

INDUSTRIAL TRAINING REPORT ON PYTHON

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE
AWARD OF THE OF **PUNJAB STATE BOARD OF TECHNICAL EDUCATION**

DIPLOMA (POLYTECHNIC)
(Computer Science & Engineering)
BATCH :- 2022-2025



SUBMITTED TO :
Dr. Hardeep Singh Jawanda (HOD Sir - CSE Dept.)

SUBMITTED BY :
NAME :- JEET SEHGAL (1453)

Registration No. : 220179511143
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
GURU NANAK DEV POLYTECHNIC COLLEGE , LUDHIANA

Dated: 21/08/2024

TO WHOM IT MAY CONCERN

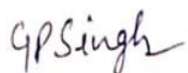
This is to certify that **Mr. Jeet Sehgal S/o Mr. Deepak Kumar** studying in **Guru Nanak Dev Polytechnic College**, branch **Computer Science and Engineering** has undergone industrial training for the period of six weeks.

He was working on a technology based on **Python MySQL and Advanced Python**. Duration from **8 July 2024 to 21 August 2024**. During his training period, he worked on following projects:

1. Quiz about python.
2. Graph using CSV data set.
3. Student database project MySQL database.
4. Chatbot console base.

During the period of internship program, his performance was excellent.
We wish his continued success in his future career endeavours.

Yours Sincerely,



Gurpreet Singh
(Project Coordinator)



Khalasoft

ISO 9001:2015 CERTIFIED

470 – L, First Floor, Model Town, Ludhiana (Punjab) - 141002

Mobile: (+91) 965077479, info@khalasoft.com, gurpreet@khalasoft.com

ACKNOWLEDGEMENT

I would like to express our sincere gratitude to all those who have contributed to the successful completion of this project report on Python Training.

First and foremost, we would like to extend our heartfelt appreciation to our Instructor, Mr. Gurpreet Singh, for their invaluable guidance, continuous support, and insightful feedback throughout the training. Their expertise and mentorship played a pivotal role in shaping our understanding and mastery of Python programming.

I would like to thank Respected **HOD Dr. Hardeep Singh Jawanda Sir (CSE Dept.)** who gave me their immense support in completion of my training

We would also like to thank the participants who engaged actively during the sessions, providing valuable feedback and insights that contributed to the refinement of our learning experience. Their enthusiasm and commitment to learning were greatly appreciated.

Lastly, we would like to acknowledge our family and friends for their unwavering support and understanding throughout the duration of this training. Their encouragement and belief in our abilities have been instrumental in our learning journey.

Although it is not possible to mention everyone individually, we are sincerely grateful to all those who have directly or indirectly contributed to this training experience. Thank you for being a part of this endeavor and for helping us in making this project report a reality.

Jeet Sehgal

Branch (CSE)

Roll No : 1453

Registration No : 220179511143

Python Training Participial

ABOUT INSTITUTE

I have completed my training of python from khalsa soft training cetner. In this training center our mentor Mr. Gurpreet Singh Sir provided us profound knowledge about python. Having completed my training at the Python Training Center, I can confidently say that the experience was instrumental in shaping my understanding of Python's core concepts. The curriculum was thorough, covering everything from basic data types and control structures to more advanced topics like object-oriented programming. What stood out the most was the emphasis on practical, hands-on exercises that allowed me to apply what I learned in real-world scenarios. The instructors were knowledgeable and supportive, always available to clarify doubts and guide me through complex topics. This training not only solidified my Python skills but also gave me the confidence to tackle programming challenges with a deeper understanding of the language.

Graduating from the Python Training Center was a transformative experience that deeply enriched my programming journey. The center's curriculum was meticulously crafted, starting from the very basics like variables, data types, and loops, and progressing to more advanced concepts such as functions, modules, and object-oriented programming. Each topic was introduced with clear explanations, followed by practical examples that demonstrated their application in real-world scenarios.

Table of Contents

Ch No	Topic	Page No
	Certificate	2
	Acknowledgement	3
	About Institute	4
1	Python and It's Basic Concepts	6
2	Conditional Statement and Loops	11
3	Operater and Method	15
4	SQL and It's Commands	20
5	CSV , JSON and Module	30
6	Module Use To Manipulate The Database	34
	"PROJECT OF PYTHON INDUSTRIAL TRAINING"	37

TRAINING DESCRIPTION

CHAPTER – 1

PYTHON AND IT'S BASIC CONCEPTS

1.1 Python :- Python is a high-level, interpreted programming language known for its simplicity and readability, making it accessible to both beginners and experienced developers. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python's extensive standard library and large ecosystem of third-party packages enable developers to build a wide variety of applications, from web development and data analysis to automation and artificial intelligence. Python's design philosophy emphasizes code readability and simplicity, allowing developers to write clear and concise code, which is one of the reasons it has become one of the most popular programming languages in the world.

1.2 Python History :- Python was created by Guido van Rossum and was first released in 1991. The development of Python began in the late 1980s as a successor to the ABC programming language, which was designed for teaching programming but had limitations. Guido van Rossum, working at the Centrum Wiskunde & Informatica (CWI) in the Netherlands, aimed to create a language that would overcome these limitations while maintaining simplicity and ease of use.

1.3 Type Casting :- Type casting, also known as type conversion, is the process of converting a value from one data type to another in programming. In Python, this is typically done using built-in functions that explicitly convert data types. There are two types of type conversion: implicit and explicit.

1.3.1 Implicit Type Casting :- Implicit type casting, also known as implicit type conversion or coercion, is the automatic conversion of one data type to another by the Python interpreter during an operation. This happens when a situation arises where two different types of data are involved, and Python needs to ensure that the operation can be performed correctly. The conversion is done without any explicit instruction from the programmer.

1.3.2 Explicit Type Casting :- Explicit type casting, also known as explicit type conversion or type coercion, involves manually converting a value from one data type to another in Python using built-in functions. This process gives the programmer full control over how and when the conversion happens, ensuring that data types are converted according to the specific needs of the program.

