nucture

* Credit test- II →

Que-1) : Explain the regular expression in Python.

Ans → i) "Regular expression" is also written as "RegEx".
ii) Regular expression is a sequence of characters, which forms a search pattern.
iii) Regular expression is used to find a string or set of strings.
iv) Regular expression can be used to check, if a string contains, the specified search patterns.
v) It can detect the presence or absence of a text, by matching it, with a particular pattern.
vi) The RegEx can also split a pattern, into one or more sub-patterns.

- RegEx Module →
  Python has a built in package, called as re, which can be used to work with a regular expression.

- Importing the re-module →

```
import re
```

- Regular Expression in python →
  After importing the re-module, you can start using a regular expression.

Example —

```
import re
# To check that, a string starts with "The"
and ends with "Spain".
```

Q.1) Regular expression - [Credit test-2 (Q.1) & external QB - Q.24]

example :-

① import re
str1 = "my name is kumud & my lucky num is 12"
print ("Occurance of <<<u>>> in given string:", re.findall ("u", str1))

o/p - Occurance of <<<u>>> in a given string : ['u', 'u', 'u', 'u']

② import re
str1 = "my name is kumud"
print (re.split (" ", str1))

o/p - ['my', 'name', 'is', 'kumud']

```
a = " The rain in Spain"
x = re.search("^ The. * Spain $", a)
if x:
    print ("Yes! We have a match!")
else:
    print ("No match")
```

- The re-module offers a set of functions that allow us to search a string for a match:

Function	Description
1) findall	Returns a list containing all matches.
2) search	Returns a match object, if we found any match in a string
3) split	Returns a list, where string has been split
4) sub	Replace one or many matches with a string.

Que-2) Describe the exception in detail.

Ans → Exception –

i) An exception is an event, which occurs, during the execution of a program.

ii) The exception may disturb, the normal flow of a program.

iii) Exception is a python object, that represents an error.

iv) When an exception occurs, it should be either handled or quit the program flow.

try → Run this as a normal part of a program

Except → Execute this when there is exception

else → execute this only, if no exceptions are raised

Finally → Finally always executes

Example - 1)

```
try:
    print("Hello")
except:
    print(" Exception occured ! Something went
          wrong")
else:
    print(" Nothing went wrong")
```

o/p - Hello
Nothing went wrong

→ expln - If try block does not raise any error, then else block will execute.

