Python LOOPS

Soumyadeep Sinha

Overview

In Python, loops are used to execute a block of code repeatedly, either for a specified number of times or until a particular condition is met.

Python primarily supports two types of loops:

1. **For Loop**
2. **While Loop**

Additionally, Python provides control statements to modify the flow of loops:

1. break
2. continue
3. pass

For Loop

The for loop in Python is used to iterate over a sequence (like a list, tuple, dictionary, set, or string). It executes a block of code for each element in the sequence.

Example:

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

For Loop using range() function

The range() function is used to generate a sequence of numbers for looping. It takes three parameters:

range(start, stop, step)

- start: Starting value (inclusive)
- stop: Ending value (exclusive)
- step: Step interval (can be positive or negative)

```
for i in range(1, 11, 2):
    print(i)
```

Nested For Loops

```python
for i in range(3):
    for j in range(2):
        print(f"i = {i}, j = {j}")
```

```
i = 0, j = 0
i = 0, j = 1
i = 1, j = 0
i = 1, j = 1
i = 2, j = 0
i = 2, j = 1
```

While Loop

A while loop executes a block of code as long as a specified condition is true. It's useful when the number of iterations is not known beforehand.

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

Infinite Loops

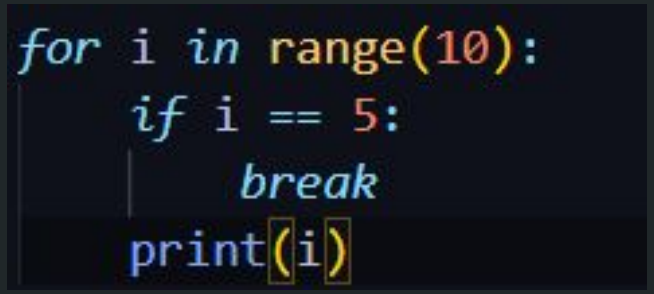
If the condition of a while loop never becomes false, it creates an infinite loop.

Control Statements

Control statements can alter the flow of a loop.

Break Statement

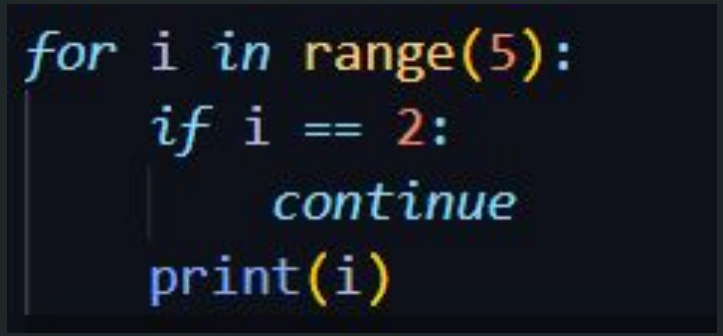
The break statement terminates the loop immediately and transfers control to the statement following the loop.



Control Statements

Continue Statement

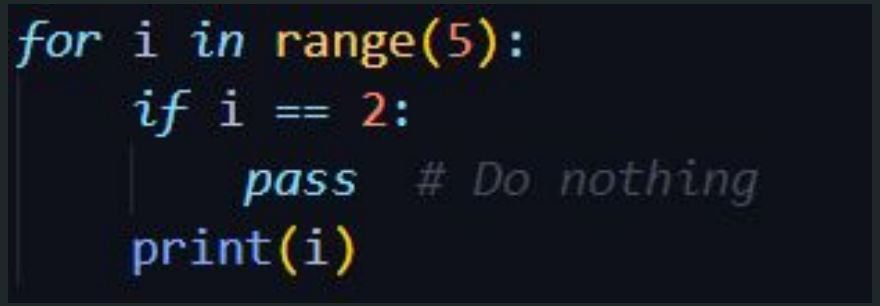
The continue statement skips the current iteration and proceeds to the next iteration of the loop.



Control Statements

Pass Statement

The pass statement is a null operation. it is syntactically required when we do not want to execute any code. It is often used as a placeholder.



List Comprehensions

Python supports a concise way to create lists using list comprehensions, which can replace some for loops.

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

Advanced Loop Techniques

Enumerate

The enumerate() function adds a counter to an iterable, allowing us to loop through both the index and the value at the same time.

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

for index, fruit in enumerate(fruits):
    print(f"Index: {index}, Fruit: {fruit}")
```

Advanced Loop Techniques

Zip

The zip() function allows us to iterate over multiple iterables (like lists) in parallel. This is particularly useful for combining data from different sources.

```
names = ["Alice", "Bob", "Charlie"]
scores = [85, 90, 95]

for name, score in zip(names, scores):
    print(f"{name} scored {score}")
```

Advanced Loop Techniques

Using else with Loops

Python allows us to use an else statement after a loop. The else block is executed when the loop completes normally (i.e., it doesn't encounter a break statement).

```
for i in range(5):
    print(i)
else:
    print("Loop completed without interruption.")
```

Advanced Loop Techniques

Looping through Dictionaries

We can loop through a dictionary in several ways: keys, values, or key-value pairs.

```
person = {"name": "Alice", "age": 30, "city": "New York"}

# Loop through keys
for key in person:
    print(key)

# Loop through values
for value in person.values():
    print(value)

# Loop through key-value pairs
for key, value in person.items():
    print(f"{key}: {value}")
```

Advanced Loop Techniques

List Comprehension with Conditions

List comprehensions can include conditions, allowing for more complex and elegant solutions.

```
evens = [x for x in range(21) if x % 2 == 0]
print(evens)
```

Looping through Strings

Strings in Python can be iterated over character by character, which is useful for various string manipulations.

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

We can also iterate with indices to manipulate or analyze specific positions.

```
for index in range(len(text)):
    print(f"Character at index {index} is {text[index]}")
```

THANK YOU