Python Practice Programs Assignment

Intern: Sumit Dhar
Date: 15th September, 2025

Question 1: Create a table with a large number of records (you can find it with a google search or use this link - https://github.com/datacharmer/test_db). Use MySQL Database. One can setup MySQL on localhost. Write some basic queries using python. Suppose you want to process/fetch a large number of records using python while keeping your memory usage low. Think of approaches on how to accomplish this and Implement.

Hint: Use Generator

Solution:

Step 1: Preparing our SQL files

We kept three files in the same folder (`mySQL_data`):

1. `employees.sql` – creates database `employees` and tables `employees` and `salaries`.
2. `load_employees.sql` – inserts employee records.
3. `load_salaries.sql` – inserts salary records.

All files were ready in the same directory.

Step 2: Running MySQL using Docker

Since we didn't want to install MySQL natively, we will:

1. Open PowerShell in the folder containing our SQL files.
2. Run the Docker command:

```
docker run --name mysql-employees `
    -e MYSQL_ROOT_PASSWORD=<our_password> `
    -p 3306:3306 `
    -v ${PWD}:/docker-entrypoint-initdb.d `
    -d mysql:8.0
```

Explanation:

- `--name mysql-employees` → names our container.
- `-e MYSQL_ROOT_PASSWORD=<our_password>` → sets root password.
- `-p 3306:3306` → maps local port 3306 to container 3306.
- `-v ${PWD}:/docker-entrypoint-initdb.d` → mounts our current folder to container, so MySQL automatically executes SQL files on first run.
- `-d mysql:8.0` → runs MySQL 8.0 in detached mode.

Docker automatically **created the database, tables, and loaded our data** using our SQL files.

```
(testing_DG_1) PS F:\Internship\Python\practice_programs\mySQL_data> docker run --name mysql-employees `
>>     -e MYSQL_ROOT_PASSWORD=root `
>>     -p 3306:3306 `
>>     -v ${PWD}:/docker-entrypoint-initdb.d `
>>     -d mysql:8.0
Unable to find image 'mysql:8.0' locally
8.0: Pulling from library/mysql
48934deb9770: Pull complete
8a27c0ce790f: Pull complete
72a465986d66: Pull complete
021b6107b9d0: Pull complete
13ed16089ebc: Pull complete
27fa9cc59961: Pull complete
500d7b2546c4: Pull complete
e32dcaa70f77: Pull complete
390885da77e4: Pull complete
d8f78235dcb8: Pull complete
1ca2ca504238: Pull complete
Digest: sha256:d2fdd0af28933c6f28475ff3b7defdbc0e0475d9f7346b5115b8d3abf8848a1d
Status: Downloaded newer image for mysql:8.0
ff42005c3668005d22a0b4169b15f4e8dc3d2162dab29c2f36c55df7735acec7
```

Step 3: Checking MySQL in Docker

To verify it's running:

`docker ps`

```
(testing_DG_1) PS F:\Internship\Python\practice_programs\mySQL_data> docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS          PORTS                                                  NAMES
ff42005c3668   mysql:8.0   "docker-entrypoint.s…"   15 seconds ago   Up 14 seconds   0.0.0.0:3306->3306/tcp, [::]:3306->3306/tcp   mysql-employees
(testing_DG_1) PS F:\Internship\Python\practice_programs\mySQL_data>
```

- Shows container status is **Up**.
- If container stops, use `docker start <container_name>, here mysql-employees`.

