
WELCOME
TechPro

[Nasos Lentzas]

[Control Flow]

[10/10/2024]

TechPro

In this Course

1. Control flow
2. Conditional statements
3. Looping

TechPro

Control Flow in programming

Control flow refers to the order in which statements and instructions are executed in a program. It determines how the program progresses from one instruction to another based on certain conditions and logic. Control flow mechanisms allow developers to create dynamic and flexible programs that can make decisions, repeat tasks, and respond to various inputs.

TechPro

Control flow types

- Conditional statements:
  - If-else
  - Switch-case
- Looping Statements:
  - For
  - While
  - Do-while
- Jump Statements:
  - Break
  - Continue
  - Return
  - goto

TechPro

Conditional statements

Conditional statements in programming are used to control the flow of a program based on certain conditions. These statements allow the execution of different code blocks depending on whether a specified condition evaluates to true or false, providing a fundamental mechanism for decision-making in algorithms

- If
- If-else
- If-else-if
- Switch
- Ternary operator

TechPro

If conditional statement

The if statement is the most basic form of conditional statement. It checks if a condition is true. If it is, the program executes a block of code.

- Use cases
  - Checking a single condition and executing code based on its result.
  - Performing actions based on user input.
- Simple & straight forward
- Useful for basic decision logic

TechPro

```
if (condition) {
    // code to execute if condition is true
}
```

If-else conditional statement

The if-else statement extends the if statement by adding an else clause. If the condition is false, the program executes the code in the else block.

- Use cases:
  - Executing one block of code if a condition is true and another block if it’s false.
  - Handling binary decisions.
- Clear and concise syntax.
- Handles binary decisions efficiently.

TechPro

```
if (condition) {
    // code to execute if condition is true
} else {
    // code to execute if condition is false
}
```

If-else if conditional statement

The if-else if statement allows for multiple conditions to be checked in sequence. If the if condition is false, the program checks the next else if condition, and so on. In else if statements, the conditions are checked from the top-down, if the first block returns true, the second and the third blocks will not be checked, but if the first if block returns false, the second block will be checked. This checking continues until a block returns a true outcome.

- Handling multiple conditions sequentially.
- Implementing multi-way decision logic.
- Reduces the need for nested if-else statements.

TechPro

```
if (condition1) {
    // code to execute if condition1 is true
} else if (condition2) {
    // code to execute if condition2 is true
} else {
    // code to execute if all conditions are false
}
```

Switch conditional statement

The switch statement is used when you need to check a variable against a series of values. It’s often used as a more readable alternative to a long if-else if chain.
In switch expressions, each block is terminated by a break keyword. The statements in switch are expressed with cases.

- Selecting one of many code blocks to execute based on the value of a variable.
- Handling multiple cases efficiently.
- Improves code readability when dealing with many conditions.

```
switch (variable) {
    case value1:
        // code to execute if variable equals value1
        break;
    case value2:
        // code to execute if variable equals value2
        break;
    default:
        // code to execute if variable doesn't match any value
}
```

TechPro

Ternary expression conditional statement

The ternary operator is a shorthand way of writing an if-else statement. It takes three operands: a condition, a result for when the condition is true, and a result for when the condition is false.

- Concise conditional assignment.
- Inline conditional assignment.
- Suitable for simple conditional assignments.
- Can reduce code readability, especially for complex conditions or expressions.

TechPro

```
condition ? result_if_true : result_if_false
```

Best practices

- **Keep it simple**: Avoid complex conditions that are hard to understand. Break them down into simpler parts if necessary.
- **Use meaningful names**: Your variable and function names should make it clear what conditions you're checking.
- **Avoid deep nesting**: Deeply nested conditional statements can be hard to read and understand. Consider using early returns or breaking your code into smaller functions.
- **Comment your code**: Explain what your conditions are checking and why. This can be especially helpful for complex conditions.

Quiz

Write a program that given a number ([1,31]) representing the day, will print the day of the week.

Quiz

Write an algorithm that given a year will determine if its leap or not.

