's Details:
Magestic Technology Solutions (P) Ltd.

Edition Details: First Edition

ISBN: 978-93-91303-63-1

COPYRIGHT

Jupiter Publications Consortium
22/102, Second Street, Virugambakkam
Chennai 600 092. Tamil Nadu. India

Imprint Magestic Technology Solutions (P) Ltd

Printed on acid-free paper
International Standard Book Number (ISBN): 978-93-91303-63-1 (Paperback)
Digital Object Identifier (DOI): 10.47715/JPC.B.978-93-91303-63-1

Visit the Jupiter Publications Consortium Web site at
http://www.jpc.in.net

DEDICATION

Er. A C S. ARUNKUMAR
B.Tech (Hons)., LMISTE., MIET.,(UK)., LMCSI.,
President
Dr. M. G. R. Educational and Research Institute
Chennai, Tamil Nadu, India.

- We take immense pride and heartfelt reverence in dedicating this book to **Er. A C S. Arunkumar, B.Tech (Hons)., LMISTE., MIET.,(UK)., LMCSI.,** who holds the esteemed position of President at our illustrious Dr. M. G. R. Educational and Research Institute, located in the culturally vibrant city of Chennai, Tamil Nadu, India.

Our President's unwavering devotion to cultivating academic excellence and fostering the expansion of knowledge is a testament to his global vision. His educational philosophy not only stimulates us but is a beacon that has helped light the path towards academic and personal growth for countless students, leaving an indelible impact on the landscape of academia.

Our gratitude for our President's leadership is profound, as his guidance persistently propels us to strive for pinnacle of excellence in all aspects of our

pursuits. It is more than an honour; it is indeed a privilege to dedicate this book to such a luminary a tangible expression of our respect, admiration, and appreciation.

We extend our deepest gratitude to you, sir, for your extraordinary contributions to the field of education, and for ceaselessly inspiring us all with your visionary leadership. Your legacy, like this book, shall serve as a beacon of inspiration for future generations.

Dr. B. Swapna
Mrs. A. Maheswari
Mrs. Chinchu Nair
Mr. R. Balaji
(Authors)

ACKNOWLEDGEMENT

We express our profound gratitude and reverence to the Almighty whose abundant grace has been our guiding light, leading us to the successful completion of this book.

We are very much grateful to our esteemed Founder and Chancellor **Thiru. A.C. Shanmugam** B.A, B.L, FRCPS (Glasgow). His unwavering support and inspiration have been invaluable to our efforts. We also convey our deepest gratitude to our young dynamic President Er. A.C.S. Arun Kumar B.Tech (Hons), LMISTE, MIET (UK), LMCSI, whose encouragement and assistance have been instrumental in the successful publication of this book.

We extend our earnest thanks to Vice Chancellor, **Dr. S. Geethalakshmi**, Pro Vice Chancellor **Dr. S. Ravichandran**, Registrar **Dr. C. B. Palanivelu**, Provost **Dr. G. Gopalakrishnan**, and **Prof. Dr. D. Viswanathan**, Rector - Research and Development. Their perpetual inspiration, motivation, and encouragement have been of immense significance to our endeavour. Special acknowledgements are due to Joint Registrar (FY E&T) **Dr. N. S. Shubhashree**, Dean (E&T) **Dr. N. Ethiraj** and HOD/CSE **Dr. S. Geetha**. Their unwavering motivation, encouragement, and support have been pivotal in transforming our hard work into a successful outcome.

We seize this opportunity to express our thanks to all our Joint Registrars, Deans, and Heads of various departments for their unflinching support. We extend our thanks to the entire teaching and non-teaching staff and our students, whose constant support and encouragement have played a crucial role in our journey.

- AUTHORS

This Page Intentionally Left Blank

PREFACE

In the dazzling cosmos of software development and programming, there are languages that define the fundamental stratum and languages that pave the way for future innovations. The synchronicity of understanding both types of languages equip one with a comprehensive understanding of the coding universe. "Bi-Lingual Programming: C and Python for Modern Coders" is a meticulously designed endeavour that seeks to unravel the intricacies of these two vital languages – the stalwart C, and the dynamic Python.

