Break and Continue Statement in Python

Break and Continue Statement in Python

In Python, both break and continue are **loop control statements** — they let you **change the flow** of a loop before it finishes naturally. They are used in **for** loops and **while** loops.

What it does:

- The **break** statement is used to **exit (stop)** the loop immediately.
- It **ends** the loop even if the loop condition is still true.
- After break, control moves to the **next line after the loop**.

Purpose of break

It immediately stops the current loop (either for or while) and jumps to the first line of code after the loop.

Why use break

- To stop a loop when a certain condition is met.
- To avoid unnecessary iterations.
- Useful in search operations, menus, or infinite loops.

Rules of Break Statement

Rule No.	Rule	Explanation / Behavior
1	Used only inside loops	break must be inside a for or while loop; using it outside causes a SyntaxError.
2	Exits the loop immediately	When break is executed, the loop stops, regardless of the loop condition.
3	Affects only the innermost loop	In nested loops, break exits only the current (innermost) loop.
4	Commonly used with if condition	break is usually used with if to stop the loop when a condition is met.
5	Skips the loop's else block	If break is executed, the else part of the loop is not executed.
6	Code after the loop still executes	After breaking from the loop, the program continues with the next statements normally.
7	Doesn't pause or wait	break immediately jumps out of the loop—it doesn't wait or delay.
8	Works in both for and while loops	Can be used in any type of loop for early termination.

Break and Continue Statement in Python

Solved Example 1: Using break in a for loop

```
for i in range(1, 11):
    if i == 6:
        break
    print(i)
```

Output:

1
2
3
4
5

Explanation:

- The loop starts from 1 and goes up to 10.
- When i becomes 6, break stops the loop immediately.
- So, numbers after 5 are not printed.

Solved Example 2: Using break in a while loop

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

Output:

1
2
3
4
5

Explanation:

The loop was supposed to run till 10, but it stops when i becomes 5.

When to use break:

- To stop a loop early when a condition is met.
- To exit a search when you've found what you need.
- To avoid unnecessary looping after a goal is achieved.

Break and Continue Statement in Python

Solved Example 3: Stop loop when an item is found

```
fruits = ["apple", "banana", "cherry", "mango"]

for fruit in fruits:
    if fruit == "cherry":
        print("Cherry found!")
        break
    print("Checking:", fruit)
```

Output:

```
Checking: apple
Checking: banana
Cherry found!
```

Solved Example 4: Using break in a menu-based program

```
while True:
    print("\n1. Say Hello")
    print("2. Say Bye")
    print("3. Exit")
    choice = input("Choose an option: ")

    if choice == '1':
        print("Hello!")
    elif choice == '2':
        print("Bye!")
    elif choice == '3':
        print("Exiting...")
        break
    else:
        print("Invalid choice!")
```

```
1. Say Hello
2. Say Bye
3. Exit
Choose an option:
```

Solved Example 5: Using break - Stop when user enters "stop"

```
while True:
    name = input("Enter a name (or type 'stop' to end): ")
    if name.lower() == "stop":
        print("Loop ended by user.")
        break
    print("You entered:", name)
```

Output:

```
Enter a name (or type 'stop' to end): Sanjeev
You entered: Sanjeev
```

Break and Continue Statement in Python

```
Enter a name (or type 'stop' to end): Aman
You entered: Aman
Enter a name (or type 'stop' to end): stop
Loop ended by user.
```

Explanation:
- The loop keeps asking for names.
- When the user types "stop", the break statement stops the loop.

Solved Example 6: Using break - Stop when user enters a negative number

```
while True:
    num = int(input("Enter a number (negative number to stop): "))
    if num < 0:
        print("Negative number entered. Stopping loop.")
        break
    print("Square of", num, "is", num ** 2)
```

Output:

```
Enter a number (negative number to stop): 5
Square of 5 is 25
Enter a number (negative number to stop): 2
Square of 2 is 4
Enter a number (negative number to stop): -1
Negative number entered. Stopping loop.
```

Explanation:
- The loop keeps running until the user enters a negative number.
- The break statement stops the loop immediately.

Break and Continue Statement in Python

The Continue Statement

What it does:
- The **continue** statement **skips the current iteration** of the loop.
- The loop **doesn't stop**; it simply **jumps to the next cycle.**
- It's used when you want to **ignore** some values or conditions.

Solved Example 1: Using continue in a for loop

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

Output:

```
1
2
4
5
```

Explanation:
When i is 3, the loop **skips** that iteration — so 3 isn't printed.

Solved Example 2: Using continue in a while loop

```
i = 0
while i < 5:
    i += 1
    if i == 2:
        continue
    print("Number:", i)
```

Output:

```
Number: 1
Number: 3
Number: 4
Number: 5
```

Explanation:
When i == 2, the loop **skips printing** and goes to the next cycle.

When to use continue:
- When you want to **skip certain values** or conditions.
- To **ignore errors or unwanted data** while processing.

Break and Continue Statement in Python

- To **filter results** during looping.

Solved Example 3: Skip even numbers

```
for i in range(1, 11):
    if i % 2 == 0:
        continue
    print(i)
```

Output:

```
1
3
5
7
9
```

Explanation:
The loop **skips even numbers**, printing only odd ones.

Solved Example 4: Using continue - Skip even numbers

```
for i in range(1, 11):
    if i % 2 == 0:
        continue
    print(i)
```

Output:

```
1
3
5
7
9
```

Solved Example 5: Using continue - Skip numbers less than 10

```
for i in range(5):
    num = int(input("Enter a number: "))
    if num < 10:
        print("Too small, skipping...")
        continue
    print("Number accepted:", num)
```

Output:

```
Enter a number: 5
```

Break and Continue Statement in Python

```
Too small, skipping...
Enter a number: 12
Number accepted: 12
Enter a number: 8
Too small, skipping...
```

Explanation:

- If user enters a number less than 10, that iteration is **skipped**.
- For valid numbers (≥10), the loop continues normally.

Solved **Example 6: Combined break and continue**

Write a Python program that repeatedly asks the user to enter a number until the user enters 0.

- If the entered number is negative, the program should ignore it and display the message "Negative number ignored!".
- If the entered number is positive, the program should display its square.
- When the user enters 0, the program should stop and display "Loop stopped by user."

Use the break and continue statements appropriately to control the loop.

Solution:

```
while True:
    num = int(input("Enter a number (0 to stop): "))
    if num == 0:
        print("Loop stopped by user.")
        break
    if num < 0:
        print("Negative number ignored!")
        continue
    print("Square is:", num ** 2)
```

Explanation:

- If user enters **negative**, that input is **skipped** (continue).
- If user enters **0**, the loop **stops completely** (break).

Break and Continue Statement in Python

Difference Between Continue and Break Statement

Feature / Aspect	continue Statement	break Statement
Purpose	Skips the current iteration and moves to the next one	Exits the loop entirely
Loop Execution	Loop continues after skipping rest of current iteration	Loop is terminated immediately
Affects	Only the current iteration	The entire loop
Usage Scenario	When you want to skip some specific values or conditions	When a condition is met and further looping is unnecessary
Works with	for and while loops	for and while loops
Position in Loop	Usually inside a conditional statement	Usually inside a conditional statement
Example	if x == 3: continue → skips when x == 3	if x == 3: break → stops when x == 3
Effect on Loop Counter	Continues with the next iteration	Stops loop, counter doesn't increment further