Loops in python :

Loops in Python

Loops in Python are used to execute a block of code repeatedly for a specified number of times. They are essential for tasks that require iteration, such as:

1. Iterating over a sequence (list, tuple, string, etc.).
2. Performing a task repeatedly until a condition is met.
3. Processing data in batches.

Types of Loops in Python:

1. **For Loop**: Used to iterate over a sequence (list, tuple, string, etc.) or other iterable objects.
2. **While Loop**: Used to execute a block of code as long as a condition is true.

For Loop

Syntax:

```
pythonCopy code
for variable in iterable:
    # Code block to be executed
```

Example:

```
pythonCopy code
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
```

```
    print(fruit)
```

Explanation:

The loop iterates over the `fruits` list, assigning each value to the `fruit` variable, and prints it.

Types of For Loops:

1. **Simple For Loop**: Iterates over a single sequence.
2. **Nested For Loop**: Iterates over multiple sequences, one inside the other.
3. **For-Else Loop**: Executes an else block when the loop finishes normally (without a break).

Simple For Loop

Use Case: Iterating over a single sequence.

Example:

```
pythonCopy code
numbers = [1, 2, 3]
for num in numbers:
    print(num)
```

Explanation:
This loop will print each number in the `numbers` list.

Nested For Loop

Use Case: Iterating over multiple sequences.

Example:

```
pythonCopy code
colors = ['red', 'green']
```

```
shapes = ['circle', 'square']
for color in colors:
    for shape in shapes:
        print(color, shape)
```

Explanation:
This loop will print each combination of colors and shapes.

For-Else Loop

Use Case: Executes an else block when the loop finishes normally (without a break).

Example:

```
pythonCopy code
numbers = [1, 2, 3]
for num in numbers:
    print(num)
else:
    print("Loop finished")
```

Explanation:
This loop will print each number in the `numbers` list, and after the loop finishes, it will print "Loop finished".

Where to Use For Loop:

1. **Iterating over a list:**

```
pythonCopy code
my_list = [10, 20, 30]
for item in my_list:
    print(item)
```

1. **Iterating over a dictionary:**

```
pythonCopy code
my_dict = {'a': 1, 'b': 2, 'c': 3}
for key, value in my_dict.items():
    print(key, value)
```

1. **Iterating over a string:**

```
pythonCopy code
my_string = "hello"
for char in my_string:
    print(char)
```

1. **Iterating over a range:**

```
pythonCopy code
for i in range(1, 10):
    print(i)
```

1. **Iterating over a file:**

```
pythonCopy code
with open('file.txt') as f:
    for line in f:
        print(line)
```

Code Examples:

Simple For Loop

```
pythonCopy code
numbers = [1, 2, 3]
for num in numbers:
    print(num)
```

Nested For Loop

```
pythonCopy code
colors = ['red', 'green']
shapes = ['circle', 'square']
for color in colors:
    for shape in shapes:
        print(color, shape)
```

For-Else Loop

```
pythonCopy code
numbers = [1, 2, 3]
for num in numbers:
    print(num)
else:
    print("Loop finished")
```

Summary

For Loops are essential in Python for tasks that require iteration. They allow you to iterate over sequences and other iterable objects, perform tasks repeatedly, and process data in batches. There are different types of for loops, including simple for loops, nested for loops, and for-else loops. Understanding these concepts and practicing their usage will help you become proficient in writing efficient and effective Python code.