Connect to MySQL inside the container:

```
docker exec -it mysql-employees mysql -u root -p
```

- Enter password: `<our_password>`.

```
(testing_DG_1) PS F:\Internship\Python\practice_programs\mySQL_data> docker exec -it mysql-employees mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 8.0.43 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

- Check database:

```
USE employees;
SELECT * FROM employees LIMIT 5;
SELECT * FROM salaries LIMIT 5;
```

```
mysql> use employees;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from employees limit 5;
+--------+------------+------------+-----------+--------+------------+
| emp_no | birth_date | first_name | last_name | gender | hire_date  |
+--------+------------+------------+-----------+--------+------------+
|  10001 | 1953-09-02 | Georgi     | Facello   | M      | 1986-06-26 |
|  10002 | 1964-06-02 | Bezalel    | Simmel    | F      | 1985-11-21 |
|  10003 | 1959-12-03 | Parto      | Bamford   | M      | 1986-08-28 |
|  10004 | 1954-05-01 | Chirstian  | Koblick   | M      | 1986-12-01 |
|  10005 | 1955-01-21 | Kyoichi    | Maliniak  | M      | 1989-09-12 |
+--------+------------+------------+-----------+--------+------------+
5 rows in set (0.01 sec)

mysql> select * from salaries limit 5;
+--------+--------+------------+------------+
| emp_no | salary | from_date  | to_date    |
+--------+--------+------------+------------+
|  10001 |  60117 | 1986-06-26 | 1987-06-26 |
|  10001 |  62102 | 1987-06-26 | 1988-06-25 |
|  10001 |  66074 | 1988-06-25 | 1989-06-25 |
|  10001 |  66596 | 1989-06-25 | 1990-06-25 |
|  10001 |  66961 | 1990-06-25 | 1991-06-25 |
+--------+--------+------------+------------+
5 rows in set (0.02 sec)

mysql>
```

We verified our data successfully loaded.

Step 4: Connecting Python to MySQL

Installing Python MySQL connector :

```
pip install mysql-connector-python
```

Connecting to the Docker MySQL database:

```
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="<our_password>",
    database="employees"
)
```

Step 5: Writing a generator to fetch data

Goal: **low memory usage** → fetch large tables in chunks (batches).

```
def fetch_employees(batch_size=1000, limit=10):
    cursor = conn.cursor(dictionary=True)
    cursor.execute(f"SELECT * FROM employees LIMIT {limit};")

    while True:
        rows = cursor.fetchmany(batch_size)
        if not rows:
            break
        for row in rows:
            yield row  # yield one row at a time

    cursor.close()
    conn.close()
```

- `fetchmany(batch_size)` → fetches small chunks from database.
- `yield row` → generator produces rows **one at a time**.
- Loop continues until all rows in the query are fetched.

Step 6: Displaying the data in table format

We used **tabulate** for nice CLI tables:

```
(base) PS F:\Internship\Python\practice_programs> python .\1_generatorSql.py
Enter the no. of records you want to see: 10
Enter the no. of records you want to batch together: 6

Batch 1:

+----------+--------------+--------------+-------------+----------+--------------+
|   emp_no | birth_date   | first_name   | last_name   | gender   | hire_date    |
+==========+==============+==============+=============+==========+==============+
|    10001 | 1953-09-02   | Georgi       | Facello     | M        | 1986-06-26   |
+----------+--------------+--------------+-------------+----------+--------------+
|    10002 | 1964-06-02   | Bezalel      | Simmel      | F        | 1985-11-21   |
+----------+--------------+--------------+-------------+----------+--------------+
|    10003 | 1959-12-03   | Parto        | Bamford     | M        | 1986-08-28   |
+----------+--------------+--------------+-------------+----------+--------------+
|    10004 | 1954-05-01   | Chirstian    | Koblick     | M        | 1986-12-01   |
+----------+--------------+--------------+-------------+----------+--------------+
|    10005 | 1955-01-21   | Kyoichi      | Maliniak    | M        | 1989-09-12   |
+----------+--------------+--------------+-------------+----------+--------------+
|    10006 | 1953-04-20   | Anneke       | Preusig     | F        | 1989-06-02   |
+----------+--------------+--------------+-------------+----------+--------------+

Batch 2:

