PPS SAMPLE PAPER -1

(ANSWER KEY)

1. What is the first step in the problem-solving process?

(A) Generate potential solutions

(B) Define the problem

(C) Implement the solution

(D) Evaluate alternatives

2. What is an algorithm?

(A) A step-by-step procedure to solve a problem

(B) A graphical representation of a process

(C) A programming language

(D) À flowchart symbol

3. Which of the following is NOT a characteristic of a good algorithm?

(A) Finiteness

(B) Determinism

(C) Ambiguity

(D) Feasibility

4. What is a flowchart?

(A) A programming language

(B) A step-by-step procedure to solve a problem

(C) A graphical representation of algorithm

(D) A function in programming language

5. Which flowchart symbol is used to represent a decision point in a
process?

(A) Rectangle

(B) Circle

(C) Arrow

(D) Diamond

6.What does the scanfO function do in C?

(A) Prints output to the console

(B) Reads input from the user

(C) Copies one string to another

(D) Concatenates two strings

7**. How do you compare two strings in C?**

(A) strl = str2

(B) **stremp(strl, str2)**

(C) strcomp(str1, str2)

(D) strcompare(str1, str2)

8.What will be the output of the expression 11 ^ 5?

(A) 12

(B) 13

(C) 10

(D) 1

9. What is the purpose of the strlen function in C?

(A) To compare two strings

(B) To concatenate two strings
(C) To calculate the length of a string
(D) To convert a string to upper case

10. Which library should be included to use the strepy function in C?

(A) stdlib.h

(B) stdio.h

(C) string.h

(D) math.h

11. What is the correct way to declare a string variable in C?

(A) string s

(B) char s[]

(C) char* s

(D) str s

12. s = 'foobar',

Then what is the output of s[-2]

(A) o

(B) r

(C) b

(D) a

13. We can combine strings and numbers by using the method

(A) combine()

(B) concat()

(C) index()

(D) format()

14. Which formatted output value is used to print hexadecimal values?

(A) %s

(B) %o

(C) %h

(d)%x

15. Which string function converts a character into an integer?

(A) len()

(B) str()

(C) ord()

(D) char()

16. With respect to String operation, the + operator____________ the given strings.

(A) Adds

***(*B) Concatenates**

(C) Multiplies

(D) Compares

17._________is used when data is in Tabular format.

A) NumPy

(B) Pandas

(C) MatPlotlib

(D) Random

18. The Interface of Matplotlib used for data visualization is

(A) Seaborn

(B) Matlab

(C) Pyplot

(D) Pandas

19. _______is a high level API built on TensorFlow.

(A) PyBrain

(B) Keras

(C) Pandas

(D) Scrapy

20. Which of the following module is to be imported to create Series?

(A) NumPy

(B) MatPlotlib

(C) Pandas

(D) Random

Q-21

(a) Define problem-solving and discuss the six steps involved in problem- solving

Problem solving is the act of defining a problem; determining the cause of the problem; identifying, prioritizing, and selecting alternatives for a solution; and implementing a solution.

1) Understand the Problem

•This step involves a deep analysis of the problem to comprehend its nature, causes, and impact.

•Gather all relevant data and ensure you're addressing the root cause, not just the symptoms.

2). Define the Problem

•Clearly articulate the problem in a structured, specific, and measurable way.

•Ensure the problem definition aligns with the scope of the task at hand.

3. Define Boundaries and Constraints

•Determine the constraints such as budget, time, resources, or environmental factors.

•Establish the boundaries within which the solution must be developed.

4. Plan the Solution

•Propose multiple possible solutions and evaluate their feasibility using engineering principles.

•Select the most suitable solution based on criteria like cost-effectiveness, efficiency, and reliability.

5. Implement the Solution

•Execute the planned solution using appropriate tools, technologies, and methods.

•Continuously monitor progress to avoid deviations from the plan.

6. Evaluate Results and Optimize

•Assess the implemented solution's effectiveness by comparing it with the defined goals.

•Collect feedback from stakeholders and analyse performance metrics.

Q-21(b)

(i) Define algorithm, and describe the characteristics of an algorithm?

An **algorithm** is a finite, step-by-step procedure or set of rules that are followed to perform a specific task or solve a problem. It is a well-defined sequence of instructions that leads to the solution of a problem in a finite amount of time. Algorithms are the foundation of computer science and are used in programming to solve various computational tasks.