Additional Examples:

1. **List Comprehension**: A concise way to create lists.

```
squares = [x**2 for x in range(10)]
print(squares)  # Output: [0, 1, 4, 9, 16, 25, 36, 49, 64, 8
1]
```

1. **Loop with Condition**: Filtering elements within the loop.

```
pythonCopy code
even_squares = [x**2 for x in range(10) if x % 2 == 0]
print(even_squares)  # Output: [0, 4, 16, 36, 64]
```

1. **Iterating Over Multiple Sequences with `zip`**:

```
names = ['Alice', 'Bob', 'Charlie']
ages = [25, 30, 35]
for name, age in zip(names, ages):
    print(f"{name} is {age} years old.")
# Output:
# Alice is 25 years old.
# Bob is 30 years old.
# Charlie is 35 years old.
```

By using these various types and methods of loops in Python, you can handle a wide range of tasks efficiently.

while loop use :

. Simple Counting Loop

A `while` loop can be used for simple counting, where a variable is incremented or decremented until a condition is met.

Example:

```
# Count from 1 to 5
count = 1
while count <= 5:
    print(count)
    count += 1
```

Output:

```
Copy code
1
2
3
4
5
```

2. Loop Until a Condition Changes

A `while` loop can continue running until a certain condition becomes `False`. This is useful for waiting on user input or processing data until a specific event occurs.

Example:

```
# Keep asking for input until a valid number is provided
```

```
number = None
while number is None:
    try:
        number = int(input("Enter a number: "))
    except ValueError:
        print("That's not a valid number.")
```

3. Processing a List

You can use a `while` loop to process elements in a list or other iterable until it's empty.

Example:

```
# Remove and print elements from a list one by one
fruits = ["apple", "banana", "cherry"]
while fruits:
    print(fruits.pop(0))
```

Output:

```
Copy code
apple
banana
cherry
```

4. Infinite Loops

A `while` loop can be designed to run indefinitely, often used in situations where the loop should continue until an external condition stops it, like waiting for user input.

Example:

```
# Infinite loop waiting for user to type "exit"
while True:
    command = input("Type 'exit' to quit: ")
    if command == "exit":
        break
```

5. Conditional Loops

You can use `while` loops to perform actions based on complex conditions that might change during execution.

Example:

```
# Loop until a number is less than 100
number = 50
while number < 100:
    number += int(input("Add a number: "))
    print(f"New number: {number}")
```

6. Using `while` with `else`

A `while` loop can be combined with an `else` statement, which executes once the loop condition becomes `False` (and no `break` occurs).

Example:

```
# Loop with an else block
count = 0
while count < 5:
    print(count)
    count += 1
```

```
else:
    print("Loop finished.")
```

Output:

```
vbnetCopy code
0
1
2
3
4
Loop finished.
```

7. Simulating a `for` Loop

You can use a `while` loop to iterate over a sequence of numbers, similar to a `for` loop.

Example:

```
# Simulating for loop behavior
i = 0
while i < 5:
    print(i)
    i += 1
```

Output:

```
Copy code
0
1
2
```

```
3
4
```

8. Nested `while` Loops

A `while` loop can be nested inside another `while` loop, useful for working with multi-dimensional data structures like matrices.

Example:

```
# Nested while loop to print a matrix
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
i = 0
while i < len(matrix):
    j = 0
    while j < len(matrix[i]):
        print(matrix[i][j], end=" ")
        j += 1
    print()
    i += 1
```

Output:

```
Copy code
1 2 3
4 5 6
7 8 9
```

9. `while` with Multiple Conditions

You can use multiple conditions in a `while` loop, combining them with logical operators like `and`, `or`, etc.

Example:

```
# Loop until one of the conditions becomes false
x, y = 0, 10
while x < 5 and y > 5:
    print(f"x: {x}, y: {y}")
    x += 1
    y -= 1
```

Output:

```
yamlCopy code
x: 0, y: 10
x: 1, y: 9
x: 2, y: 8
x: 3, y: 7
x: 4, y: 6
```

10. `while` Loop with Early Exit (`break`)

You can use `break` to exit a `while` loop prematurely when a certain condition is met.