Syntax :- `var_name=datatype(expression)`

1.4 Scope of Python :- Python has an extensive scope in the modern tech landscape, making it one of the most versatile and widely-used programming languages. Its simplicity, readability, and extensive library support have enabled

Python to penetrate various domains, ranging from web development to artificial intelligence. Here's an overview of Python's scope:

1. Web Development

- Python is widely used in web development, with powerful frameworks like Django, Flask, and Pyramid simplifying the creation of robust and scalable web applications. Python's clean syntax and rapid development capabilities make it an ideal choice for startups and large enterprises alike.

2. Data Science and Analytics

- Python is the dominant language in data science, thanks to libraries such as Pandas, NumPy, SciPy, and Matplotlib. These tools, combined with the simplicity of Python, allow data scientists to analyze large datasets, perform statistical analyses, and visualize data effectively. Python's role in data science extends to machine learning and AI as well.

3. Artificial Intelligence and Machine Learning

- Python is a leading language in AI and machine learning, with popular libraries like TensorFlow, Keras, PyTorch, and Scikit-learn. Its ease of use, combined with the extensive community support, has made Python the go-to language for developing machine learning models, neural networks, natural language processing applications, and more.

4. Automation and Scripting

- Python's simplicity and readability make it perfect for automation and scripting tasks. It's often used for automating repetitive tasks, such as file management, web scraping, and task scheduling. Python's vast library ecosystem further enhances its capabilities in this area.

5. Scientific Computing

- Python is extensively used in scientific research and computing. Libraries such as NumPy, SciPy, and SymPy allow scientists to perform complex mathematical computations, simulations, and modeling. Python is also a preferred language in fields like bioinformatics and computational physics.

6. Embedded Systems and IoT

- Python is making inroads into the embedded systems and IoT space, particularly with microcontroller platforms like Raspberry Pi and MicroPython.

Python's ease of use allows developers to quickly prototype and develop IoT applications.

1.5 Naming Conventions :- In Python, naming conventions often refer to specific styles like PascalCase, camelCase, and others. Here's a breakdown of these common naming styles and where they are typically used:

1. PascalCase

- **Description:** Also known as UpperCamelCase, in PascalCase, each word in the name starts with an uppercase letter, and there are no underscores or spaces between words.
- **Usage:** PascalCase is commonly used for naming classes and exception classes in Python.
- **Example:**
 - ClassName
 - UserProfile
 - DatabaseConnection

2. camelCase

- **Description:** In camelCase, the first word starts with a lowercase letter, and each subsequent word starts with an uppercase letter. Like PascalCase, there are no underscores or spaces.
- **Usage:** camelCase is less commonly used in Python compared to languages like JavaScript or Java. However, it may sometimes be used for naming instance variables or methods, though the Python community generally prefers snake_case for these.
- **Example:**
 - instanceVariable
 - calculateSum
 - getUserInfo

3. snake_case

- **Description:** In snake_case, all letters are lowercase, and words are separated by underscores.
- **Usage:** This is the most common naming convention for variables, functions, and methods in Python. It's favored for its readability.
- **Example:**
 - variable_name
 - calculate_sum
 - get_user_info

1.6 Comments :- Comments in Python are used to annotate code, making it easier to understand for both the original programmer and others who might read the code later. Comments are ignored by the Python interpreter, so they do not affect the execution of the program. Python supports single-line comments, multi-line comments, and docstrings for documenting code. Here's a breakdown of how comments are used in Python:

1. Single-Line Comments

- **Syntax:** Use the # symbol at the beginning of a line or after a line of code to create a single-line comment.
- **Usage:** Single-line comments are typically used to explain a particular line of code or provide brief notes.

2. Multi-Line Comments

- **Syntax:** There is no specific syntax for multi-line comments in Python, but you can create them by using multiple single-line comments or by using triple quotes (though triple quotes are technically meant for docstrings).
- **Usage:** Multi-line comments are used to explain larger sections of code or provide more detailed explanations

1.7 Data types :- Python offers a wide variety of built-in data types that are used to store and manipulate different kinds of data. Understanding these data types is fundamental to programming in Python. Here's a detailed overview of the key data types in Python:

- **int:** Integer numbers, which can be positive or negative and have no fractional part.
Example: `x = 5`, `y = -100`
- **float:** Floating-point numbers, which are real numbers with a decimal point.
Example: `pi = 3.14`, `gravity = 9.8`
- **complex:** Complex numbers, which have a real and an imaginary part, represented as `a + bj`.
Example: `z = 3 + 5j`, `c = -2j`
- **str:** Strings, which are sequences of characters enclosed in single, double, or triple quotes.
Example: `name = "Alice"`, `greeting = 'Hello, world!'`
- **list:** Lists, which are ordered, mutable sequences of elements. Lists can contain elements of different types.
Example: `fruits = ["apple", "banana", "cherry"]`, `numbers = [1, 2, 3, 4]`
- **tuple:** Tuples, which are ordered, immutable sequences of elements. Like lists, they can contain elements of different types.
Example: `coordinates = (10, 20)`, `person = ("John", 30, "Engineer")`

- **set:** Sets, which are unordered collections of unique elements. Sets do not allow duplicate elements.
Example: `unique_numbers = {1, 2, 3, 4, 5}`, `letters = {'a', 'b', 'c'}`
- **bool:** Boolean values, which can be either True or False. Booleans are often used in conditional statements and expressions.
Example: `is_active = True`, `is_valid = False`
- **dict:** Dictionaries, which are unordered, mutable collections of key-value pairs. Each key is unique within the dictionary.
Example: `person = {"name": "Alice", "age": 25, "city": "New York"}`, `scores = {"math": 90, "science": 85}`

CHAPTER – 2

CONDITIONAL STATEMENT AND LOOPS

2.1 Conditional Statement :- Conditional statements in Python allow you to execute certain blocks of code based on whether a condition is True or False. They are a fundamental part of controlling the flow of a program. The most common conditional statements in Python are if, elif, and else.