Characteristics of an Algorithm:

1. **Finiteness:**
   - An algorithm must have a finite number of steps. It should eventually terminate after executing a limited number of operations.
2. **Definiteness (Clarity):**
   - Every step in the algorithm must be precisely and unambiguously defined. There should be no ambiguity in the instructions, ensuring that each step is clear and understandable.
3. **Input:**
   - An algorithm should have zero or more inputs, which are provided before the algorithm starts or during its execution. These inputs are the initial data needed for the algorithm to function.
4. **Output:**
   - An algorithm must produce at least one output, which is the result or solution to the problem. The output should be related to the inputs and provide useful information.
5. **Effectiveness:**
   - The steps of the algorithm must be basic enough to be carried out, in principle, by a human using only pen and paper in a reasonable amount of time. This ensures that the algorithm is practical and can be implemented.
6. **Generality:**
   - An algorithm should be applicable to a general class of problems, not just a specific instance. It must solve all possible instances of the problem, within the defined constraints.

(ii) Mention flowchart and discuss the symbols/shapes are commonly used in flowcharts.

A **flowchart** is a visual representation of a process or algorithm, used to illustrate the sequence of steps required to solve a problem. It uses various symbols connected by arrows to represent the flow of control and data in a system or process. Flowcharts are widely used in programming, process management, and systems analysis to simplify complex processes and communicate them clearly.

Commonly Used Symbols/Shapes in Flowcharts:

1. **Oval (Terminator):**
   - **Purpose:** Represents the start and end points of the flowchart.
   - **Description:** The oval shape is used to indicate the beginning (Start) or the end (End) of the process.
   - **Example:** "Start" or "End"
2. **Rectangle (Process):**
   - **Purpose:** Represents a process, operation, or action.
   - **Description:** This symbol indicates a task or operation that needs to be performed, such as a calculation or assignment.
   - **Example:** "Add two numbers," "Set variable to value"
3. **Parallelogram (Input/Output):**
   - **Purpose:** Represents input or output operations.
   - **Description:** This shape is used when the process involves receiving input or providing output (e.g., reading data or displaying results). o **Example:** "Read input from user," "Display result"
4. **Diamond (Decision):**
   - **Purpose:** Represents a decision point in the flowchart.

- **Description:** The diamond shape is used to show a decision or condition, typically a yes/no or true/false question, that determines the next step. o **Example:** "Is the number positive?" (Yes/No)

5. **Arrow (Flowline):**
    - **Purpose:** Indicates the flow or direction of the process.
    - **Description:** Arrows connect different symbols to show the sequence of operations or decisions, indicating the order in which steps are executed.
    - **Example:** Connecting "Start" to the first process or decision point.

Q-22

(a) Analyze the various Storage Classes in C with suitable examples

In the C programming language, **storage classes** define the scope, lifetime, and visibility of variables and functions. They determine how and where the variables are stored in memory, as well as how long they retain their values. C has four primary storage classes:

1. **Auto**
2. **Register**
3. **Static**
4. **Extern**

Each of these storage classes has its specific characteristics that control the variable’s scope, lifetime, and memory location.

1. Auto Storage Class:

- **Default Storage Class**: The auto storage class is the default for local variables in C. Variables declared inside a function or block without any storage class are automatically assumed to be auto.
- **Scope**: Local to the function or block where they are declared.
- **Lifetime**: The lifetime of an auto variable is limited to the function or block in which it is defined. The variable is created when the block is entered and destroyed when the block is exited.
- **Memory**: The memory for auto variables is allocated on the **stack**.

2. Register Storage Class:

- **Purpose**: The register storage class is used for variables that are frequently accessed. It suggests to the compiler to store the variable in a **CPU register** instead of RAM to speed up access.
- **Scope**: Local to the function or block where they are declared.
- **Lifetime**: Same as auto (limited to the block where they are declared).
- **Memory**: The register keyword asks the compiler to store the variable in a register, but it's not guaranteed. If no registers are available, the compiler will treat the variable as an auto variable.