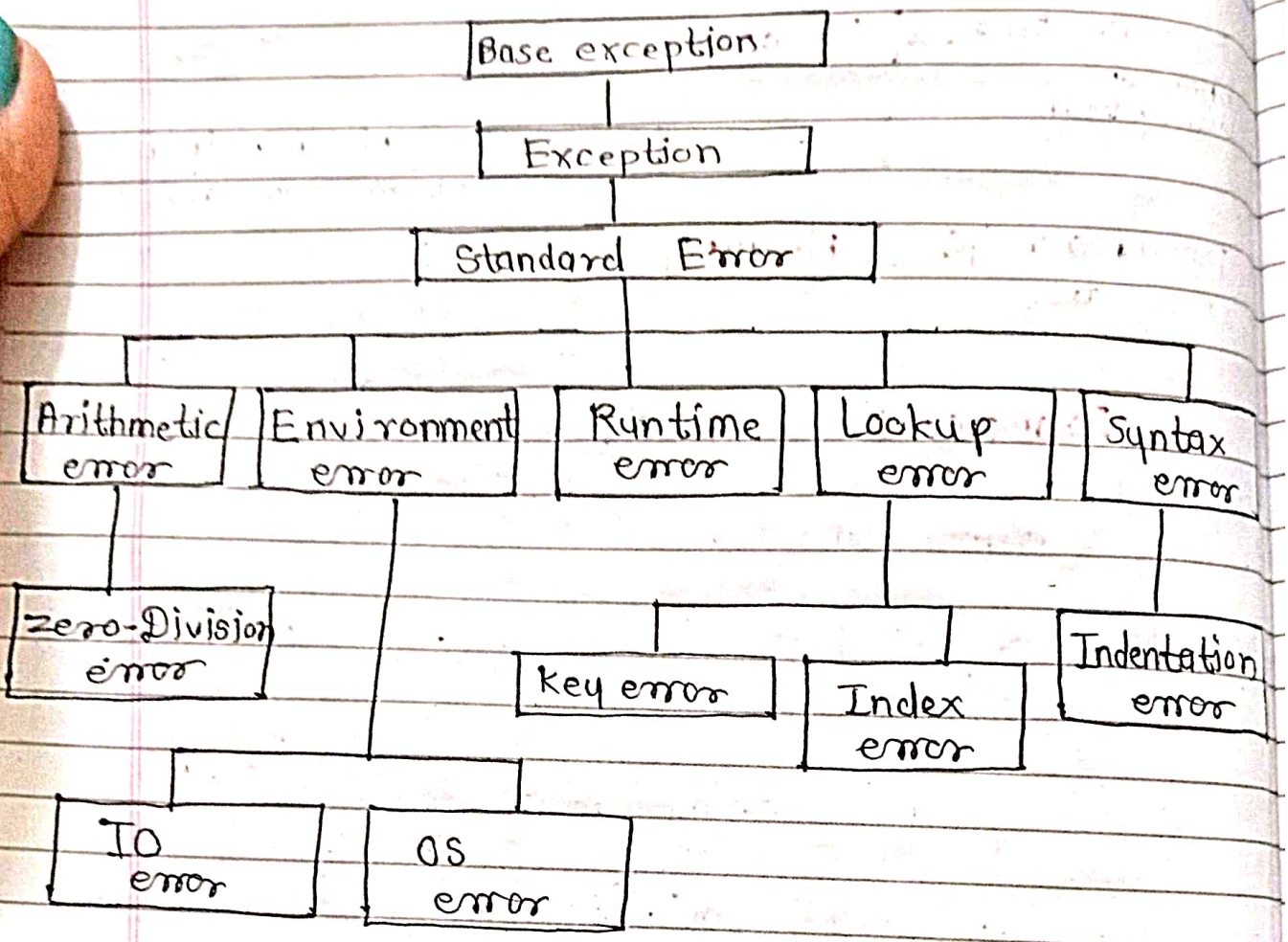
Example-2)

```
try:
    print(2/0)
except:
    print("something went wrong! Error occured!")
else:
    print("Nothing went wrong")
```

o/p - Something went wrong! Error Occured!

→ expln - DivideByZero error is there, hence exception occured.

• Different types of errors in a python-



Que-3) Define Lambda function & map function.

Ans (A) Lambda function →

i) Python lambda functions are anonymous functions.

ii) Anonymous means a function is without name.

iii) As we already know that, 'def' is a keyword which is used to define a function, in a python.

iv) Similarly the 'lambda' keyword is used to define an anonymous function in a python.

v) Syntax - lambda arguments: expression

example-

```
string1 = "HelloWorld"
upper = lambda string1: string1.upper()
print(upper(string1))
```

```
string2 = 'HelloWorld'
lower = lambda string2: string.lower()
print(lower(string2))
```

o/p - HELLOWORLD
helloworld

vi) The lambda function can have any number of arguments, but only one expression.
vii) One is free to use lambda functions, whenever function objects are required.
viii) The lambda function is restricted to a single expression.

3) Map function →

i) The map function returns the map object, which is the iterator, of the result.
ii) The map function (map()) executes a specified function, for each item, in an iterable.
iii) The item is send to the function as a parameter.
iv) Syntax : map(function, iterables)
Parameters → (fun, iter)
(a) function → It is a function, to which map passes each element of a given iterable.
(b) Iterables → It is an iterable, which will be mapped.
v) We can pass one or more iterables, to a map() function.

