# CBSE Computer Science Practicals

- Name : Aahnik Daw
- Class : XII-E
- School : Kalyani Public School, Barasat, West Bengal
- Roll :
- Find this online : https://github.com/aahnik/cbse-xii-cs-proj/

---

Scan this QR Code to visit the page online

---

## CBSE Computer Science Practicals

### Python

i. A program to calculate the n-th term of Fibonacci series using function.
ii. Program to search any word in given string/sentence using function.
iii. Read a text file line by line and display each word separated by a # .
iv. Read a text file and display the number of vowels/consonants/uppercase/lowercase characters and digits in the file.
v. Read a text file and display the largest word and maximum number of characters present in a line from text file.
vi. Write a program using Dictionary and Text file to store roman numbers and find their equivalent.
vii. Write a menu driven program to perform read and write operations using a text file called "student.txt" counting student roll_no, name and address.
viii. Create a binary file with name and roll number. Search for a given roll number and display the name, if not found display appropriate message.
ix. Create a binary file with roll number, name and marks. Input a roll number and update the marks.
x. Remove all the lines that contain the character `a' in a file and write it to another file.
xi. Write a random number generator that generates random numbers between 1 and 6 (simulates a dice).
xii. Take a sample of ten phishing e-mails (or any text file) and find most commonly occurring word(s).
xiii. Program to create CSV file and store empno,name,salary and search any empno and display name,salary and if not found appropriate message.
xiv. Program to implement Stack in Python using List.
xv. Program to implement Queue in Python using List.
xvi. Program to create 21 Stick Game so that computer always wins
xvii. Program to connect with database and store record of employee and display records.
xviii. Program to connect with database and search employee number in table employee and display record, if empno not found display appropriate message.
xix. Program to connect with database and update the employee record of entered empno.
xx. Program to connect with database and delete the record of entered employee number.
Helper scripts

### SQL

i. To display the all information of Sales department.
ii. To display all information about the employees whose name starts with 'K'.
iii. To list the name of female employees who are in Finance department.
iv. To display name and sex of all the employees whose age is in the range of 40 to 50 in ascending order of their name.
v. To count the number of female employees with age greater than 20 and who are in Accounts department.
vi. To display the name of all Games with their GCodes.
vii. To display details of those games which are having PrizeMoney more than 7000.
viii. To display the content of the GAMES table in ascending order of ScheduleDate.
ix. To display sum of PrizeMoney for each of the Numberof participation groupings ( as shown in column number 2 or 4).
x. To display the sum of prize money of all games.
xi. Display the sum of all Loan Amount whose Interest rate is greater than 10.
xii. Display the Maximum Interest from Loans table.
xiii. Display the count of all loan holders whose name ends with 'SHARMA'.
xiv. Display the count of all loan holders whose Interest is NULL.
xv. Display the Interest-wise details of Loan Account Holders.
xvi. Display the Interest-wise details of Loan Account Holders with at least 10 installments remaining
xvii. Display the Interest-wise count of all loan holders whose Installment due is more than 5 in each group.
xviii. Add one more column name 'Address' to the LOANS table.
xix. Reduce Interest rate by 1 of all loan holders whose Interest is not NULL.
xx. Delete the record of customer whose account number is 105.

# Python

## i. A program to calculate the n-th term of Fibonacci series using function.

***Code***

```python
''' A program to calculate the n-th term of Fibonacci series using function.
'''


def fibo(n: int, memo: dict = {}) -> int:
    ''' The name 'Fibonacci' is due to a 13th-century Italian mathematician
    Leonardo of Pisa, who later came to be known as Fibonacci.
    However, what we popularly call 'Fibonacci numbers' find their earliest
mention in the 2nd century BCE work of Acharya Pingala.

    By definition,the 0-th term of the series is zero, and the 1-st term is
1.
    Any other term is the sum of previous two terms.

    This function uses the concept of memoization to decrease time
complexity
    and increase speed.

    Returns the n-th term of fibbonaci series. Returns -1 for invalid input.

    Args:
        n (int): the term

    Returns:
        int: the nth fibonacci number
    '''
    if n in memo:
        return memo[n]

    if n < 0:
        print(f'Invalid Input \n{fibo.__doc__}')
        raise ValueError('You cannot calculate fibonacci number for n < 0')

    if n <= 2:
        return 1

    memo[n] = fibo(n-1, memo) + fibo(n-2, memo)

    return memo[n]


if __name__ == "__main__":
    # Testing whether the function works correctly

    assert fibo(1) == 1
    assert fibo(2) == 1
    assert fibo(10) == 55
    assert fibo(20) == 6765
```

```
    print(f'The 50th fibonacci number is {fibo(50)}')
    print(f'The 100th fibonacci number is {fibo(100)}')

```

***Output***

```

> python -i q01_fibo.py
The 50th fibonacci number is 12586269025
The 100th fibonacci number is 354224848179261915075
>>> fibo(1)
1
>>> fibo(20)
6765
>>> fibo(34)
5702887
>>> fibo(69)
117669030460994
>>> fibo(475)
831082459908702935293955784701120993704369028200651613859972830080739980541065544674812034151699525
>>>

```

## ii. Program to search any word in given string/sentence using function.

***Code***

```
''' Program to search any word in given string/sentence using function
'''


def search(string: str, word: str) -> list:
    ''' Searches given word in a given string and returns search result.

    Args:
        string (str): the given string/sentence in which to search
        word (str): the word to search

    Returns:
        list: list containing indexes of occurence of the word (empty if not found)
    '''

    index = -1
    result = []

    while True:
        index = string.find(word, index+1)
        if index == -1:
            break
        result.append(index)

    return result


if __name__ == "__main__":
    # Testing whether the function works correctly
    assert(search('I am a donkey', 'donkey') == [7])
    assert(search('Foo bar foo bar spam egg', 'bar') == [4, 12])
    assert(search('Bharat Mahan', 'pakistan') == [])

```

***Output***

```

> python -i q02_wordSearch.py
>>> search('I love Computer Science','love')
[2]
>>> search('FAANG rules the world','India')
[]
>>> search('Aeio u aeio aieo','a')
[7, 12]
>>>

```

## iii. Read a text file line by line and display each word separated by a # .

***Code***

```
''' Read a text file line by line and display each word separated by a # .
'''


path = input('Enter path of the file to read \n >>> ')

try:
    with open(path, 'r') as file:
        while True:
            line = file.readline()
            if line == '':
                break
            for word in line.split():
                print(word, end='#')

except FileNotFoundError:
    print("Sorry ! File does not exist")

finally:
    print("Done")

```

***Output***

```

> python q03_textRead.py
Enter path of the file to read
 >>> data/text.txt
RSA#algorithm#From#Simple#English#Wikipedia,
kfksf#kjkfjsjflksjk#fkjsdlif#jk#kjjfkljslkfjjklj#ksdfkjskl#l#lkjijfiijkkdflij
grofkjporglrj#kljrjiodngl#pkjgjlj#kjglkjf#jlgkmflj#iodjglkjg#jgldjgljglmtioj5
o9m#okptg#;jgojg#oglg;od;
```

## iv. Read a text file and display the number of vowels/consonants/uppercase/lowercase characters and digits in the file.