1. The if Statement

- **Usage:** The if statement is used to test a condition. If the condition is True, the code block under the if statement is executed.
- **Syntax:**
if condition:
 # code block to execute if condition is True
- **Example:**
x = 10
if x > 5:
 print("x is greater than 5")
 - **Output:** x is greater than 5

2. The elif Statement

- **Usage:** The elif (short for "else if") statement allows you to check multiple conditions. If the first if condition is False, Python checks the next elif condition. You can have multiple elif statements.
- **Syntax:**
if condition1:
 # code block to execute if condition1 is True
elif condition2:
 # code block to execute if condition2 is True
- **Example:**
x = 10
if x > 10:
 print("x is greater than 10")
elif x == 10:
 print("x is equal to 10")
 - **Output:** x is equal to 10

3. The else Statement

- **Usage:** The else statement is used as a fallback. If none of the if or elif conditions are True, the code block under else is executed.
- **Syntax:**

```
if condition1:  
    # code block to execute if condition1 is True  
elif condition2:  
    # code block to execute if condition2 is True  
else:  
    # code block to execute if all previous conditions are False
```
- **Example:**

```
x = 10  
if x > 10:  
    print("x is greater than 10")  
elif x == 10:  
    print("x is equal to 10")  
else:  
    print("x is less than 10")
```

▪ **Output:** x is equal to 10

4. Nested Conditional Statements

- **Usage:** You can nest conditional statements within each other to check multiple levels of conditions.
- **Syntax:**

```
if condition1:  
    if condition2:  
        # code block to execute if both condition1 and condition2 are True  
    else:  
        # code block to execute if condition1 is True and condition2 is False  
else:  
    # code block to execute if condition1 is False
```
- **Example:**

```
x = 10  
y = 20  
if x > 5:  
    if y > 15:  
        print("x is greater than 5 and y is greater than 15")  
    else:  
        print("x is greater than 5 but y is not greater than 15")
```

```
else:  
    print("x is not greater than 5")
```

- **Output:** x is greater than 5 and y is greater than 15

2.2 Loops :- Loops in Python allow you to execute a block of code repeatedly, either a set number of times or until a certain condition is met. Python primarily uses two types of loops: for loops and while loops. Each has its own use case and can be accompanied by control statements like break, continue, and else.

1. The for Loop

- **Usage:** The for loop in Python is used to iterate over a sequence (such as a list, tuple, string, or range) and execute a block of code for each element in that sequence.
- **Syntax:**
for variable in sequence:
 # code block to execute

- **Example:**

```
fruits = ["apple", "banana", "cherry"]  
for fruit in fruits:  
    print(fruit)
```

- **Output:**

```
apple  
banana  
cherry
```

- **Using range() in a for loop:**

```
for i in range(5):  
    print(i)
```

- **Output:**

```
0  
1  
2  
3  
4
```

2. The while Loop

- **Usage:** The while loop is used to repeatedly execute a block of code as long as a specified condition is True.
- **Syntax:**
while condition:
 # code block to execute
- **Example:**
i = 1
while i < 5:
 print(i)
 i += 1
 - **Output:**
1
2
3
4

CHAPTER –3

OPERATER AND METHODS

3.1 OPERATER IN PYTHON :- Operators in Python are special symbols that perform operations on variables and values. They can be categorized into several types based on their functionality. Here's an overview of the different types of operators in Python:

1. Arithmetic Operators

- **Used for basic mathematical operations.**
- **Operators:**
 - **+** : Addition
 - **-** : Subtraction
 - ***** : Multiplication
 - **/** : Division (floating-point division)
 - **//** : Floor Division (division that rounds down to the nearest integer)
 - **%** : Modulus (remainder after division)
 - ****** : Exponentiation (power of)

2. Comparison (Relational) Operators

- **Used to compare two values.**
- **Operators:**
 - **==** : Equal to
 - **!=** : Not equal to
 - **>** : Greater than
 - **<** : Less than
 - **>=** : Greater than or equal to
 - **<=** : Less than or equal to

3. Logical Operators

- **Used to combine conditional statements.**
- **Operators:**
 - **and** : Returns True if both statements are true
 - **or** : Returns True if one of the statements is true
 - **not** : Reverses the result, returns False if the result is true

4. Assignment Operators

- **Used to assign values to variables.**
- **Operators:**
 - **=** : Assigns value
 - **+=** : Adds and assigns
 - **-=** : Subtracts and assigns

- `*=` : Multiplies and assigns
- `/=` : Divides and assigns
- `//=` : Floor divides and assigns
- `%=` : Modulus and assigns
- `**=` : Exponentiates and assigns

5. Bitwise Operators

- **Used to perform operations on binary numbers.**
- **Operators:**
 - `&` : Bitwise AND
 - `|` : Bitwise OR
 - `^` : Bitwise XOR
 - `~` : Bitwise NOT
 - `<<` : Bitwise left shift
 - `>>` : Bitwise right shift

6. Identity Operators

- **Used to compare the memory locations of two objects.**
- **Operators:**
 - `is` : Returns True if both variables are the same object
 - `is not` : Returns True if both variables are not the same object

7. Membership Operators

- **Used to test if a sequence is present in an object.**
- **Operators:**
 - `in` : Returns True if a sequence with the specified value is present in the object
 - `not in` : Returns True if a sequence with the specified value is not present in the object

3.2 Methods :- In Python, methods are functions that are defined inside a class and are associated with the objects of that class. They operate on the data within an object, which is typically called the instance of a class. **Syntax:** Functions in Python are defined using the `def` keyword, followed by the function name, parentheses `()`, and a colon `:`. The code block within the function is indented.

```
def greet(name):
    print("Hello", name)
```

Types of methods :-

1. Built-in Methods

- **Description:** Python comes with a set of built-in functions that are always available for use. These functions perform common tasks like converting data types, performing mathematical calculations, and more.

- **Examples:**
 - `print()`: Outputs data to the console.
 - `len()`: Returns the length of an object.
 - `type()`: Returns the type of an object.
 - `sum()`: Sums up the elements of an iterable.

2. User-defined Functions

- **Description:** These are functions that you define yourself to perform specific tasks. They are created using the `def` keyword, followed by the function name, parentheses `()`, and a colon `:`. The function body is indented.

