Lesson 3: Control Structures in Procedural Programming

Control structures are fundamental constructs in procedural programming that allow programmers to control the flow of execution within a program. They enable decisions to be made and repetitive tasks to be performed based on specific conditions. Control structures provide the means to create dynamic and interactive programs.

Conditional statements (if-else, switch)

Conditional statements are essential control structures in procedural programming that allow you to make decisions and execute different blocks of code based on specific conditions. Two common types of conditional statements are "if-else" and "switch" statements.

if-else statement:

The "if-else" statement is used to make binary decisions, executing one block of code if a certain condition is true and another block of code if the condition is false. The syntax for the "if-else" statement is as follows:

```
if condition:
    # Code block executed if the condition is true
else:
    # Code block executed if the condition is false
```

The "else" block is optional, and if the condition in the "if" statement is true, only the code within the "if" block is executed.

```
age = 25
if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

switch statement (or case statement):

The "switch" statement is used to select one of many code blocks to be executed based on the value of an expression. While "switch" statements are prevalent in some programming languages (like C and C++), they are not natively supported in Python.

In Python, you typically use a series of "if-elif-else" statements to achieve similar functionality as a "switch" statement.

Example in Python (equivalent of a switch statement):

```
day_of_week = 3

if day_of_week == 1:
    print("Monday")
elif day_of_week == 2:
    print("Tuesday")
elif day_of_week == 3:
    print("Wednesday")
elif day_of_week == 4:
    print("Thursday")
elif day_of_week == 5:
    print("Friday")
else:
    print("Weekend")
```

In this example, the program determines the day of the week based on the value of the "day_of_week" variable.

Note: Some programming languages, like Python, do not have a direct "switch" statement, as the "if-elif-else" construct is often sufficient to handle conditional scenarios. If you need more complex decision-making, you can utilize nested "if-else" statements or other data structures like dictionaries for mapping keys to values.

Looping structures (for, while, do-while)

Looping structures are control structures in procedural programming that allow you to repeat a block of code multiple times based on specific conditions. They are used to perform iterative tasks, process collections of data, and implement repetitive operations efficiently. The three common types of looping structures in procedural programming are "for," "while," and "do-while" loops.

for loop:

The "for" loop is used when you know the number of iterations you want to perform. It allows you to execute a block of code a fixed number of times. The syntax for a "for" loop is as follows:

```
   for variable in range(start, stop, step):
       # Code block executed in each iteration
```

- The `**variable**` is a loop control variable that takes values from the `**start**` value to `**stop-1**` value with a specified `**step**`. By default, `**start**` is 0, and `**step**` is 1.
- The "for" loop is particularly useful when dealing with sequences like lists, strings, or arrays.

```
for i in range(1, 6):
    print("Iteration:", i)
```

while loop:

The "while" loop is used when you want to repeat a block of code as long as a certain condition is true. It keeps executing the code block until the condition becomes false. The syntax for a "while" loop is as follows:

```
while condition:
    # Code block executed as long as the condition is true
```

- The "while" loop may execute indefinitely if the condition never becomes false, so it's essential to ensure that the condition eventually becomes false to exit the loop.

```
count = 0
```

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

do-while loop:
The "do-while" loop is not natively available in Python and some other languages. Unlike the "while" loop, a "do-while" loop executes the code block first and then checks the condition. It ensures that the code block is executed at least once, even if the condition is false initially.

While Python doesn't have a built-in "do-while" loop, a similar behavior can be achieved using a "while" loop with a break statement.

Example in Python (equivalent of a do-while loop):

```
count = 0
while True:
    print("Count:", count)
    count += 1
    if count >= 5:
        break
```

In procedural programming, the choice of looping structure depends on the specific problem and the type of data being processed. "for" loops are ideal for iterating over sequences with a known number of elements, while "while" loops are useful when the exact number of iterations is uncertain or determined by a dynamic condition.

Using control structures for decision-making and repetition

Control structures are essential tools in procedural programming for decision-making and repetition. They allow you to create dynamic and interactive programs by executing different blocks of code based on specific conditions and repeating tasks until certain criteria are met. Let's explore how control structures can be used for decision-making and repetition:

Decision-Making with Control Structures:

Conditional statements, such as "if-else" and "switch" (if available), are used for decision-making in procedural programming. They allow the program to make choices and execute different blocks of code based on specific conditions.

Example: A simple program to check if a number is positive or negative.

```
# Decision-Making with "if-else" statement
number = int(input("Enter a number: "))

if number > 0:
    print("The number is positive.")
elif number < 0:
    print("The number is negative.")
else:
    print("The number is zero.")
```

Repetition with Control Structures:

Looping structures like "for" and "while" loops enable repetition in procedural programming. They allow you to execute a block of code multiple times until a certain condition is met.

Example: A program to calculate the sum of numbers from 1 to n using a "for" loop.

```
# Repetition with "for" loop
n = int(input("Enter a positive integer: "))
sum_result = 0

for i in range(1, n + 1):
    sum_result += i

print("The sum of numbers from 1 to", n, "is:", sum_result)
```

Example: A program to calculate the factorial of a number using a "while" loop.

```
# Repetition with "while" loop
num = int(input("Enter a non-negative integer: "))
factorial = 1
i = 1

while i <= num:
    factorial *= i
    i += 1

print("The factorial of", num, "is:", factorial)
```

By combining decision-making with conditional statements and repetition with looping structures, you can build powerful procedural programs to solve a wide range of problems. Control structures make your programs more flexible, interactive, and capable of handling diverse scenarios, allowing you to efficiently process data, perform calculations, and implement various algorithms.

Handling nested control structures

Handling nested control structures involves using one or more control structures (e.g., conditional statements or loops) within another control structure. Nested control structures are commonly used in procedural programming to solve complex problems that require multiple levels of decision-making or iteration.

The key to effectively handling nested control structures is to ensure proper indentation and logical organization of code. Indentation is essential to visually represent the hierarchy of control structures and helps maintain code readability.

Here are some common examples of nested control structures:

Nested "if-else" statements:

You can have one or more "if-else" statements nested within another "if" or "else" block. This is useful when you need to consider multiple conditions or make more fine-grained decisions based on different cases.

Example: A program to determine the grade based on marks scored.

```
marks = int(input("Enter your marks: "))

if marks >= 90:
    print("Grade: A+")
else:
    if marks >= 80:
        print("Grade: A")
    else:
        if marks >= 70:
            print("Grade: B")
        else:
            print("Grade: C or below")
```

Nested loops:

You can have one or more loops (e.g., "for" or "while" loops) nested within another loop. This is useful when you need to process multidimensional data or perform repetitive tasks in a structured manner.

Example: A program to print a pattern using nested loops.

```
for i in range(5):
    for j in range(i + 1):
        print("*", end="")
    print()  # Move to the next line after each row
```

Combination of "if" and loops:

You can combine conditional statements and loops to create more sophisticated control flow in your programs.

Example: A program to print odd numbers up to a specified limit.

```
limit = int(input("Enter the limit: "))

for num in range(1, limit + 1):
    if num % 2 != 0:
        print(num, end=" ")
```

When dealing with nested control structures, it is essential to pay attention to code indentation and maintain proper formatting to ensure the logic is clear and easy to follow. Additionally, consider the performance implications of nested loops, as they can lead to increased execution time for larger data sets.

Nested control structures are a powerful tool in procedural programming, as they allow you to solve complex problems by breaking them down into smaller, more manageable steps. However, it's crucial to strike a balance between complexity and readability, ensuring that the code remains understandable and maintainable.