***Code***

```
''' Read a text file and display the number of
vowels/consonants/uppercase/lowercase
characters and digits in the file.
'''


def analyse(path: str) -> dict:
    ''' Analyses a text file and displays the number of
vowels/consonants/uppercase/lowercase characters and digits in the file.

    Args:
        path (str): path of the file to analyse

    Returns:
        dict: dictionary containing the analysis
    '''

    with open(path, 'r') as file:
        content = file.read()

    analysis = {'vowel': 0,
                'consonant': 0,
                'uppercase': 0,
                'lowercase': 0,
                'digit': 0
                }

    def count(categ: str) -> None:
        '''
        Helper function to count
        '''
        analysis[categ] += 1

    for chr in content:
        if chr.isupper():
            count('uppercase')
        if chr.islower():
            count('lowercase')
        if chr.isdigit():
            count('digit')

        if chr.isalpha():
            if chr.lower() in 'aeiou':
                count('vowel')
            else:
                count('consonant')

    return analysis

```

```
48
49 if __name__ == "__main__":
50     # Test the function
51     result = analyse(input('Enter file path\n>>> '))
52     print(result)
53
```

***Output***

```
1 > python q04_txtalyser.py
2 Enter file path
3 >>> data/text.txt
4 {'vowel': 4388, 'consonant': 8344, 'uppercase': 535, 'lowercase': 12182,
  'digit': 341}
```

## v. Read a text file and display the largest word and maximum number of characters present in a line from text file.

***Code***

```
''' Read a text file and display the largest word and maximum number of characters
present in a line from text file.
'''


def analyse(path: str) -> None:
    ''' Analyses a text file and display the largest word and maximum number of characters present in a line from text file.

    Args:
        path (str): path of the file to analyse
    '''

    with open(path, 'r') as file:
        content = file.read()

    for line_no, line in enumerate(content.splitlines(), start=1):

        if len(line) != 0:
            print(
                f'''Line:{line_no} Character Count : {len(line)}''', end='\t')
            words = line.split()
            if words:
                print(f'Largest Word: {max(words)}')
            else:
                print('This line has no words')

        else:
            print(f'Line:{line_no} Empty Line')


if __name__ == "__main__":
    # Run the function
    analyse(input('Enter file path\n>>> '))

```

***Output***

```

Enter file path
>>> data/text.txt
Line:1 Empty Line
Line:2 Character Count : 13     Largest Word: algorithm
Line:3 Character Count : 52     Largest Word: the
Line:4 Character Count : 32     Largest Word: to
Line:5 Empty Line
Line:6 Character Count : 197    Largest Word: write
```

```
Line:7 Character Count : 566    Largest Word: when
Line:8 Empty Line
Line:9 Character Count : 263    Largest Word: with
Line:10 Empty Line

```

## vi. Write a program using Dictionary and Text file to store roman numbers and find their equivalent.

***Code***

```
''' Write a program using Dictionary and Text file to store roman numbers and find their equivalent.
'''


# base roman numbers and their integer equivalents
ROMAN = {
    'I': 1,
    'V': 5,
    'X': 10,
    'L': 50,
    'C': 100,
    'D': 500,
    'M': 1000,
}


def parse_roman(roman_num: str) -> int:
    ''' Parses a string which is roman numeral and returns equivalent integer.

    Args:
        roman_num (str): Roman numeral to parse

    Raises:
        ValueError: Invalid character in roman numeral
        ValueError: Character occured more than 3 times consecutivel
        ValueError: Invalid roman numeral, incorrect subtractive notation

    Returns:
        int: Integer equivalent
    '''

    # stripping any space
    roman_num = roman_num.strip()

    # convert string to uppercase
    roman_num = roman_num.upper()

    # list of parsed characters
    prev = []

    # total is the value to be returned
    total = 0

    # checker for valid roman string
    largest = 0

    # iterating the roman numeral from right to left
    for chr in roman_num[::-1]:

```

```
        # get the integer value of the character, None if not in ROMAN
        curr_num = ROMAN.get(chr)

        # if current character does not exist in ROMAN dictionary
        if not curr_num:
            raise ValueError(f'Invalid character "{chr}" in roman numeral')

        # list of last 3 characters parsed
        last3 = prev[-3:]

        # if last 3 characters exist
        if len(last3) == 3:
            # if all of the last 3 characters are same as current one
            if all(chr == last for last in last3):
                raise ValueError(
                    f'Invalid roman numeral, "{chr}" occured more than 3 times consecutively')

        # if atleast one character have been already parsed
        if prev:
            # numeric value of last character parsed
            last_num = ROMAN.get(prev[-1])

            # if last character is numerically smaller or equal to current one
            if last_num <= curr_num:
                total += curr_num

                # checking validity of roman string
                if curr_num > largest:
                    largest = curr_num
                elif curr_num < largest:
                    raise ValueError(
                        'Invalid roman numeral, incorrect subtractive notation')
            else:
                total -= curr_num

        # parsing the first character, ie the last character of the roman string
        else:
            total += curr_num
            largest = curr_num

        prev.append(chr)
    return total


if __name__ == "__main__":
    # Checking whether our algorithm passes all test cases

    with open('data/romans.txt') as file:
        for line in file:
            roman, decimal = line.split(',')
            print(f'\nTesting if "{roman}" is same as {decimal.strip()}')
            try:
                assert parse_roman(roman) == int(decimal)
                print('True')
```

```
            except ValueError as err:
                print(err)
            except AssertionError:
                print(f'False. The correct decimal is
{parse_roman(roman)}')

```

***Output***

```

> cat data/romans.txt
xii,12
v,5
c,100
cii,102
iiv,3
xv,12

> py -i q06_roman.py

Testing if "xii" is same as 12
True

Testing if "v" is same as 5
True

Testing if "c" is same as 100
True

Testing if "cii" is same as 102
True

Testing if "iiv" is same as 3
Invalid roman numeral, incorrect subtractive notation

Testing if "xv" is same as 12
False. The correct decimal is 15
>>>
>>> parse_roman('xvii')
17
>>> parse_roman('ii')
2
>>> parse_roman('iiii')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/aahnik/Projects/cbse-xii-cs-
proj/practicals/python/q06_roman.py", line 64, in parse_roman
    raise ValueError(
ValueError: Invalid roman numeral, "I" occured more than 3 times
consecutively
>>> parse_roman('c')
100
>>> parse_roman('cxvii')
117
>>>

```

## vii. Write a menu driven program to perform read and write operations using a text file called “student.txt” counting student roll_no, name and address.

***Code***

```
''' Write a menu driven program to perform read and write operations using
a text file
called “student.txt” counting student roll_no, name and address.
'''


import os
from utils import drive_menu

filename = ''


def init(path: str) -> None:
    ''' Creates an file `student.txt` in desired directory, if not exists.

    Args:
        path (str): Directory path
    '''

    try:
        os.makedirs(path)
        print("directory created")
    except FileExistsError:
        print("directory exists")

    global filename
    filename = os.path.join(path, 'student.txt')

    if not os.path.isfile(filename):
        # create file if does not exist
        with open(filename, 'w+') as file:
            file.write('Roll,Name,Address')
            # writing the headers in first line ( line no = 0)
            print("file `student.txt` created")
            return

    print("file `student.txt` exists in desired directory")


def search_student(roll: int) -> list:
    ''' Searches the roll no. in the text file.

    Args:
        roll (int): The roll number of student

    Returns:
        list: The record list which looks like [roll,name,address]
            None if student's record is absent
    '''
```

```
    with open(filename, 'r') as file:
        content = file.read()

    lines = content.splitlines()

    def record(line): return line.split(',')

    for line in lines:
        if record(line)[0] == str(roll):
            return record(line)

    return None


def record_student() -> None:
    ''' Records a new student in the text file.
        - Roll numbers must be unique.
        - If roll number already exists, returns False
    '''
    try:
        roll = int(input("Enter Student's roll number\n>>> "))
    except ValueError:
        print("Roll number must be an Integer")
        return
    name = input("Enter name\n>>> ")
    address = input("Enter address\n>>> ").replace('\n', ';')
    # new line in address is not allowed

    if roll <= 0:
        print('Invalid roll no')
        return
    if search_student(roll):
        print('Student already exists')
        return

    with open(filename, 'a') as file:
        file.write(f'\n{roll},{name},{address}')
        # multi line address is converted to single line
        print('Successfully Recorded')


def read_data() -> None:
    ''' Displays the details of the student searhced
    '''
    roll = input("Enter roll no. to search\n>>> ")
    record = search_student(roll)
    if not record:
        print("Record not found")
    else:
        print(f'''
        Name : {record[1]}
        --------------------
        Roll no. : {record[0]}

        Address: {record[2]}
        ''')