Example:

```
# Loop until a certain condition, then break
i = 0
while i < 10:
    if i == 5:
```

```
        break
    print(i)
    i += 1
```

Output:

```
Copy code
0
1
2
3
4
```

Summary:

- **Simple Counting:** Incrementing or decrementing counters.
- **Conditional Loops:** Run until a condition changes.
- **Processing Collections:** Iterating through lists, strings, etc.
- **Infinite Loops:** Run indefinitely, often with a break condition.
- `else` **with** `while`**:** Execute a block of code after the loop finishes normally.
- **Simulating** `for` **Loop:** Iterate using a `while` loop like a `for` loop.
- **Nested Loops:** Handling multi-dimensional data or complex conditions.
- **Multiple Conditions:** Combine logical conditions in a loop.
- **Early Exit:** Use `break` to exit early from the loop.

break and continue :

Example: Breaking the Loop When `x` Reaches a Certain Value

```
pythonCopy code
x = 4
while True:
    print(x)

    # Add a condition to break the loop
    if x >= 10:
        break  # Exit the loop when x reaches 10

    x += 1  # Increment x to avoid an infinite loop
```

Explanation:

- `while True:` creates an infinite loop that will run until it's explicitly broken.
- `break` is used to exit the loop when a certain condition is met. In this case, when `x` reaches or exceeds 10, the loop will break.
- `x += 1` increments the value of `x` by 1 in each iteration, ensuring that the loop eventually meets the break condition.

Using `break` in a `while` Loop

```
pythonCopy code
x = 0
while x < 10:
    print("Current value of x:", x)
```

```
    if x == 5:
        print("Breaking the loop as x reached 5")
        break  # Exit the loop when x is 5

    x += 1  # Increment x
```

Explanation:

- `while x < 10:` runs the loop as long as `x` is less than 10.
- `if x == 5:` checks if `x` has reached 5. If so, the loop will break.
- `break` is used to exit the loop when the condition is met.
-

Example using `while` Loop: Cricket Over with Break Condition

```
pythonCopy code
balls_faced = 0
runs_scored = 0

while balls_faced < 6:  # Over has 6 balls
    balls_faced += 1
    run = int(input(f"Enter runs scored on ball {balls_face
d}: "))

    if run == -1:  # -1 indicates getting out
        print("You're out! Over ends.")
        break  # Exit the loop, over ends

    runs_scored += run

print(f"Total runs scored in the over: {runs_scored}")
```

Explanation:

- `while balls_faced < 6:`: Loops until all 6 balls in the over are faced.
- `if run == -1:`: Checks if the player is out. If true, the loop breaks, and the over ends early.

Example using `for` Loop: Cricket Over with Break Condition

```
pythonCopy code
runs_scored = 0

for ball in range(1, 7):  # Over has 6 balls
    run = int(input(f"Enter runs scored on ball {ball}: "))

    if run == -1:  # -1 indicates getting out
        print("You're out! Over ends.")
        break  # Exit the loop, over ends

    runs_scored += run

print(f"Total runs scored in the over: {runs_scored}")
```

Example of `continue` in a `while` loop:

In this example, we'll use the `continue` statement to skip even numbers and only print odd numbers.

```
pythonCopy code
x = 0
while x < 10:
    x += 1
```

```
    if x % 2 == 0:
        continue  # Skip the rest of the loop for even number
s

    print(x)  # This will only print odd numbers
```

Explanation:

- `x % 2 == 0` checks if `x` is an even number.
- `continue` skips the current iteration of the loop if `x` is even, so the print statement is only executed for odd numbers.