3. Static Storage Class:

- **Purpose**: The static storage class is used to retain the value of a variable between function calls. A static variable keeps its value even after the function exits and is preserved when the function is called again.
- **Scope**: If declared inside a function, it is local to that function. If declared outside, it is global but accessible only within the file (file scope).
- **Lifetime**: Static variables are initialized only once, and their values are preserved throughout the program's execution, unlike auto variables, which are reinitialized each time the function is called.
- **Memory**: Static variables are stored in the **data segment** of the memory.

4. Extern Storage Class:

- **Purpose**: The extern storage class is used to declare a global variable or function in another file. It allows variables or functions to be shared between different source files.
- **Scope**: Global, across different files (if the variable is declared as extern in one file, it can be accessed in others).
- **Lifetime**: The lifetime of an extern variable is the entire duration of the program.
- **Memory**: extern variables are stored in the **data segment** (like static variables) but are accessible across multiple files.

Summary of Storage Classes in C:

Storage Class	**Scope**	**Lifetime**	**Memory Location**	**Visibility**
Auto	Local to function/block	Limited to the block	Stack	Local
Register	Local to function/block	Limited to the block	Register (if available)	Local
Static	Local (if inside a function) or Global (if outside)	Throughout program execution	Data segment	Limited (if inside function) or Local (if outside function)
Storage Class	**Scope**	**Lifetime**	**Memory Location**	**Visibility**
Extern	Global across files	Throughout program execution	Data segment	Global across files

Q-22

(b) Explain with example +ti, itt, -i, & i-.

1. Prefix Increment (++i)

- **Operation:** The value of the variable is incremented first, and then the result is used in the expression. **Eg:**- int i = 5; int result = ++i; print(result);

  Output:
  After this code executes, i = 6 and result = 6.

2. Postfix Increment (i++)

- **Operation:** The value of the variable is used in the expression first, and then the variable is incremented. Eg:- int i = 5; int result = i++; printf("%d",result);

  Output:
  After this code executes, i = 6 and result = 5.

3. Prefix Decrement (--i)

- **Operation:** The value of the variable is decremented first, and then the result is used in the expression. **Eg:-**

  int i = 5; int result =

  --i;

  printf(“%d”,result);

  Output:
  After this code executes, i = 4 and result = 5.

4.Postfix Decrement (i--)

- **Operation:** The value of the variable is used in the expression first, and then the variable is decremented. **Eg:**-

  int i = 5; int result =

  i--;

  printf(“%d”,result);

Q-23

(a) Describe function and its types in C? And explain recursion function with an example.

A **function** in C is a block of code that performs a specific task. Functions allow us to break down a complex problem into smaller, manageable pieces, making the code easier to maintain and reuse. Functions can take inputs, process those inputs, and return outputs.

<u>Syntax of a Function in C:</u>

```
#include <stdio.h>
return_type function_name(formal parameters); int
main()
 {
     //Statement Block
     function_name(actual parameters) //Function Calling

}
return_type function_name(parameters)
 {

   // body of the function
// code to be executed
```

```
   return value;  // if function has a return type
}
```

- **Return type**: The data type of the value the function will return (e.g., int, float, void).
- **Function name**: A unique name to identify the function.
- **Parameters**: Optional values passed to the function to perform operations.
- **Body**: The block of code that defines what the function does.
- **Return statement**: Returns a value from the function to the caller (only applicable if the return type is not void).

Types of Functions in C:

✞ **Pre-Defined Functions:** These are pre-defined functions that are part of the C standard library. These functions perform common tasks like input/output operations, mathematical operations, and string manipulations.

 Examples: printf(), scanf(), strlen(), sqrt(), malloc(), free().

✞ **User-Defined Functions:** These are functions that you define to perform specific tasks within a program. They allow you to write modular, reusable code.

- **Function Declaration/Prototype**: Declares the function before it is used.
- **Function Definition**: Provides the actual implementation of the function.
- **Function Call**: Invokes the function to execute it.