3.3 List Methods :- In Python, methods are functions that are associated with objects. When it comes to lists, Python provides a variety of built-in methods to perform operations on list objects. Here's a list of commonly used list methods:

- **`append(x)`:** Adds an item `x` to the end of the list.
- **`extend(iterable)`:** Extends the list by appending all items from the given iterable.
- **`insert(i, x)`:** Inserts an item `x` at a specified position `i` in the list.
- **`remove(x)`:** Removes the first occurrence of the item `x` from the list.
- **`pop([i])`:** Removes and returns the item at position `i`. If `i` is not provided, it removes and returns the last item.
- **`clear()`:** Removes all items from the list, leaving it empty.
- **`index(x[, start[, end]])`:** Returns the index of the first occurrence of the item `x`. Optional `start` and `end` parameters can be used to limit the search to a specific subrange.
- **`count(x)`:** Returns the number of times the item `x` appears in the list.
- **`sort(key=None, reverse=False)`:** Sorts the items of the list in place. The `key` parameter can specify a function to determine the sort order. The `reverse` parameter can be set to `True` to sort in descending order.
- **`reverse()`:** Reverses the order of the items in the list in place.
- **`copy()`:** Returns a shallow copy of the list, creating a new list with the same elements.

3.4 Dictionary Methods :- Here's a list of common dictionary methods in Python, along with brief descriptions:

1. **`clear()`:** Removes all items from the dictionary, leaving it empty.
2. **`copy()`:** Returns a shallow copy of the dictionary.
3. **`fromkeys(iterable, value=None)`:** Creates a new dictionary with keys from the given iterable and assigns a default value to all keys (default value is `None` if not specified).

4. **get(key, default=None)**: Returns the value for the specified key if it exists in the dictionary. If the key is not found, it returns the default value (default is None if not specified).
5. **items()**: Returns a view object that displays a list of dictionary's key-value tuple pairs.
6. **keys()**: Returns a view object that displays a list of all the keys in the dictionary.
7. **pop(key, default=None)**: Removes the specified key and returns the corresponding value. If the key is not found, it returns the default value (if specified) or raises a `KeyError`.
8. **popitem()**: Removes and returns the last inserted key-value pair as a tuple. Raises a `KeyError` if the dictionary is empty.
9. **setdefault(key, default=None)**: Returns the value of the specified key if it exists in the dictionary. If the key is not found, it inserts the key with the specified default value and returns the value.
10. **update([other])**: Updates the dictionary with the key-value pairs from another dictionary or an iterable of key-value pairs. If a key exists in the original dictionary, its value is updated.
11. **values()**: Returns a view object that displays a list of all the values in the dictionary.

3.5 Exception Handling :- Exception handling in Python is a mechanism that allows developers to manage and respond to errors or exceptional conditions in their code. Instead of allowing an error to crash the program, you can catch and handle exceptions to ensure your program runs smoothly, even when something unexpected occurs.

Key Concepts in Python Exception Handling:

1. **Exceptions:**
 - a. Exceptions are errors detected during execution. When an exception occurs, the normal flow of the program is disrupted, and the program may terminate unless the exception is handled.
2. **try and except Blocks:**
 - a. **try block:** Contains the code that might raise an exception. Python executes this block and if an error occurs, the execution is transferred to the except block.
 - b. **except block:** Contains the code that runs if an exception occurs in the try block. You can specify the type of exception you want to handle (e.g., `ValueError`, `TypeError`, etc.).
3. **else Block:**
 - a. An optional block that can be used after except blocks. The code inside the else block runs if no exceptions were raised in the try block.
4. **finally Block:**

- a. An optional block that is always executed, regardless of whether an exception was raised or not. It's commonly used for cleanup actions, like closing files or releasing resources.

5. Raising Exceptions:

- a. You can raise exceptions intentionally using the raise keyword. This is useful for testing or when you want to trigger an error condition.

CHAPTER – 4

SQL AND IT'S COMMANDS

4.1 SQL :- SQL, or Structured Query Language, is the foundational language used for managing and manipulating relational databases. It provides a standardized method for performing various database operations, including querying, updating, inserting, and deleting data. With SQL, users can interact with databases to retrieve specific information, manage data integrity, and maintain database structures efficiently. It supports complex queries, data aggregation, and sophisticated data manipulation through commands like SELECT, INSERT, UPDATE, and DELETE. Additionally, SQL facilitates database administration tasks such as creating and modifying tables, managing permissions, and ensuring data consistency with transactions. Its powerful capabilities and standardized syntax make SQL an essential tool for data management and analysis in numerous applications across diverse industries. sql commands are divided into 4 categories mainly which are :-

1. DDL :- DDL is a subset of SQL used for defining and managing the structure of database objects. These commands are crucial for creating, modifying, and deleting structures in a database, which include tables, indexes, views, and schemas.

- **CREATE**

- **Purpose:** Used to define new database objects such as tables, indexes, views, or schemas.
- **Functionality:**
 - **Tables:** Specifies the structure of a new table, including its columns and data types.
 - **Indexes:** Creates an index to improve query performance by enabling faster data retrieval.
 - **Views:** Defines a virtual table based on a query, which simplifies complex data retrieval.
 - **Schemas:** Establishes a new schema to organize database objects.

- **ALTER**

- **Purpose:** Modifies the structure of existing database objects.
- **Functionality:**
 - **Tables:** Allows changes such as adding, deleting, or modifying columns, or changing data types.
 - **Indexes:** Can be used to modify the properties of an existing index.
 - **Views:** Updates the definition of a view.

- **Schemas:** Alters schema properties or transfers objects between schemas.

- **DROP**

- **Purpose:** Removes existing database objects from the database.
- **Functionality:**
 - **Tables:** Deletes a table and all its data.
 - **Indexes:** Removes an index from a table.
 - **Views:** Deletes a view from the database.
 - **Schemas:** Deletes a schema and all objects contained within it.

- **TRUNCATE**

- **Purpose:** Removes all records from a table but retains the table structure for future use.
- **Functionality:**
- **Tables:** Quickly deletes all rows in a table, resetting the table to its empty state while keeping the table structure intact. It is generally faster than using the DELETE command because it does not log individual row deletions.