Example of `continue` in a `for` loop:

Now, we'll use the `continue` statement in a `for` loop to skip numbers divisible by 3.

```
pythonCopy code
for i in range(1, 11):
    if i % 3 == 0:
        continue  # Skip the rest of the loop for numbers div
isible by 3

    print(i)  # This will print numbers not divisible by 3
```

```
balls_faced = 0
runs_scored = 0

while balls_faced < 6:  # Over has 6 balls
    run = int(input(f"Enter runs scored on ball {balls_faced + 1
```

```
    if run == -1:  # -1 indicates a wide ball
        print("Wide ball! It doesn't count.")
        continue  # Skip this iteration and go to the next ball

    runs_scored += run
    balls_faced += 1

print(f"Total runs scored in the over: {runs_scored}")
```

Pass :

Example 1: Using `pass` in a Loop

```
for i in range(5):
    if i == 3:
        pass  # Placeholder, nothing happens when i == 3
    else:
        print(i)
```

Example 2: Using `pass` in a Function

```
def placeholder_function():
    pass  # This function does nothing
```

```
# You can call the function without any action
placeholder_function()
```

Example 3: Using `pass` in a Class

```
class MyClass:
    pass  # Empty class definition

# You can create an instance of the class without any behavio
r
my_instance = MyClass()
```

Example 4: Using `pass` in an `if-else` Statement

```
x = 10

if x > 0:
    pass  # Placeholder for future code
else:
    print("Negative number")
```

Exercise 1:

Question 1: Sum of Even Numbers

Problem: Write a Python program to find the sum of all even numbers from 1 to 50. Use a `for` loop and `continue` statement.

Solution:

```
total_sum = 0
for num in range(1, 51):
    if num % 2 != 0:
        continue  # Skip odd numbers
    total_sum += num

print("Sum of even numbers from 1 to 50:", total_sum)
```

Question 2: First Multiple of 7

Problem: Write a Python program to find the first multiple of 7 in the range from 10 to 50 using a `for` loop and `break` statement.

Solution:

```
for num in range(10, 51):
    if num % 7 == 0:
        print("First multiple of 7 between 10 and 50:", num)
        break  # Stop the loop after finding the first multip
le
```

Question 3: Count the Occurrences of a Character

Problem: Write a Python program to count how many times the letter 'a' appears in a given string. Use a `for` loop and `continue` statement.

Solution:

```
text = "An apple a day keeps the doctor away"
count_a = 0

for char in text:
    if char != 'a' and char != 'A':
        continue  # Skip characters other than 'a' or 'A'
    count_a += 1

print("Number of 'a' in the text:", count_a)
```

Question 4: Find Prime Numbers

Problem: Write a Python program to print all prime numbers between 10 and 30 using a `for` loop and `break` statement.

Solution:

```
for num in range(10, 31):
    if num > 1:
        for i in range(2, num):
            if num % i == 0:
                break  # Not a prime number
        else:
            print(num)  # Prime number
```

Question 5: Skip Multiples of 5

Problem: Write a Python program to print numbers from 1 to 30 but skip the multiples of 5. Use a `for` loop and `continue` statement.

Solution:

```
for num in range(1, 31):
    if num % 5 == 0:
        continue  # Skip multiples of 5
    print(num)
```

Question 6: Find the First Negative Number

Problem: Write a Python program to find the first negative number in a list using a `for` loop and `break` statement.

Solution:

```
numbers = [10, 20, -3, 15, -7, 25]

for num in numbers:
    if num < 0:
        print("First negative number:", num)
        break  # Stop the loop after finding the first negati
ve number
```

Question 7: Sum of Positive Numbers Until Zero

Problem: Write a Python program to sum positive numbers in a list until a zero is encountered using a `for` loop and `break` statement.

Solution:

```
numbers = [10, 20, 5, 0, 15, 25]
```

```
total_sum = 0

for num in numbers:
    if num == 0:
        break  # Stop summing when zero is encountered
    total_sum += num

print("Sum of positive numbers until zero:", total_sum)
```

Question 8: Print Non-Alphabet Characters

Problem: Write a Python program to print all non-alphabet characters from a string using a `for` loop and `continue` statement.

Solution:

```
text = "Hello, World! 123"

for char in text:
    if char.isalpha():
        continue  # Skip alphabet characters
    print(char)
```

Question 9: Find the Largest Even Number

Problem: Write a Python program to find the largest even number in a list using a `for` loop and `continue` statement.