```

```

def display_all() -> None:
    ''' Display all records.
    '''
    with open(filename, 'r') as file:
        print(file.read())


def main():
    ''' Drive the application.
    '''

    path = input('Enter directory path to store/retrieve data\n>>> ')
    init(path)

    menus = {}
    menus['1'] = {'desc': 'Add new student',
                  'func': record_student}
    menus['2'] = {'desc': 'Display details of all students',
                  'func': display_all}
    menus['3'] = {'desc': 'Search student by roll no',
                  'func': read_data}
    drive_menu('Student Management Portal', menus)


if __name__ == "__main__":
    main()

```

***Output***

```

> python q07_student.py
Enter directory path to store/retrieve data
>>> data/students.txt
directory created
file `student.txt` created

Press [ENTER] to continue or CTRL+C to quit

```

*( the screen gets cleared at this point and menu is displayed )*

```
        MENU for Student Management Portal

╒═══════════╤═══════════════════════════════╕
│   Choice  │ Description                   │
╞═══════════╪═══════════════════════════════╡
│         1 │ Add new student               │
├───────────┼───────────────────────────────┤
│         2 │ Display details of all students │
├───────────┼───────────────────────────────┤
│         3 │ Search student by roll no     │
╘═══════════╧═══════════════════════════════╛
    Enter your choice or X to quit

```

```
>>>
```

*( the user chooses menu 1 )*

```
>>> 1
Enter Student's roll number
>>> 45
Enter name
>>> Horrible Haru
Enter address
>>> Mars
Successfully Recorded

Press [ENTER] to continue or CTRL+C to quit
```

*( the screen gets cleared at this point and menu is re-displayed )*

```
>>> 3
Enter roll no. to search
>>> 2
Record not found

Press [ENTER] to continue or CTRL+C to quit
```

*( the screen gets cleared at this point and menu is re-displayed )*

```
>>> 2
Roll,Name,Address
45,Horrible Haru,Mars
23,Asdff,jklls
```

*( the screen gets cleared at this point and menu is re-displayed )*

```
>>> 3
Enter roll no. to search
>>> 45

        Name : Horrible Haru
        --------------------
        Roll no. : 45

        Address: Mars


Press [ENTER] to continue or CTRL+C to quit
```

## viii. Create a binary file with name and roll number. Search for a given roll number and display the name, if not found display appropriate message.

***Code***

```python
''' Create a binary file with name and roll number. Search for a given roll
number and display the name, if not found display appropriate message.
'''


import pickle
import os
from utils import drive_menu

students = {}
filename = ''


def init(path: str) -> None:
    ''' Load the file. If file does not exist, creates it.

    Args:
        path (str): file path
    '''

    global students
    if not os.path.isdir(path):
        os.makedirs(path)
    global filename
    filename = os.path.join(path, 'student.bin')
    if not os.path.isfile(filename):
        with open(filename, 'wb') as file:
            pickle.dump(students, file)
    else:
        with open(filename, 'rb') as file:
            students = pickle.load(file)


def record() -> None:
    ''' Record a new student.
    '''
    roll = input('Enter roll: ')
    name = input('Enter name: ')
    student = {roll: name}
    try:
        students.update(student)
        with open(filename, 'wb') as file:
            pickle.dump(students, file)
        print('Successfully recorded student')
    except Exception as e:
        print(f'Failed to record student due to error \n {e}')


def search() -> None:
```

```
    ''' Search for an existing student.
    '''
    roll = input('Enter roll to search student: ')
    try:
        print(f'Student found : {students[roll]}')
    except KeyError:
        print('Student not found in records')


def main():
    ''' Driving the app.
    '''

    path = input('Enter directory path to store/retrieve data\n >>> ')
    init(path)

    menus = {}

    menus['1'] = {'desc': 'Record new student',
                  'func': record}
    menus['2'] = {'desc': 'Search student by roll',
                  'func': search}

    drive_menu('Student Management', menus)


if __name__ == "__main__":
    main()

```

***Output***

```
> python q08_studentBin.py
Enter directory path to store/retrieve data
 >>> data

Press [ENTER] to continue or CTRL+C to quit
```

*( the screen gets cleared at this point and menu is displayed )*

```

        MENU for Student Management

╒═══════════╤═══════════════════════════╕
│   Choice  │ Description               │
╞═══════════╪═══════════════════════════╡
│         1 │ Record new student        │
├───────────┼───────────────────────────┤
│         2 │ Search student by roll    │
╘═══════════╧═══════════════════════════╛
    Enter your choice or X to quit

>>>
```

*( the user chooses menu 1 )*

```
>>> 1
Enter roll: 10
Enter name: Gangabati Das
Successfully recorded student

```

*( the screen gets cleared at this point and menu is re-displayed )*

```
>>> 2
Enter roll to search student: 12
Student found : Jack Dorsey

Press [ENTER] to continue or CTRL+C to quit
```

## ix. Create a binary file with roll number, name and marks. Input a roll number and update the marks.

***Code***

```
''' Create a binary file with roll number, name and marks. Input a roll
number and update the marks.
'''


import pickle
from utils import drive_menu
from tabulate import tabulate
import os

# in data folder of current directory
filename = os.path.join('data', 'marks.bin')

students = {}  # dictionary containing students


def load_file() -> None:
    with open(filename, 'rb') as file:
        global students
        students = pickle.load(file)


def write_to_file() -> None:
    with open(filename, 'wb') as file:
        pickle.dump(students, file)


def record_student() -> None:
    global students
    roll, name, marks = input(
        'Enter roll, name and marks seperated by comma\n> ').split(',')
    students[roll] = [name, marks]
    write_to_file()
    print('Sucessfully recorded')


def update_marks() -> None:
    roll, marks = input(
        'Enter roll, and new marks seperated by comma\n ').split(',')
    if roll in students.keys():
        students[roll][1] = marks
        write_to_file()
        print('Sucessfully updated')
    else:
        print('Student does not exist in records')


def display() -> None:
    table = []
    for key, value in students.items():
        table.append([key, value[0], value[1]])
```

```
    print(tabulate(table, tablefmt='fancy_grid',
                   headers=['Roll', 'Name', 'Marks']))


def main():
    ''' Driving the app.
    '''
    if not os.path.isfile(filename):
        write_to_file()
    else:
        load_file()

    menus = {}

    menus['1'] = {'desc': 'Record new student',
                  'func': record_student}
    menus['2'] = {'desc': 'Update marks of existing student',
                  'func': update_marks}
    menus['3'] = {'desc': 'Display all records',
                  'func': display}

    drive_menu('Marks Manager', menus)


if __name__ == "__main__":
    main()