Example -

```
def myfun(a):
    return len(a)
x = map(myfun, ('apple', 'banana', 'cherry'))
print(x)
print(list(x))
```

Lambda ()→

```
num = int(input ("Enter a num:"))
cube = lambda num : num*num*num
print ("cube of a num is:", cube
                          (num))
```

o/p - Enter a num: 3
cube of a num is : 27

map() → [double all the num]

```
num = (1, 2, 3, 4, 5)
result = map(lambda x: x+x,
                numbers)
print(list(result))
```

contain no.

1+1=2
2+2=4
3+3=6
4+4=8
5+5=10

o/p - [2, 4, 6, 8, 10]

filter() →

```
series = [1, 2, 3, 4, 5, 6]
even = filter (lambda n : n%2==0,
                series)
list(even)
```

o/p - [2, 4, 6]

converting the map into a list, for redability -

off of code :-

[5, 6, 6]
↓ ↓ ↳ cherry
apple banana len(6).
len(5) len(6)

Que-1). Define the filter and reduce functions

Ans - (A) filter() →

i) The filter() in python, filter an elements, from an iterable, based on a given condition.
ii) The function returns a new ~~element~~ iterable, with the filtered element.
iii) The filter function, filter the given sequence with the help of a function, which test each element.
iv) It may be true or false.

Syntax - filter(function, sequence)

Parameters -

(a) function - function that tests, if each element of a sequence is true or not
(b) sequence - The sequence which needs to be filtered. It can be set, list, tuple or container.
(c) Returns - Returns is an iterator, that is already filtered.

example -

```
ages = [5, 12, 17, 18, 24, 32]  ← list
def myfuc(x):
    if x<18:
        return False
```

```
else:
    return True

adults = filter(myfun, ages)

for x in adults:
    print(x)
```

o/p
18
24
32.

(B) reduce() →

i) reduce() function, is a built in function in python.
ii) reduce() is applied to a given function, to the iterable element, by reducing them, to a single value.
iii) This function is defined in "functools" module
iv) The reduce() can also be combined with an operator function.
v) The reduce() can also be used to calculate the summation of a sequence elements.
vi) reduce() stores the intermediate result & only returns the final summation value.
vii) reduce() has two arguments - (fun, seq).

Syntax - functools.reduce(function, iterable [initializer])

Example -

```
from functools import reduce

# function defined for sum of two numbers----
def add(a,b):
    return a+b

# our_iterable - - ->>> list - - -
num_list = [1,2,3,4,5,6,7,8,9,10]
```

```
sum = reduce(add, num_list)
```

(add fun) as a 1st argument

num_list as a 2nd argument

```
print("Sum of integers of num_list:", sum)
```

(1+2+3+4+5+6+7+8+9+10)

55

Now passing 10, as an initial value ----

```
sum = reduce(add, num_list, 10)
print("Sum of integers of num_list, with initial value 10:", sum)
```

o/p - Sum of integers of num_list : 55
Sum of integers of num_list with initial value 10 : 65

30) explain in detail about modules.

ns - i) Python module is a set of built in ~~content~~ functions which you want to include, in your application.

ii) There are many modules available in python., and of them have own specific work.

iii) Module file contains definations & statements.

iv) It also include runnable code.

v) It allows to manage & reuse a code.
vi) To use the python modules, in our script, it is important to import the required module.

- **Module creation →**

i) To create a module in python, it is important to write the code in a file, having .py extension.

example -

my_module.py

```
def greet(name):
    return("Hello, {name}")
def add(a, b):
    print("Addition of a & b:", (a+b))
add(10,20)
```

- **Import a module in Python →**

i) We can import the functions & classes defined in a module, to another module, using the import statement, in same other python source file.

syntax -

```
import module_name
```

— This only import the module. Not the function & classes.

