CHAPTER 2

Control Flow Structures

Topics:

1. **Introduction to Control Structures**

Decision Making Structures :-

1. **if, if-else statements**
2. **Nested if-else and if-elif-else statements**
3. **switch statement**

Loops

1. **for loop**
2. **while loop**
3. **Nested loops**
4. **break, continue and pass statements**

1. **Introduction to Control Structures**

=> Control structures in Python are fundamental for directing the flow of a program. They enable you to make decisions, repeat actions, and handle different scenarios

Conditional Statements

If if-else elif

Loops

for while

Control Flow Statements

Beak Continue pass

Decision Making Structures :-

Conditional statements allow you to control the flow of your program based on certain conditions.

1. **if, if-else statements**

=> **1. if Statement**

The if statement executes a block of code if its condition is true. If the condition is false, the block of code inside the if statement is skipped.

Syntax:

```
if condition:
 # Code to execute if the condition is true
```

Example:

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

=> **2. Nested if-else and if-elif-else statements**

The if-else statement allows you to execute one block of code if the condition is true and another block if the condition is false.

Syntax:

```
if condition:
   # Code to execute if the condition is true
else:
   # Code to execute if the condition is false
```

Example:

```
age = 16
if age >= 18:
   print("You are an adult.")
else:
   print("You are a minor.")
```

=> **Nested if-else Statements**

You can also nest if-else statements inside one another to handle more complex conditions.

```
age = 20
if age >= 18:
        if age < 21:
                print("You are an adult, but not old enough to drink alcohol
                in the US.")
        else:
                print("You are an adult and can drink alcohol in the US.")
else:
        print("You are a minor.")
```

=> **Multiple if-elif-else Statements**

For handling multiple conditions, you can use if-elif-else statements. elif stands for "else if" and allows you to check multiple conditions sequentially.

```
temperature = 30
if temperature < 0:
    print("It's freezing cold!")
elif 0 <= temperature < 20:
    print("It's cold.")
elif 20 <= temperature < 30:
    print("It's warm.")
else:
    print("It's hot.")
```

3. **switch statement**

=> Python does not have a built-in switch statement like some other programming languages such as C or Java. However, you can achieve similar functionality using dictionaries or a series of if-elif-else statements.

=> **Match Statements (Python 3.10+)**

Starting from Python 3.10, you can use the match statement, which is similar to a switch statement and can be more expressive and readable.

Example:

```
case_number = 2
match case_number:
    case 1:
        result = "This is case 1."
    case 2:
        result = "This is case 2."
    case 3:
        result = "This is case 3."
    case _:
        result = "This is the default case."

print(result)  # Output: This is case 2.
```

Loops :-

1. **for loop**

=> In Python, a for loop is used to iterate over a sequence (like a list, tuple, dictionary, set, or string) or other iterable objects. It allows you to execute a block of code multiple times, once for each item in the sequence.

Basic Syntax

```
for variable in iterable:
        # Code to execute for each item
```

Examples

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


for fruit in fruits:
   print(fruit)
```

op
```
apple
banana
cherry
```

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

op
```
0
1
2
3
4
```

2. **while loop**

=> In Python, a while loop repeatedly executes a block of code as long as a given condition is true. It's useful for situations where you don't know in advance how many times you need to repeat an action but have a condition that determines when to stop.

Basic Syntax

```
while condition:
    # Code to execute while the condition is true
```

Example

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

3. **Nested Loop in Python**

=> Nested loops in Python involve placing one loop inside another. This allows you to perform complex iterations and operations that require multiple levels of looping. You can nest both for loops and while loops.

Syntax:

```
for outer_variable in outer_iterable:
        for inner_variable in inner_iterable:
                # Code to execute for each combination of outer and inner
                variables
```

Example:

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

op

```
i: 0, j: 0
i: 0, j: 1
i: 1, j: 0
i: 1, j: 1
i: 2, j: 0
i: 2, j: 1
```

4. **break, continue and pass statements**

=> **1. break Statement**

The break statement is used to exit from the innermost loop that encloses it. When break is encountered, the loop terminates immediately, and control is transferred to the statement following the loop.

Example:

```
for i in range(5):
        if i == 3:
                break
        print(i)
```

op

```
0
1
2
```

=> **2. continue Statement**

The continue statement is used to skip the remaining code inside the current iteration of the loop and proceed to the next iteration. It effectively skips the rest of the code in the loop for the current iteration and jumps to the next iteration.

Example:

```
for i in range(5):
        if i == 3:
                continue
        print(i)
```

op

```
0
1
2
4
```

=> **3. pass Statement**

The pass statement is a placeholder that does nothing. It is used where syntax requires a statement but where no action is needed. It is often used as a placeholder in loops, functions, classes, or conditionals during development.

Example:

```
for i in range(5):
   if i == 3:
      pass  # Placeholder for future code
   else:
      print(i)
```

op

```
0
1
2
4
```