```

***Output***

```
> python q09_marks.py

Press [ENTER] to continue or CTRL+C to quit
```

*( the screen gets cleared at this point and menu is displayed )*

```

        MENU for Marks Manager

╒═══════════╤═══════════════════════════════════╕
│   Choice  | Description                       |
╞═══════════╪═══════════════════════════════════╡
│         1 | Record new student                |
├───────────┼───────────────────────────────────┤
│         2 | Update marks of existing student  |
├───────────┼───────────────────────────────────┤
│         3 | Display all records               |
╘═══════════╧═══════════════════════════════════╛
    Enter your choice or X to quit

>>>
```

*( the user chooses 1)*

```
>>> 1
Enter roll, name and marks seperated by comma
> 13,Jay Bhatt, 80
Sucessfully recorded

Press [ENTER] to continue or CTRL+C to quit
```

*( the screen gets cleared at this point and menu is displayed )*

```
>>> 2
Enter roll, and new marks seperated by comma
 13,90
Sucessfully updated
```

*( the screen gets cleared at this point and menu is re-displayed, and now the user tries an invalid update )*

```
>>> 2
Enter roll, and new marks seperated by comma
 100,90
Student does not exist in records
```

*( the screen gets cleared at this point and menu is re-displayed)*

```


>>> 3
╒═══════╤═══════════╤═══════╕
│   Roll │ Name      │   Marks │
╞═══════╪═══════════╪═══════╡
│      1 │ Hitesham  │      90 │
├───────┼───────────┼───────┤
│     34 │ Ramesha   │      89 │
├───────┼───────────┼───────┤
│     13 │ Jay Bhatt │      90 │
╘═══════╧═══════════╧═══════╛

Press [ENTER] to continue or CTRL+C to quit

```

## x. Remove all the lines that contain the character `a' in a file and write it to another file.

***Code***

```
''' Remove all the lines that contain the character `a' in a file and write
it
to another file.
'''


def move(old: str, new: str):
    ''' The function that does the job

    Args:
        old (str): file path of original file
        new (str): file path of new file
    '''

    with open(old, 'r') as old_file:
        lines = old_file.readlines()
        a_lines = [line for line in lines if 'a' in line]
        not_a_lines = [line for line in lines if 'a' not in line]

    with open(old, 'w') as old_file:
        old_file.writelines(not_a_lines)

    with open(new, 'w') as new_file:
        new_file.writelines(a_lines)


def main():
    old = input('Enter the old file path: ')
    new = input('Enter the new file path: ')
    move(old, new)
    print('Done! ')


if __name__ == "__main__":
    main()

```

***Output***

*Let's first see the original file.*

```
> cat data/lines.txt
A good donkey was grazing
No body took notice
But Andrew was eating ice-cream
Hans was jumping with joy
November is the month of winter
Jacob was lost
```

*Now let's run the program.*

```
> python q10_moveA.py
Enter the old file path: data/lines.txt
Enter the new file path: data/new_lines.txt
Done!
```

*Now let's see the old file again.*

```
> cat data/lines.txt
No body took notice
November is the month of winter
```

*The new file is as follows.*

```
> cat data/new_lines.txt
A good donkey was grazing
But Andrew was eating ice-cream
Hans was jumping with joy
Jacob was lost
```

## xi. Write a random number generator that generates random numbers between 1 and 6 (simulates a dice).

***Code***

```
''' Write a random number generator that generates random numbers
between 1 and 6 (simulates a dice).
'''

import random

while True:
    print('Throwing a dice ...')
    print(random.randint(1, 6))
    choice = input('Press ENTER to throw again, or X to quit')
    if choice == 'X':
        break

```

***Output***

```
> python q11_dice.py
Throwing a dice ...
3
Press ENTER to throw again, or X to quit
Throwing a dice ...
2
Press ENTER to throw again, or X to quit
Throwing a dice ...
4
Press ENTER to throw again, or X to quit
Throwing a dice ...
3
Press ENTER to throw again, or X to quitX
```

## xii. Take a sample of ten phishing e-mails (or any text file) and find most commonly occurring word(s).

***Code***

```
''' Take a sample of ten phishing e-mails (or any text file) and find most
commonly occurring word(s)

the file `data/phishing.txt` contains the text extracted from 10 phishing
emails
the samples are taken from https://security.berkeley.edu/
'''

from collections import Counter
from tabulate import tabulate
import os

# in data folder of current directory
filename = os.path.join('data', 'phishing.txt')

with open(filename, 'r') as file:
    content = file.read()

# take all the words
words = Counter(content.split())

# count the most common words
most_common = words.most_common(20)

print('Top 20 Commonly used words\n')
print(tabulate(most_common, tablefmt='fancy_grid'))

```

***Output***

> **Note** : the file `data/phishing.txt` contains the text extracted from 10 phishing emails
> the samples are taken from https://security.berkeley.edu/

```
> python q12_phishy.py
Top 20 Commonly used words

╒═══════════╤════╕
│ to        │ 32 │
├───────────┼────┤
│ bank      │ 29 │
├───────────┼────┤
│ the       │ 24 │
├───────────┼────┤
│ immediate │ 24 │
├───────────┼────┤
│ urgent    │ 20 │
├───────────┼────┤
│ you       │ 14 │
├───────────┼────┤
│ for       │ 10 │
```

```
├──────────────┼─────┤
│ in           │  10 │
├──────────────┼─────┤
│ my           │   9 │
├──────────────┼─────┤
│ a            │   9 │
├──────────────┼─────┤
│ is           │   8 │
├──────────────┼─────┤
│ of           │   8 │
├──────────────┼─────┤
│ have         │   7 │
├──────────────┼─────┤
│ will         │   7 │
├──────────────┼─────┤
│ I            │   7 │
├──────────────┼─────┤
│ ID           │   7 │
├──────────────┼─────┤
│ email        │   6 │
├──────────────┼─────┤
│ an           │   6 │
├──────────────┼─────┤
│ from         │   6 │
├──────────────┼─────┤
│ on           │   6 │
└──────────────┴─────┘
```

## xiii. Program to create CSV file and store empno,name,salary and search any empno and display name,salary and if not found appropriate message.

***Code***

```
''' Program to create CSV file and store empno,name,salary and search any empno and
display name,salary and if not found appropriate message.
'''

import os
import csv
from utils import drive_menu

filename = os.path.join('data', 'employee.csv')


def init():
    ''' Create files if not present '''
    if not os.path.isfile(filename):
        with open(filename, 'w') as file:
            file.write('empno,name,salary')


def store() -> None:
    ''' Store the record of employee '''
    record = input('Enter empno, name and salary seperated by comma\n>>> ')
    with open(filename, 'a') as file:
        file.write(f'\n{record}')
    print('Employee recorded')


def retrieve() -> None:
    ''' Retrieve the record of existing employee '''
    empno = input('Enter empno to search\n>>> ')
    with open(filename, 'r') as file:
        employees = csv.DictReader(file)
        for row in employees:
            if row['empno'] == empno:
                print(f"Name: {row['name']}\nSalary: {row['salary']}")
                return
        print('Employee not found in records')


def main():
    init()
    menus = {}
    menus['1'] = {'desc': 'Store new Employee', 'func': store}
    menus['2'] = {'desc': 'Search Employee', 'func': retrieve}
    drive_menu('Employee Management', menus)


if __name__ == "__main__":
    main()
```

```
49
```

