Conditionals and Recursion in Python

Conditionals in Python

Conditionals are fundamental control structures in Python that allow the execution of different code blocks based on conditions. They are primarily implemented using if, elif, and else statements.

if Statement

The if statement evaluates a condition and executes a block of code only if the condition is True.

```
x = 10
if x > 5:
    print("x is greater than 5")
```

if-else Statement

The if-else statement provides an alternative execution path when the condition is False.

```
x = 3
if x > 5:
    print("x is greater than 5")
else:
    print("x is not greater than 5")
```

if-elif-else Statement

The if-elif-else structure allows checking multiple conditions sequentially.

```
x = 7
if x > 10:
    print("x is greater than 10")
elif x > 5:
    print("x is greater than 5 but less than or equal to 10")
else:
    print("x is 5 or less")
```

Nested Conditionals

Conditionals can be nested inside one another for more complex decision-making.

```
x = 8
y = 3
```

```
if x > 5:
    if y < 5:
        print("x is greater than 5 and y is less than 5")
```

Logical Operators

Python supports logical operators (and, or, not) to combine multiple conditions.

```
x = 8
y = 6
if x > 5 and y > 5:
    print("Both x and y are greater than 5")
```

Recursion in Python

Recursion is a programming technique where a function calls itself to solve a problem. It is commonly used for problems that can be broken down into smaller subproblems of the same type.

Basic Recursive Function

A simple example of recursion is a function to calculate the factorial of a number.

```
def factorial(n):
    if n == 0 or n == 1:  # Base case
        return 1
    else:
        return n * factorial(n - 1)  # Recursive call


print(factorial(5))  # Output: 120
```

Understanding Recursion

1. **Base Case:** A condition that stops the recursion.
2. **Recursive Case:** The function calls itself with modified arguments.
3. **Stack Overflow:** Excessive recursive calls can lead to a stack overflow error due to deep recursion.

Fibonacci Sequence Using Recursion

The Fibonacci sequence can be generated recursively.

```
def fibonacci(n):
    if n <= 0:
```

```
    return 0
  elif n == 1:
    return 1
  else:
    return fibonacci(n - 1) + fibonacci(n - 2)

print(fibonacci(6))  # Output: 8
```

Advantages and Disadvantages of Recursion

Advantages:

- Simplifies code for problems like tree traversal and backtracking.
- Provides a clear, logical approach to divide-and-conquer problems.

Disadvantages:

- Consumes more memory due to function call stack.
- Slower than iterative approaches for some problems.