Why Looping

In Python, looping is used to repeatedly execute a block of code multiple times. It is a fundamental concept in programming that allows developers to automate repetitive tasks and process large amounts of data efficiently.

Python provides two main types of loops:

1. **for loop**: A for loop is used to iterate over a sequence (such as a list, tuple, or string) and execute a block of code for each item in the sequence. The syntax for a for loop in Python is:

   ```
   for item in sequence:
       # do something with item
   ```

2. **while loop**: A while loop is used to repeatedly execute a block of code as long as a certain condition is true. The syntax for a while loop in Python is:

   ```
   while condition:
       # do something
   ```

Loops are a powerful tool in programming because they allow you to write code that can process large amounts of data without having to write the same code over and over again. By automating repetitive tasks with loops, you can save time and reduce the chances of making errors in your code.

While Loop

In Python, a while loop is used to repeatedly execute a block of code as long as a certain condition is true. The basic syntax for a while loop in Python is:

```
while condition:
    # execute this code while condition is True
```

Example 1 : Write a python to prints the numbers from 1 to 5 using while loop

```
i = 1
while i <= 5:
    print(i)
    i += 1
```

In this example, the loop starts with **i** equal to 1. The **while** condition **i** <= **5** is true, so the block of code inside the loop is executed. The value of **i** is printed, and then **i** is incremented by 1 with **i** += **1**. The loop then checks the condition again. This process continues until the condition is false (when **i** is greater than 5).

Example 2: Here's another example of a while loop that prompts the user to enter a number until they enter a valid integer:

```
valid_input = False
while not valid_input:
    user_input = input("Enter an integer: ")
    if user_input.isdigit():
        valid_input = True
    else:
        print("Invalid input. Please enter an integer.")
```

In this example, the loop starts with **valid_input** set to **False**. The loop continues to prompt the user for input and checks whether the input is a valid integer with the **isdigit()** method. If the input is valid, the loop sets **valid_input** to **True**, and the loop exits. If the input is invalid, the loop prints an error message and continues to prompt the user for input.

for loop

In Python, a for loop is used to iterate over a sequence (such as a list, tuple, or string) and execute a block of code for each item in the sequence. The basic syntax for a for loop in Python is:

```
for item in sequence:
    # execute this code for each item in the sequence
```

Example 1 : you can use a for loop to iterate over the characters in a string:

```
for char in "hello":
    print(char)
```

In this example, the loop starts with the first character in the string ("h"), prints it, and then moves on to the next character ("e") and executes the block of code again, printing "e". This process continues until the loop has iterated over all characters in the string.

You can also use the **range()** function to create a sequence of numbers and iterate over them with a for loop. For example, the following code prints the numbers from 0 to 4:

```
for i in range(5):
    print(i)
```

In this example, **range(5)** creates a sequence of numbers from 0 to 4. The loop starts with the first number in the sequence (0), prints it, and then moves on to the next number (1) and executes the block of code again, printing 1. This process continues until the loop has iterated overall numbers in the sequence.

Difference between the while and for loop

Feature	**`while` Loop**	**`for` Loop**
Structure	while condition: # body of the loop # code after the loop	for variable in iterable: # body of the loop # code after the loop

Example	count = 0
	while count < 5:
	    print(count)
	    count += 1
	print("Loop ended")
		numbers = [1, 2, 3, 4, 5]
		for num in numbers:
		        print(num)
		print("Loop ended")

Condition	Uses a condition to control the loop's execution. The loop runs as long as the condition is True.	Iterates over elements in an iterable one by one until the iterable is exhausted.
Infinite Loop	Possible to create an infinite loop if the condition never becomes False. Care must be taken to ensure the loop eventually terminates.	No risk of creating an infinite loop since it iterates over a finite number of elements in the iterable.
Control within Loop	Need to handle the increment or update of the loop variable (if applicable) manually within the loop block.	No need to handle iteration control manually; the loop automatically advances to the next element.
Use of Range function	Can be used with range() to achieve similar functionality as a for loop.	Primarily used with range() to control the number of iterations.

Example with Range	start = 1
	end = 6
	
	while start < end:
	    print("Iteration:", start)
	    start += 1
		`for i in range(5):`
		`    print(i)`

Break

In Python, **break** is a keyword used within **loops** (such as for or while loops) to immediately terminate the loop and resume execution at the next statement after the loop.

Here's an **example** of using **break** in a **while loop** to **exit** the loop when a certain condition is met:

```
i = 0
while i < 10:
    if i == 5:
        break
    print(i)
    i += 1
```

Output:

```
0
1
2
3
4
```

In this example, the loop iterates through the values of **i** from 0 to 9. However, when **i** becomes equal to 5, the **break** statement is executed, causing the loop to immediately terminate and resume execution at the statement after the loop.

Continue

In Python, **continue** is a **keyword** used **within loops** (such as for or while loops) to skip the rest of the current iteration and move on to the next iteration.

Example of using **continue** in a for loop to skip printing even numbers:

```
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
```

Output:

```
1
3
5
7
9
```

In this example, the **loop** iterates through the values of **i** from 0 to 9. However, when **i** is even, the continue statement is executed, causing the rest of the current iteration to be skipped and moving on to the next iteration.