— To access the functions inside module, the dot (.) operator is used.

eg.

```
import my_module
print(my_module.greet "Kumu"))
print(my_module.add(2, 3))
```

o/p

Hello Kumu

- The module is loaded once, even if the num of lines are imported.

- **Naming a module** → You can name the module file, whatever you like.
- But it must have the file extension .py.

- **Renaming a module** → You can create an alias, when you import a module, by using the as keyword.

eg. create an alias for mymodule callad mx:

```
import mymodule as mx
a = mx.person1["age"]
print(a)
```

o/p - 36.

- **Import all names** →
- You can use the from statement, to import all name, from module, into current namespace.
- This is done by using a - '*' Symbal.

Syntax - from module_name import *

eg
```
from mymodule import *
print(greet("Kumu"))
print(add(10,20))
```

o/p - Hello kumu
30

→ **Some python well known modules** →

ⓐ Mathematics ⓑ Os ⓒ Pandas ⓓ Numpy
ⓔ Datasets ⓕ PyQT ⓖ Tkinter ⓗ keras
ⓘ seaborn ⓙ Matplotlib ⓚ statistics ⓛ Time, etc

Q 31) Describe Python MYSQL in detail.

Ans → i) Python MySQL connecter is a python driver, that helps to integrate python and MYSQL.

ii) This python MYSQL library, allows the conversion between, python and MYSQL data types.

iii) MYSQL connecter API, is implemented using pure python, and doesnot require any third-party library.

① Installation →

- To install Python MYSQL connecter module, It is mandetory to have python & PIP preinstalled on their system.
- If python & pip are already installed, then type the below command in the terminal.

```
-pip3 install mysql-connector-python
```

② Connection → first you need to establish a connection, to your MYSQL database.

- You need to provide host, username, password & optionally the database name.

```
- import mysql.connector
connection = mysql.connecter.connect(
                host = "localhost",
                user = "user",
                passwd = "password"
            )
```

③ Cursor → Once the connection is established, you have create object of the cursor

- Cursor is used to execute SQL queries.

```
-Cursor= connection.cursor()
```

* Create a new database - mycursor.execute(
"create database mydb")

④ Executing queries → We can execute SQL queries using the cursors 'execute()' method.

- cursor.execute("create table if not exists
users (id int Auto_increment
primary key,
name varchar(235),
age int)")

⑤ Committing changes → If you have made changes to the database, then you need to commit those changes, using the "commit()" method
- connection.commit()

⑥ Fetching the data → After executing, select query you can fetch the results, using cursors - fetchall(), fetchone() or fetchmany() methods.
- cursor.execute("Select * from users")
rows = cursor.fetchall()
for row in rows:
print(row)

⑦ Closing Connections → Once you are done, with the database operations, it is good to close the cursor & connection.
- cursor.close()
connection.close()

Q.32) - Demonstrate insert and read operation, in Python MYSQL.

Ans →

(A) Insert operation → The insert operation is need to perform, when we want to create a record, in a database table.

It fills a data(record) in a database.

```
import MySQLdb

db = MySQL.connect("localhost", "testuser",
     "test123", "TestDB")

mycursor = db.cursor() --- prepare cursor object using a cursor method
mycursor.execute("create table, if not exists
                  customers(name varchar(20) not null,
                  age int)")
sql = "insert into emp (name, age) values
      (%s, %d)"
val = ("John", 20)

try:
    mycursor.execute(sql, val)
    db.commit()
except:
    db.rollback() --- Rollback in case, if there is any error.
db.close() --- disconnect from server.
```

open db connection

(B) Read operation →

a) Read operation is used to fetch the data, from the database.

b) You can use various methods, to fetch values from database table.

① fetchone() - fetchone() method is used to retrieve next row, of a query result set.
- If there are no more rows to fetch, it returns None.

② fetchall() - It fetch all the rows in a result set,
- If some rows have been extracted already, from the result set, then it retrieve the remaining rows, from a result set.