***Output***

```
1
2          MENU for Employee Management
3
4   ╔═══════════╦═══════════════════════╗
5   ║   Choice  ║ Description           ║
6   ╠═══════════╬═══════════════════════╣
7   ║         1 ║ Store new Employee    ║
8   ╠═══════════╬═══════════════════════╣
9   ║         2 ║ Search Employee       ║
10  ╚═══════════╩═══════════════════════╝
11      Enter your choice or X to quit
12
13  >>>
14
```

*( the user chooses 2 )*

```
1  >>> 2
2  Enter empno to search
3  >>> 13
4  Employee not found in records
5
6  Press [ENTER] to continue or CTRL+C to quit
```

*( the screen is cleared, and menu re-displayed )*

```
1  >>> 1
2  Enter empno, name and salary seperated by comma
3  >>> 12, Akshay Kumar, 10000
4  Employee recorded
5
6  Press [ENTER] to continue or CTRL+C to quit
```

*( the screen is cleared, and menu re-displayed )*

```
1  >>> 2
2  Enter empno to search
3  >>> 12
4  Name:  Akshay Kumar
5  Salary:  10000
6
7  Press [ENTER] to continue or CTRL+C to quit
```

## xiv. Program to implement Stack in Python using List.

***Code***

```
''' Program to implement Stack in Python using List
'''

from tabulate import tabulate
from utils import drive_menu


class Stack():
    def __init__(self, limit=9999) -> None:
        self.stk = []
        if (type(limit) != int) or (limit <= 0):
            print(f'Invalid Limit : must be int greater than zero')
            return
        self.limit = limit

    def is_empty(self):
        return self.stk == []

    def peek(self):
        if self.is_empty():
            print('Nothing to peek: Stack is empty')
            return
        return self.stk[len(self.stk)-1]

    def push(self, data=None):
        if not data:
            data = input('Enter data to push: ')
        if len(self.stk) == self.limit:
            print('Stack Overflow : Size of stack exceeded limit')
            return
        self.stk.append(data)

    def pop(self):
        if self.is_empty():
            print('Stack Underflow : Cannot pop from empty stack')
            return
        return self.stk.pop()

    def display(self):
        if self.is_empty():
            return
        else:
            print('top')
            print(tabulate([[item] for item in self.stk[::-1]],
                           tablefmt='fancy_grid'))


def main():
    stack = Stack()
    menus = {}
    menus['1'] = {'desc': 'Push', 'func': stack.push}
    menus['2'] = {'desc': 'Pop', 'func': stack.pop}
```

```
    menus['3'] = {'desc': 'Peek', 'func': stack.peek}
    menus['4'] = {'desc': 'Display', 'func': stack.display}
    drive_menu('Stack Operations', menus)


if __name__ == "__main__":
    main()

```

***Output***

```
        MENU for Stack Operations

╒═══════════╤═══════════════════╕
│   Choice  │ Description       │
╞═══════════╪═══════════════════╡
│         1 │ Push              │
├───────────┼───────────────────┤
│         2 │ Pop               │
├───────────┼───────────────────┤
│         3 │ Peek              │
├───────────┼───────────────────┤
│         4 │ Display           │
╘═══════════╧═══════════════════╛
    Enter your choice or X to quit

>>>
```

*( during the execution of the program the screen is cleared and the menu is displayed several times, for an aesthetic experience. To keep stuff clean, the same menu is not being repeated here)*

```
>>> 1
Enter data to push: hoch

>>> 1
Enter data to push: poch

>>> 1
Enter data to push: ghosh

>>> 3
ghosh

>>> 4
top
╒═══════╕
│ ghosh │
├───────┤
│ poch  │
├───────┤
│ hoch  │
╘═══════╛

>>> 2
ghosh

```

```
>>> 4
top
 ┍━━━━━━┑
 │ poch │
 ├──────┤
 │ hoch │
 ┕━━━━━━┙

```

## xv. Program to implement Queue in Python using List.

***Code***

```
''' Program to implement Queue in Python using List
'''

from tabulate import tabulate
from utils import drive_menu


class Queue():
    def __init__(self, length=9999) -> None:
        self.q = []
        if (type(length) != int) or (length <= 0):
            print(f'Invalid Limit : must be int greater than zero')
            return
        self.length = length

    def is_empty(self):
        return self.q == []

    def front(self):
        if self.is_empty():
            print('Empty Queue : No front element')
            return
        return self.q[0]

    def rear(self):
        if self.is_empty():
            print('Empty Queue : No rear element')
            return
        return self.q[len(self.q)-1]

    def enqueue(self, data=None):
        if not data:
            data = input('Enter data to enqueue: ')
        if len(self.q) == self.length:
            print('Queue Overflow : Size of queue exceeded length')
            return
        self.q.append(data)

    def dequeue(self):
        if self.is_empty():
            print('Queue Underflow : Empty queue, nothing to dequeue')
            return
        rm = self.q[0]
        del self.q[0]
        return rm

    def display(self):
        if self.is_empty():
            return
        print('front')
        print(tabulate([self.q], tablefmt='fancy_grid'))

```

```
53
54  def main():
55      qu = Queue()
56      menus = {}
57      menus['1'] = {'desc': 'Enqueue', 'func': qu.enqueue}
58      menus['2'] = {'desc': 'Dequeue', 'func': qu.dequeue}
59      menus['3'] = {'desc': 'Peek (front)', 'func': qu.front}
60      menus['4'] = {'desc': 'Rear', 'func': qu.rear}
61      menus['5'] = {'desc': 'Display', 'func': qu.display}
62      drive_menu('Queue Operations', menus)
63
64
65  if __name__ == "__main__":
66      main()
67
```

***Output***

```
1
2           MENU for Queue Operations
3
4   ╒═══════════╤═══════════════════╕
5   │   Choice  │ Description       │
6   ╞═══════════╪═══════════════════╡
7   │         1 │ Enqueue           │
8   ├───────────┼───────────────────┤
9   │         2 │ Dequeue           │
10  ├───────────┼───────────────────┤
11  │         3 │ Peek (front)      │
12  ├───────────┼───────────────────┤
13  │         4 │ Rear              │
14  ├───────────┼───────────────────┤
15  │         5 │ Display           │
16  ╘═══════════╧═══════════════════╛
17      Enter your choice or X to quit
18
19  >>>
20
```

*( during the execution of the program the screen is cleared and the menu is displayed several times, for an aesthetic experience. To keep stuff clean, the same menu is not being repeated here)*

```
1   >>> 1
2   Enter data to enqueue: utopia
3
4   >>> 1
5   Enter data to enqueue: distopia
6
7   >>> 1
8   Enter data to enqueue: ultadanga
9
10  >>> 5
11  front
12  ╒═══════╤═══════════╤═══════════╕
13  │ utopia │ distopia │ ultadanga │
14  ╘═══════╧═══════════╧═══════════╛
```

```
>>> 3
utopia

>>> 4
ultadanga

>>> 2
utopia

>>> 5
front
╒═══════════╤═══════════╕
│ distopia  │ ultadanga │
╘═══════════╧═══════════╛