C, the timeless classic, forms the bedrock of programming. It is a language that has been shaping the digital world for decades, its low-level capabilities making it a powerful tool in the right hands. Python, on the other hand, represents the ever-evolving landscape of modern programming. Its flexibility, ease of learning, and wide range of applications makes it one of the most popular languages among new and seasoned programmers alike.

This book is a collaborative endeavour of Dr. B. Swapna, Mrs. A. Maheswari, Mrs. Chinchu Nair, and Mr. R. Balaji. It brings together our diverse experiences and deep-rooted passion for coding, pedagogy, and knowledge dissemination. The aim is to facilitate an enriching learning experience that explores these languages in detail, elucidating their fundamental aspects, advanced capabilities, and unique applications.

"Bi-Lingual Programming: C and Python for Modern Coders" presents a well-structured, topic-wise layout, making it easy to follow and digest. The first two units delve into the core of C programming, exploring everything from its basic character sets and variables to intricate concepts like structures and binary I/O functions. Unit III ventures into the realm of Python, elucidating its basic concepts, data types, modularity, and I/O functionality, among others.

The journey from C's preciseness and complexity to Python's simplicity and versatility is intentionally crafted to provide learners with a panoramic understanding of both the languages. We have taken great pains to maintain a smooth and cohesive transition between concepts, punctuating the text with examples, exercises, and snippets to enhance comprehension and retention.

Whether you're a beginner starting your coding journey or an experienced coder aiming to expand your skills, this book promises to be a reliable companion. It's our hope that readers will not only gain knowledge from it, but also cultivate a deep appreciation for programming languages and their pivotal role in our increasingly digital world.

We, the authors, eagerly invite you to embark on this enriching journey of bi-lingual programming with us. Here's to unlocking a new realm of coding prowess, together!

Dr. B. Swapna
Mrs. A. Maheswari
Mrs. Chinchu Nair
Mr. R. Balaji

ABSTRACT

In the rapidly evolving landscape of software development, the understanding and effective utilization of programming languages serve as critical skills for modern coders. This book, "Bi-Lingual Programming: C and Python for Modern Coders", presents an in-depth exploration of two highly influential programming languages: the fundamental and efficient C, and the versatile and intuitive Python.

Our comprehensive guide begins by delving into the structure and intricacies of C programming, from its basic syntax and data types to more complex constructs like pointers, structures, and binary I/O functions. The focus then shifts to the user-friendly Python, uncovering its historical evolution, basic concepts, structured data types, and control statements, before advancing to its unique modularity and I/O functionality.

Both languages are approached with a balanced blend of theoretical insights and practical applications, ensuring an effective learning experience for beginners and seasoned programmers alike. By combining C's foundation-laying strengths with Python's flexibility and simplicity, the book provides readers with a thorough understanding of the programming spectrum, thereby enhancing their coding proficiency.

Keywords: Programming Languages, C Programming, Python, Binary I/O Functions, Data Types, Control Statements, Modularity, I/O Functionality, Software Development, Coding Proficiency.

This Page Intentionally Left Blank

PROLOGUE

The march of technology is relentless, and the fields of software development and data science move at a pace that can be dizzying. Yet, amidst this rapid evolution, programming languages remain our steadfast tools, our translators between human thought and digital action. Among the plethora of languages that have been developed over the years, two have consistently demonstrated their value and versatility: C and Python. This book, "Bi-Lingual Programming: C and Python for Modern Coders", is a testament to the timelessness and adaptability of these two languages.

C, a language born in the late 1960s, was revolutionary in its day, providing programmers with an unprecedented blend of high-level functionality and low-level access. Over the years, it has become a cornerstone in the world of programming, its principles serving as a foundation for numerous modern languages. In this volume, we pay homage to C's impact and delve into its fundamental constructs and nuanced features.

Python, on the other hand, represents a newer generation of programming languages. With its simple yet expressive syntax, Python has garnered a reputation for enabling quick development and being an excellent tool for beginners. However, these traits do not limit its power; Python is a force to be reckoned with in areas as diverse as web development, data analysis, artificial intelligence, and more.