Solution:

```
numbers = [15, 22, 9, 18, 30, 11]
largest_even = None
```

```
for num in numbers:
    if num % 2 != 0:
        continue  # Skip odd numbers
    if largest_even is None or num > largest_even:
        largest_even = num

print("Largest even number:", largest_even)
```

Question 10: Find the Index of First Odd Number

Problem: Write a Python program to find the index of the first odd number in a list using a `for` loop and `break` statement.

Solution:

```
numbers = [2, 4, 6, 7, 8, 10]

for i, num in enumerate(numbers):
    if num % 2 != 0:
        print("Index of the first odd number:", i)
        break  # Stop the loop after finding the first odd nu
mber
```

Exercise 2 with while loops:

Question 1: Print Numbers from 1 to 10

Problem: Write a Python program to print numbers from 1 to 10 using a `while` loop.

Solution:

```
num = 1
while num <= 10:
    print(num)
    num += 1
```

Question 2: Sum of First N Natural Numbers

Problem: Write a Python program to find the sum of the first N natural numbers using a `while` loop.

Solution:

```
n = 10
total_sum = 0
num = 1

while num <= n:
    total_sum += num
    num += 1

print("Sum of first", n, "natural numbers is:", total_sum)
```

Question 3: Reverse a Number

Problem: Write a Python program to reverse a given number using a `while` loop.

Solution:

```
number = 12345
reverse_number = 0

while number > 0:
    remainder = number % 10
    reverse_number = (reverse_number * 10) + remainder
    number //= 10

print("Reversed number is:", reverse_number)
```

Question 4: Find the Factorial of a Number

Problem: Write a Python program to find the factorial of a given number using a `while` loop.

Solution:

```
n = 5
factorial = 1

while n > 0:
    factorial *= n
    n -= 1

print("Factorial is:", factorial)
```

Question 5: Print Fibonacci Sequence

Problem: Write a Python program to print the first N numbers of the Fibonacci sequence using a `while` loop.

Solution:

```
n = 10
a, b = 0, 1
count = 0

while count < n:
    print(a)
    a, b = b, a + b
    count += 1
```

Question 6: Find the Greatest Common Divisor (GCD)

Problem: Write a Python program to find the GCD of two numbers using a `while` loop.

Solution:

```
a = 56
b = 98

while b != 0:
    a, b = b, a % b

print("GCD is:", a)
```

Question 7: Check if a Number is Prime

Problem: Write a Python program to check if a given number is prime using a `while` loop.

Solution:

```
num = 29
```

```
is_prime = True
i = 2

while i <= num // 2:
    if num % i == 0:
        is_prime = False
        break
    i += 1

if is_prime:
    print(num, "is a prime number")
else:
    print(num, "is not a prime number")
```

Question 8: Guess the Number Game

Problem: Write a Python program for a simple "Guess the Number" game. The program will generate a random number between 1 and 100, and the user has to guess it.

Solution:

```
import random

random_number = random.randint(1, 100)
guess = None

while guess != random_number:
    guess = int(input("Guess the number (between 1 and 100):
"))

    if guess < random_number:
        print("Too low!")
    elif guess > random_number:
```

```
        print("Too high!")

print("Congratulations! You guessed the number:", random_numb
er)
```

Question 9: Find the Sum of Digits of a Number

Problem: Write a Python program to find the sum of the digits of a given number using a `while` loop.

Solution:

```
number = 1234
sum_of_digits = 0

while number > 0:
    sum_of_digits += number % 10
    number //= 10

print("Sum of the digits is:", sum_of_digits)
```

Question 10: Display a Countdown Timer

Problem: Write a Python program to display a countdown timer from 10 to 1 using a `while` loop.

