RAMAIAH
Institute of Technology

Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science & Engineering

Data Visualization with Python Lab(CSL48)

USN: **Week #: 04**

Semester: **Section:** **Date:**

Instructions:

- **Implement the following programs using python language.**

Topic: Regular Expressions: Introduction to regex, pattern matching and practical applications.
Error Handling & Exceptions, Iterators & Generators

Programs:

1. a. Write a Python program to check if a given string contains only letters (a-z, A-Z) using regex.
   b. Write a Python program to validate a strong password (at least 8 characters, including uppercase, lowercase, numbers, and special characters) using regex.

2. a. Write a Python program to handle multiple exceptions (e.g., `ZeroDivisionError`, `ValueError`, and `TypeError`).
   b. Write a Python program that prompts the user for an integer input and handles the case where the input is not an integer.

3. a. Write a Python program to validate a strong password (at least 8 characters, including uppercase, lowercase, numbers, and special characters) using regex.
   b. Write a Python program to find all words in a given string that start with a vowel using regex.

4. a Write a Python program that manually iterates over a list using an iterator object.
   b. Write a Python class that implements an iterator to return even numbers up to a given limit.

5. a. Write a Python generator function that yields even numbers up to a given limit.
   b. Write a Python generator that generates an infinite sequence of Fibonacci numbers.

6. Write a Python program to search for and extract all phone numbers from a given block of text. Assume phone numbers can be in formats like `(123) 456-7890` or `123-456-7890`.

7. Write a Python program that demonstrates the use of try-except-finally blocks. For example, attempt to open a file, process its content, and ensure that the file is closed regardless of whether an exception occurs.

8. Write a Python program that demonstrates the use of the `iter()` function on a list and manually retrieves elements using the `next()` function until a `StopIteration` exception is encountered.

📘 1. Regular Expressions (Regex) in Python

✅ What is Regex?

- A **Regular Expression (regex)** is a sequence of characters that forms a search pattern.
- Used for **pattern matching**, **searching**, **validating**, and **extracting** text data.

🔧 Key Functions (from `re` module):

- `re.match()`: Checks for a match only at the **beginning** of the string.
- `re.search()`: Searches the **entire string** for a match.
- `re.findall()`: Returns all **non-overlapping matches** of a pattern.
- `re.fullmatch()`: Ensures the **entire string** matches the pattern.

🧩 Common Metacharacters:

Symbol	Meaning
`.`	Any character except newline
`^`	Start of string
`$`	End of string
`*`	0 or more occurrences
`+`	1 or more occurrences
`?`	0 or 1 occurrence
`{n}`	Exactly n repetitions
`[]`	Set of characters
`	`
`()`	Grouping
`\d`	Digit (`0-9`)
`\D`	Non-digit
`\w`	Word character (`a-z`, `A-Z`, `0-9`, `_`)
`\W`	Non-word character
`\s`	Whitespace
`\S`	Non-whitespace

📌 Applications of Regex:

- Form validation (email, phone number, passwords)
- Data extraction (e.g., extracting dates, URLs)
- Log parsing
- String substitution and cleanup

⚠️ 2. Error Handling & Exceptions

✅ Why Handle Errors?

- Prevents program crashes.
- Allows **graceful degradation** — handles issues without stopping the program.

⚙️ Basic Structure:

```
try:
    # Code that might cause an exception
except ExceptionType:
    # Code to handle the exception
else:
    # Optional: Executes if no exception occurs
finally:
    # Always executed, used for cleanup (like closing files)
```

🧩 Common Exception Types:

Exception	Trigger
`ZeroDivisionError`	Dividing by zero
`ValueError`	Invalid type/value (e.g., converting `abc` to int)
`TypeError`	Unsupported operation between different types
`IndexError`	Index out of range
`KeyError`	Key not found in dictionary
`FileNotFoundError`	Trying to open a non-existent file

🧠 Best Practices:

- Catch **specific exceptions**.
- Use `finally` to release resources.
- Avoid catching general `Exception` unless necessary.

🔁 3. Iterators in Python

✅ What is an Iterator?

- An object that implements:
  - `__iter__()` → returns the iterator object itself
  - `__next__()` → returns the next value, raises `StopIteration` when done

🔁 How Iterators Work:

```
mylist = [1, 2, 3]
it = iter(mylist)
print(next(it))  # 1
```

🧠 Built-in Iterable Types:

- Lists, Tuples, Dictionaries, Strings, Sets, etc.

🧩 Advantages of Iterators:

- Memory efficiency (especially for large data)
- Enables **lazy evaluation**
- Supports custom traversal logic using classes

🌱 4. Generators in Python

✅ What is a Generator?