✞ **Functions with a Return-Type and Arguments**
✞ **Functions with a Return-Type and without Arguments**
✞ **Functions without a Return-Type and without Arguments** ✞ **Functions without a Return-Type and with Arguments**

✞ **Recursive Function:-**

✞ **Recursion** is a process in which a function calls itself. Recursive functions are often used to solve problems that can be broken down into smaller, similar subproblems, such as tree traversal, factorial computation, Fibonacci series, etc.
✞ A recursive function must have two key elements:
Example:-

```
#include <stdio.h>
```

```
// Recursive function to calculate factorial int

factorial(int n) {     if (n == 0) {  // Base case:

factorial of 0 is 1         return 1;

    }

    else {  // Recursive case: n * factorial of (n-1)

return n * factorial(n - 1);

    }

}


int main() {     int number = 5;     printf("Factorial of %d is %d\n",

number, factorial(number));     return 0;

}
```

Q-23

(b) Write a C Program to replace all VOWELS with star (*) and print the output.

```
#include <stdio.h> int

main() {     char str[100];

printf("Enter a string: ");

scanf("%[^\n]%*c",str);

    for (int i = 0; str[i] != '\0'; i++) {

        if (str[i] == 'a' || str[i] == 'e' || str[i] == 'i' || str[i] == 'o' || str[i] == 'u' ||

str[i] == 'A' || str[i] == 'E' || str[i] == 'I' || str[i] == 'O' || str[i] == 'U') {

str[i] = '*';

        }

    }
    printf("Modified string: %s", str);
return 0;
}
```

Output:-

```
Enter the string : programming
Modified string : pr*gr*mm*ng
```

Q-24

(a) Describe about the different types of data structures in python.

data structures are used to organize, store, and manage data efficiently. Python offers several built-in types of data structures that allow for different ways of storing and manipulating data.

1. List

A list is an ordered collection of elements, which can be of different types (e.g., integers, strings, objects). Lists are mutable, meaning their elements can be modified.

Syntax: list_name = [element1, element2, element3, ...] **Characteristics**:

1)Ordered (preserves the insertion order).

2)Allows duplicate values

3)Supports indexing, slicing, and various methods (like append(), insert(),etc 4)Allows

mixed data types.

Eg :-my_list = [1, 2, 3, "hello", 5.5]

2. Tuple

Definition: A tuple is similar to a list but is **immutable** (i.e., its elements cannot be changed once assigned).

Syntax: tuple_name = (element1, element2, element3, ...) **Characteristics**:

1)Ordered.

2)Allows duplicates.

3)Cannot be changed after creation.

4)Can be used as keys in dictionaries due to immutability.

3. Set

Definition: A set is an unordered collection of unique elements. It does not allow duplicate values.

Syntax: set_name = {element1, element2, element3, ...} **Characteristics**:

1) Unordered (does not guarantee order).

2) No duplicates.

3) Supports mathematical set operations like union, intersection, difference.

4) Mutable.

Eg:- tuple = (1, 2, 3, "hello")

3. Set

Definition: A set is an unordered collection of unique elements. It does not allow duplicate values.

Syntax: set_name = {element1, element2, element3, ...} **Characteristics**:

1) Unordered (does not guarantee order).

2) No duplicates

3) Supports mathematical set operations like union, intersection, difference.

4) Mutable.

Eg:- set = {1, 2, 3, 4}

4. Dictionary (dict)

Definition: A dictionary is an unordered collection of key-value pairs. It allows fast lookups by key and is useful for associative data. **Syntax**: dict_name = {key1: value1, key2: value2, ...} **Characteristics**:

1) Unordered (since Python 3.7, insertion order is preserved, but still does not guarantee sorting).

2) Keys must be unique and immutable (e.g., strings, numbers, tuples).

3) Values can be any data type.

Eg :-dict = {"name": "John", "age": 25, "city": "New York"}

Q-24

(b) Explain the significant features of the Pandas and Numpy library.

Python is a powerful language for data analysis, and two of its most essential libraries are **Pandas** and **NumPy**. These libraries provide a vast range of tools for data manipulation, analysis, and numerical computations, making them indispensable for scientific computing and data science tasks.

