key papers R22CS7240E

python programming

PART-A

Q-1) a) what symbol is used for comments in python?

Ans. In python, the symbol used for comments is the hash symbol (#)

b) Differentiate between integer and floating-point data types.

Ans. 1. Integer (int): Represents whole numbers without any decimal point

Ex: 5, -3, 100

2. floating-point (float): Represents numbers that have a decimal point

Ex: 3.14,

c) what is the purpose of the try keyword?

Ans: The try keyword is used to handle errors by placing code that may cause an exception inside it

d) why are modules used in python?

Ans: To organize code and reuse functions, Making programs easier to manage.

e) How multithreading improves performance?

Ans: Executing multiple tasks concurrently, making programs run faster and more efficiently.

f) why is synchronization required in multithreading?

Ans: synchronization is acquired to prevent multiple threads from accessing shared resources at the same time, avoiding errors and data conflicts.

g) what is Tkinter used for?

Ans: Tkinter is used to create graphical user interfaces

h) Define CGI?

Ans: CGI (common Gateway Interface) is a standard method for web servers to run programs

i) what are parameterized queries?

Ans: parameterized queries are safe queries that use placeholders to safely pass user input and sql injection.

j) How a database connection is established in python?

Ans: A database connection in Python is established by importing a database module sqlite3, connect() functions

PART – B

Q 2. Explain Python's core data types with suitable examples.

Ans: Python provides several built-in core data types that are used to store different kinds of values.

These data types help in writing efficient and structured programs.

1. Integer (int): used to store whole numbers (positive or negative) without decimal points

Ex: x = 20

2. Float: Represents numbers with decimal points and is used for real-number calculations

Ex: y = 3.14

3. string:
A sequence of characters enclosed within quotes. strings are used to store text data.

Ex:
name = "python programming"

4. Boolean (bool):
Represents truth values: True or false. useful in conditions and decision-making.

Ex:
is_valid = True

5. List:
An ordered, mutable collection that can store different data types

Ex: num = [10, 15, 20]

6. Tuple:
similar to list but immutable (cannot be changed after creation)

point = (4, 5)

7. Dictionary (dict):

stores data in key-value pairs, useful for fast lookups.

Ex: student = {"name": "amit", "age": 20}

8. Set:

An unordered collection of unique elements.

Ex: {1, 2, 3}.

Q3(a) Describe the behaviors of type conversion functions like int(), float(), and str()

Ans: int() function:

The int() function converts a value into an integer

* it removes the decimal part when converting a float.
* It can convert numeric strings like "us"

Ex: int(3.9) output: 3
int("20") output: 20

float() function:

The float() function converts a value into a floating-point number.

* converts integers to floats by adding a decimal poi

Ex: float(5) output: 5.0
float("4.5") output: 4.5

str() function:

The str() function converts any data type into a string

* Can convert numbers, lists, tuples, and almost all python objects.

Ex: str(100) output: "100"

str(3.10) output: "3.10"

3(b) Explain the difference between mutable and immutable data types.

① mutable data types: mutable data types allow modification of their content without changing their identity

Ex: list
Dictionary
set

Ex: num = [1, 2, 3]

num, [0] = 10

② immutable datatypes: immutable data types cannot be changed once created.

if you try to modify them, python creates a new object.

Ex: int, float, string, Tuple.

name = "python"

name, = name + "3"

Q4 Implement a python program to read and write in file and counts number of vowels.

```
# writing to a file.
text = " Hello, this is example of file to . count vowels"
file = open(" sample.txt", "w")
file. write (text)
file. close()
file = open (" sample. txt", "r")
content = file. read ()
file. close()
vowels = " aeiouAEIOU"
count = 0
for char in content:
    if char in vowels:
        count + = 1
print(" file content:", content)
print(" number of vowels:", count)
```

file content: Hello, this is example of file to count vowels.

number of vowels : 11

Q(5) Discuss about python manages files internally. Explain the difference between modules and packages.

Ans: Python uses a file handling mechanism that allows reading, writing, and managing files through a set of built-in functions. Internally, Python follows

1. Opening a file:

Python uses the open() function to connect a file with a file object.

ex: f = open("data.txt", "r")

2. File object creation:

when a file is opened, Python creates a file object stored in memory.

* file name
* mode (read/write)
* file pointer position
* encoding information

3. File pointer:

python maintains a file pointer that shows the current read/write position.