The journey between these two languages is a passage through time, a voyage that connects the early days of programming to its future. This book serves as your guide on this journey, offering a comprehensive look at each language and the programming principles they embody.

This prologue, though brief, is a glimpse into the worlds we will traverse. Whether you are a seasoned programmer looking to expand your knowledge or a beginner starting from scratch, we hope this book will serve as a helpful companion, illuminating the path forward as you delve into the enduring language of C and the versatile realm of Python.

As we embark on this exploration, we are reminded of a fundamental truth in technology: while languages may come and go, the principles they embody endure. Understanding these principles is the key to becoming not just a coder, but a master of the craft. It is our hope that "Bi-Lingual Programming: C and Python for Modern Coders" will be your gateway to this mastery.

INTRODUCTION

In the grand concert of technology and digital innovation, programming languages play the role of pivotal instruments, each contributing its unique sound to the harmonious symphony of software development. Among the many instruments, two have always been at the forefront, either for their foundational importance or for their flexibility in adapting to changing paradigms - the stalwart C and the innovative Python. "Bi-Lingual Programming: C and Python for Modern Coders" is a comprehensive guide that delves deep into these two significant languages, elucidating their nuances and applications for a diverse audience of readers.

The language of C, revered for its efficiency and control, has been the foundation upon which many current high-level languages are built. It is often likened to the mother of programming languages, the understanding of which provides a sturdy grounding for any coder. In this book, we explore the basic character sets, identifiers, and data types in C, before gradually transitioning into more complex topics like operators, looping statements, arrays, functions, pointers, and structures. The structured format and thorough analysis of C will help beginners establish a strong programming base and provide a refresher for experienced programmers looking to revisit the basics.

Contrastingly, Python, with its emphasis on code readability and syntax simplicity, has swiftly emerged as the go-to language for quick prototyping, data analysis, machine learning, and web development. It represents the dynamic and evolving landscape of modern programming, offering a high degree of flexibility and wide application range. The third unit of the book is dedicated entirely to Python, starting with its history and basic concepts, moving on to structured data types, control statements, exception handling, and program flow modification. We further delve into modularity principles,

callable objects, and I/O functionality in Python, providing readers with a well-rounded understanding of this versatile language.

Our objective in crafting this book is not merely to impart knowledge about C and Python but to foster an understanding of the broader programming landscape. The book balances theoretical concepts with practical applications, enhancing learning through numerous examples, exercises, and program snippets. The seamless transition from one language to another offers a unique perspective and will equip readers with a bi-lingual programming competency. Whether you're a novice just embarking on your coding journey, a hobbyist looking to expand your coding horizons, or a seasoned programmer aiming to brush up your basics and update your knowledge, this book is designed to meet your needs. The comprehensive and detailed approach used in "Bi-Lingual Programming: C and Python for Modern Coders" ensures that it serves as a valuable resource in your programming journey.

Welcome to a bi-lingual exploration of the fascinating world of coding, where tradition meets innovation, providing a solid foundation for the programmers of today and tomorrow.

Dr. B. Swapna
Mrs. A. Maheswari
Mrs. Chinchu Nair
Mr. R. Balaji

Table of Contents

This Page is intentionally Left Blank.

UNIT – 1 INTRODUCTION

Fundamentals, C Character set, Identifiers and Keywords, Data Types, Variables and Constants, Structure of a C Program, Executing a C Program.

C programming is a general-purpose, procedural, imperative computer programming language developed in 1972 by Dennis M. Ritchie at the Bell Telephone Laboratories to develop the UNIX operating system. C is the most widely used computer language.

Key Advantages of Learning C Programming:

- Easy to learn
- Structured language
- It produces efficient programs
- It can handle low-level activities
- It can be compiled on a variety of computer platforms

Applications of C

- Operating Systems
- Language Compilers
- Assemblers
- Text Editors
- Print Spoolers
- Network Drivers
- Modern Programs
- Databases
- Language Interpreters
- Utilities

C Character Set

A character set is a set of alphabets, letters and some special characters that are valid in C language.

Alphabets

Uppercase: A B C ..... X Y Z

Lowercase: a b c .....x y z

C accepts both lowercase and uppercase alphabets as variables and functions.

