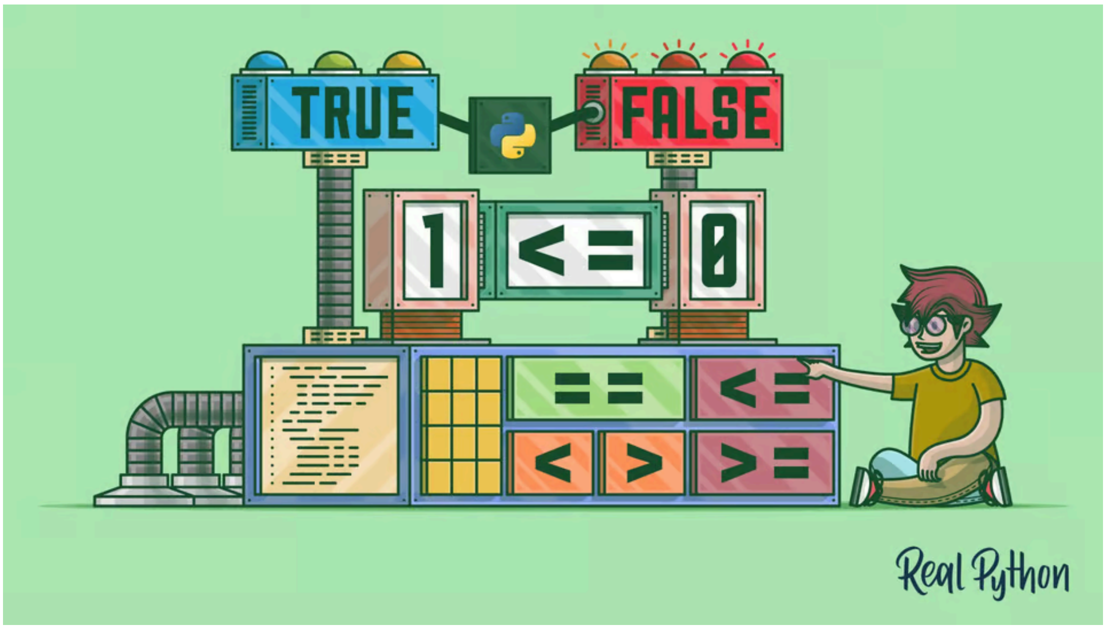
6. Booleans and Comparison Operators

In this section, we'll explore the concept of booleans and how they are used in Python to represent true or false values. We'll also delve into comparison operators, which are used to evaluate conditions and make decisions within your programs.

We'll learn about the following topics:

- 6.1. Booleans
- 6.2. Comparison Operators
- 6.3. Chained Comparison Operators



6.1. Booleans:

Booleans in Python are a fundamental data type that can only hold two values: `True` and `False`. They are often used to represent logical conditions or make decisions within your programs.

Name	Type in Python	Description
Booleans	bool	Logical value indicating True or False

```
In [1]: type(True), type(False)
```

```
Out[1]: (bool, bool)
```

6.2. Comparison Operators:

These operators will allow us to compare variables and output a Boolean value (True or False).

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

- **Equal**

```
In [2]: 5 == 5
```

```
Out[2]: True
```

```
In [3]: 8 == 9
```

```
Out[3]: False
```

Note that `==` is a *comparison* operator, while `=` is an *assignment* operator.

- **Not Equal**

```
In [4]: 5 != 6
```

```
Out[4]: True
```

```
In [5]: 5 != 5
```

```
Out[5]: False
```

These two operations (`==` `!=`) can be done on lists, tuples, dictionaries, and sets.

```
In [6]: {'key1':1, 'key2':2} != {'key2':2, 'key1':1}
```

```
Out[6]: False
```

```
In [7]: [1, 2] == [2, 1]
```

```
Out[7]: False
```

- **Greater Than**

```
In [8]: 6>4
```

```
Out[8]: True
```

```
In [9]: 2>3
```

```
Out[9]: False
```

- **Less Than**

```
In [10]: 2 < 6
```

```
Out[10]: True
```

```
In [11]: 5 < 2
```

```
Out[11]: False
```

- **Greater Than or Equal to**

```
In [12]: 3 >= 3
```

```
Out[12]: True
```

```
In [13]: 1 >= 7
```

```
Out[13]: False
```

- **Less than or Equal to**

```
In [14]: 4 <= 4
```

```
Out[14]: True
```

```
In [15]: 8 <= 4
```

```
Out[15]: False
```

6.3. Chained Comparison Operators:

An interesting feature of Python is the ability to chain multiple comparisons to perform a more complex test.

```
In [16]: 1 < 2 < 3
```

```
Out[16]: True
```

The above statement checks if 1 was less than 2 and if 2 was less than 3. We could have written this using an and statement in Python:

```
In [17]: 1<2 and 2<3
```

```
Out[17]: True
```

```
In [18]: 1 < 3 > 2
```

```
Out[18]: True
```

The above checks if 3 is larger than both of the other numbers, so you could use and to rewrite it as:

```
In [19]: 1<3 and 3>2
```

```
Out[19]: True
```

It's important to note that Python is checking both instances of the comparisons. We can also use **or** to write comparisons in Python. For example:

```
In [20]: 1==2 or 2<3
```

```
Out[20]: True
```

```
In [21]: 1==1 or 100==1
```

```
Out[21]: True
```