4. Buffering:

python uses an internal buffer to improve performance.

Data is not directly read/written to the disk

5. Reading and writing:

Python provides functions like read(), readline(), readlines(), write() and writelines() to manage file data.

6. closing the file.

Python clears the buffer and releases the file object using close()

f.close.

modules:

* A module is a single python file (.py) that contains functions, variables, or classes.

* it helps break large programs into smaller components

Ex: math.py

packages:

* A package is a collection of multiple modules stored inside a directory

* it must contain an __init__.py file.

Ex: mypackage
__init__.py.

Q6 compare threading and multiprocessing approaches. Define regular expressions and their purpose.

Ans: Threading and multiprocessing use two techniques used for achieving parallelism, but they work in different ways

Threading:

* Threading uses multiple threads within the same process

* All threads share the same memory space, which makes communication easier

* Suitable for I/O - bound tasks.

multiprocessing:

* multiprocessing creates separate process each with its own memory space.

* Bypasses the GIL, allowing true parallel execution on multiple cpu cores

* Best of cpu - bound tasks, such as mathematical computations or data processing.

* more memory - intensive than threads

Regular expression and their purpose

* A regular expression (regex) is a special sequence of characters that defines a pattern for searching and manipulating text

purpose of regular expression:

① pattern matching: find whether a specific pattern exists inside a string

Ex: check if an email address is valid

② Searching:

locate the position of patterns within text

re.search("python", text)

3. Extraction:

pull out useful data like phone numbers, date.

4. Replacing Text:

substitute patterns with new text

Ex: Replace multiple spaces with a single space.

5. Splitting strings:

Break a string using pattern-based separators instead of fixed characters.

Q7. Discuss how deadlocks can occur in multithreaded programs

In python, deadlocks commonly occur when threads use shared resources with locking mechanisms such as threading.lock Block or other synchronization tools

condition for deadlock.

1. mutual exclusion : only one thread can access ushared resource at a time.

2. Hold and wait : A thread holds one resource and waits for another

3. no pre-emption:

Resources cannot be forcibly taken away from a thread.

4. circular wait:

Two or more threads form cycle of waiting for each other's resources

if thread A waits for a resource held by thread B and thread B waits for a resource held by A, both remain waiting forever.

Example scenario:

* Thread 1 acquired lock A, then tries to acquire lock B.

* Thread 2 acquires lock B, then tries to acquire lock A.

Both threads hold one lock and wait for the other, causing deadlock.

Example :-

```
import threading

lock1 = threading.Lock()
lock2 = threading.Lock()

def thread1():
    lock1.acquire()
    lock2.acquire()
    lock2.release()
    lock1.release()

def thread2():
    lock2.acquire()
    lock1.acquire()
    lock1.release()
    lock2.release()

t1 = threading.Thread(target=thread1)
t2 = threading.Thread(target=thread2)

t1.start()
t2.start()
```

Reasons why deadlock happens.

1. improper lock ordering — different threads acquire locks in different orders.

2. Nested locks - a thread holding on lock tries to acquire another.

3. long-running critical sections - resources stay locked for too long.

4. lack of timeout - locks without timeout keep threads waiting forever

preventing deadlock.

1. use consistent lock ordering across all threads

2. Avoid holding multiple locks at the same time.

3. use try/lock with timeout to avoid infinite waiting.

4. prefer higher-level synchronization tools, like semaphore, queue or Rlock

Q(8) Explain about Tkinter and module for GUI development. Explain the main features of Tkinter.

Ans. Tkinter and GUI development.

Tkinter is python's standard library for creating graphical user interface. It provides a simple way to design windows, buttons, labels, text boxes, menus, and other GUI elements in python programs

* Tkinter is a wrapper around Tcl/Tk GUI toolkit, allowing python developers to build cross platform GUI applications

* it comes pre-installed with python, so no separate installation is required

* The GUI elements in Tkinter are called widgets

```
import tkinter as tk
root = tk.Tk()
root.title("my app")
label = tk.Label(root, text="Hello, Tkinter!")
label.pack()
root.mainloop()
```

main features of Tkinter

1. Cross-platform:

   works on windows, macOS, and Linux without changes to the code.

2. Rich set of widgets:

   Tkinter provides buttons, labels, checkboxes, radio buttons, text boxes, frames, menus, listboxes, and more

3. Event-Driven programming.

Tkinter uses an event loop (mainloop()) to handle user actions like clicks, key presses, or mouse events

4. simple and lightweight.