Difference between break and continue statement

Feature	**break Statement**	**continue Statement**
Keyword	**break**	**continue**
Purpose	It is used to exit the current loop prematurely when a condition is encounter.	It is used to skip the rest of the current iteration and continue to the next iteration.
Applicable Loops	Works with for, while, and nested loops.	Works with for and while loops.
Loop Termination	Exits the loop entirely when the break statement is encountered.	Skips the remaining code in the current iteration and proceeds to the next iteration.

Example	`for i in range(1, 6):`
	`    if i == 3:`
	`        break`
	`    print(i)`
		`for i in range(1, 6):`
		`    if i == 3:`
		`        continue`
		`    print(i)`

Output	The loop stops when i equals 3 and only prints 1 and 2.	The loop skips printing 3 but continues to print 1, 2, 4, and 5.
Use Cases	Useful when you want to exit the loop prematurely based on a specific condition.	Useful when you want to skip certain iterations based on a condition, but not terminate the loop entirely.

while with else

In Python, a **while loop** can be used with an **else** block, which is executed when the condition of the while loop becomes false.

Example of a while loop with an else block:

```
i = 0
while i < 5:
    print(i)
    i += 1
else:
    print("The loop has ended.")
```

 Output:

```
0
1
```

```
2
3
4
 The loop has ended.
```

In this example, the **while** loop iterates through the values of **i** from 0 to 4 and prints each value. After the loop terminates because the condition **i < 5** becomes false, the **else** block is executed and "The loop has ended." is printed.

The **else** block in a **while** loop is useful when you want to execute some code only after the loop has completed all iterations without being terminated by a **break** statement. If a **break** statement is executed inside the loop, the **else** block will not be executed.

Difference between while and while with else loop

Feature	**`while` Loop**	**`while` Loop with `else`**

Syntax	`while condition:`	`while condition:`
		` # Loop block`
		`else:`
		` # Else block`

Purpose	It is used for repetitive tasks.	It is used when you want to execute some code if the loop completes without encountering a `break` statement.
Else Block	Not applicable.	The `else` block executes only if the loop completes without a `break` statement.

Example	`count = 0`
	`while count < 5:`
	`  print(count)`
	`  count += 1`
		`count = 0`
		`while count < 5:`
		` print(count)`
		` count += 1`
		`else:`
		` print("Loop completed without a break.")`

Break Statement Effect	The `break` statement can be used to exit the loop prematurely.	The `break` statement can be used to exit the loop prematurely and skip the `else` block. If a `break` statement is encountered, the `else` block will not be executed.

Infinite while Loop in Python

If the condition of a loop is always True, the loop runs for infinite times (until the memory is full).

Example

age = 36

```
# the test condition is always True
while age > 18:
    print('You can vote')
```

In this program the condition always evaluates to True. Hence, the loop body will run for infinite times.

Example Problems

Example 1 : Write a program to take the input from user and prints the sum of the entered numbers until the user enters a negative number:

```
total = 0
while True:
    num = int(input("Enter a number: "))
    if num < 0:
        break
    total += num

print("The sum of the numbers entered is:", total)
```

In this program, the **while** loop continues to prompt the user to enter a number until the user enters a negative number. If the user enters a negative number, the break statement is executed, causing the loop to terminate. Otherwise, the entered number is added to the total variable.

Finally, the sum of the entered numbers is printed.

Example 2 Write a program that uses a `while` loop to repeatedly prompt the user to enter a password until the correct password is entered:

```
password = "abc123"

while True:
    guess = input("Please enter the password: ")
    if guess == password:
        print("Password accepted!")
        break
    else:
        print("Incorrect password. Please try again.")
```

In this program, the **while** loop runs indefinitely until the correct password is entered. Inside the loop, the program prompts the user to enter a password using the **input()** function. If the user's guess matches the correct password (which is stored in the password variable), the program uses the **break** statement to **exit** the loop. If the user's guess does not match the correct password, the program prints an error message and continues the loop.

Once the correct password is entered, the program prints a message indicating that the password was accepted.

Example 3 : Write a program that uses a while loop to check if a given number is equal to its reverse:

```
n = int(input("Please enter a number: "))
origin = n
reverse_n = 0

while n > 0:
   digit = n % 10
   reverse_n = reverse_n * 10 + digit
   n = n // 10

if origin == reverse_n:
   print("The number is a palindrome.")
else:
   print("The number is not a palindrome.")
```

In this program, the user is prompted to enter a number using the **input()** function, which is then converted to an integer using the **int()** function. The program creates two variables, **origin** and **reverse_n**, to store the original number and its reverse, respectively. The variable n is used to keep track of the remaining digits of the number during the loop.

Inside the **while** loop, the program repeatedly extracts the rightmost digit of n using the modulus operator (**%**) and adds it to **reverse_n** multiplied by 10 to shift the existing digits to the left. The program then updates n by dividing it by 10 (using integer division //) to remove the rightmost digit.