Digits

0 1 2 3 4 5 6 7 8 9

Special Characters like !<> ( ) & % # $ { } etc.

Identifiers

Identifiers are names for entities in a C program, such as variables, arrays, functions, structures, unions and labels. An identifier can be composed only of uppercase, lowercase letters, underscore and digits, but should start only with an alphabet or an underscore.

Rules for naming identifiers

A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.

The first letter of an identifier should be either a letter or an underscore.

You cannot use keywords as identifiers.

There is no rule on how long an identifier can be. However, you may run into problems in some compilers if the identifier is longer than 31 characters.

No special characters or symbols are allowed.

Keywords

Keywords are predefined, reserved words used in programming that have special meanings to the compiler. Keywords are part of the syntax and they cannot be used as an identifier.

C Keywords			
Auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	volatile

Data Types

Data types in c refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The types in C can be classified as follows −

Sr.No.	Types & Description
1	**Basic Types** They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.
2	**Enumerated types** They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program.
3	**The type void** The type specifier *void* indicates that no value is available.
4	**Derived types** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

```
    print(x)
    x -= 1
```
In this example, the loop continues indefinitely with while True. However, when x becomes 0, the break statement is encountered, and it terminates the loop, immediately transferring the control to the next statement after the loop.

Example use of "pass":

```
while True:
  if x < 10:
    pass
  elif x == 10:
    # Do something with x
    print("x is 10")
  else:
    # Do something else with x
    print("x is greater than 10")
  break
```
In this example, the loop continues indefinitely with while True. When x is less than 10, the pass statement is encountered, indicating that no action is needed at that point. When x is equal to 10, the code block under the elif statement is executed. When x is greater than 10, the code block under the else statement is executed. The break statement is included to terminate the loop after one iteration.

Case Structure in Python

In Python, there is no direct "case" or "switch" statement like in some other programming languages. However, you can achieve similar functionality using various techniques. Here are a few approaches to implement a case-like structure in Python:

if-elif-else ladder:

One way to emulate a case structure is by using an if-elif-else ladder. You can use multiple if-elif statements to check different conditions and execute corresponding blocks of code.

Example:

```
x = 2
if x == 1:
   print("Case 1")
elif x == 2:
   print("Case 2")
elif x == 3:
   print("Case 3")
else:
   print("Default case")
```

In this example, each condition is checked using if-elif statements, and the corresponding block of code is executed based on the condition that evaluates to true.

Dictionary-based approach:

You can use a dictionary to map keys to functions or values. The keys represent the cases, and the associated values can be functions or values to be executed or returned.

Example:

```
def case_1():
   print("Case 1")

def case_2():
   print("Case 2")
def case_3():
   print("Case 3")
cases = {
   1: case_1,
   2: case_2,
   3: case_3
}
x = 2
if x in cases:
   cases[x]()
else:
```

```
    print("Default case")
```
In this example, functions case_1, case_2, and case_3 are defined, and a dictionary cases is created to map the cases to their corresponding functions. The value of x is used as the key to retrieve the corresponding function from the dictionary and execute it.

Using the match-case statement (Python 3.10+):

Starting from Python 3.10, a new feature called "match-case" is introduced as a more concise and expressive way to handle pattern matching. It allows you to match values against specific patterns and execute corresponding code blocks.

Example:

```
x = 2
match x:
    case 1:
        print("Case 1")
    case 2:
        print("Case 2")
    case 3:
        print("Case 3")
    case _:
        print("Default case")
```
In this example, the match statement is used to match the value of x against different cases. The code block under the matching case is executed, and if none of the cases match, the _ case acts as the default case.

These are a few approaches to implement a case-like structure in Python. Each technique has its advantages and can be used based on your specific requirements and the version of Python you are working with.

Modularity

Modularity is a fundamental aspect of Python programming, allowing you to break down your code into smaller, manageable modules. Python provides several mechanisms to achieve modularity, which include:

Modules:

Python modules are files that contain Python code. They serve as a way to organize related code and provide a means to reuse functionality across different programs. Modules can be imported into other modules or scripts using the import statement. By importing a module, you gain access to its functions, classes, and variables.