```

## xvi. Program to create 21 Stick Game so that computer always wins

***Code***

```
''' Program to create 21 Stick Game so that computer always wins

21 Matchstick Puzzle game
- In this Puzzle there are 21 Match Sticks.
- You and Computer will pick up the sticks one by one.
- Sticks can be picked from 1 to 4.
- The who, picked up the last stick, is the loser.
'''

from utils import drive_menu, clear_screen


def display_rules():
    print(__doc__)


def game():
    ''' The game '''

    sticks = 21

    while sticks != 1:
        clear_screen()
        print(f'Currently there are {sticks} sticks')

        # ensure user enters an integer
        try:
            user_choice = int(input('Choose from 1 to 4 sticks\n>>> '))
        except ValueError:
            print('You have entered a non integer value')
            return 'Game Aborted'

        # ensure user choice is valid
        try:
            assert user_choice in (1, 2, 3, 4)
        except AssertionError:
            print('You can choose only between 1 to 4 sticks')
            return 'Game Aborted'

        # calculate remaining no. of sticks
        sticks -= user_choice
        print(f'Now we have {sticks} sticks left. Its my turn now')

        # strategy to win
        computer_choice = 5-user_choice
        sticks -= computer_choice
        print(f'I have picked {computer_choice} sticks')

    print('\nThere is only one stick left. By the rule, you loose 😁')
    print('Better Luck next time !\n')
    return 'Game Ended'
```

```
52
53
54  def main():
55      menus = {}
56      menus['1'] = {'desc': 'Play the game', 'func': game}
57      menus['2'] = {'desc': 'See the rules', 'func': display_rules}
58      drive_menu('21 Stick Game', menus)
59
60
61  if __name__ == "__main__":
62      main()
63
```

***Output***

Rules of the 21 Matchstick Puzzle game

- In this Puzzle there are 21 Match Sticks.
- You and Computer will pick up the sticks one by one.
- Sticks can be picked from 1 to 4.
- The who, picked up the last stick, is the loser.

*( during the execution of the program the screen is cleared and the menu is displayed several times, for an aesthetic experience. To keep stuff clean, the same menu is not being repeated here)*

```
1          MENU for 21 Stick Game
2
3  ╔═══════════╤═══════════════════╗
4  │   Choice  | Description       |
5  ╠═══════════╪═══════════════════╣
6  │         1 | Play the game     |
7  ├───────────┼───────────────────┤
8  │         2 | See the rules     |
9  ╚═══════════╧═══════════════════╝
10     Enter your choice or X to quit
11
12 >>> 1
13
14 Press [ENTER] to continue or CTRL+C to quit
15
16 Currently there are 21 sticks
17 Choose from 1 to 4 sticks
18 >>> 5
19 You can choose only between 1 to 4 sticks
20 Game Aborted
21
22 Press [ENTER] to continue or CTRL+C to quit
23
24 # MENU IS RE-DISPLAYED
25 >>> 1
26
27 Press [ENTER] to continue or CTRL+C to quit
28
29 Currently there are 21 sticks
30 Choose from 1 to 4 sticks
31 >>> 2
32 Now we have 19 sticks left. Its my turn now
```

```
I have picked 3 sticks

Currently there are 16 sticks
Choose from 1 to 4 sticks
>>> 4
Now we have 12 sticks left. Its my turn now
I have picked 1 sticks


Currently there are 11 sticks
Choose from 1 to 4 sticks
>>> 2
Now we have 9 sticks left. Its my turn now
I have picked 3 sticks


Currently there are 6 sticks
Choose from 1 to 4 sticks
>>> 4
Now we have 2 sticks left. Its my turn now
I have picked 1 sticks

There is only one stick left. By the rule, you loose 😔
Better Luck next time !

Game Ended

```

## xvii. Program to connect with database and store record of employee and display records.

***Code***

```
''' Program to connect with database and store record of employee and
display records.
'''

from sqlTor import SqlTor
import mysql.connector
from mysql.connector import errorcode
from tabulate import tabulate
from utils import clear_screen


def input_employee_details():
    while True:
        try:
            name = input('name: ')
            assert 5 < len(name) < 20
            department = input('department: ')
            assert len(department) < 20
            salary = int(input('salary: '))
            assert salary >= 0
        except Exception as err:
            print(f'Please enter valid details. {err}')
        else:
            break

    return name, department, salary


def input_emp_id():
    while True:
        try:
            emp_id = int(input('Enter employee id: '))
        except ValueError:
            print('Invalid Employee id. It must be integer.')
        else:
            break
    return emp_id


def create_table(cursor):
    ''' Takes the cursor object and creates table '''

    table_creation = ("CREATE TABLE employees(\
                      emp_id integer NOT NULL PRIMARY KEY,\
                      name char(20) NOT NULL,\
                      department char(20) NOT NULL,\
                      salary integer NOT NULL);")

    try:
        cursor.execute(table_creation)
    except mysql.connector.Error as err:
```

```
        if err.errno == errorcode.ER_TABLE_EXISTS_ERROR:
            print('table already exists')
        else:
            print(err)
    else:
        print('Created table `employees` successfully')


def display_all(cursor):
    ''' Display all employees '''

    query = "SELECT * FROM employees"

    try:
        cursor.execute(query)
    except Exception as err:
        print(err)
    else:
        employees = cursor.fetchall()
        if employees:
            print(f'''\n\nHere is the list of all employees
            \n{tabulate(employees,tablefmt='fancy_grid',headers=
['emp_id','name','department','salary'])}\n''')
        else:
            print('No employees recorded yet')


def record_new(cursor):
    ''' Record a new employee '''

    print('Enter the details to add new employee.\n')

    emp_id = input_emp_id()

    name, department, salary = input_employee_details()

    insert_employee = f"INSERT INTO employees \
                        VALUES({emp_id},\
                            '{name}','{department}',{salary})"

    try:
        cursor.execute(insert_employee)
    except Exception as err:
        if err.errno == errorcode.ER_DUP_ENTRY:
            print('Duplicate entry. emp_id must be unique.')
    else:
        print('New employee added successfully 😀')


if __name__ == "__main__":

    with SqlTor() as my_con:
        cursor = my_con.cursor()
        create_table(cursor)
        while True:
            clear_screen()
            display_all(cursor)
            print('RECORD NEW EMPLOYEES')
```

```
            record_new(cursor)
            my_con.commit()

```

***Output***

```
> python q17_dbRecord.py
table already exists

Press [ENTER] to continue or CTRL+C to quit

# screen gets cleared

No employees recorded yet
RECORD NEW EMPLOYEES
Enter the details to add new employee.

Enter employee id: 12
name: Hans Chen
department: Sales
salary: 10000
New employee added successfully 😀

Press [ENTER] to continue or CTRL+C to quit

# screen gets cleared

Here is the list of all employees

╒═══════════╤═══════════╤═══════════════╤══════════╕
│   emp_id  │ name      │ department    │   salary │
╞═══════════╪═══════════╪═══════════════╪══════════╡
│       12  │ Hans Chen │ Sales         │    10000 │
╘═══════════╧═══════════╧═══════════════╧══════════╛

RECORD NEW EMPLOYEES
Enter the details to add new employee.

Enter employee id: 13
name: Jay Chandran
department: Coding
salary: 100000000
New employee added successfully 😀

Press [ENTER] to continue or CTRL+C to quit

# screen gets cleared

Here is the list of all employees

╒═══════════╤═══════════╤═══════════════╤═══════════╕
│   emp_id  │ name      │ department    │    salary │
╞═══════════╪═══════════╪═══════════════╪═══════════╡
│       12  │ Hans Chen │ Sales         │     10000 │
├───────────┼───────────┼───────────────┼───────────┤
│       13  │ Jay Chandran │ Coding     │ 100000000 │
╘═══════════╧═══════════╧═══════════════╧═══════════╛

```

```
52
53 RECORD NEW EMPLOYEES
54 Enter the details to add new employee.
55
56 Enter employee id: ^C
57 Interrupt recieved. Quitting.
58
```

## xviii. Program to connect with database and search employee number in table employee and display record, if empno not found display appropriate message.