1. <u>Pandas Library:</u>

Pandas is an open-source data manipulation and analysis library built on top of NumPy. It provides data structures like **DataFrame** and **Series** that make handling structured data much easier. Here are some of the key features of Pandas: **a. Data Structures:**

- **DataFrame**: The core data structure in Pandas, a DataFrame is a twodimensional labeled data structure, similar to a table or spreadsheet. It supports rows and columns, making it easy to store and manipulate structured data.
- **Series**: A one-dimensional array-like object, it can hold any data type (integers, strings, floats, Python objects). It's essentially a single column of a DataFrame. **b. Data Handling:**
- **Reading and Writing Data**: Pandas allows reading from and writing to various file formats like CSV, Excel, SQL databases, and more.

c. Data Cleaning and Transformation:

- **Handling Missing Data**: Pandas has powerful tools for handling missing or null data with functions like .isna(), .dropna(), and .fillna().
- **Data Filtering**: Easily filter rows and columns using conditions, such as selecting rows where age is greater than 30.

d. Data Grouping and Aggregation:

- **GroupBy**: The .groupby() function allows you to group data based on one or more columns and apply aggregation functions like sum, mean, count, etc.

e. Time Series Data:

- **Datetime Handling**: Pandas has robust support for handling time series data. It can easily parse dates, perform date arithmetic, and resample data (e.g., changing the frequency of time series).

f. Merging and Joining Data:

- Pandas provides functions like .merge(), .join(), and .concat() for combining different datasets, similar to SQL join operations.

2. <u>NumPy Library:</u>

NumPy is the foundational package for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. **a. N-dimensional Array (ndarray):**

- The core data structure in NumPy is the **ndarray** (n-dimensional array), which allows efficient storage and manipulation of large data sets. It supports a wide range of operations such as arithmetic, matrix operations, and element-wise computations.

b. Fast Array Operations:

- NumPy arrays are faster than traditional Python lists due to their optimized C implementation. You can perform arithmetic operations on entire arrays at once (called **vectorization**), avoiding the need for explicit loops.

c. Array Broadcasting:

- NumPy supports **broadcasting**, which allows operations to be performed on arrays of different shapes, making it very efficient when working with multidimensional arrays.

d. Mathematical Functions:

- NumPy provides a vast library of mathematical functions that can operate on arrays, such as sin(), cos(), log(), sqrt(), sum(), mean(), etc.

e. Linear Algebra Operations:

- NumPy includes a variety of functions for performing linear algebra operations like matrix multiplication, dot product, eigenvalues, etc.

f. Random Number Generation:

- NumPy provides a powerful random module (np.random) for generating random numbers, random samples, and random distributions (uniform, normal, etc.). **g. Array Manipulation:**
- NumPy provides numerous functions for reshaping, concatenating, splitting, and modifying arrays.

Comparison of Pandas and NumPy:

Feature	**Pandas**	**NumPy**
Primary Data Structure	DataFrame, Series	ndarray
Data Types	Heterogeneous data types (strings, ints, etc.)	Homogeneous data types (numbers)
Feature	**Pandas**	**NumPy**
Handling Missing Data	Supports missing data handling (NaN, None)	Does not have built-in support for missing data
Functionality	Data manipulation, data cleaning, merging, grouping	Numerical and matrix operations
Performance	Optimized for data manipulation on tables, but slower than NumPy for numerical operations	Optimized for numerical operations, faster than Pandas for large datasets
Flexibility	Highly flexible for structured data analysis (e.g., tables, CSV files)	More focused on large numerical arrays or matrices

Q-25

(a) Explain function and module with suitable example.

Explanation of Function and Module in Python:-

Function:

A function in Python is a block of code that is designed to perform a specific task. Functions help to organize code into reusable, manageable, and modular blocks. They can take input, process it, and return output. Functions are defined using the def keyword, followed by the function name and parameters.

Syntax: def

```
function_name(parameters):
   # function body
return result
```

Example:

```
# Function that adds two numbers
def add_numbers(a, b):
   return a + b


# Calling the function result
=     add_numbers(5,     3)
print(result)  # Output: 8
```

In this example:

add_numbers is a function that takes two parameters a and b and returns their sum.

Functions help reduce repetition in code by allowing us to reuse them whenever needed.

Module:

A module is a file containing Python definitions and statements. It can contain functions, classes, and variables that can be reused across different programs. A module is simply a Python file with a .py extension. You can import and use its contents in other programs.

Syntax:

```
import module_name
```

Example: Let's say we have a module math_operations.py containing a function add_numbers.

```
# math_operations.py (module) def

add_numbers(a, b):

   return a + b
```

Now, in another Python file, we can import and use the add_numbers function from the math_operations module:

```
# main.py import math_operations result =

math_operations.add_numbers(5, 7)

print(result)  # Output: 12
```

In this case:

math_operations.py is the module that defines the add_numbers function.

