Python Help Book By Coolboymannu

Unit – 1 **(6 hours)**
Introduction to Python Programming
Problem solving strategies; Structure of a Python program; Syntax and semantics; Python interpreter/shell, indentation; Executing simple programs in Python.

Unit – 2 **(12 hours)**
Creating Python Programs
Identifiers and keywords; literals, numbers, and strings; Operators and expressions; Input and output statements; control structures (conditional statements, loop control statements, break, continue and pass), Errors and exception handling.

Unit – 3 **(9 hours)**
User Defined Functions
Defining functions, passing arguments and returning values, default arguments

Unit – 4 **(18 hours)**
Built-in Data Structures
Strings, Lists, Tuples, Sets, Dictionaries; their built-in functions, operators and operations

Essential Readings

1. Kamthane, A. N., & Kamthane, A.A. Programming and Problem Solving with Python, McGraw Hill Education, 2017.
2. Balaguruswamy E. “Introduction to Computing and Problem Solving using Python”,2nd edition, McGraw Hill Education, 2018.
3. Taneja, S., Kumar, N. Python Programming- A modular Approach, Pearson Education India, 2018.

Certainly! Below is an enhanced and more understandable version of the syllabus for DSC-2 (Programming Fundamentals Using Python):

Unit-1: Introduction to Python Programming

1.1 Problem-Solving Strategies:

- Defining Problem-Solving in Python
- Key Strategies for Effective Problem-Solving
    - Breaking Down Problems
    - Understanding Requirements
    - Pseudocode as a Planning Tool

1.2 Structure of a Python Program:

- Importance of a Well-Structured Program

- Components of a Python Program
- Syntax and Semantics
- Python Interpreter/Shell and Its Role
- Proper Indentation and Its Significance
- Executing Simple Programs in Python

Unit-2: Creating Python Programs

2.1 Identifiers and Keywords:

- Understanding Identifiers
- Significance of Keywords in Python

2.2 Literals, Numbers, and Strings:

- Overview of Literals
- Exploration of Numbers and Strings

2.3 Operators and Expressions:

- Introduction to Operators
- Understanding Expressions

2.4 Input and Output Statements:

- Usage of Input and Output Statements
- Interactive Examples for Clarity

2.5 Control Structures:

- Conditional Statements (if, else, elif)
- Loop Control Statements (for, while)
- Break, Continue, and Pass Statements
- Errors and Exception Handling Introduction

Unit-3: User Defined Functions

3.1 Defining Functions:

- Importance and Role of Functions
- Syntax and Structure

3.2 Passing Arguments and Returning Values:

- Understanding Function Parameters
- Return Values and Their Significance

3.3 Default Arguments:

- Explanation and Usage

Unit-4: Built-in Data Structures

4.1 Overview of Data Structures:

- Strings, Lists, Tuples, Sets, Dictionaries

4.2 Functions and Operators for Each Data Structure:

- Exploring Built-in Functions and Operators
- Practical Examples of Operations

Short Questions and answers

Frequent Questions

1. **What is the significance of data structures in Python programming?**
2. **Differentiate between lists, tuples, dictionaries, and sets in Python.**

3. **Define Python and list three key properties that make it popular.**
4. **How does Python achieve readability in its code?**
5. **Explain the concept of dynamic typing in Python.**
6. **What are the key operations that can be performed on strings in Python?**
7. **Discuss the immutability concept in Python strings.**
8. **Define control structures in Python and provide an example.**
9. **What is the purpose of looping statements in Python?**
10. **Differentiate between for and while loops in Python with examples.**
11. **What are conditional statements in Python? Provide examples.**
12. **How can errors and exceptions be handled in Python programming?**
13. **Explain the try, except, and finally blocks in Python exception handling.**
14. **Provide the syntax for defining a function in Python.**
15. **How many types of arguments can be used in a Python function, and what are they?**
16. **Enumerate and explain the types of operators in Python.**
17. **Define a list in Python and provide an example.**
18. **Explain the characteristics of sets in Python and their use cases.**
19. **Define a tuple in Python and explain its properties.**
20. **How can you access the key and value in a Python dictionary?**

PYQs Link click here

Answers

1. Significance of Data Structures:
    - Data structures in Python organize and store data efficiently, improving code performance and readability.
2. Differentiate Lists, Tuples, Dictionaries, and Sets:
    - Lists are mutable ordered sequences, tuples are immutable ordered sequences, dictionaries are mutable key-value pairs, and sets are mutable, unordered collections of unique elements.
3. Define Python and Its Key Properties:
    - Python is a high-level, interpreted programming language known for readability, simplicity, and versatility.