***Code***

```
''' Program to connect with database and search employee number in table employee
and display record, if empno not found display appropriate message. '''

from utils import clear_screen
from sqlTor import SqlTor
from q17_dbRecord import input_emp_id


def get_employee(cursor) -> tuple or None:
    ''' Input employee id and fetch details of employee.
    Returns a tuple or None if not found '''

    emp_id = input_emp_id()

    query = f'SELECT * FROM employees WHERE emp_id={emp_id}'

    try:
        cursor.execute(query)
    except Exception as err:
        print(err)
    else:
        employees = cursor.fetchall()
        if employees:
            return employees[0]


if __name__ == "__main__":

    with SqlTor() as my_con:
        cursor = my_con.cursor()

        while True:
            clear_screen()
            print('SEARCH EMPLOYEE')
            emp = get_employee(cursor)
            if emp:
                print('Record found 🥰')
                print(f'''
                        name: {emp[1]},
                        department: {emp[2]},
                        salary: {emp[3]}''')
            else:
                print('Employee Not found ☹')

```

***Output***

```
SEARCH EMPLOYEE
```

```
Enter employee id: 12
Record found 🥰

                        name: Hans Chen,
                        department: Sales,
                        salary: 10000

Press [ENTER] to continue or CTRL+C to quit

# screen gets cleared

SEARCH EMPLOYEE
Enter employee id: 100
Employee Not found ☹

Press [ENTER] to continue or CTRL+C to quit
^C
Interrupt recieved. Quitting.

```

## xix. Program to connect with database and update the employee record of entered empno.

***Code***

```
''' Program to connect with database and update the employee record of
entered empno. '''

from utils import clear_screen
from sqlTor import SqlTor

from q18_dbSearch import get_employee
from q17_dbRecord import input_employee_details


def update_employee(cursor):
    ''' Update an employee '''

    emp = get_employee(cursor)

    if not emp:
        print('Employee does not exist.')
        return

    print('Enter new details of employee.')
    name, department, salary = input_employee_details()

    employee_updation = f"UPDATE employees \
                        SET name='{name}',\
                            department='{department}',\
                            salary={salary} \
                        WHERE emp_id={emp[0]};"

    try:
        cursor.execute(employee_updation)
    except Exception as err:
        print(err)
    else:
        print('Update Successful!')


if __name__ == "__main__":
    with SqlTor() as my_con:
        cursor = my_con.cursor()
        while True:
            clear_screen()
            print('UPDATE EMPLOYEE')
            update_employee(cursor)
            my_con.commit()

```

***Output***

```

UPDATE EMPLOYEE
```

```
Enter employee id: 12
Enter new details of employee.
name: Aahnik Daw
department: Machine Learning
salary: 10
Update Successful!

Press [ENTER] to continue or CTRL+C to quit

# screen gets cleared

UPDATE EMPLOYEE
Enter employee id: 100
Employee does not exist.

Press [ENTER] to continue or CTRL+C to quit
^C
Interrupt recieved. Quitting.

```

## xx. Program to connect with database and delete the record of entered employee number.

***Code***

```
''' Program to connect with database and delete the record of entered
employee number. '''

from sqlTor import SqlTor
from utils import clear_screen
from q18_dbSearch import get_employee


def delete_employee(cursor):
    ''' Delete an employee '''

    emp = get_employee(cursor)

    if not emp:
        print('Employee does not exist.')
        return

    employee_deletion = f'DELETE FROM employees WHERE emp_id={emp[0]}'

    try:
        cursor.execute(employee_deletion)
    except Exception as err:
        print(err)
    else:
        print('Successfully deleted.')


if __name__ == "__main__":

    with SqlTor() as my_con:
        cursor = my_con.cursor()
        while True:
            clear_screen()
            print('DELETE EMPLOYEE')
            delete_employee(cursor)
            my_con.commit()

```

***Output***

```
DELETE EMPLOYEE
Enter employee id: 12
Successfully deleted.

Press [ENTER] to continue or CTRL+C to quit

# screen gets cleared

DELETE EMPLOYEE
Enter employee id: 100
```

```
Employee does not exist.

Press [ENTER] to continue or CTRL+C to quit
^C
Interrupt recieved. Quitting.
```

## Helper scripts

**utils.py**

```

''' General purpose utility module, to reduce number of lines of code in solution
Enables my code to be DRY (Dont Repeat Yourself)
'''

import os
from tabulate import tabulate
import sys
import signal


def handle_interrupt(*args):
    print('\nInterrupt recieved. Quitting.')
    sys.exit(0)


def clear_screen():

    # handle user interrupt
    signal.signal(signal.SIGTERM, handle_interrupt)
    signal.signal(signal.SIGINT, handle_interrupt)

    # wait for user to see current screen
    input('\nPress [ENTER] to continue or CTRL+C to quit\n')

    if os.name == 'posix':
        # for Linux and Mac
        os.system('clear')
    else:
        # for Windows
        os.system('cls')


def drive_menu(heading: str, menus: dict) -> None:
    ''' Function to allow a menu driven program

    Args:
        heading (str): heading to be displayed on top of menu
        menus (dict): dictionary of menus containing
        key (menu id) value (another dictionary having `desc` and `func` )
    '''

    table = [[ch, menu['desc']] for ch, menu in menus.items()]
    menu_chart = f'''
        MENU for {heading}
    \n{tabulate(table,tablefmt='fancy_grid',headers=['Choice','Description'])}
    Enter your choice or X to quit
    \n>>> '''
    choice = ''
    while choice != 'X':
```

```
        clear_screen()
        choice = input(menu_chart)
        if choice in menus.keys():
            val = menus[choice]['func']()
            if val:
                print(val)
        elif choice == 'X':
            print('Bye 🤗')
        else:
            print('INVALID CHOICE')

```

**sqlTor.py**

```
'''
An utility module that helps to connect to the my sql database
'''

import mysql.connector
import yaml

# read the config file, and load it into a dict
with open('config.yaml') as f:
    config = yaml.full_load(f)


class SqlTor():
    ''' Context manager to enable easy connection to database
    '''

    def __init__(self) -> None:
        self.conn = mysql.connector.connect(**config)

    def __enter__(self):
        ''' Entry point '''
        if self.conn.is_connected():
            return self.conn
        else:
            raise Exception('Not connected to MySQL')

    def __exit__(self, exception_type, exception_value, traceback):
        ''' Exit '''
        self.conn.close()

```

# SQL

> *the full `emp` table*

## i. To display the all information of Sales department.

```sql
SELECT
    *
FROM
    emp
WHERE
    department = 'Sales';
```

| Id | Name | Age | Department | Sal | Sex |
|---|---|---|---|---|---|
| 1 | Arprit | 62 | Sales | 38000 | M |
| 3 | Kareem | 32 | Sales | 17000 | M |
| 8 | Zareen | 45 | Sales | 28000 | F |
| 10 | Shilpa | 23 | Sales | 22000 | F |
| NULL | NULL | NULL | NULL | NULL | NULL |

## ii. To display all information about the employees whose name starts with 'K'.

```sql
SELECT
    *
FROM
    emp
WHERE
    name LIKE 'K%';