2. DML :- DML is a subset of SQL used for managing and manipulating data within database tables. Unlike DDL commands, which focus on the structure of the database, DML commands deal with the actual data stored in the tables. The primary DML commands include INSERT, DELETE, and UPDATE.

- **INSERT**

- **Purpose:** Adds new records to a table.
- **Functionality:**
 - **Records:** Inserts one or more new rows into a table with specified values for each column. This command is essential for populating tables with data.
 - **Syntax Variations:** Can insert a single row or multiple rows in one command, and can also insert data derived from a SELECT query.

- **DELETE**

- **Purpose:** Removes existing records from a table.
- **Functionality:**
 - **Records:** Deletes one or more rows from a table based on specified criteria. This command can be used to remove specific records or all records from a table.

- **Criteria:** Uses a WHERE clause to specify which records to delete. Omitting the WHERE clause will remove all rows in the table, but the table structure remains intact.

- **UPDATE**

- **Purpose:** Modifies existing records in a table.
- **Functionality:**
 - **Records:** Updates the values of one or more columns in existing rows based on specified criteria. This command is used to correct or change data without altering the table structure.
 - **Criteria:** Uses a WHERE clause to specify which rows should be updated. Omitting the WHERE clause will result in updating all rows in the table.

3. DCL :- DCL is a subset of SQL used to control access and permissions on database objects. It includes commands that manage user privileges and security within the database. The primary DCL commands are GRANT and REVOKE.

- **GRANT**

- **Purpose:** Provides specific privileges to users or roles, allowing them to perform certain actions on database objects.
- **Functionality:**
 - **Privileges:** Grants permissions such as SELECT, INSERT, UPDATE, and DELETE on tables or views. It can also grant administrative privileges like creating tables or managing other users.
 - **Users/Roles:** Assigns these permissions to individual users or roles, enabling them to perform specific operations based on their granted rights.
- **Usage:**
 - **Tables:** Can grant permissions on tables to allow users to view, modify, or manage data.
 - **Views:** Can grant access to views, enabling users to query and work with virtual tables.
 - **Other Objects:** Can also grant permissions on other database objects, such as stored procedures or functions.

- **REVOKE**

- **Purpose:** Removes specific privileges from users or roles, restricting their access or ability to perform certain actions on database objects.
- **Functionality:**

- **Privileges:** Revokes permissions that were previously granted, effectively limiting or removing access to tables, views, or other objects.
- **Users/Roles:** Applies to individual users or roles, modifying their access rights as needed.
- **Usage:**
 - **Tables:** Can revoke access to tables, preventing users from querying, inserting, or modifying data.
 - **Views:** Can remove access to views, restricting users from interacting with virtual tables.
 - **Other Objects:** Can also revoke permissions on other database objects, such as stored procedures or functions.

4. TCL :- TCL is a subset of SQL used to manage transactions within a database. Transactions are sequences of SQL operations executed as a single unit, ensuring data integrity and consistency. TCL commands control how these transactions are handled, including committing changes or rolling them back.

- **COMMIT**

- **Purpose:** Permanently saves all changes made during the current transaction.
- **Functionality:**
 - **Data Integrity:** Ensures that all changes made during a transaction are applied to the database. Once a COMMIT is issued, the changes become permanent and are visible to other users.
 - **Transaction End:** Marks the end of a transaction, confirming that all operations within the transaction have been completed successfully.
- **Usage:**
 - **Successful Completion:** Used when all operations within a transaction have been executed correctly, and the changes should be made permanent.

- **ROLLBACK**

- **Purpose:** Undoes all changes made during the current transaction, reverting the database to its previous state.
- **Functionality:**
 - **Error Handling:** Allows the cancellation of a transaction if an error occurs or if certain conditions are not met. This ensures that partial or erroneous changes are not applied to the database.

- **Transaction Reversal:** Restores the database to the state it was in before the transaction began, effectively discarding all changes made during the transaction.
- **Usage:**
 - **Error Recovery:** Used when an error is detected or when a transaction needs to be aborted. It ensures that incomplete or incorrect data is not saved

4.2 Data Type in SQL :- Data types in SQL define the kind of data that can be stored in a table's columns. Each column in a table is assigned a specific data type, which dictates the nature of the data (such as numeric, text, or date/time) that can be stored in that column. SQL provides various data types, which can generally be categorized into several broad groups.

- **BOOLEAN:** Stores true/false values. In some SQL implementations, TRUE is represented as 1 and FALSE as 0.
- **DATE:** Stores a date value (year, month, day).
- **TIME:** Stores a time value (hours, minutes, seconds).
- **TEXT:** Stores large strings of variable length. Often used for storing large text blocks like descriptions or content.
- **VARCHAR(n):** Stores a string of variable length with a maximum size of n characters.
- **CHAR(n):** Stores a string of fixed length n. If the string is shorter than n, it is padded with spaces.
- **INT or INTEGER:** Stores whole numbers, typically 4 bytes in size.

4.3 Aggregate functions :- Aggregate functions in SQL are used to perform calculations on multiple rows of a table's data and return a single result. These functions are often used in conjunction with the GROUP BY clause to group rows that share certain characteristics before applying the aggregate function. They are essential for summarizing and analyzing data.

1. COUNT()

- **Purpose:** Counts the number of rows in a result set or the number of non-null values in a specified column.
- **Usage:**
 - **Counting Rows:** COUNT(*) counts all rows in a table, including rows with NULL values.
 - **Counting Non-Null Values:** COUNT(column_name) counts only the non-null values in a specific column.

2. SUM()

- **Purpose:** Calculates the total sum of a numeric column.
- **Usage:**
 - **Summing Values:** SUM(column_name) adds up all the values in a specified numeric column. It is often used to calculate totals like sales, revenue, or expenses.

3. AVG()

- **Purpose:** Calculates the average value of a numeric column.
- **Usage:**
 - **Calculating Average:** AVG(column_name) returns the average of all the values in a specified numeric column, providing insights into typical values in a dataset.

4. MIN()

- **Purpose:** Returns the smallest value in a specified column.
- **Usage:**
 - **Finding Minimum:** MIN(column_name) is used to identify the lowest value in a dataset, such as the smallest price, earliest date, or lowest score.

