Loop

Kittikun Jitpairod

Introduction to Loops

Loops in Python provide a mechanism for executing a set of instructions repeatedly. They are fundamental constructs in programming that allow for efficient handling of iterative tasks. Python offers two primary types of loops: 'for' loops and 'while' loops, each serving different purposes in code execution.

The `for` Loop

The `for` loop in Python is designed to iterate over a sequence (such as a list, tuple, string, or range) or other iterable objects. It executes a set of statements once for each item in the sequence.

Basic syntax:

```
for item in sequence:
    # code to be executed
```

Example:

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

```
0
1
2
3
4
```

`range()` function return a sequence of a specific integer.

Loop througn list items.

```python
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

```
apple
banana
cherry
```

The `while` Loop

The `while` loop executes a set of statements as long as a given condition is true. It's particularly useful when the number of iterations is not known beforehand.

Basic syntax:

```python
while condition:
    # code to be executed
```

Example:

```python
count = 0
while count < 5:
    print(count)
    count += 1
```

```
0
1
2
3
4
```

Loop Control Statements

Python provides several statements to control the flow of loops:

1. `break`: Terminates the loop prematurely
2. `continue`: Skips the rest of the code inside the loop for the current iteration
3. `else`: Specifies a block of code to be executed when the loop is exhausted

Example:

```
for num in range(10):
    if num == 5:
        break
    print(num)
else:
    print("Loop completed normally")
```

```
0
1
2
3
4
```

```
for num in range(10):
    if num <= 7:
        continue
    print(num)
else:
    print("Loop completed normally")
```

```
8
9
Loop completed normally
```

Nested Loops

Loops can be nested inside other loops, allowing for more complex iterations:

```
A = [1, 2, 3]
B = ['a', 'b', 'c', 'd']

for i in A:
    for j in B:
        print(f"({i}, {j})")
```

```
(1, a)
(1, b)
(1, c)
(1, d)
(2, a)
(2, b)
(2, c)
(2, d)
```

```
(3, a)
(3, b)
(3, c)
(3, d)
```

List Comprehensions

Python offers a concise way to create lists based on existing lists, known as list comprehensions:

```
squares = [x**2 for x in range(10)]
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

This is equivalent to:

```
squares = []
for x in range(10):
    squares.append(x**2)
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```