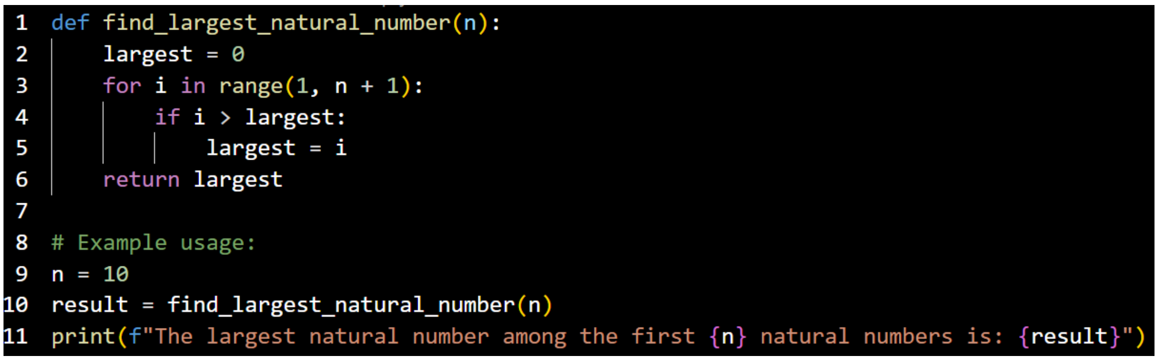
4. Python's Code Readability:
    - Python achieves code readability through indentation and a clean, easy-to-understand syntax.
5. Dynamic Typing in Python:
    - Dynamic typing allows variables to change types during runtime, offering flexibility but requiring careful consideration.
6. Key Operations on Strings:
    - String operations include concatenation, slicing, indexing, and various methods like `len()`, `upper()`, `lower()`.
7. Immutability Concept in Python Strings:
    - Strings in Python are immutable, meaning their values cannot be changed once assigned.
8. Control Structures in Python:
    - Control structures manage program flow. For example, `if` statements execute code based on conditions.
9. Purpose of Looping Statements:
    - Looping statements allow the execution of a block of code repeatedly, making tasks more efficient.
10. Difference Between For and While Loops:
    - `for` loops iterate over a sequence, and `while` loops execute as long as a condition is true.
11. Conditional Statements in Python:
    - Conditional statements (if, else, elif) execute code blocks based on specified conditions.
12. Handling Errors and Exceptions:
    - Errors and exceptions are handled using `try`, `except`, and `finally` blocks to manage unexpected situations gracefully.
13. Try, Except, and Finally Blocks in Exception Handling:
    - `try` contains code that might raise an exception, `except` handles the exception, and `finally` contains code executed regardless of exceptions.
14. Syntax for Defining a Function in Python:
    - Functions are defined using the syntax `def function_name(parameters):`.
15. Types of Arguments in Python Functions:
    - Three types of arguments: positional, keyword, and default arguments.
16. Types of Operators in Python:
    - Arithmetic, Comparison, Logical, Assignment, Bitwise, Membership, Identity operators.
17. Defining a List in Python:
    - Lists are defined using square brackets, e.g., `my_list = [1, 2, 3]`.
18. Characteristics of Sets in Python:

- Sets are unordered, mutable collections of unique elements, useful for tasks requiring unique values and set operations.

19. Defining a Tuple in Python:
    - Tuples are defined using parentheses, e.g., `my_tuple = (1, 2, 3)`.
20. Accessing Key and Value in a Python Dictionary:
    - Dictionary keys and values can be accessed using square bracket notation.