5. MAX()

- **Purpose:** Returns the largest value in a specified column.
- **Usage:**
 - **Finding Maximum:** MAX(column_name) is used to identify the highest value in a dataset, such as the highest salary, latest date, or maximum score.

6. COUNT(DISTINCT)

- **Purpose:** Counts the number of distinct (unique) non-null values in a column.
- **Usage:**
 - **Counting Unique Values:** COUNT(DISTINCT column_name) counts how many different (unique) values exist in a specified column, which is useful for counting unique items, categories, or occurrences.

4.4 Joins :- Joins in SQL are used to combine rows from two or more tables based on a related column between them. Joins allow you to retrieve data that is distributed across multiple tables, presenting it in a single result set. There are several types of joins, each serving a specific purpose in querying relational databases.

1. INNER JOIN

- **Purpose:** Retrieves rows that have matching values in both tables involved in the join.
- **Usage:**
 - **Data Matching:** Only rows with matching values in the specified columns of both tables are returned in the result set.
 - **Common Use:** Often used when you need to find records that exist in both tables, such as finding orders that have been placed by customers.

2. LEFT JOIN (or LEFT OUTER JOIN)

- **Purpose:** Retrieves all rows from the left table and the matching rows from the right table. If there is no match, the result will include NULL for columns from the right table.
- **Usage:**
 - **Data Inclusion:** Ensures that all rows from the left table are included in the result, regardless of whether there is a matching row in the right table.
 - **Common Use:** Useful for queries where you want to see all records from the left table, such as finding all customers and the orders they've placed, including those who haven't placed any orders.

3. RIGHT JOIN (or RIGHT OUTER JOIN)

- **Purpose:** Retrieves all rows from the right table and the matching rows from the left table. If there is no match, the result will include NULL for columns from the left table.
- **Usage:**
 - **Data Inclusion:** Ensures that all rows from the right table are included in the result, regardless of whether there is a matching row in the left table.
 - **Common Use:** Often used when you want to see all records from the right table, such as finding all orders and their customers, including orders that may not be associated with any customer.

4. FULL JOIN (or FULL OUTER JOIN)

- **Purpose:** Retrieves all rows when there is a match in either left or right table. If there is no match, the result will include NULL for columns from the table without a match.
- **Usage:**
 - **Data Combination:** Combines the results of both LEFT JOIN and RIGHT JOIN, returning all records from both tables and filling in NULL where there are no matches.

- **Common Use:** Useful for queries where you want to retrieve all records from both tables, such as combining two datasets with potentially unmatched rows.

5. CROSS JOIN

- **Purpose:** Returns the Cartesian product of the two tables, meaning all possible combinations of rows.
- **Usage:**
 - **Data Combination:** Each row from the first table is combined with every row from the second table, which can lead to a very large result set.
 - **Common Use:** Typically used when there is no explicit relationship between the tables, or when you need to generate all possible combinations, such as creating test data.

6. SELF JOIN

- **Purpose:** Joins a table to itself, often used to compare rows within the same table.
- **Usage:**
 - **Data Comparison:** The table is aliased to create two instances, which are then joined based on a related column within the same table.
 - **Common Use:** Used in scenarios like finding duplicate records, hierarchical data structures (like employee-manager relationships), or comparing rows within the same table.

4.5 Clauses :- SQL clauses are used to refine queries, enabling more precise data retrieval and organization. Each clause serves a specific purpose in filtering, sorting, grouping, and limiting data.

1. WHERE Clause

- **Purpose:** Filters rows based on specified conditions.
- **Functionality:**
 - **Condition Filtering:** Retrieves only those rows that meet the conditions defined in the WHERE clause.
 - **Common Use:** Used to select specific records that match criteria, such as finding customers from a particular city or products with a price above a certain value.
 - **Example:** Selects records where the value of a column meets a specified condition, like WHERE age > 18.

2. ORDER BY Clause

- **Purpose:** Sorts the result set based on one or more columns.
- **Functionality:**
 - **Sorting:** Arranges the rows in ascending (ASC) or descending (DESC) order based on the values of specified columns.
 - **Common Use:** Used to sort data by specific criteria, such as sorting customers by last name or orders by date.
 - **Example:** ORDER BY last_name ASC sorts the results by last name in alphabetical order.

3. GROUP BY Clause

- **Purpose:** Groups rows that have the same values in specified columns into summary rows.
- **Functionality:**
 - **Data Aggregation:** Often used with aggregate functions (COUNT, SUM, AVG, etc.) to summarize data by specific criteria.
 - **Common Use:** Used to group data, such as grouping sales data by product category or customer orders by customer ID.
 - **Example:** GROUP BY department groups employees by their department, allowing aggregate functions like SUM(salary) to calculate the total salary for each department.

4. HAVING Clause

- **Purpose:** Filters groups created by the GROUP BY clause based on specified conditions.
- **Functionality:**
 - **Post-Grouping Filtering:** Similar to WHERE, but used after the data has been grouped. It filters groups based on aggregate conditions.
 - **Common Use:** Used to filter groups that meet certain conditions, such as finding departments with a total salary above a certain amount.
 - **Example:** HAVING SUM(salary) > 100000 filters out groups where the total salary is less than 100,000.

5. LIMIT Clause

- **Purpose:** Restricts the number of rows returned in the result set.
- **Functionality:**
 - **Result Limiting:** Specifies the maximum number of rows to be returned by the query.
 - **Common Use:** Useful for paging through results or retrieving a specific subset of data, such as showing the top 10 highest-paid employees.

- **Example:** LIMIT 10 returns only the first 10 rows of the result set.

CHAPTER – 5

CSV, JSON AND MODULES

5.1 CSV file :- A **CSV (Comma-Separated Values)** file is a plain text file that stores tabular data, such as a spreadsheet or database. Each line in a CSV file corresponds to a row of data, and each value in a line is separated by a comma, which represents the columns. CSV files are widely used because they are simple, easy to understand, and compatible with most applications that handle data, including Excel, Google Sheets, and various programming languages.