Looping

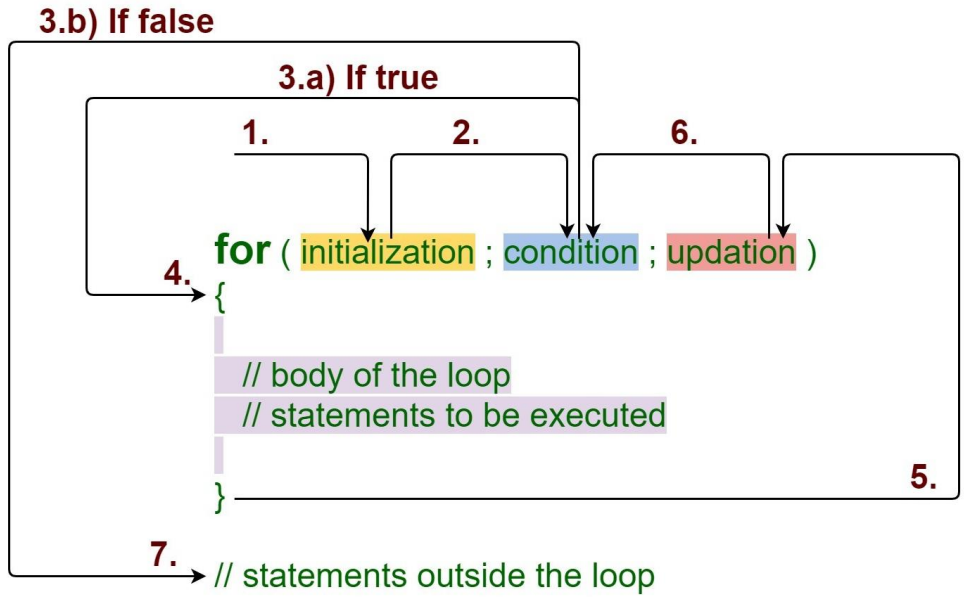
Loops, also known as iterative statements, are used when we need to execute a block of code repetitively. Loops in programming are control flow structures that enable the repeated execution of a set of instructions or code block as long as a specified condition is met. Loops are fundamental to the concept of iteration in programming, enhancing code efficiency, readability and promoting the reuse of code logic.

- Entry-Controlled loops: the test condition is checked before entering the main body of the loop. For Loop and While Loop is Entry-controlled loops.
- Exit-Controlled loops: the test condition is evaluated at the end of the loop body. The loop body will execute at least once, irrespective of whether the condition is true or false. Do-while Loop is an example of Exit Controlled loop.

TechPro

For Loop

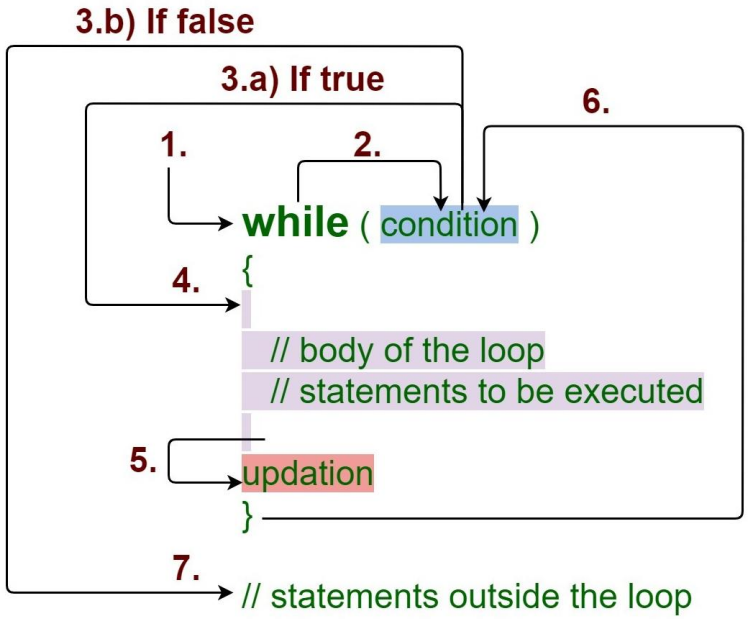
For loop in programming is a control flow structure that iterates over a sequence of elements, such as a range of numbers, items in a list, or characters in a string. The loop is entry-controlled because it determines the number of iterations before entering the loop.