Solution:

```
import time

countdown = 10

while countdown > 0:
```

```
    print(countdown)
    time.sleep(1)  # Sleep for 1 second
    countdown -= 1

print("Time's up!")
```

Regex :

2. Using Python's `re` Module

```
import re
```

2.1 Basic Matching Functions

- `re.match()`: Determines if the regex matches at the beginning of the string.
- `re.search()`: Scans through a string, looking for any location where the regex matches.
- `re.findall()`: Finds all substrings where the regex matches, and returns them as a list.
- `re.finditer()`: Similar to `findall()`, but returns an iterator yielding match objects.
- `re.sub()`: Replaces the matches with a given string.

- `re.split()`: Splits the string by the occurrences of the regex.

```
pythonCopy code
# Example usage of re.match()
pattern = r"hello"
text = "hello world"
match = re.match(pattern, text)
if match:
    print("Match found!")
else:
    print("No match.")
```

1. `re.match()`

The `re.match()` function checks for a match only at the beginning of the string.

```
pythonCopy code
import re

pattern = r"\d+"  # Matches one or more digits
string = "123abc456"

match = re.match(pattern, string)
if match:
    print("Match found:", match.group())
else:
    print("No match found.")
```

Output:

```
sqlCopy code
Match found: 123
```

Explanation:

`re.match()` only checks if the string starts with a number, and since `"123"` is at the beginning, it returns a match.

2. `re.search()`

The `re.search()` function scans through the string and returns the first match it finds.

```
pythonCopy code
import re

pattern = r"\d+"  # Matches one or more digits
string = "abc123def456"

search = re.search(pattern, string)
if search:
    print("Match found:", search.group())
else:
    print("No match found.")
```

Output:

```
sqlCopy code
Match found: 123
```

Explanation:

`re.search()` looks for the first occurrence of digits anywhere in the string. It finds `"123"` after `"abc"`.

3. `re.findall()`

The `re.findall()` function returns all matches of the pattern in the string as a list.

```
pythonCopy code
import re

pattern = r"\d+"  # Matches one or more digits
string = "abc123def456ghi789"

matches = re.findall(pattern, string)
print("All matches:", matches)
```

Output:

```
lessCopy code
All matches: ['123', '456', '789']
```

Explanation:

`re.findall()` finds all sequences of digits in the string and returns them as a list.

4. `re.finditer()`

The `re.finditer()` function returns an iterator yielding match objects for each match.

```
pythonCopy code
import re

pattern = r"\d+"  # Matches one or more digits
string = "abc123def456ghi789"

matches = re.finditer(pattern, string)
for match in matches:
    print("Match found:", match.group(), "at position:", matc
h.span())
```

Output:

```
arduinoCopy code
Match found: 123 at position: (3, 6)
Match found: 456 at position: (9, 12)
Match found: 789 at position: (15, 18)
```

Explanation:

`re.finditer()` returns an iterator of match objects, each containing details of the match, including its position in the string.

5. `re.sub()`

The `re.sub()` function replaces all occurrences of the pattern with a specified string.

```
pythonCopy code
import re

pattern = r"\d+"  # Matches one or more digits
string = "abc123def456ghi789"

new_string = re.sub(pattern, "#", string)
print("String after replacement:", new_string)
```

Output:

```
arduinoCopy code
String after replacement: abc#def#ghi#
```

Explanation:

`re.sub()` replaces all digit sequences in the string with `"#"`.