- A **special type of iterator** that uses `yield` instead of `return`.
- Automatically implements `__iter__()` and `__next__()`.

🔧 How it works:

```
def my_gen():
    yield 1
    yield 2

gen = my_gen()
print(next(gen))  # 1
```

🧠 Key Benefits:

- **Lazy Evaluation**: Values are produced only when needed.
- **Memory Efficient**: Suitable for large datasets or infinite sequences.
- More readable and compact than writing custom iterator classes.

🧩 Use Cases:

- Reading large files line by line
- Producing an infinite sequence (e.g., Fibonacci, prime numbers)
- Stream processing in real-time applications

Topic 1: Regular Expressions (Regex)

- **Regular Expressions (regex)** allow for pattern matching in strings.
- Common regex symbols:
  - `.`: any character except newline
  - `^`: beginning of string
  - `$`: end of string
  - `[]`: character set
  - `\d`: digit, `\D`: non-digit
  - `\w`: word character, `\W`: non-word
  - `\s`: whitespace, `\S`: non-whitespace
  - `+`: one or more
  - `*`: zero or more
  - `{n}`: exactly n times
  - `|`: or
  - `()`: group

Programs:

a. Check if string contains only letters:

```
import re

def only_letters(s):
    return bool(re.fullmatch(r"[A-Za-z]+", s))

print(only_letters("Hello"))  # True
print(only_letters("Hello123"))  # False
```

b. Validate strong password:

```
import re

def validate_password(password):
    pattern =
r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*#?&])[A-Za-z\d@$!#%*?&]
{8,}$'
    return bool(re.fullmatch(pattern, password))

print(validate_password("Strong1@Pass"))  # True
```

c. Find all words starting with a vowel:

```
import re

def words_starting_with_vowel(text):
    return re.findall(r'\b[aeiouAEIOU]\w*', text)

print(words_starting_with_vowel("An apple is on the umbrella"))  #
['An', 'apple', 'is', 'on', 'umbrella']
```

d. Extract all phone numbers:

```
import re

text = "Call me at (123) 456-7890 or 987-654-3210."
pattern = r'\(?\d{3}\)?[- ]\d{3}-\d{4}'
print(re.findall(pattern, text))  # ['(123) 456-7890',
'987-654-3210']
```

Topic 2: Error Handling & Exceptions

- Python uses `try-except` blocks to handle exceptions.
- `finally` block always runs (for cleanup).
- Can catch specific exceptions like `ZeroDivisionError`, `ValueError`.

Programs:

a. Handle multiple exceptions:

```
try:
    a = int("abc")  # ValueError
    b = 5 / 0       # ZeroDivisionError
    c = "2" + 2     # TypeError
except ValueError:
    print("Value Error occurred")
except ZeroDivisionError:
    print("Zero Division Error occurred")
except TypeError:
    print("Type Error occurred")
```

b. User input with validation:

```
try:
    num = int(input("Enter an integer: "))
    print(f"You entered: {num}")
except ValueError:
    print("That's not an integer!")
```

c. Try-except-finally block:

```
try:
    f = open("example.txt", "r")
    content = f.read()
    print(content)
except FileNotFoundError:
    print("File not found.")
finally:
    print("Closing file (if open).")
    try:
        f.close()
    except:
        pass
```

Topic 3: Iterators

- Iterators use `__iter__()` and `__next__()` methods.
- `iter()` returns an iterator object.
- `next()` retrieves the next element, raises `StopIteration` when done.

Programs:

a. Manual iteration using iterator:

```
lst = [1, 2, 3]
it = iter(lst)

while True:
    try:
        print(next(it))
    except StopIteration:
        break
```

b. Custom iterator for even numbers:

```
class EvenIterator:
    def __init__(self, limit):
        self.limit = limit
        self.num = 0

    def __iter__(self):
        return self

    def __next__(self):
        while self.num <= self.limit:
            if self.num % 2 == 0:
                result = self.num
                self.num += 1
                return result
            self.num += 1
        raise StopIteration

for even in EvenIterator(10):
    print(even)
```

Topic 4: Generators

- Generators use `yield` to return a value and pause the function state.
- More memory efficient than lists for large sequences.

✅ Programs:

a. Generator for even numbers:

```
def even_gen(limit):
    for i in range(0, limit+1, 2):
        yield i

for num in even_gen(10):
    print(num)
```

b. Infinite Fibonacci generator:

```
def fibonacci():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

fib = fibonacci()
for _ in range(10):
    print(next(fib))
```