Chapter 5 Python Basics to Advanced Control Flow

Control Flow directs program execution through structures like loops, conditionals, and functions, determining the order and path of operations.

IF Statements:

An if statement in Python checks whether a condition is true or false. If the condition is true, the code inside the if block runs. If False, the code is skipped. It's used to make decisions in the program, executing specific actions based on conditions.

example:

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

Else and elif statements:

In Python, else and elif statement are used alongside if to handle multiple conditions and alternative actions.

- **elif (else if):**
  checks another condition if the previous if was False. You can have multiple elif statements

- **else:**
  Runs when none of the elif or if conditions are same true. Its the 'default' action.

example:

```
temperature = 75

if temperature > 85:
    print ("Its too hot outside")
elif temperature > 76:
    print ("Its a nice day")
elif temperature > 50:
    print ("Its a bit chilly")
else:
    print("Its cold outside")
```

Nested if Statement:

A nested 'if statement in Python is an 'if statement inside and if. It lets you check multiple related conditions in sequence.

For example:

If you first check the weather and its sunny, you can then check how many guests are coming. Depending on the number of guests, you decide between different activities, like a barbecue or picnic.

If the weather isn't sunny, you skip the nested checks and go straight to an alternative action, like staying indoors.

```
temperature = 78
humidity = 65

if temperature > 70:
    printf "The weather is warm")
    if humidity > 60:
        print ("Its also quite humid")
    else:
        print("The humidity is low.")
else:
    print("The weather is cool")
```

@curious_.programmer

```
if humidity > 60:
    print("Its humid despite the cool")
else:
    print("The weather is cool and dry")
```

The while loop:

A 'while loop' in Python repeatedly executes a block of code as long as a specified condition is true.

It first checks the condition; if true, the code inside runs. After each iteration, the condition is rechecked. The loop continues until the condition becomes false.

For example:

A while loop can keep counting up as long as the count is below a certain number. It's useful for scenarios where you don't know in advance how many iterations are needed.

```
count = 0
while count < 5:
    print("Count is :", count)
    count += 1

print("Loop ended")
```

@ curious_.programmer

The For Loop :

A For loop in Python is used to iterate over a sequence, such as a list, tuple, or range, executing a block of code for each item in the sequence.

Unlike a while loop which runs untill a condition is false, a for loop runs a set number of times based on the length of the sequence.

For example:

It can go through a list of numbers, processing each one in turn. It's ideal for repetitive tasks like iterating over data collections.

example:

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

For fruit in fruits:
    print (fruit)

print("Loop ended")
```

Loop Control Statements:

Loop control statements in Python allow

you to alter the flow of loop execution. They include:

break:

This is statement immediately exits the loop, regardless of whether the loop's condition is still true. It's useful for stopping a loop when a specific condition is met, like when searching for an item in a list and finding it before the loop has iterated through the entire list.

Example:

```
for i in range(10):
    if i==5:
        break
    print("Current number:", i)
print("Loop ended")
```

Continue:

This statement skips the rest of the current loop iteration and proceeds to the next iteration. It's helpful for bypassing certain parts of the loop based on a condition, like skipping even numbers in a loop that processes a range of numbers.

```
for i in range(10):
    if i % 2 == 0:
        continue
    print("Odd number:", i)
print("Loop ended")
```

Using Range() in for loop:

The range() function in Python is commonly used in for loops to iterate over a sequence of numbers.

```
for i in range(5):
    print("Number:", i)
```

Here's a basic rundown of how it works:

start: The starting value of the sequence (inclusive). If ommitted, it defaults to 0.

stop: The ending value of the sequence (exclusive). The loop will run until it reaches this value.

step: The amount by which the sequence is incremented. If ommitted, it defaults to 1.

You can use range() in a for loop for various tasks like iterating through lists, generating sequencing of numbers, or performing repetitive actions a specific number of times.

example:

```
For i in range(10, 0, -1):
    print("Number:", i)
```