Condition

Kittikun Jitpairod

Introduction to Conditional Statements

Conditional statements in Python, as in many programming languages, enable decision-making processes within code. These statements facilitate the execution of different code blocks based on varying conditions.

The primary mechanism for implementing conditional statements in Python is the `if` statement, complemented by `else` and `elif` clauses. These constructs allow for the selective execution of code blocks contingent upon the truth value of specified conditions.

The `if` Statement

The `if` statement represents the most fundamental form of a conditional statement. It permits the execution of a code block only when a specified condition evaluates to true.

Consider this example:

```
age = 20

if age >= 18:
    print("You are an adult.")

print("This will always be printed.")
```

```
You are an adult.
This will always be printed.
```

In this instance, the condition `age >= 18` evaluates to True, resulting in the execution of the indented code block. The final print statement, being outside the if block, is executed regardless of the condition's truth value.

It is important to note that Python utilizes indentation to define code blocks. The indented code under the `if` statement is executed only when the condition evaluates to True.

```
temperature = 15

if temperature < 20:
    print("It's a bit chilly.")
    print("You might want to wear a jacket.")

print("Enjoy your day!")
```

```
It's a bit chilly.
You might want to wear a jacket.
Enjoy your day!
```

The `else` Clause

The `else` clause provides an alternative execution path when the condition in the `if` statement evaluates to False:

```
age = 15

if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

```
You are a minor.
```

The `else` clause specifies an alternative code block to be executed when the `if` condition is False.

```
is_raining = True

if is_raining:
    print("Remember to take an umbrella!")
else:
    print("Enjoy the sunny weather!")
```

```
Remember to take an umbrella!
```

For scenarios requiring multiple condition checks, Python offers the `elif` (short for “else if”) clause as an efficient alternative to nested if-else statements:

```
score = 65

if score >= 80:
    print("A")
elif score >= 70:
    print("B")
elif score >= 60:
    print("C")
elif score >= 50:
    print("D")
else:
    print("F")
```

```
C
```

```
day = "Wednesday"

if day == "Monday":
    print("Start of the work week.")
elif day == "Friday":
    print("TGIF!")
elif day == "Saturday" or day == "Sunday":
    print("Weekend!")
else:
    print("Midweek.")
```

```
Midweek.
```

Multiple `elif` clauses can be employed as needed. Python evaluates each condition sequentially and executes the code block associated with the first True condition encountered.

Nested Conditional Statements

Conditional statements can be embedded within other conditional statements, a practice known as nesting:

```
is_sunny = True
temperature = 28

if is_sunny:
    if temperature > 25:
        print("It's a hot, sunny day!")
    else:
```

```
        print("It's a mild, sunny day.")
else:
    print("It's not sunny today.")
```

```
It's a hot, sunny day!
```

While nested conditionals can facilitate more complex decision-making processes, excessive nesting should be avoided as it can compromise code readability.

Conditional Expressions (Ternary Operator)

Python provides a concise syntax for simple if-else statements, often referred to as the ternary operator:

```
age = 20

status = "adult" if age >= 18 else "minor"
print(status)
```

```
adult
```

The syntax follows the pattern: `value_if_true if condition else value_if_false`

While this construct can be useful for simple conditions, full if-else statements are generally preferred for more complex logic to maintain code readability.