Some important Programs

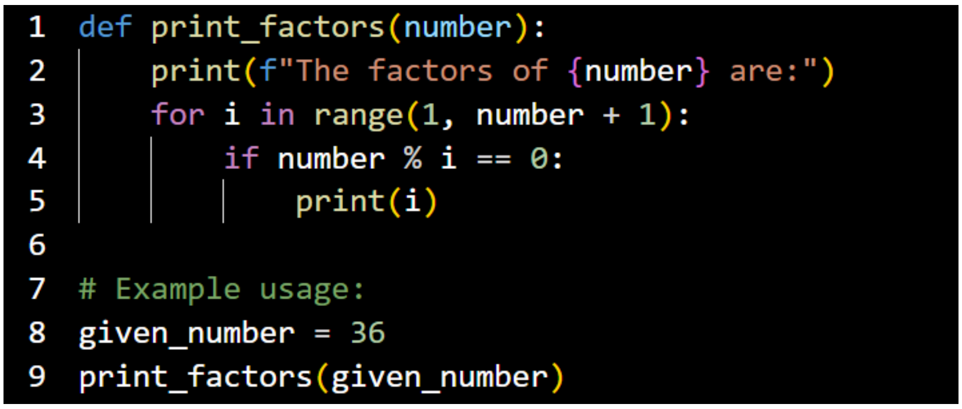
1. Find the largest of n natural numbers:



```python
def find_largest_natural_number(n):
    largest = 0
    for i in range(1, n + 1):
        if i > largest:
            largest = i
    return largest

# Example usage:
n = 10
result = find_largest_natural_number(n)
print(f"The largest natural number among the first {n} natural numbers is: {result}")
```

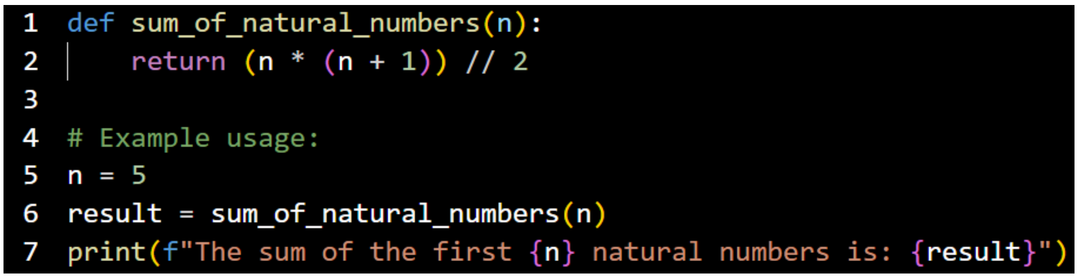
2. WAP to print factors of a given number.



```python
def print_factors(number):
    print(f"The factors of {number} are:")
    for i in range(1, number + 1):
        if number % i == 0:
            print(i)

# Example usage:
given_number = 36
print_factors(given_number)
```

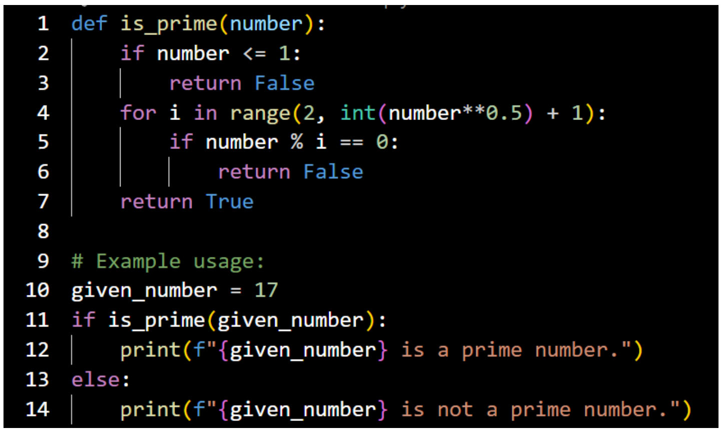
3. WAP to add N natural numbers and display their sum.



```python
def sum_of_natural_numbers(n):
    return (n * (n + 1)) // 2

# Example usage:
n = 5
result = sum_of_natural_numbers(n)
print(f"The sum of the first {n} natural numbers is: {result}")
```

4. Check whether a given number is prime or not:



```python
def is_prime(number):
    if number <= 1:
        return False
    for i in range(2, int(number**0.5) + 1):
        if number % i == 0:
            return False
    return True

# Example usage:
given_number = 17
if is_prime(given_number):
    print(f"{given_number} is a prime number.")
else:
    print(f"{given_number} is not a prime number.")
```