③ rowscount() - This is a read-only attribute & it returns the number of rows, which was affected by execute() method.

eg -

```
import mysqldb
db = MySQLdb.connect("localhost", "testuser",
                     "test123", "TestDB")
cursor = db.cursor()
sql = " select * from emp"
try:
    cursor.execute(sql)
    result = cursor.fetchall()
    for row in result:
        print(row)
except:
    print("error: Unable to fetch data")
```

Q.33) Demonstrate update and delete operations in python MySQL.

Ans →

(A) Update Operation → The update operation allows to update one or more records, which are already available in the database.

```
import mysqldb
db = MySQLdb.connect ("localhost", "testuser",
                      "test123", "TestDB")
cursor = db.cursor()
sql = "update emp set age = %d where
                      name = %s"
                      val = (21, "Kumud")
try:
    cursor.execute (sql, val)
    db.commit()
except:
    db.rollback()
db.close()
```

(B) Delete Operation → Delete operation is required when you want to delete some records from your database.

```
import mysqldb
db = MySQLdb.connect ("localhost", "testuser",
                      "test123", "Test.db")
cursor = db.cursor()
Sql = "Delete from emp where name = %s"
      val = ("kumud")
try:
```

```
        cursor.execute(sql,val)
        db.commit()
except:
        db.rollback()
db.close()
```

extra → Join Operations :-

① Inner Join → It returns rows from both tables, that satisfy the condition specified, in the clause.

It involves matching values in col^ms beth tables.

syntax -
```
select columns
from table1
inner join table2 on table1.
    column = table2.column;
```

② Left Join → It returns all the rows from left table, along with matching rows, from the right table.

syntax -
```
select columns
from table1
left join table2 on table1.column=
    table2.column;
```

③ Right Join → It returns all the rows from right, table, along with matching rows, from the left table.

syntax -
```
select columns
from table1
right join table2 on table1.columns=
    table2.columns;
```

④ Full Join → It returns all matching rows from both tables.

Syntax - sql = "select users.name As user, product.name As favourite from users left join products on users.favourite = products.id"

Q.34) - Describe flask operation in python.

Ans → Flask →

a) Flask is a microweb framework for python.
b) Flask is a lightweight and flexible web framework for python.
c) It is designed to be simple & easy.
d) Flask is used ~~for~~, ~~while~~ ~~providing~~ to build web applications, because it provide the necessary tools, to build a web application.
e) Flask is developed by Armin Ronacher, who leads international group of python, enthusi~~~~ named as pocco.
f) Flask is based on the WSGI toolkit & Jinja2 template engine. Both are pocco projects.

Flask operations →

① Installation → flask can be installed using pip, pythons package manager.

pip install flask

② Importing flask → To use Flask, in your python application, you need to import it.

```
python
from flask import Flask
```

③ Creating Application Instance → You create an instance of a flask class, to represent you web application.

```
python
app = Flask(__name__)
```

④ Defining Routes → Routes are URL patterns,
- Flask use routes, to map URL's to python functions, which are known as view functions
- We can define route using the - @app.route() decorater

```
@app.route('/')
def index():
    return 'Hello, World!'
```

⑤ Running the application → To run a flask appln, we typically use the app.run() method.

```
if __name__ == '__main__':
    app.run(debug=True)
```

⑥ View functions → Python view functions handle the requests and return responses.

```
@app.route('/user/<username>')
def show_user_profile(username):
    return f'User {username}'
```

⑦ Templates → Flask support rendering HTML templates, using template engines like - Jinja2.

render_template() is used to render a template.

```
from flask import render_template
@app.route('/hello')
@app.route('/hello/<name>')
def hello(name = None):
    return render_template('hello.html', name=name)
```

⑧ Request Handling → Flask provides objects, to handle incoming requests.

```
from flask import request
@app.route('/login', methods=['POST', 'GET'])
def login():
    if request.method == 'POST':
        # Handle the login request ----
    else:
        # Render the login form ----
```

⑨ Response Handling → Flask provides various ways to generate HTTP responses.