```

| Id | Name | Age | Department | Sal | Sex |
|---|---|---|---|---|---|
| 3 | Kareem | 32 | Sales | 17000 | M |
| 6 | Kettaki | 26 | Finance | 60000 | F |
| 9 | Kush | 29 | Accounts | 32000 | M |

## iii. To list the name of female employees who are in Finance department.

```sql
SELECT
    name
FROM
    emp
WHERE
    sex = 'F' AND department = 'Finance';
```

| name |
| --- |
| Kettaki |
| Ankita |

### iv. To display name and sex of all the employees whose age is in the range of 40 to 50 in ascending order of their name.

```
SELECT
    name, sex
FROM
    emp
WHERE
    age BETWEEN 40 AND 50
ORDER BY name;
```

| name | sex |
| --- | --- |
| Arun | M |
| Zareen | F |

### v. To count the number of female employees with age greater than 20 and who are in Accounts department.

```
SELECT
    COUNT(*) 'female emp older than 20 in accounts'
FROM
    emp></div>
WHERE
    age > 20 AND department = 'Accounts';
```

| female emp older than 20 in |
| --- |
| 3 |

> *the full `games` table*

## vi. To display the name of all Games with their GCodes.

```
SELECT
    gamename, gcode
FROM
    games;
```

| gamename | gcode |
|---|---|
| Carom Board | 101 |
| Badminton | 102 |
| Table Tennis | 103 |
| Chess | 105 |
| Lawn Tennis | 108 |

## vii. To display details of those games which are having PrizeMoney more than 7000.

```
SELECT
    *
FROM
    games
WHERE
    prizemoney > 7000;

```

| GCode | GameName | Number | PrizeMoney | ScheduleDate |
|---|---|---|---|---|
| 103 | Table Tennis | 4 | 8000 | 14-Feb-2021 |
| 105 | Chess | 2 | 9000 | 02-Jan-2021 |
| 108 | Lawn Tennis | 4 | 25000 | 19-Mar-2021 |

## viii. To display the content of the GAMES table in ascending order of ScheduleDate.

```
SELECT
    *
FROM
    games
ORDER BY scheduledate;
```

| GCode | GameName | Number | PrizeMoney | ScheduleDate |
|---|---|---|---|---|
| 105 | Chess | 2 | 9000 | 02-Jan-2021 |
| 102 | Badminton | 2 | 1200 | 12-Dec-2020 |
| 103 | Table Tennis | 4 | 8000 | 14-Feb-2021 |
| 108 | Lawn Tennis | 4 | 25000 | 19-Mar-2021 |
| 101 | Carom Board | 2 | 5000 | 23-Jan-2021 |

**ix. To display sum of PrizeMoney for each of the Numberof participation groupings ( as shown in column number 2 or 4).**

```
SELECT
    number, SUM(prizemoney)
FROM
    games
GROUP BY number;
```

| number | SUM(prizemoney) |
|---|---|
| 2 | 15200 |
| 4 | 33000 |

**x. To display the sum of prize money of all games.**

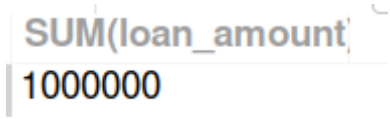
```
SELECT
    SUM(prizemoney)
FROM
    games;

```

| SUM(prizemoney) |
|---|
| 48200 |

> *the full `loans` table*

## xi. Display the sum of all Loan Amount whose Interest rate is greater than 10.

```
SELECT
    SUM(loan_amount)
FROM
    loans
WHERE
    int_rate > 10;
```




## xii. Display the Maximum Interest from Loans table.

```
SELECT
    MAX(int_rate)
FROM
    loans;
```


## xiii. Display the count of all loan holders whose name ends with ‘SHARMA’.

```
SELECT
    COUNT(cust_name)
FROM
    loans
WHERE
    cust_name LIKE '%SHARMA';
```


## xiv. Display the count of all loan holders whose Interest is NULL.

```
SELECT
    COUNT(cust_name)
FROM
    loans
WHERE
    int_rate IS NULL;
```

| COUNT(cust_name) |
| --- |
| 2 |

## xv. Display the Interest-wise details of Loan Account Holders.

```
SELECT
    *
FROM
    loans
ORDER BY interest;
```

| AccNo | Cust_Name | Loan_Amoun | Installments | Int_Rate | Start_Date | Interest | Adress |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 101 | R.K.GUPTA | 300000 | 36 | 11.00 | 19-07-2019 | 1200 | NULL |
| 103 | K.P.JAIN | 300000 | 36 | NULL | 08-03-2017 | 1600 | NULL |
| 102 | S.P.SHARMA | 500000 | 48 | 9.00 | 22-03-2018 | 1800 | NULL |
| 104 | M.P.YADAV | 800000 | 60 | 9.00 | 06-12-2018 | 2250 | NULL |
| 106 | P.SHARMA | 700000 | 60 | 11.50 | 05-06-2018 | 3500 | NULL |
| 107 | K.S.DHALL | 500000 | 48 | NULL | 05-03-2018 | 3800 | NULL |

## xvi. Display the Interest-wise details of Loan Account Holders with at least 10 installments remaining

```
SELECT
    *
FROM
    loans
WHERE
    installments >= 10
ORDER BY interest;

```

| AccNo | Cust_Name | Loan_Amoun | Installments | Int_Rate | Start_Date | Interest | Adress |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 101 | R.K.GUPTA | 300000 | 36 | 11.00 | 19-07-2019 | 1200 | NULL |
| 103 | K.P.JAIN | 300000 | 36 | NULL | 08-03-2017 | 1600 | NULL |
| 102 | S.P.SHARMA | 500000 | 48 | 9.00 | 22-03-2018 | 1800 | NULL |
| 104 | M.P.YADAV | 800000 | 60 | 9.00 | 06-12-2018 | 2250 | NULL |
| 106 | P.SHARMA | 700000 | 60 | 11.50 | 05-06-2018 | 3500 | NULL |
| 107 | K.S.DHALL | 500000 | 48 | NULL | 05-03-2018 | 3800 | NULL |

## xvii. Display the Interest-wise count of all loan holders whose Installment due is more than 5 in each group.

```sql
SELECT
    int_rate, COUNT(*)
FROM
    loans
GROUP BY
    int_rate
HAVING
    SUM(installments)>5;

```

| int_rate | COUNT(*) |
|---|---|
| 11.00 | 1 |
| 9.00 | 2 |
| NULL | 2 |
| 11.50 | 1 |

## xviii. Add one more column name ‘Address’ to the LOANS table.

```sql

ALTER TABLE loans
ADD (Adress TEXT);

```

```
28  03:35:16  ALTER TABLE loans ADD (Address TEXT)   0 row(s) affected
                                                     Records: 0  Duplicates: 0  Warnings: 0   1.175 sec
```

## xix. Reduce Interest rate by 1 of all loan holders whose Interest is not NULL.

```sql
UPDATE loans
SET
    int_rate = int_rate - 1
WHERE
    int_rate IS NOT NULL;
```

```
29  03:35:46  UPDATE loans SET  int_rate = int_rate - 1 WHE...  4 row(s) affected
                                                               Rows matched: 4  Changed: 4  Warnings: 0   0.144 sec
```

## xx. Delete the record of customer whose account number is 105.

```sql
DELETE FROM loans
WHERE
    accno = 105;

```

```
30  03:36:30  DELETE FROM loans WHERE  accno = 105   0 row(s) affected   0.00039 sec
```