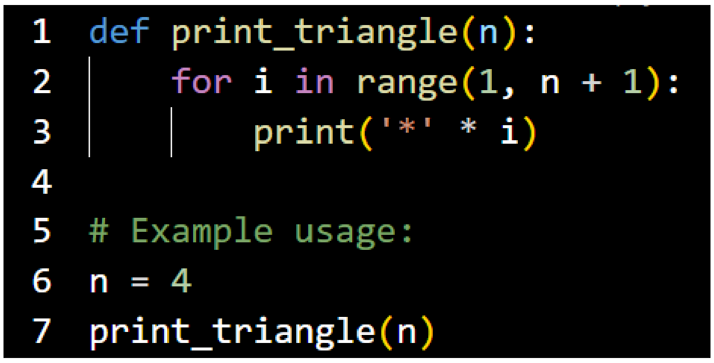
5. Write a program that takes a positive integer n and the produce n lines of output as shown:

```
*

**

***

****
```



```python
def print_triangle(n):
    for i in range(1, n + 1):
        print('*' * i)

# Example usage:
n = 4
print_triangle(n)
```

Practical Questions

1. WAP to calculate total marks, percentage and grade of a student. Marks obtained in each of three subjects are to be input by the user. Assign grades according to the following criteria: Grade A : if Percentage >=80 Grade B : if Percentage >=60 and Percentage <80 54 | Page Grade C : if Percentage >=40 and Percentage <60 Grade D : if Percentage <=40

2. Write a menu driven program using user defined functions to print the area of rectangle, square, circle and triangle by accepting suitable input from user.

3. WAP to print the series and its sum: (use functions)

   1/1! + 1/2! + 1/3! …….1/n!

4. WAP to perform the following operations on an input string
   a. Print length of the string
   b. Find frequency of a character in the string
   c. Print whether characters are in uppercase or lowercase

5. WAP to create two lists: one of even numbers and another of odd numbers. The program should demonstrate the various operations and methods on lists.

6. WAP to create a dictionary where keys are numbers between 1 and 5 and the values are the cubes of the keys.

7. . WAP to create a tuple t1 = (1,2,5,7,2,4). The program should perform the following:
   a. Print tuple in two lines, line 1 containing the first half of tuple and second line having the second half.
   b. Concatenate tuple t2 = (10,11) with t1.program to test inheritance of this class.

C++ Programming Language Overview

Unit-1: Introduction to Python Programming

1.1 Problem-Solving Strategies:

- Defining Problem-Solving in Python:
  - Introduction to the problem-solving process in the context of Python programming. Understanding how Python can be employed to address various challenges.
- Key Strategies for Effective Problem-Solving:
  - Delving into strategies such as algorithmic thinking and decomposition, which are fundamental to solving problems efficiently with Python.
- Breaking Down Problems:
  - Techniques for breaking down complex problems into smaller, more manageable parts. Discussing the importance of a systematic approach.
- Understanding Requirements:
  - Highlighting the significance of comprehending and articulating clear requirements before initiating the development of Python programs.
- Pseudocode as a Planning Tool:
  - Introducing pseudocode as a planning tool, emphasizing its role in structuring Python code before actual implementation.

1.2 Structure of a Python Program:

- Importance of a Well-Structured Program:
  - Discussing the advantages of organizing Python programs in a structured manner, including improved readability and maintainability.
- Components of a Python Program:
  - Overview of key components, such as functions, classes, and modules, contributing to the structure of a Python program.
- Syntax and Semantics:
  - Explaining the correct syntax and semantics required for writing effective Python code. Understanding the rules governing program structure.
- Python Interpreter/Shell and Its Role:
  - Shedding light on the Python interpreter and interactive shell, their roles in executing Python code, and their significance during development.
- Proper Indentation and Its Significance:

    - Emphasizing the importance of proper indentation in Python for code readability. Discussing how indentation defines the structure of the code.
- Executing Simple Programs in Python:
    - Practical exercises involving the execution of basic Python programs to reinforce theoretical concepts. Demonstrating the process of running Python code.

Unit-2: Creating Python Programs

2.1 Identifiers and Keywords:

- Understanding Identifiers:
    - Defining identifiers and their role as names assigned to variables, functions, and other entities in Python programs.
- Significance of Keywords in Python:
    - Identifying and explaining Python keywords, reserved terms with predefined meanings in the language.