TechPro

While Loop

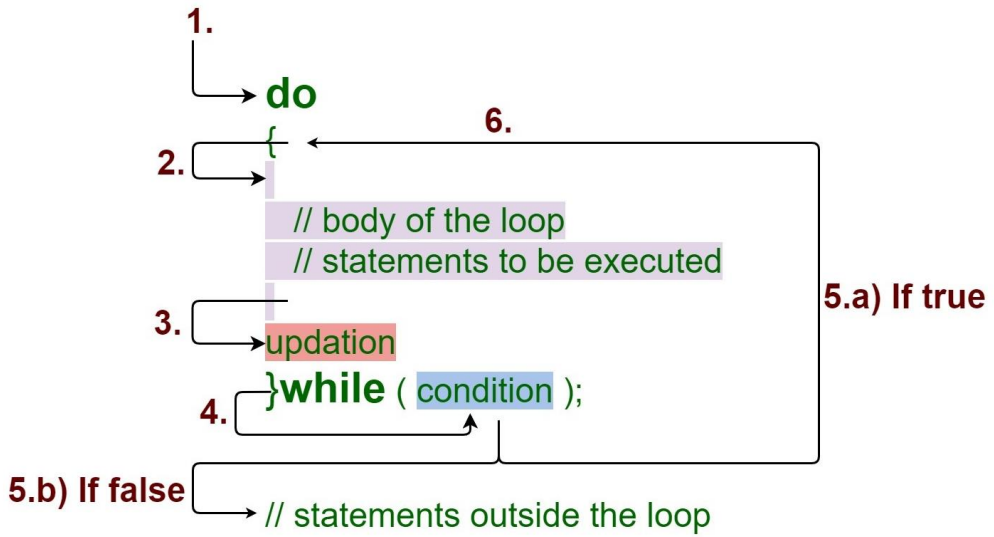
A while loop in programming is an entry-controlled control flow structure that repeatedly executes a block of code as long as a specified condition is true. The loop continues to iterate while the condition remains true, and it terminates once the condition evaluates to false.



TechPro

Do-While Loop

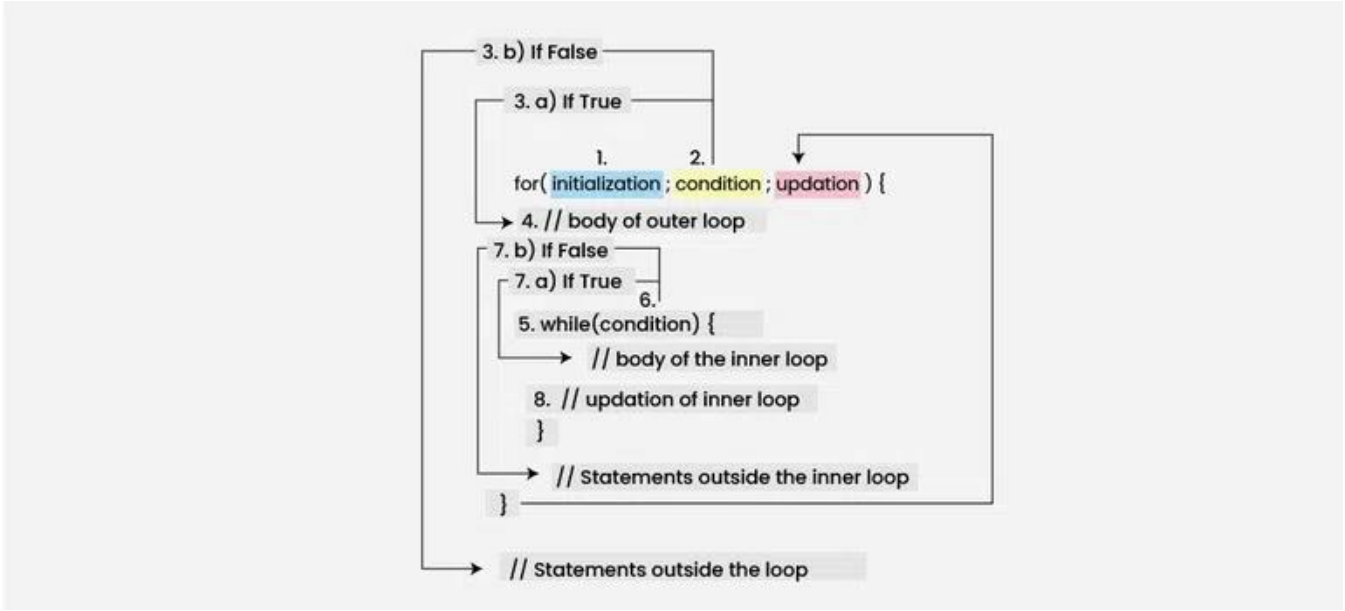
A do-while loop in programming is an exit-controlled control flow structure that executes a block of code at least once and then repeatedly executes the block as long as a specified condition remains true. The distinctive feature of a do-while loop is that the condition is evaluated after the code block, ensuring that the block is executed at least once, even if the condition is initially false.



