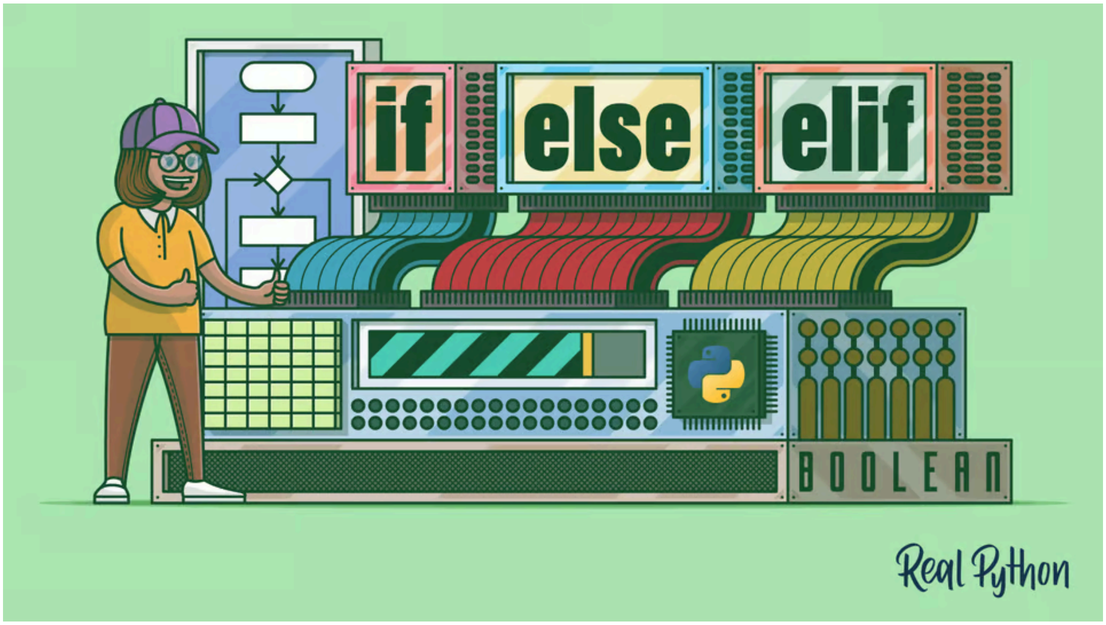
8. Conditional Statements

Conditional statements are essential for creating dynamic and flexible Python programs. They allow you to make decisions based on specific conditions and execute different code blocks accordingly.

We'll learn about the following topics:

- 8.1. Creating Sets
- 8.2. If Statement
- 8.3. Else Statement
- 8.4. Elif Statement
- 8.5. Logical Operators
- 8.6. Conditional Expressions (Ternary Operators)



8.1. Introduction to Python Statements:

- **Indentation**: Python uses indentation to define blocks of code. It is essential for maintaining the structure and readability of your code. Here's an example:

In [1]:
```
if 5 > 3:
    print('Five is greater than three')
```

```
Five is greater than three
```

In this example, the print statement is indented with four spaces. The indentation indicates that the print statement is part of the if block. If the condition 5 > 3 evaluates to True, the indented code block will execute. After `:` if you push Enter, it will automatically creates the indent.

8.2. If Statement:

The **if** statement allows you to execute a block of code conditionally. It is structured with the following components:

1. The `if` keyword: This is the keyword that marks the beginning of the if statement.

2. Condition: After the if keyword, you write a condition that evaluates to either True or False. This condition determines whether the associated block of code should be executed. You can use various comparison operators (such as <, >, ==, !=, <=, >=, etc.) to create conditions.

3. Colon (`:`): After the condition, you need to include a colon (:) at the end of the line. The colon is a syntax requirement and indicates that the code block associated with the if statement follows.

4. Indented code block: The code block associated with the if statement must be indented. It typically consists of one or more lines of code that are executed when the condition is True. Python uses indentation (usually four spaces or a tab) to determine the beginning and end of the code block. It's crucial to maintain consistent indentation throughout your code.

`if condition: code block to execute if the condition is True`

Here's an example:

In [2]:
```
x = 5

if x > 3:
    print('x is greater than 3')
```

```
x is greater than 3
```

In this example, the condition x > 3 checks if the value of x is greater than 3. If the condition is True, the code inside the indented block will be executed. In this case, it will print the message 'x is greater than 3'.

In [3]:
```
x = 2

if x > 3:
    print('x is greater than 3')
```

If the condition is False, the code block is skipped.

```
In [4]: x = 5

        if x > 3:
            print('x is greater than 3')
            print('This line is also inside the if block')

        print('This line is outside the if block')
```

```
x is greater than 3
This line is also inside the if block
This line is outside the if block
```

In this example, both print statements are indented and will be executed if the condition x > 3 is True. The last print statement, which is not indented, will be executed regardless of the condition.

```
In [5]: x = 0
        y = 5

        if x < y:
            print('yes')
```

```
yes
```

```
In [6]: if 'hello' in ['hello', 'world', 'python']:
            print('Condition is true')
```

```
Condition is true
```

8.3. Else Statement:

The **else** statement is used in conjunction with an if statement and provides an alternative block of code to execute when the condition of the if statement is False. Here's the general syntax:

` if condition: code block to execute if the condition is True else: code block to execute if the condition is False`

The else statement does not require a condition. It is executed only when the preceding if statement's condition is False. Only one of the two code blocks (either the one following the if statement or the one following the else statement) will be executed.

```
In [7]: x = 5

        if x > 10:
            print('x is greater than 10')
        else:
            print('x is less than or equal to 10')
```

```
x is less than or equal to 10
```

In this example, if x is greater than 10, the first print statement will be executed. Otherwise, if x is less than or equal to 10, the second print statement (inside the else block) will be executed.