The csv module is part of Python's standard library and provides functionality to read from and write to CSV files.

Key Operations:

- **Reading a CSV File:**
 - You can read a CSV file line by line and process each row using the `csv.reader` method.
 - The `DictReader` class allows you to read a CSV file into a dictionary format, where the first row becomes the keys of the dictionary.
- **Writing to a CSV File:**
 - The `csv.writer` method is used to write rows to a CSV file.
 - You can also write dictionaries to a CSV file using the `DictWriter` class.

Example Operations:

- **Reading:**
 - Open the CSV file, create a `csv.reader` object, and iterate through the rows.
 - Use `DictReader` to read the CSV as a list of dictionaries for easier data manipulation.
- **Writing:**
 - Open a file in write mode, create a `csv.writer` object, and use it to write rows of data.
 - Use `DictWriter` to write dictionaries to the CSV, specifying the fieldnames.

5.2 JSON file :- **JSON (JavaScript Object Notation)** is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate. JSON is widely used for transmitting data between a server and a client, particularly in web applications. It represents data as key-value pairs, arrays, and nested objects.

Structure of a JSON File

- **Objects:** Collections of key-value pairs enclosed in curly braces {}.
- **Arrays:** Ordered lists of values enclosed in square brackets [].
- **Values:** Can be strings, numbers, booleans, arrays, or other JSON objects.

Example JSON Structure:

```
{
  "name": "John Doe",
  "age": 30,
  "email": "john.doe@example.com",
  "is_active": true,
  "skills": ["Python", "JavaScript", "SQL"],
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "zip": "12345"
  }
}
```

Python provides built-in support for working with JSON data through the `json` module. This module allows you to convert between JSON strings and Python dictionaries (and other data structures), making it easy to read, write, and manipulate JSON data.

1. Reading JSON Data

You can read JSON data from a file or a string and convert it into Python objects using the `json` module.

Key Functions:

- **`json.load()`:**
 - Reads JSON data from a file and converts it into a Python dictionary.
 - Example: `data = json.load(file)`
- **`json.loads()`:**
 - Parses a JSON string and converts it into a Python dictionary.
 - Example: `data = json.loads(json_string)`

Common Use:

- Loading configuration files, API responses, or data files that are in JSON format.

2. Writing JSON Data

You can write Python objects to a JSON file or convert them into a JSON string using the json module.

Key Functions:

- **json.dump():**
 - Converts a Python object into a JSON string and writes it to a file.
 - Example: `json.dump(data, file)`
- **json.dumps():**
 - Converts a Python object into a JSON string.
 - Example: `json_string = json.dumps(data)`

Common Use:

- Saving configuration settings, logging data, or sending data over a network in JSON format.

3. Manipulating JSON Data

Once JSON data is loaded into a Python dictionary, you can easily manipulate it as you would with any dictionary.

Key Operations:

- **Accessing Data:**
 - Use standard dictionary syntax to access values by their keys.
 - Example: `name = data["name"]`
- **Modifying Data:**
 - Modify values by assigning new data to existing keys or adding new key-value pairs.
 - Example: `data["age"] = 31`
- **Adding or Removing Keys:**
 - Add new keys by assigning values, or remove keys using the `del` statement.
 - Example: `data["new_key"] = "new_value"`
 - Example: `del data["email"]`

3.3 Modules :- A **module** in Python is a file that contains Python definitions and statements, such as functions, classes, and variables, that you can use in other Python programs. Modules help organize and reuse code, making it more modular, maintainable, and easier to manage.

Key Features of Modules

1. **Modularity:**
 - a. Breaks down large programs into smaller, manageable, and organized pieces.
 - b. Encourages code reusability by allowing you to import and use code from one module in multiple programs.
2. **Namespace Separation:**
 - a. Modules create a separate namespace, which helps avoid naming conflicts between different parts of a program.
3. **Standard Library Modules:**
 - a. Python comes with a rich standard library of modules, providing a wide range of functionalities like file I/O, system calls, data manipulation, and more.
 - b. Examples include math for mathematical operations, os for operating system interactions, sys for system-specific parameters and functions, and datetime for date and time manipulation.
4. **Third-Party Modules:**
 - a. Python supports a vast ecosystem of third-party modules, available through the Python Package Index (PyPI).
 - b. These modules can be installed using package managers like pip.
 - c. Examples include requests for HTTP requests, numpy for numerical computations, and pandas for data manipulation.

CHAPTER – 6

MODULE USE TO MANIPULATE THE DATABASE

6.1 Module use to Manipulate MySQL :- To manipulate MySQL databases in Python, the most commonly used module is **mysql-connector-python**. This module allows you to connect to a MySQL database server, execute SQL queries, and manage the database directly from your Python code.

mysql-connector-python Module

mysql-connector-python is an official MySQL driver developed and maintained by Oracle. It provides an easy-to-use interface for interacting with MySQL databases and is fully compatible with Python.

6.2 Code use to connect with database :- For that thing you have to import the mysql.connector module . And after that you have to use connect() method to get connection . If you got connected with the database you have to get a cursor through cursor() method which is present in the object which is created during the time of connection with database .

Syntax :-

```
import mysql.connector

connection = mysql.connector.connect(

    host="localhost", # Replace with your host, e.g., "127.0.0.1"
    user="your_username", # Replace with your MySQL username
    password="your_password", # Replace with your MySQL password
    database="your_database" # Replace with your database name )

myCursor= connection.cursor()
```



6.3 Cursor Method :- The cursor() method in Python's mysql-connector-python module is used to create a cursor object, which allows you to execute SQL queries

and fetch data from a MySQL database. The cursor object provides various methods to interact with the database. Here are some of the key methods of the cursor object:

Key Methods of cursor()

1. **execute(query, params=None)**

- a. **Purpose:** Executes a single SQL query.
- b. **Parameters:**
 - i. query: The SQL query string to execute.
 - ii. params: Optional parameters for parameterized queries (e.g., to prevent SQL injection).
- c. **Example:**

```
cursor.execute("INSERT INTO employees (name, salary) VALUES (%s, %s)", ("Alice", 60000))
```