Which include returning strings, returning templates, redirecting & returning JSON data.

⑩ Error Handling → Flask provides ways to handle errors and exceptions, in your appln :- Such as:

@app.errorhandler() decorator

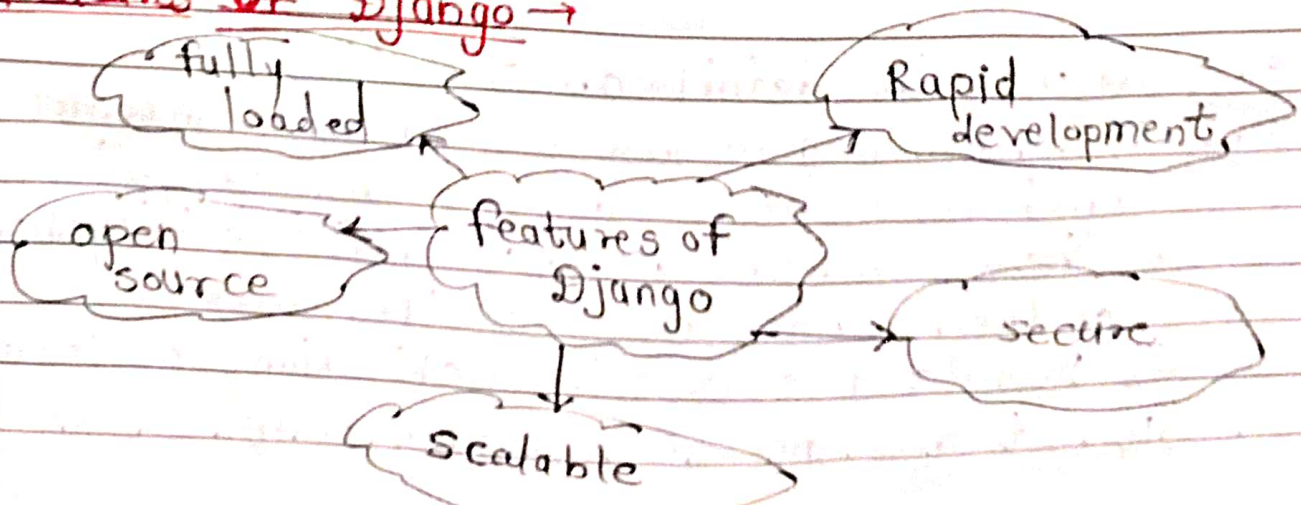
These are the fundamental operations performed on flask.

5) Explain Django framework in python.

ns → Simple Django Framework →

a) Django is a simple web application framework, written in Python programming language.
b) It is based on MVT (Model View Template) design pattern.
c) The Django is very demanding, due to its rapid development feature.
d) It take less time, to build application, after collecting client requirement.
e) This framework uses a famous tag line: The web framework, for perfections, with deadlines.
f) Django is designed in such a manner, that it handles many configure things automatically.
g) Django was designed & developed by - Lawrence Journal World in 2003.
h) It was publically released under BSD license, in July 2005.
i) It's current version is 2.0.3, which was released on 6 march 2018.
j) Django is widely accepted, and used by variou well-known sites, such as- Instagram, Mozilla Pinterest, etc.

Features of Django →



① Rapid development → Django is designed to make a web application frame in less time. The project implementation phase takes a long time, but Django creates it rapidly.

② Secure → Django takes security seriously and it help the developers to avoid many common security mistakes. It has a secure user authentication, to manage user accounts & passwords.

③ Scalable → Django is scalable in nature, and it has ability to quickly switch, from small to large scale application project.

④ Open source → Django is an open source, web application framework. It is publicaly available without cost. It can be downloaded with a source code.

⑤ Fully loaded → Django includes various helping task modules, and libraries. Which can be used to handle common web development tasks.