   Easy to learn and use for small to medium GUI application

5. Geometry management

   Tkinter supports layout managers (pack, grid, place)

6. Customizable widgets:

fonts, colors, size, and images can be easily customized for each widget

7. Integration with python libraries.

works well with other python modules like pil for images, matplotlib for plots, and sqlite3 for database integration.

Q(9): Evaluate python's suitability for GUI-based software development.

Ans: python is widely considered a suitable and powerful language for developing GUI based applications due to its simplicity, large library support, and strong community, its features and ecosystem make GUI creation efficient even for beginners

1) Availability of multiple GUI frameworks.

python provides several GUI libraries such as.

* Tkinter (built-in, simple and lightweight)
* pyqt/ pyside (powerful, professional UI tools)
* kivy (used for mobile and touch applications)
* wxpython (native-looking desktop apps)

2. Easy and readable syntax.

python's clean and easy-to-read syntax reduces development time.

GUI programs become easier to write, maintain, and extend

3. Cross-platform capability.

most python GUI frameworks run on:

* windows
* macOS
* linux

4. Rapid application development

Python supports quick prototyping

* High-level syntax
* built-in debugging tools
* interactive development environments

5. Integration with other technologies.

* Databases (sqlite3, MySQL, postgreSQL)
* web API
* Data analysis libraries like pandas, numpy
* plotting tools like matplotlib

6. Strong community and documentation

python has extensive online resources, tutorials and community supports

7. limitations:

* not ideal for heavy, graphic, intensive application
* python might be slower than languages like c++ because it is interpreted
* packaging GUI apps into standalone

Q(10) Justify the choice of sqLite for embedded python applications what are ORM frameworks?

Ans: sqLite is one of the most preferred databases for embedded python applications because of its simplicity, reliability, and lightweight nature, Python includes built-in support for sqLite through the sqlite3.

1. lightweight and self-contained:

sqLite done not require a server process, it stores the entire database in a single file, making it ideal for small devices and embedded systems.

2. Zero configuration:

This makes it extremely small, portable, and suitable for embedded systems,

3. Built-in support in python

· python provided the sqlite3 module in its standard library. this means no installation or configuration.

4. Fast Read/write performance for local Data-

for small to medium datasets, SQLite provides excellent performance because it avoids network overhead. This makes it ideal for real-time embedded application like sensors, desktop utilities.

5. ACID compliance and Reliability!

SQLite supports, atomicity, consistency, Isolation, Durability, ensuring safe transactions even if the system shuts down unexpectedly.

6. low memory and CPU consumption.

SQLite uses very little RAM and simple files, they can be moved across windows, Linux, and macOS without modification. This simplifies distribution of embedded python applications

7. Cross-platform portability!

Because sqlite databases are simple files, they can be moved across windows, linux, and macos.

8. Zero configuration and Simple Deployment.

No installation, configuration, or management is required. A python script with an sqlite file is enough to run the whole application.

ORM (Object - Relational mapping) frameworks allow programmers to interact with a database using python objects and classes instead of SQL queries

* Database tables → Python classes
* table rows → python objects.
* Table columns → object attributes

advantages of ORM frameworks.

1. Reduces SQL complexity: Developers write less raw SQL, and more python code
2. improves readability and maintainability: Database operation become object-oriented

Q(ii). Explain about DB-API sqlite in detail.

Ans: DB-API defines a uniform way for python applications to connect to databases, execute queries, handle results, and manage transactions

* Connection objects
* Cursor objects
* Error and Exception classes
* Methods for executing SQL commands

connecting to SQLite Database

Using DB-API, a connection is created with

```
import sqlite3
conn = sqlite3.connect("example.db")
```

creating a cursor object:

A cursor allows python to send sql commands to the database.

Python:

```
cursor = conn.cursor()
```

Executing sql queries:-

DB-API supports execute() and executemany()

```
cursor.execute("create table students (name text,
                age integer)")
cursor.execute("insert into students values
                (?, ?)", ("john", 20))
```

Fetching Records:

DB-API provides standard methods

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

Transaction management:

SQLite supports DB-API transaction control.

```
conn.commit
```

conn.rollback()

closing the cursor and connection

DB-API ensures proper resource cleanup

cursor.close()

conn.close()