+----------+--------------+--------------+-------------+----------+--------------+
|   emp_no | birth_date   | first_name   | last_name   | gender   | hire_date    |
+==========+==============+==============+=============+==========+==============+
|    10007 | 1957-05-23   | Tzvetan      | Zielinski   | F        | 1989-02-10   |
+----------+--------------+--------------+-------------+----------+--------------+
|    10008 | 1958-02-19   | Saniya       | Kalloufi    | M        | 1994-09-15   |
+----------+--------------+--------------+-------------+----------+--------------+
|    10009 | 1952-04-19   | Sumant       | Peac        | F        | 1985-02-18   |
+----------+--------------+--------------+-------------+----------+--------------+
|    10010 | 1963-06-01   | Duangkaew    | Piveteau    | F        | 1989-08-24   |
+----------+--------------+--------------+-------------+----------+--------------+
(base) PS F:\Internship\Python\practice_programs>
```

Now output shows **records batch by batch**, memory efficient and readable.

Summary

1. Prepared SQL schema + data files.
2. Ran MySQL inside Docker, auto-loaded SQL/DUMP files.
3. Verified data using `docker exec`.
4. Connected Python to Docker MySQL using `mysql.connector`.
5. Used a **generator** to fetch rows in batches (low memory).
6. Displayed results **batch-wise** using `tabulate`.

Question 2: Define a class Person and its two child classes: Male and Female. All classes have a method "get_gender" which can print "Male" for Male class and "Female" for Female Class. Make class Person an abstract class and make get_gender an abstract method in the same class. The two child classes must inherit and implement get_gender. i.,e, When trying to initialize an object of class Person, the program must throw an error.

Hint: Use abc library (comes natively with Python3) https://www.python-course.eu/python3_abstract_classes.php

Solution:

```
# ABC prevents misuse (creating meaningless base objects) and ensures the base class only serves as a template.
# @abstractmethod is a contract: it says any subclass of an abstract class must implement this method.

from abc import ABC, abstractmethod

class Person(ABC):
    @abstractmethod
    def get_gender(self): # this abstract method must be implemented by the child classes.
        pass

class Male(Person):
    def get_gender(self):
        return 'Male'

class Female(Person):
    def get_gender(self):
        return 'Female'

m = Male()
print(f'Child class "Male" returns get_gender: {m.get_gender()}\n')

f = Female()
print(f'Child class "Female" returns get_gender: {f.get_gender()}\n')

print(f'Throws error when we try to create obj of abstract class:')
try:
    p = Person()
except TypeError as e:
    print(f"Error: {e}")
```

Output:

```
(base) PS F:\Internship\Python\practice_programs> python .\2_abstract.py
Child class "Male" returns get_gender: Male

Child class "Female" returns get_gender: Female

Throws error when we try to create obj of abstract class:
Error: Can't instantiate abstract class Person without an implementation for abstract method 'get_gender'
(base) PS F:\Internship\Python\practice_programs>
```

Question 3: With a given list [12,24,35,24,88,120,155,88,120,155], write a program to print this list after removing all duplicate values with original order reserved.

Hint: Use set() to store a number of values without duplicates

Solution:

```
# We are using set, so that we can keep track of visited numbers,
# and only appending the first occurance of any number.
# Thus the original order is reserved.

given_list = [12,24,35,24,88,120,155,88,120,155]

filter_list = []
visited = set()

for item in given_list:
    if item not in visited:
        visited.add(item)
        filter_list.append(item)

print(f'Original List: {given_list}')
print(f'After removing duplicates: {filter_list}')
```

Output:

```
(base) PS F:\Internship\Python\practice_programs> python .\3_remDuplicate.py
Original List: [12, 24, 35, 24, 88, 120, 155, 88, 120, 155]
After removing duplicates: [12, 24, 35, 88, 120, 155]
(base) PS F:\Internship\Python\practice_programs>
```

Question 4: Write a program to generate a 3*5*8 3D array whose each element is 0.

Solution:

```python
# Write a program to generate a 3*5*8 3D array whose each element is 0.