TechPro

Nested Loops

Nested loops in programming are when one loop is placed inside another. This allows for the iteration of one or more loops within the body of another loop. Each time the outer loop executes, the inner loop completes its full iteration. Nested loops are commonly employed for tasks involving multidimensional data processing, pattern printing, or handling complex data structures. Efficient use of nested loops is essential, considering potential impacts on code readability and execution time for larger datasets.



TechPro

Loops common mistakes

- **Infinite Loops**: Not ensuring that the loop condition will eventually become false can lead to infinite loops, causing the program to hang.
- **Off-by-One Errors**: Mismanaging loop counters or conditions can lead to off-by-one errors, either skipping the first iteration or causing an extra, unintended iteration.
- **Uninitialized Variables**: Forgetting to initialize loop control variables properly can result in unpredictable behavior.
- **Inefficient Nesting**: Excessive or inefficient use of nested loops can lead to performance issues, especially with large datasets.
- **Modifying Loop Variable:** Inside the Loop: Changing the loop variable inside the loop can lead to unexpected behavior. For example, modifying the iterator variable in a for loop.

TechPro

Jump statements

Jump statements in programming are used to change the flow of control within a program. They allow the programmer to transfer program control to different parts of the code based on certain conditions or requirements.

- Break: primarily used to exit from loops prematurely. When encountered inside a loop, it terminates the loop's execution and transfers control to the statement immediately following the loop.
- Continue: used to skip the current iteration of a loop and proceed to the next iteration
- Return: used to exit a function and optionally return a value to the caller.
- Goto: allows transferring control to a labeled statement within the same function or block of code. The use of goto is generally discouraged due to its potential for creating unreadable and unmaintainable code.

TechPro

Armstrong Numbers

Write an algorithm that will determine whether a number is Armstrong or not. Armstrong numbers are numbers that are equal to the sum of their digits to the power of the number of digits.
e.g. 153 is Armstrong since: $1^3+5^3+3^3 = 153$

TechPro

Kaprekar Numbers

Write an algorithm that will determine whether a number is Kaprekar or not. A number is Kaprekar if the square of the number can be split into two parts A & B such as the sum of the two parts equals the initial number. Each part can have zeros before a number, part A can be 0 but part B must be >0.
e.g.

- 9 is Kaprekar: $9^2 = 81$. 81 can be split into A = 8 and B=1 => A+B =9.
- 297 is Kaprekar: $297^2 = 88209$. 88209 can be split into A = 88 and B=209 => A+B =297.

TechPro

Prime Factors

Every integer > 1 can be written as the product of prime numbers (e.g. 6 = 2*3). Write a program that given an integer num (>2) will print in ascending order its prime number factors.
e.g.

- 7 -> 7
- 8-> 2 2 2
- 147->3 7 7

TechPro

Feedback Discussion

TechPro

Thank you

TechPro