Python Fundamentals

Loops in programming are control structures that allow you to execute a block of code repeatedly. They are essential for automating repetitive tasks, iterating through collections of data, and performing operations until a specific condition is met. Loops help you avoid writing the same code multiple times and make your programs more efficient and concise.

Loop control refers to the techniques and statements used to manage the flow of execution within loops, such as for and while loops, in programming. Loop control allows you to control how many times a loop iterates, skip certain iterations, or prematurely exit the loop based on specific conditions. In Python, you can achieve loop control using various statements and techniques. Let's explore loop control statements and their syntax with examples for both for and while loops.

Types of Loops

In programming, there are primarily two types of loops: for loops and while loops. These loops serve different purposes and are used in various situations depending on the requirements of your code.

For Loops:

Syntax: For loops are used when you want to iterate over a sequence (such as a list, tuple, string, or range) and perform a block of code for each item in the sequence.

for item in sequence:

```
# Code to be executed for each item
```

Examples

```
# Looping through a list of names:

names = ['Alice', 'Bob', 'Charlie']
for name in names:
    print(name)

# Looping through a range of numbers:

for num in range(1, 5):
    print(num)
```

Increment & Decrement

```
x = 5
x += 1  # Increment x by 1
print(x)  # Output: 6

y = 10
y -= 1  # Decrement y by 1
print(y)  # Output: 9

# Incrementing a variable using a for loop
incremented_value = 0  # Initial value

for _ in range(5):  # Loop 5 times
    incremented_value += 1  # Increment by 1 in each iteration

print("Incremented Value:", incremented_value)
```

In this code, we start with an initial value of 0 and use a for loop to increment it by 1 in each iteration. After the loop, the value of incremented_value will be 5.

Decrementing a Variable using a For Loop:

```
# Decrementing a variable using a for loop
decremented_value = 10  # Initial value

for _ in range(5):  # Loop 5 times
    decremented_value -= 1  # Decrement by 1 in each iteration

print("Decremented Value:", decremented_value)
```

In this code, we start with an initial value of 10 and use a for loop to decrement it by 1 in each iteration. After the loop, the value of decremented_value will be 5.

While Loops

Syntax: While loops are used when you want to execute a block of code as long as a certain condition is true. The code inside the loop will continue to execute until the condition becomes false.

while condition:

```
# Code to be executed repeatedly
```

Examples

```
# Looping until a condition is met:

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

# Looping until a condition is met:

count = 10
while count > 5:
    print(count)
    count -= 1

# Prompting user input until a valid response is given:

while True:
    user_input = "yes"
    if user_input.lower() == 'yes' or user_input.lower() == 'no':
        break
    else:
        print("Invalid input. Please enter 'yes' or 'no'.")
        break
```

In addition to these two primary loop types, Python also provides loop control statements like break and continue to modify the behavior of loops. These statements allow you to exit a loop prematurely (break) or skip the current iteration and move to the next one (continue) based on specific conditions.

Different types of loops are chosen based on the problem you're trying to solve. for loops are commonly used for tasks that involve iterating over known sequences, while while loops are used when you need to repeatedly execute code until a certain condition is satisfied.

Sequence Generation using Loops

```
# -20, -17, -14, -11, .... 10
a0 = -20
d = 3
terms = 10
for i in range(terms+1):
    term = a0 + i*d
    print(term)

# 2, 6, 18, 54, 162, 486,
a0 = 2
r = 3
terms = 12
for i in range(terms+1):
```

```
    term = a0*(r**i)
    print(term)


# 1, 4 , 3, 8, 5, 12, 7, 16, ...
a0 = 1
d1 = 2
b0 = 4
d2 = 4

terms = 15
combined_sequence = []
for i in range(terms+1):
    if i%2 ==0:
        term = a0 + (i//2)*d1
        print(term)
    else:
        term = b0 + (i//2)*d2
        print(term)
    combined_sequence.append(term)

print(combined_sequence)

# -10, 2, 0, -5, 4, 1, 0, 6, 2, 5, 8, 3, 10, 10, 4,  15, 12, 5

a0= -10
d1 = 5
b0 = 2
d2 = 2
c0 = 0
d3 = 1

terms = 20

for i in range(terms+1):
    if i%3 == 0:
        term = a0 + (i//3)*d1
        print(term)
    elif i%3 == 1:
        term = b0 + (i//3)*d2
        print(term)
    else:
        term = c0 + (i//3)*d3
        print(term)


2, 3, 5, 6, 8, 12, 11, 24, 14, 48, 17, 96, 20, 192, 23

a0 = 2
d1= 3
b0 = 3
```

```
r = 2

terms = 12

for i in range(terms+1):
    if i%2 == 0:
        term = a0 + (i//2)*d1
        print(term)
    else:
        term = b0*r**(i//2)
        print(term)
```