6. `re.split()`

The `re.split()` function splits the string at each occurrence of the pattern.

```
pythonCopy code
import re

pattern = r"\d+"  # Matches one or more digits
string = "abc123def456ghi789"

split_list = re.split(pattern, string)
print("Split string:", split_list)
```

Output:

```
csharpCopy code
Split string: ['abc', 'def', 'ghi', '']
```

2.2 Special Sequences

- `\d`: Matches any digit (0-9).
- `\D`: Matches any non-digit character.
- `\w`: Matches any alphanumeric character plus underscore.
- `\W`: Matches any non-alphanumeric character.
- `\s`: Matches any whitespace character.
- `\S`: Matches any non-whitespace character.

```

# Example: Matching a phone number
phone_pattern = r"\d{3}-\d{3}-\d{4}"
phone_text = "My phone number is 123-456-7890."
match = re.search(phone_pattern, phone_text)
```

```
if match:
    print(f"Phone number found: {match.group()}")
```

4. Common Regex Patterns

- **Email Validation:** `r"[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+"`
- **URL Validation:** `r"http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*(),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+"`
- **IP Address Validation:** `r"\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b"`

5. Practical Examples

5.1 Extracting All Emails from a Text

```
text = "Please contact us at support@example.com or sales@exa
mple.co.uk."
email_pattern = r"[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9
-.]+"
emails = re.findall(email_pattern, text)
print(emails)  # Output: ['support@example.com', 'sales@examp
le.co.uk']
```

5.2 Replacing All Digits with a Character

```
text = "My phone number is 123-456-7890."
masked_text = re.sub(r"\d", "*", text)
print(masked_text)  # Output: My phone number is ***-***-****
```

5.3 Splitting a String by Multiple Delimiters

```
text = "apple, orange; banana|grape"
split_pattern = r"[,;|]"
fruits = re.split(split_pattern, text)
print(fruits)  # Output: ['apple', 'orange', 'banana', 'grap
e']
```

Pratice Questions :

1. Extracting Dates from Text

If you need to extract dates from a text, regex can be very helpful.

```
import re

text = "The event is scheduled on 2023-10-15, and the registr
ation closes on 2023-09-30."

# Regex pattern to match dates in YYYY-MM-DD format
date_pattern = r"\b\d{4}-\d{2}-\d{2}\b"

# Find all dates in the text
dates = re.findall(date_pattern, text)
print(dates)  # Output: ['2023-10-15', '2023-09-30']
```

2. Validating a Password

Suppose you want to ensure that a password meets certain criteria, such as being at least 8 characters long and containing at least one uppercase letter, one lowercase letter, and one digit.

```
import re

password = "Passw0rd"

# Regex pattern for password validation
password_pattern = r"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[A-Za-z
\d]{8,}$"

# Validate password
if re.match(password_pattern, password):
    print("Password is valid")
else:
    print("Password is invalid")
```

3. Finding All Hashtags in a Tweet

If you're analyzing social media content, you might need to extract all hashtags from a tweet.

```
import re

tweet = "Loving the #Python community! #coding #programming"

# Regex pattern to match hashtags
hashtag_pattern = r"#\w+"

# Find all hashtags in the tweet
```

```
hashtags = re.findall(hashtag_pattern, tweet)
print(hashtags)  # Output: ['#Python', '#coding', '#programmi
ng']
```

4. Replacing Phone Numbers with a Placeholder

In scenarios where you need to anonymize phone numbers in text (e.g., for privacy reasons), regex can be used to replace them with a placeholder.

```
import re

text = "Contact me at 123-456-7890 or 987-654-3210."

# Regex pattern to match phone numbers
phone_pattern = r"\b\d{3}-\d{3}-\d{4}\b"

# Replace phone numbers with a placeholder
anonymized_text = re.sub(phone_pattern, "[REDACTED]", text)
print(anonymized_text)  # Output: Contact me at [REDACTED] or
[REDACTED].
```

5. Splitting a Paragraph into Sentences

You can use regex to split a large block of text into individual sentences, which can be useful in natural language processing.

```
import re

paragraph = "Hello world! How are you? This is a test."

# Regex pattern to split by sentence-ending punctuation
sentence_pattern = r"(?<=[.!?])\s+"
```

```
# Split paragraph into sentences
sentences = re.split(sentence_pattern, paragraph)
print(sentences)  # Output: ['Hello world!', 'How are you?',
'This is a test.']
```