Packages:

Packages are a way to organize related modules into a hierarchical directory structure. A package is simply a directory that contains an __init__.py file, indicating that it is a package. Packages allow for a more organized and structured approach to modularity in larger projects. They can be nested to create a hierarchy of modules.

Functions:

Functions are a way to encapsulate a block of code that performs a specific task. By defining functions, you can break down your program logic into reusable units. Functions help in achieving modularity by promoting code reuse, separation of concerns, and encapsulation of functionality.

Classes:

Classes provide a blueprint for creating objects that have attributes (variables) and methods (functions). They encapsulate related data and behavior into a single entity. Classes are instrumental in achieving object-oriented modularity, allowing for code reuse, encapsulation, and separation of concerns.

Libraries and Frameworks:

Python has an extensive standard library and a vast ecosystem of third-party libraries and frameworks. These libraries and frameworks provide pre-built modules and functions that you can leverage to achieve modularity in your programs. They offer a wide range of functionality for various domains, allowing you to focus on your specific problem without reinventing the wheel.

Modularity Principles:

Beyond the language constructs and features, adhering to modularity principles is essential in Python programming. Following principles such as the Single Responsibility Principle (SRP) and Separation of Concerns (SoC) can guide you in structuring your code in a modular fashion.

By utilizing modules, packages, functions, and classes effectively, along with the principles of modularity, you can create Python programs that are modular, reusable, maintainable, and scalable. Modularity in Python promotes code organization, separation of concerns, and collaboration among developers, leading to cleaner and more efficient codebases.

Callable Objects

In Python, a callable object is any object that can be called as a function. This includes user-defined functions, built-in functions, class objects, class instance methods, and class instances. When a callable object is called, it is invoked with a series of arguments (if any) and returns a value (unless an exception is raised).

User-Defined Functions:

User-defined functions are created using the def keyword, followed by the function name, a list of arguments (optional), and a block of executable statements. They can also include a return statement to specify the value to be returned.

```
def demo_function(message):
    print(message)
    return 1
```

Classes:

Classes in Python can also be callable objects. They define a blueprint for creating objects that have attributes and methods. Class methods are defined within the class using the def keyword, just like user-defined functions. The first argument of a class method is conventionally named self, which refers to the instance of the class.

Example:

```
class MyClass:
    def __init__(self, width):
        self.width = width
        self.inputExpected = True
    def to_string(self):
        return "Input is expected " + str(self.inputExpected)
```

In the above example, MyClass is a class with an __init__ method (constructor) and a to_string method. The self argument represents the instance of the class.

Invocation of Callable Objects:

When calling a callable object, the parameters are passed as arguments. The callable object is invoked by using parentheses () after its name, followed by the argument values. The return value of the callable object is the result of the call, which includes None unless an exception is raised.

Example:

```
result = demo_function("Hello")
print(result)  # Output: 1
obj = MyClass(10)
obj_str = obj.to_string()
print(obj_str)  # Output: "Input is expected True"
```

In the above example, the demo_function and to_string methods are called by using parentheses () and providing the required arguments.

Default and Keyword Parameters:

Python supports default parameters and keyword parameters. Default parameters are values assigned to function or method arguments when they are defined. Keyword parameters allow you to pass arguments using the parameter name, which provides more clarity and flexibility in function calls.

Example:

```
def multiply(a, b=1):
    return a * b

result1 = multiply(5)         # Uses default value for b
result2 = multiply(5, b=3)    # Uses keyword parameter for b
```

In the above example, the multiply function has a default parameter b assigned to 1. The function can be called without providing a value for b, in which case it uses the default value. Alternatively, a keyword argument can be passed to explicitly specify the value of b.
Callable objects in Python are versatile and allow you to encapsulate functionality and perform operations by invoking them as functions. They provide a flexible way to define reusable code and enable code organization and encapsulation within classes and functions.

IO Functionality:

Input and output (IO) functionality is crucial in programming as it enables communication between the program and the user or external data sources. In Python, IO operations can be performed through console IO and file IO. Let's explore each of them:

Console I/O:

Input:

Input from the console is accomplished using the raw_input() function (in Python 2) or the input() function (in Python 3).
raw_input() takes an optional prompt as its parameter and waits for the user to enter input. The entered value is returned as a string.