# We are usng numpy for speed and performance.
import numpy as np

multi_dim_arr = np.zeros((3,5,8), dtype=int)
print(f"A 3*5*8 3D array:\n{multi_dim_arr}")
```

Output:

```
(base) PS F:\Internship\Python\practice_programs> python .\4_generateArray.py
A 3*5*8 3D array:
[[[0 0 0 0 0 0 0 0]
  [0 0 0 0 0 0 0 0]
  [0 0 0 0 0 0 0 0]
  [0 0 0 0 0 0 0 0]
  [0 0 0 0 0 0 0 0]]

 [[0 0 0 0 0 0 0 0]
  [0 0 0 0 0 0 0 0]
  [0 0 0 0 0 0 0 0]
  [0 0 0 0 0 0 0 0]
  [0 0 0 0 0 0 0 0]]

 [[0 0 0 0 0 0 0 0]
  [0 0 0 0 0 0 0 0]
  [0 0 0 0 0 0 0 0]
  [0 0 0 0 0 0 0 0]
  [0 0 0 0 0 0 0 0]]]
(base) PS F:\Internship\Python\practice_programs>
```

Question 5: Write a binary search function which searches an item in a sorted list. The function should return the index of the element to be searched in the list.

Solution:

```python
# Write a binary search function which searches an item in a sorted list. The function should return the index of the element to be searched in the list.

def binary_search(given_list: list, choice: int, left: int, right: int):
    if left > right:
        return -1
    mid = (right-left)//2 + left

    if given_list[mid] == choice:
        return mid
    elif given_list[mid] > choice:
        return binary_search(given_list, choice, left, mid-1)
    else:
        return binary_search(given_list, choice, mid+1, right)

given_list = [int(i) for i in range(1, 11)]

choice = int(input("Enter the number to seach in between 1 to 10 (inclusively):"))

index = binary_search(given_list, choice, 0, len(given_list)-1)

print(f"For the list: {given_list}")
if index == -1:
    print(f"Item was not found for choice: {choice}.")
else:
    print(f"Item was found at index-'{index}' (0-based index), for choice {choice}.")
```

Output:

```
(base) PS F:\Internship\Python\practice_programs> python .\5_binarySearch.py
Enter the number to seach in between 1 to 10 (inclusively):8
For the list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Item was found at index-'7' (0-based index), for choice 8.
(base) PS F:\Internship\Python\practice_programs> python .\5_binarySearch.py
Enter the number to seach in between 1 to 10 (inclusively):15
For the list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Item was not found for choice: 15.
(base) PS F:\Internship\Python\practice_programs>
```

Question 6: Write a program using generator to print the numbers which can be divisible by 5 and 7 between 0 and n in comma separated form while n is input by console.

Solution:

```
# Write a program using generator to print the numbers which can be divisible by 5 and 7 between 0 and n in comma separated form while n is input by console.

# a generator is a special type of function that allows us to produce values one at a time instead of creating them all at once in memory.
# it is like a lazy iterator, instead of returning with return, a generator uses yield.
# print_using_generator is a generator function because it uses yield, every time Python sees yield i, it pauses the function and returns that value to the caller.
# when we use list(print_using_generator(n)) Python automatically calls next(gen) in a loop until the generator is exhausted.

def print_using_generator(n):
    for i in range(n+1):
        if i % 5 == 0 and i % 7 == 0:
            yield i

n = int(input("Enter the value of 'n':"))

numbers = list(print_using_generator(n))

print(f"Values divisible by 5 and 7 are: \n{', '.join(map(str, numbers))}")
```

Output:

```
(base) PS F:\Internship\Python\practice_programs> python .\6_generator.py
Enter the value of 'n':108
Values divisible by 5 and 7 are:
0, 35, 70, 105
(base) PS F:\Internship\Python\practice_programs> python .\6_generator.py
Enter the value of 'n':482
Values divisible by 5 and 7 are:
0, 35, 70, 105, 140, 175, 210, 245, 280, 315, 350, 385, 420, 455
(base) PS F:\Internship\Python\practice_programs>
```

Question 7: Assuming that we have some email addresses in the "username@companyname.com" format, write a program to print the company name of a given email address. Both user names and company names are composed of letters only.

Solution:

```
# Assuming that we have some email addresses in the "username@companyname.com" format, write a program to print the company name of a given email address. Both user names and company names are composed of letters only.

# We will be using split method efficiently to separate the username and company name of the email address in format: (username@companyname.com)

email = input("Enter your email-id (username@companyname.com):")

user_name, rest_of_the_part = email.split('@')
company_name, rest_of_the_part = rest_of_the_part.split('.')

print(f"User name: {user_name} \nCompany name: {company_name}")
```

Output:

```
(base) PS F:\Internship\Python\practice_programs> python .\7_email.py
Enter your email-id (username@companyname.com):sumitdhar@datagrokr.co
User name: sumitdhar
Company name: datagrokr
(base) PS F:\Internship\Python\practice_programs> python .\7_email.py
Enter your email-id (username@companyname.com):xyz@abcompany.io
User name: xyz
Company name: abcompany
(base) PS F:\Internship\Python\practice_programs>
```

Question 8: Write a program which can map() to make a list whose elements are square of numbers between 1 and 20 (both included).

Hints:
Use map() to generate a list.
Use Lambda to define anonymous functions.

Solution:

```
# Write a program that can map() to make a list whose elements are squares of numbers between
1 and 20 (both included).

# Lambda is an anonymous (nameless) function in Python, written using the lambda keyword
instead of def.
# Syntax we use -> lambda arguments: expression
# Points to note:
# 1. lambda functions are one-liners i.e. no loops, no multiple statements.
# 2. it is used for short, throwaway functions.


nums_list = [i for i in range(1,21)]
squared_list = list(map(lambda x: x*x, nums_list))

print(f'Original list: {nums_list}')
print(f'Original list: {squared_list}')
```

Output:

```
(base) PS F:\Internship\Python\practice_programs> python .\8_squares.py
Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
Original list: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400]
(base) PS F:\Internship\Python\practice_programs>
```

Question 9: Implement a program dir_tree.py that takes a directory as argument and prints all the files in that directory recursively as a tree.

Hint: Use os.listdir and os.path.isdir functions.

Solution:

```
# Implement a program dir_tree.py that takes a directory as argument and prints all the files
in that directory recursively as a tree.
# Hint: Use os.listdir and os.path.isdir functions.

# We will be using 'os.path.isdir' to check whether it is a valid dir, then 'os.listdir' to
get all the files and os.path.join(path, item) to join path with respective files.

import os

def extraxt_files_in_dir_and_print(path, indent=""):
    if not os.path.isdir(path):
        return

    items = os.listdir(path)
    for index, item in enumerate(items):
        itemPath = os.path.join(path, item)
        isLast = index==len(items)-1
        prefix = "└── " if isLast else "├── "
        print(indent + prefix + item)

        if os.path.isdir(itemPath):
            new_indent = indent + "    " if isLast else "│   "
            extraxt_files_in_dir_and_print(itemPath, new_indent)

dir_link = input("Enter the dir link, to see all the files present in it:")
extraxt_files_in_dir_and_print(dir_link)
```

Output:

```
(base) PS F:\Internship\Python\practice_programs> python .\9_dir_tree.py
Enter the dir link, to see all the files present in it:F:\Internship\Python
├── Assignment.docx
├── practice_programs
│   ├── 10_iterator.py
│   ├── 11_anti_html.py
│   ├── 1_generatorSql.py
│   ├── 2_abstract.py
│   ├── 3_remDuplicate.py
│   ├── 4_generateArray.py
│   ├── 5_binarySearch.py
│   ├── 6_generator.py
│   ├── 7_email.py
│   ├── 8_squares.py
│   ├── 9_dir_tree.py
│   ├── mySQL_data
│   │   ├── employees.sql
│   │   ├── load_employees.sql
│   │   └── load_salaries.sql
│   └── Notes.txt
└── ~$signment.docx
(base) PS F:\Internship\Python\practice_programs>
```

Question 10: Write an iterator class ReverseIter, that takes a list and iterates it from the reverse Direction.

Solution:

```python
# Write an iterator class ReverseIter, that takes a list and iterates it from the reverse direction.


# an iterator is any object in Python that has an __iter__() method (returns the iterator object itself) and has a __next__() method (returns the next item, or raises StopIteration when done).
# python internally keeps calling __next__() until StopIteration is raised.

class ReverseIter:
    def __init__(self, given_list):
        self.given_list = given_list
        self.curr = len(given_list) - 1

    def __iter__(self):
        return self

    def __next__(self):
        if self.curr >= 0:
            val = self.given_list[self.curr]
            self.curr -= 1
            return val
        else:
            raise StopIteration

my_list = [i for i in range(1, 16)]
final_list = ReverseIter(my_list)

print(', '.join(map(str, final_list)))
```

Output:

```
(base) PS F:\Internship\Python\practice_programs> python .\10_iterator.py
15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
(base) PS F:\Internship\Python\practice_programs>
```

Question 11: Write a program anti_html.py that takes a URL as argument, downloads the html from web and print it after stripping html tags.

Solution:

```
# Write a program anti_html.py that takes a URL as an argument, downloads the HTML from the
web, and prints it after stripping HTML tags.

# To achieve this we can use requests, beautifulsoup to extract contents.

import requests
from bs4 import BeautifulSoup

def extract_text_from_url(url: str):
    try:
        # fetching the page response
        response = requests.get(url)

        # raising  error for bad requests/status
        response.raise_for_status()

        # parsing the  HTML content
        soup = BeautifulSoup(response.text, "html.parser")

        # extracting text and removing HTML tags
        text = soup.get_text(separator="\n", strip=True)

        return text

    except Exception as e:
        print(f"Error: {e}")
        return None

url = 'https://datagrokr.co'
content = extract_text_from_url(url)

print(f'Text extracted from given url: "{url}"\n\n {content}')
```

Output:

```
(base) PS F:\Internship\Python\practice_programs> python .\11_anti_html.py
Text extracted from given url: "https://datagrokr.co"

 IT Strategy and Consulting Services in Bangalore | DataGrokr
Skip to content
Drop us a line at
hello@datagrokr.co
Toggle Navigation
Home
Services
Data Engineering
Full Stack Development
Cloud Engineering
Case Studies
About Us
About DataGrokr
Leadership
Life at DataGrokr
Careers
Internship Program
Contact Us
```

•
•
•

```
Full Stack Development
Accelerate your web and mobile application development with our..
Cloud Engineering
Every company’s journey to the cloud is different..
Life At DG
A thriving atmosphere is important to DataGrokr
since it helps our staff be as motivated as possible.
Comprehensive Health Care Benefits
Flexible Work Environment
Attractive Pay
Rewards & recognition
Learning & Development
Recreation room
It would be great to hear from you! If you got any questions, We are looking forward to hearing from you!
Get In Touch
INDIA
5th Floor, Bel Air Drive, Bellary Road Ganga Nagar Near Mekhri Circle, Bengaluru, Karnataka 560032
USA
340 S Lemon Ave, #4082 Walnut, CA 91789
Follow Us
© Copyright  2025 |   DataGrokr | All Rights Reserved |
Privacy Policy
Page load link
Go to Top
(base) PS F:\Internship\Python\practice_programs>
```