Once the loop finishes, the program checks if **origin** is equal to **reverse_n**. If they are equal, the program prints a message indicating that the number is a palindrome (i.e., equal to its reverse). If they are not equal, the program prints a message indicating that the number is not a palindrome.

Example 4 : Write a program that uses a for loop with the range() function to iterate over a sequence of numbers and print each number:

```
for i in range(1, 11):
   print(i)
```

In this program, the **range()** function is used to generate a sequence of numbers from 1 to 10 (inclusive) that is then iterated over by the for loop. The **range()** function takes two arguments: the starting value (1 in this case) and the ending value plus one (11 in this case). The for loop assigns each value in the sequence to the variable **i**, which is then printed using the **print()** function.

Example 4 : Write a program that uses a `while` loop to calculate the factorial of a number

```
n = int(input("Enter a number: "))
fact = 1

while n >= 1 :

  fact = fact * n

  n = n – 1

print("The factorial of", n, "is:", fact)
```

In this program, the user is prompted to enter a number using the **input()** function, and the **int()** function is used to convert the input string to an integer. The variable **fact** is initialized to **1**, and the while loop check the condition **n** is greater than or equal to n, if the condition true multiplying each number by the fact variable using the ***=** operator followed by decrement the n value by 1. Once the condition fails, the program prints a message indicating the factorial of the input number.

Example 5 : Write a program that uses a `for` loop to calculate the factorial of a number:

```
n = int(input("Enter a number: "))
fact = 1

for i in range(1, n+1):
   fact *= i
print("The factorial of", n, "is:", fact)
```

In this program, the user is prompted to enter a number using the **input()** function, and the **int()** function is used to convert the input string to an integer. The variable **fact** is initialized to **1**, and the for loop iterates over the range of numbers from **1** to **n** (inclusive), multiplying each number by the fact variable using the ***=** operator. Once the loop finishes, the program prints a message indicating the factorial of the input number.

Example 6 : Write a program that uses a `for` loop to iterate over a sequence of characters in a string:

```
string = "Hello, world!"

for char in string:
   print(char)
```

In this program, the string "Hello, world!" is assigned to the variable string. The **for** loop iterates over each character in the string, assigning each character to the variable char in turn. The **print()** function is called inside the loop to print each character on a separate line.

When you run this program, it will output the following sequence of characters:

```
H
e
```

```
l
l
o
,

w
o
r
l
d
!
```

This demonstrates how the **for** loop can be used to iterate over a sequence of elements, such as the characters in a string.

Exercise Problems

1. Write a program that prompts the user to enter a number and uses a for loop to print the first n Fibonacci numbers. The Fibonacci sequence is defined by the recurrence relation F(n) = F(n-1) + F(n-2), with F(0) = 0 and F(1) = 1.
2. Write a program that prompts the user to enter a string and uses a while loop to reverse the order of the characters in the string. For example, if the user enters the string "hello", the program should output "olleh".
3. Write a program that uses a for loop to iterate over the numbers from 1 to 100, printing "Fizz" for multiples of 3, "Buzz" for multiples of 5, and "FizzBuzz" for multiples of both 3 and 5. The program should print the number itself for all other numbers.
4. Write a program that prompts the user to enter a list of integers and uses a while loop to calculate the sum of the squares of the integers in the list.
5. Write a program that uses a for loop to iterate over the numbers from 1 to 100, printing the numbers that are divisible by 3 or 5. The program should print "Fizz" for multiples of 3, "Buzz" for multiples of 5, and "FizzBuzz" for multiples of both 3 and 5.
6. Write a program that prompts the user to enter a positive integer, and then uses a while loop to print all the even numbers from 2 up to the entered number (inclusive). If the user enters a non-positive integer, the program should print an error message and terminate using the break statement. If the loop completes without encountering a non-positive integer, the program should print "Done!" using the else clause.
7. Write a program that prompts the user to enter a string, and then uses a for loop to print each character of the string on a separate line.
8. Write a program that prompts the user to enter a positive integer, and then uses a while loop to compute the factorial of that integer.
9. Write a Python program that prompts the user to enter a positive integer and then prints the sum of all the even numbers from 1 to that integer.
10. Write a program that prompts the user to enter n integers and then uses a for loop to compute the product of those integers.
11. Write a program that prompts the user to enter a string, and then uses a while loop to count the number of vowels (i.e. 'a', 'e', 'i', 'o', and 'u') in the string.

12. Write a program that prompts the user to enter n strings and then uses a for loop to concatenate those strings into a single string.
13. Write a program that prompts the user to enter n integers and then uses a for loop to compute the average of those integers.
14. Write a program that prompts the user to enter a string, and then uses a while loop to check if the string is a palindrome (i.e. reads the same forwards and backwards).
15. Write a program that prompts the user to enter n integers and then uses a for loop to find the smallest and largest values from the n integers.
16. Write a program that prompts the user to enter a positive integer and then prints all the divisors of that integer.
17. Write a program that prompts the user to enter a password. The password must be at least 8 characters long and must contain at least one uppercase letter, one lowercase letter, and one number. If the password meets these requirements, print "Password accepted." Otherwise, print "Invalid password."