Example:

```
data = raw_input("Please enter something: ")
```

Output:

Output to the console is achieved using the print statement (in Python 2) or the print() function (in Python 3).
print accepts one or more arguments and displays them as output. By default, it adds a newline character after the last argument.

Example:

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

File I/O:

Opening a File:

File objects are created by using the open() function, which takes two parameters: the filename and the mode.

The mode specifies the purpose of opening the file, such as reading, writing, appending, or both reading and writing.

Example:

```
file = open("example.txt", "r")
```

Reading from a File:

File objects support various methods for reading data, such as read(), readline(), and readlines().

read() reads the entire content of the file as a string or a specified number of bytes.

readline() reads a single line from the file.

readlines() reads all lines of the file and returns them as a list of strings.

Example:

```
content = file.read()
line = file.readline()
lines = file.readlines()
```

Writing to a File:

File objects provide the write() method to write data to a file. It takes a string parameter and returns None.

Example:

```
file.write("Hello, World!")
```

File Position and Seeking:

File objects keep track of the current position within the file using the tell() method, which returns the current position in bytes.

The seek() method is used to change the file object's current position. It takes two parameters: the offset and the from_what argument, which specifies the reference position (0 for the beginning, 1 for the current position, and 2 for the end).

Example:
position = file.tell()
file.seek(0, 0) # Move to the beginning of the file

Closing a File:

It's important to close a file after you are done with it to release system resources. The close() method is used to close a file.
Example:
file.close()
The combination of console IO and file IO capabilities in Python allows you to interact with users, read from external data sources, and write data to files. These IO functionalities are vital for building interactive and data-driven applications. Remember to handle exceptions and close files properly to ensure efficient and reliable IO operations.

Python Programs with Output

Hello, World!:
print("Hello, World!")
Output:
Hello, World!

Sum of Two Numbers:
num1 = 5
num2 = 10
sum = num1 + num2
print("The sum is:", sum)
Output:
The sum is: 15

Factorial Calculation:
def factorial(n):

```
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
num = 5
result = factorial(num)
print("The factorial of", num, "is", result)
```

Output:

The factorial of 5 is 120

Fibonacci Series:

```
def fibonacci(n):
    sequence = [0, 1]
    while len(sequence) < n:
        next_num = sequence[-1] + sequence[-2]
        sequence.append(next_num)
    return sequence
num = 10
fib_sequence = fibonacci(num)
print("The Fibonacci sequence up to", num, "terms is:", fib_sequence)
```

Output:

vbnet

The Fibonacci sequence up to 10 terms is: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

Guessing Game:

```
import random
target_num = random.randint(1, 100)
guesses_taken = 0
while True:
    guess = int(input("Take a guess: "))
    guesses_taken += 1
    if guess < target_num:
        print("Too low!")
    elif guess > target_num:
```

```
        print("Too high!")
    else:
        print("Congratulations! You guessed the number in", guesses_taken, "guesses.")
        break
```

Output (example interaction):

```
Take a guess: 50
Too high!
Take a guess: 30
Too low!
Take a guess: 40
Congratulations! You guessed the number in 3 guesses.
```

Prime Number Checker:

```
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
num = 17
if is_prime(num):
    print(num, "is a prime number.")
else:
    print(num, "is not a prime number.")
```

Output:

```
17 is a prime number.
```

Reverse a String:

```
string = "Hello, World!"
reversed_string = string[::-1]
print("Reversed string:", reversed_string)
```

Output:

Reversed string: !dlroW ,olleH

Check Leap Year:

```
year = 2024
if year % 4 == 0 and (year % 100 != 0 or year % 400 == 0):
   print(year, "is a leap year.")
else:
   print(year, "is not a leap year.")
```

Output:

2024 is a leap year.

Sum of List Elements:

```
numbers = [1, 2, 3, 4, 5]
sum = 0
for num in numbers:
   sum += num
print("The sum of the numbers is:", sum)
```

Output:

The sum of the numbers is: 15

This Page Intentionally Left Blank