In main.py, we import the module and use the add_numbers function by calling math_operations.add_numbers(5, 7).

Key Differences:

Function: A function is a block of code that performs a specific task, and it can be part of a module.

Module: A module is a file containing Python definitions (such as functions and variables) that can be imported and reused in other Python programs.

Conclusion:

Functions make your code more modular and reusable.

Modules allow you to organize your functions and code into separate files, enhancing code maintainability and reusability.

Q-25

(b) Explain list and tuple with suitable example

Definition:

A **list** is a mutable (modifiable) collection of items. It allows you to add, remove, or change elements.

Characteristics:

1.**Mutable**: You can modify the content of a list.

2.**Ordered**: Items in a list maintain their order.

3.**Duplicates Allowed**: Lists can contain duplicate elements.

4.**Heterogeneous**: Items in a list can have different data types.

5.**Syntax**: Lists are defined using square brackets [ ].

```
# Creating a list students = ["Alice",

"Bob", "Charlie"]


# Adding an item students.append("Diana")  # List becomes ['Alice', 'Bob',

'Charlie', 'Diana']
```

# Updating an item students[1] = "Eve"   # List becomes ['Alice', 'Eve',

'Charlie', 'Diana']

# Removing an item students.remove("Alice") # List becomes ['Eve',

'Charlie', 'Diana'] print(students)   # Output: ['Eve', 'Charlie',

'Diana']

Definition:

A **tuple** is an ordered collection of elements. Like lists, tuples can hold elements of different data types. However, unlike lists, tuples are **immutable**—once created, their elements cannot be modified, added, or removed.

Syntax:

Tuples are created using **parentheses** ().

Key Characteristics of Tuples:

- **Immutable**: Tuples are immutable, meaning once a tuple is created, you cannot change, add, or remove elements from it.
- **Ordered**: Like lists, tuples also maintain the order of their elements.
- **Performance**: Due to their immutability, tuples are generally faster than lists for accessing elements.
- **Heterogeneous**: Tuples can contain elements of different data types.

Eg:-   coordinates = (10, 20, 30)

Feature	List	Tuple
Syntax	Defined with square brackets []	Defined with parentheses ()
Mutability	Mutable (elements can be modified)	Immutable (elements cannot be modified)
Performance	Slower (due to mutability)	Faster (due to immutability)
Methods	More methods available (e.g., append(), remove(), etc.)	Fewer methods available (e.g., count(), index())
Usage	Used when data needs to be modified or changed	Used when data should remain constant and not change
Memory Usage	Takes more memory (due to mutability)	Takes less memory (due

Q-26

(i) Write a C program to calculate the sum of digits of a five digit number.

```
#include <stdio.h> int
main()

 {

    int n, s = 0;  //Variable Declaration & Initialization
    printf("Enter a 5 digit number : ");

scanf("%d",&n);          //Reading a 5 digit number

    for(int i = 1;n!=0;i++)

    {

        s += n%10;                    //Adding the digits

n/=10;

    }

    printf("The Sum of Digits = %d",s); //Printing the Sum of  Digits      return 0;

 }
```

Output:-

```
Enter a 5 digit number : 12345 The
Sum of Digits = 15
```

(ii) Mr. Bob has been deputed as the Election officer for the Tamil Nadu State Election. He wanted to perform an analysis to check whether a candidate is eligible for voting when he/she enters his/her age. Write a C program to read the age of a candidate and determine it is eligible for casting his/her vote.

```
#include <stdio.h> int
main()

 {

    int a;                          //Variable Declaration

    printf("Enter the Age of the Candidate : ");

scanf("%d",&a);          //Reading the Age of the Candidate

if(a>=18)         //Checking if the Candidate is Eligible

printf("The Candidate is Eligible for Voting");

    else
          printf("The Candidate is Ineligible for Voting");
return 0;

 }
```

<u>Output:-</u>

```
Enter the Age of the Candidate : 18 The
Candidate is Eligible for Voting
```

27) What is Dictionary? Explain Python dictionaries in detail discussing its operations and methods.