2.2 Literals, Numbers, and Strings:

- Overview of Literals:
    - Explanation of literals as constant values, including numeric literals and string literals, representing data in Python.
- Exploration of Numbers and Strings:
    - In-depth exploration of numeric data types, string data types, and their respective properties and use cases in Python.

2.3 Operators and Expressions:

- Introduction to Operators:
    - Overview of operators in Python, symbols that perform operations on variables and values.
- Understanding Expressions:
    - Exploration of expressions formed by combining variables, literals, and operators. Discussing the evaluation of expressions in Python.

2.4 Input and Output Statements:

- Usage of Input and Output Statements:
    - Application of input and output statements to facilitate interaction with the user and display results in Python programs.
- Interactive Examples for Clarity:

    - Providing hands-on examples to enhance understanding of input and output statements. Engaging exercises to illustrate practical usage.

2.5 Control Structures:

- Conditional Statements (if, else, elif):
    - Introduction to conditional statements in Python, enabling the execution of code blocks based on specified conditions.
- Loop Control Statements (for, while):
    - Explanation of loop control statements, including `for` and `while` loops, for repetitive execution of code.
- Break, Continue, and Pass Statements:
    - Understanding special statements like `break`, `continue`, and `pass` within loops. Discussing their roles in modifying loop behavior.
- Errors and Exception Handling Introduction:
    - Providing an overview of errors and exception handling concepts. Introducing mechanisms for managing unexpected situations in Python programs.

Unit-3: User Defined Functions

3.1 Defining Functions:

- Importance and Role of Functions:
    - Emphasizing the significance of functions as modular and reusable units of code in Python programs.
- Syntax and Structure:
    - Defining the syntax and structure of functions in Python. Understanding the key elements of function definition.

3.2 Passing Arguments and Returning Values:

- Understanding Function Parameters:
    - Exploration of different types of function parameters, including positional parameters and keyword parameters.
- Return Values and Their Significance:
    - Discussion on the usage and significance of return values in Python functions. Understanding how functions can produce output.

3.3 Default Arguments:

- Explanation and Usage:
    - Understanding the concept and implementation of default arguments. Exploring scenarios where default values are applied to parameters.

Unit-4: Built-in Data Structures

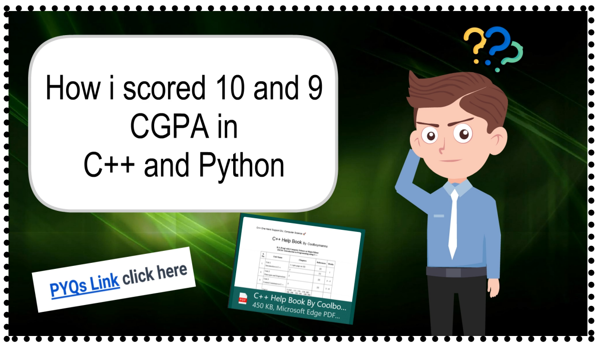
4.1 Overview of Data Structures:

- Strings, Lists, Tuples, Sets, Dictionaries:
    - Introduction to fundamental data structures in Python, including strings, lists, tuples, sets, and dictionaries. Understanding their characteristics and use cases.

4.2 Functions and Operators for Each Data Structure:

- Exploring Built-in Functions and Operators:
    - Overview of functions and operators specific to each data structure. Enhancing manipulation capabilities and providing practical tools for working with data structures.
- Practical Examples of Operations:
    - Application of built-in functions and operators through practical examples for each data structure. Hands-on exercises demonstrating operations on strings, lists, tuples, sets, and dictionaries in Python.

Strategy


How i scored 10 and 9 CGPA in C++ and Python
PYQs Link click here
C++ Help Book By Coolbo...
450 KB, Microsoft Edge PDF...

How i scored 10 and 9 CGPA in C++ and Python || #DU Ba Programme compu…

1. What is your purpose? (getting good marks , placements , learning etc.)
2. understanding syllabus (you must know all topics in syllabus)
3. watching a full video of C++ or Python other video
4. solving PYQs
5. ready short answers and some important codes
6. self test
7. How to attempt paper
    1. take 1and half margin in paper
    2. there are 7 questions 1st is compulsory 2 to 7 answer only 4 questions
    3. look up questions serially (conscious mind)
    4. attempting questions depend on you

Best of luck