2. **executemany(query, seq_of_params)**

- a. **Purpose:** Executes a SQL query against all parameter sequences in the sequence of parameters.
- b. **Parameters:**
 - i. query: The SQL query string to execute.
 - ii. seq_of_params: A sequence of parameter sequences to execute the query with.
- c. **Example:**

```
data = [("Bob", 70000), ("Charlie", 80000)]
cursor.executemany("INSERT INTO employees (name, salary) VALUES (%s, %s)", data)
```

3. **fetchone()**

- a. **Purpose:** Fetches the next row of a query result set, returning a single sequence or None if no more rows are available.
- b. **Example:**

```
row = cursor.fetchone()
print(row)
```

4. **fetchall()**

- a. **Purpose:** Fetches all (remaining) rows of a query result set, returning a list of tuples.
- b. **Example:**

```
rows = cursor.fetchall()
for row in rows:
    print(row)
```

5. **fetchmany(size)**

- a. **Purpose:** Fetches the next set of size rows of a query result, returning a list of tuples.
- b. **Parameters:**
 - i. size: The number of rows to fetch.
- c. **Example:**

```
rows = cursor.fetchmany(5)
for row in rows:
    print(row)
```

6. close()

- a. **Purpose:** Closes the cursor object and releases its resources.
- b. **Example:**

```
cursor.close()
```

“PROJECT OF PYTHON INDUSTRIAL TRAINING”

QUIZ PROJECT :- I have created a project on the topic quiz in which there are two sections one is dedicated to admin and the other is for the user which plays the quiz. In quiz there are the questions about python which will be represented to the user randomly . When the user plays the quiz the ques will appear randomly . The ques will be in form of Multiple choice questions .

The user will have to enter the option name which he/she think can be a correct option . The user will get 5 ques in a particular quiz on the basis of which the score of the user will be calculated upon . The project uses the mysql database to store the data .The parts of project the admin and the user have it's own features and rights on the quiz. The exit part is present in every section of program for which we have to enter 0 . Both parts are explained below in detail :-

```
---Quiz Game---

enter 0 for exit
enter 1 for entering as admin
enter 2 for entering as user

enter your choice :
```

```
enter you choice : 0
exiting....
```

- **ADMIN PART :-** To enter in the admin part of project we have to enter the name, id, password of the existing admin so that the admin rights can be secure . When we enter a right id, name, password the admin features got unlocked for it . In this process the program matches the entered value with the database values for admin . It has a admin table in database which have all the values . The features which are provided to the admin are described below :-

```

enter your choice : 1

entering as admin.....

enter the admin name : jeet
enter the admin id : 1
enter the password : 4812
admin is varified successfully

enter 0 for exit
enter 1 for adding questions
enter 2 for view all users record
enter 3 for adding a admin

enter you choice : █

```

1. **Add a Admin:-** the admin can add a another admin . For that the admin will specify the name and password of new admin and this value is inserted in the table and after that table returns the id which will be associated to the admin.

```

enter 0 for exit
enter 1 for adding questions
enter 2 for view all users record
enter 3 for adding a admin

enter you choice : 3
enter the admin name : vivek
enter your password : 1234
admin added successfully
the admin id is : (9,)
enter 0 for exit
enter 1 for adding questions
enter 2 for view all users record
enter 3 for adding a admin

enter you choice :

```

2. **View the Data of User :-** In this the admin can view the data of the user which have played the quiz there score, name, age, id . When the admin

select that option the program fetches all the data from user table and show this to the admin in a specific format

```
enter 1 for adding questions
enter 2 for view all users record
enter 3 for adding a admin

enter you choice : 2

all users are :

user id : 1000
user name : karan
score : 5
user age : 18

user id : 1001
user name : abhinav
score : 5
user age : 18

user id : 1004
user name : navish
score : 4
user age : 12
```

3. **Add a ques to quiz :-** Admin can add a ques to the quiz of his choice. For adding a question he/she has to specify the ques and it's options and the correct answer. At code level all this data is going to store in the quiz table and when the user play the quiz the added ques may appear to the

ques. All the data will got store in the table

```
enter you choice : 1
enter your question :

which one is not method of list
enter option a : min()
enter option b : append()
enter option c : keys()
enter option d : copy()
enter the correct option from a,b,c,d : c
inserted successfully

enter 0 for exit
enter 1 for adding questions
enter 2 for view all users record
enter 3 for adding a admin

enter you choice : █
```

- **USER PART :-** In user part there are 2 options in which one is to log in if the user has already registered for the quiz for log in the user have to provide the user id, name after that the program matches that data from the database if the data matches successfully then user got successfully logged in. If the user is not registered for the quiz then he/she will select the option to sign in . In this part the user have to fill his name and his age. After that a user id is provided to the user and at code level this data is stored in the user table in the data base so that user can log in successfully in the quz. Two features unlock when a user got log in which are described below :-

```
enter your choice : 2
entering as user....

enter 0 for exit
enter 1 for log in
enter 2 for sign in
enter your choice : █
```

Sign in ->


```
enter 0 for exit
enter 1 for log in
enter 2 for sign in
enter your choice : 2
the sign in page .....
enter user name : hanji
enter the age : 23
you have signed in successfully >>>
your id is : 1006
```

Log in ->

```
enter 0 for exit
enter 1 for log in
enter 2 for sign in
enter your choice : 1
the log in page .....
enter user name : hanji
enter the id : 1006
user is logged in >>>
enter 0 for exit
enter 1 for viewing your score
enter 2 for playing quiz
enter the choice : █
```

1. **View the score :-** using this option the user can view his score in the quiz if the user has not played the quiz yet then the score are shown null but if he has played the quiz then the user last score are shown

```
enter 0 for exit
enter 1 for viewing your score
enter 2 for playing quiz
enter the choice : 1
your score are : None
```

2. **Play the Quiz :-** When user select this option the user can play the quiz. When the quiz start then the user encounter five random question and the user have to fill the answer on the basis of which score is calculated.

```
python was developed in  
option a : 1934  
option b : 1985  
option c : 1990  
option d : 1991
```

```
enter your answer : a
```

```
*****
```

```
you scored : 1
```

```
*****
```