A **dictionary** in Python is an unordered, mutable, and indexed collection of data. It is one of the built-in data types in Python used to store collections of data in the form of key-value pairs. Dictionaries are also known as **associative arrays** or **hash maps** in other programming languages. Each key in a dictionary is unique, and it maps to a value. These keys must be immutable types (like strings, numbers, or tuples), and the values can be of any data type.

A Python dictionary is defined using curly braces {} with key-value pairs separated by colons.

The general syntax is:
my_dict = {key1: value1, key2: value2, key3: value3} **Example of**

a Dictionary:

```
person = {
    "name": "John",
    "age": 30,
    "city": "New York"
}
```

In this dictionary:

- "name", "age", and "city" are the **keys**.
- "John", 30, and "New York" are the corresponding **values**.

Operations on Python Dictionaries

Python dictionaries support several operations:

1. Accessing Values:

Values in a dictionary can be accessed using their respective keys.

```
print(person["name"])  # Output: John
```

If the key is not present in the dictionary, a KeyError will be raised.

2. Modifying Values:

You can modify the value associated with a specific key.

person["age"] = 31 print(person) # Output: {'name': 'John', 'age': 31,

'city': 'New York'}

3. Adding Items:

New key-value pairs can be added to a dictionary.

person["email"] = "john@example.com" print(person)

4. Removing Items:

You can remove items using the del statement, or use dictionary methods like pop() and popitem().

del person["city"] print(person) # Output: {'name': 'John', 'age': 31, 'email':

'john@example.com'}

# Using pop() method removed_value = person.pop("age")

print(person) # Output: {'name': 'John', 'email': 'john@example.com'}

print(removed_value) # Output: 31

# Using popitem() method (removes last inserted item)

last_item = person.popitem() print(person) # Output:

{'name': 'John'} print(last_item) # Output: ('email',

'john@example.com')

5. Checking for Key Existence:

The in keyword can be used to check if a key exists in a dictionary.

print("name" in person) # Output: True print("city"

in person) # Output: False

6. **Clearing All Items:**

The clear() method removes all items from a dictionary.

person.clear() print(person) # Output: {}

Methods in Python Dictionaries

Python dictionaries come with several built-in methods for performing operations:

1. **clear()**
   Removes all elements from the dictionary. my_dict.clear()

2. **copy()**
   Returns a shallow copy of the dictionary. new_dict =
   my_dict.copy()

3. **fromkeys()**
   Returns a new dictionary with keys from the provided iterable and a default value. new_dict = dict.fromkeys(["a", "b", "c"], 0)

   print(new_dict) # Output: {'a': 0, 'b': 0, 'c': 0}

4. **get()**
   Returns the value for the specified key if the key exists. Otherwise, it returns None or a specified default value. print(my_dict.get("key1")) # Output: value1
   print(my_dict.get("nonexistent_key", "Not Found")) # Output: Not Found

5. **items()**
   Returns a view object that displays a list of a dictionary's key-value tuple pairs.

   print(my_dict.items()) # Output: dict_items([('key1', 'value1'), ('key2', 'value2')])

6. **keys()**
   Returns a view object that displays all the keys in the dictionary.

   print(my_dict.keys()) # Output: dict_keys(['key1', 'key2'])

7. **values()**
   Returns a view object that displays all the values in the dictionary.

   print(my_dict.values()) # Output: dict_values(['value1', 'value2'])

8. **pop()**
   Removes the key-value pair with the specified key and returns its value.

   value = my_dict.pop("key1") print(value)

   # Output: value1

9. **popitem()**
   Removes and returns the last key-value pair as a tuple.

   last_item = my_dict.popitem() print(last_item) # Output:

   ('key2', 'value2')

10. **setdefault()**
    Returns the value of the key if it exists. If not, inserts the key with a specified default value.

    value = my_dict.setdefault("key3", "default_value") print(value)

    # Output: default_value

11. **update()**
    Updates the dictionary with the key-value pairs from another dictionary or an iterable of key-value pairs.

    my_dict.update({"key4": "value4"}) print(my_dict) #

    Output: {'key1': 'value1', 'key4': 'value4'}

Conclusion

A Python dictionary is a versatile data structure that allows fast access, modification, and removal of key-value pairs. It is widely used due to its efficient implementation, supporting various operations and methods that make data management easier. Understanding how to manipulate dictionaries and using their built-in methods is essential for anyone working with Python.