8.4. Elif Statement:

The **elif** (short for "else if") statement allows you to check multiple conditions after an initial if statement. It provides an additional condition to be evaluated if the preceding if or elif statements' conditions are False.

`if condition1: code block to execute if condition1 is True elif condition2: code block to execute if condition2 is True else: code block to execute if all conditions are False`

You can have multiple elif statements to check different conditions in sequence. The code blocks associated with elif statements are executed only if their respective conditions evaluate to True.

In [8]:
```python
x = 5

if x > 10:
    print('x is greater than 10')
elif x > 5:
    print('x is greater than 5')
else:
    print('x is less than or equal to 5')
```

```
x is less than or equal to 5
```

In this example, the conditions are evaluated sequentially. If x is greater than 10, the first print statement is executed. If x is not greater than 10 but is greater than 5, the second print statement is executed. If neither condition is met, the code inside the else block is executed.

In [9]:
```python
x = 10

if x > 0:
    print('x is positive')
elif x == 0:
    print('x is zero')
else:
    print('x is negative')
```

```
x is positive
```

you can have multiple if statements in Python. Each if statement will be evaluated independently, regardless of the other if statements.

In [10]:
```
x = 5

if x > 10:
    print('x is greater than 10')

if x > 4:
    print('x is greater than 5')

if x > 0:
    print('x is positive')
```

```
x is greater than 5
x is positive
```

If you want to create mutually exclusive conditions where only one block should execute, you can use if, elif, and else statements together. Keep in mind that elif statements can be more than one in mutually exclusive conditions.

In [11]:
```
x = 5

if x > 10:
    print('x is greater than 10')
elif x > 7:
    print('x is greater than 7')
elif x > 5:
    print('x is greater than 5')
elif x > 3:
    print('x is greater than 3')
else:
    print('x is less than or equal to 3')
```

```
x is greater than 3
```

In this example, we have multiple elif statements that are evaluated in sequence. Each elif statement provides an additional condition to be checked if the preceding conditions are False. Only the code block associated with the first elif statement (whose condition evaluates to True) will be executed.

Let's assume x is 5 in this example. Here's how the conditions are evaluated:

The first elif condition x > 10 is False, so the associated code block is not executed.

The second elif condition x > 7 is also False, so its code block is skipped.

The third elif condition x > 5 is False, so its code block is skipped as well.

The fourth elif condition x > 3 is True, so the associated code block print("x is greater than 3") is executed.

8.5. Logical Operators:

and operator: The first if statement checks two conditions using the and operator. The code block is executed only if both conditions (x > 3 and y < 15) are True.

or operator: The second if statement checks two conditions using the or operator. The code block is executed if at least one of the conditions (x > 7 or y > 15) is True.

not operator: The third if statement uses the not operator to negate a condition. The code block is executed only if the negation of the condition (not x > 10) is True.

Here's the example that demonstrates the use of logical operators (or, and, etc.) in conditional statements in Python.

In [12]:
```python
x = 5
y = 10

if x > 3 and y < 15:
    print('Both conditions are satisfied')

if x > 7 or y > 15:
    print('At least one condition is satisfied')

if not x > 10:
    print('Negation of the condition is satisfied')
```

```
Both conditions are satisfied
Negation of the condition is satisfied
```

Assuming x is 5 and y is 10, here's how the conditions are evaluated:

In the first if statement, both conditions are satisfied (x > 3 is True and y < 15 is True), so the code block is executed and the message "Both conditions are satisfied" is printed.

In the second if statement, the first condition x > 7 is False, but the second condition y > 15 is False as well. Since neither condition is satisfied, the code block is not executed.

In the third if statement, the condition x > 10 is False. However, the not operator negates the condition, resulting in True. Hence, the code block is executed, and the message "Negation of the condition is satisfied" is printed.

8.6. Conditional Expressions (Ternary Operators):

The purpose of the conditional expression is to provide a concise way to choose between two values or expressions based on a condition.

The general syntax of a conditional expression is as follows:

```
value_if_true if condition else value_if_false
```

1. value_if_true: This is the value or expression that will be returned if the condition evaluates to True.
2. condition: The condition is an expression that evaluates to either True or False. It determines which value (either value_if_true or value_if_false) will be returned.
3. value_if_false: This is the value or expression that will be returned if the condition evaluates to False.

In [13]:
```
x = 5

result = 'Positive' if x > 0 else 'Non-positive'
print(result)
```
```
Positive
```

In this example, the condition x > 0 is evaluated. If the condition is True, the value "Positive" is assigned to the variable result. If the condition is False, the value "Non-positive" is assigned. In this case, since x is 5 (which is greater than 0), the output will be Positive.

Conditional expressions are useful when you want to assign a value based on a condition without the need for an explicit if-else statement. They offer a concise and readable way to express simple conditional logic.

It's important to note that conditional expressions should be used judiciously to maintain code readability. If the logic becomes more complex or involves multiple conditions, it is generally recommended to use traditional if-else statements instead.

In [14]:
```
age = 13
'minor' if age < 21 else 'adult'
```
Out[14]:
```
'minor'
```

In [15]:
```
x = y = 20

z = 2 + x if x > y+1 else y + 3
z
```
Out[15]:
```
23
```

In [16]:
```
x = y = 20

z = 2 + (x if x > y+1 else y) + 3
z
```
Out[16]:
```
25
```