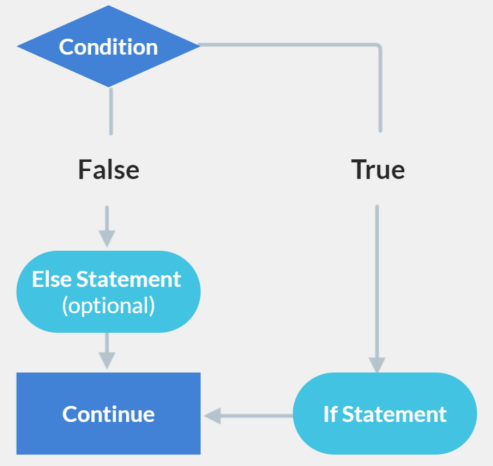
What Is Control Flow?

Control flow refers to methods of executing code blocks in a certain order based on specific conditions or criteria. These methods include **branching** and **looping**. Control flow is important in programming because it allows developers to control the behavior of their programs and make them more efficient and flexible.

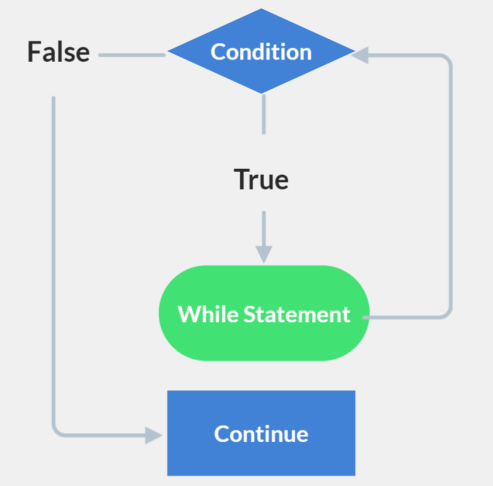
Conditional Statements (Branching)

Conditional statements allow the execution of different code paths based on certain conditions. The most common type of conditional statement is the **if statement**, which executes a block of code if a given condition is true, and otherwise executes another block of code within an **else statement**, or does nothing. This enables programmers to make their programs more responsive to different scenarios. "If" statements can be **nested** (an "if" statement within another "if" statement.)



Loops

Loops are used in programming to repeatedly execute a set of instructions. They let the programmer automate repetitive tasks and iterate over collections of data. There are two commonly used types of loops: **for loops**, which are used to iterate once per every item in a sequence of values, and **while loops**, which are used to repeatedly execute code until a certain condition is met.



Exception Handling

Exception handling allows programmers to handle errors that would otherwise crash the program. When an **exception** occurs, the program can "catch" it and take appropriate actions, such as displaying an error message or executing alternate code to handle the situation.

In Python, exception handling is done with **try** and **except** statements. If a block of code within a “try” statement would normally crash, the code execution is redirected to the corresponding “except” statement, which can handle the exception and take appropriate actions. For example, if a program attempts to divide by zero inside of a “try” block, the "except" block could display an error message instead of crashing.

Functions

Functions are blocks of code that perform a specific task, and can be **called** multiple times throughout a program. They help to break down large programs into smaller, more manageable parts. Functions easily allow for code reuse, since the same function can be called from different parts of a program with different inputs resulting in different outputs.

In Python, functions are **defined** using the "def" keyword, followed by the function name and any necessary **parameters** (variables which need to be filled with arguments when the function is called). This is followed by the **body** code, which is indented. Everything inside the body will execute every time the function is called, substituting any parameter placeholders with whatever arguments were passed into the function call.

Functions may or may not take one or more inputs (called **arguments**) and may or may not **return** an output value. In this example, the function add_numbers receives two arguments (x and y) and returns a value (sum).

```
# Defining the function
def add_numbers(x, y):
    sum = x + y
    return sum

# Calling the function
result = add_numbers(5, 7)
```