BYU IDAHO CSE 110 | Programming Building Blocks

06 Prepare: Preparation Material

Overview

In this lesson, you will continue practicing working with **`if`** statements, using even more complicated programs and logic.

Preparation Material

Watch the following videos:

- » Complex Condition Checks (4 mins)
- » Demo: Complex Conditions (8 mins)

I. TESTING MULTIPLE CONDITIONS

As shown, you can combine two conditions with **`and`** and **`or`**:

```
if x > 5 and y > 10:
    # This happens if both conditions are true

if x > 5 or y > 10:
    # This happens if either one of the conditions (or both!) are true
```

And you can mix and match any expressions that evaluate to true:

```
if x > 5 and y > 10 and word == "taco":
    # This happens if all three conditions are true
```

Just as mathematical operators have precedence rules where **`*`** happens before **`+`**, the **`and`** condition takes precedence over the **`or`**

condition, so if you want to mix them and have the or condition happen first, you need to use parentheses:

```
if (x > 5 or x < -5) and y > 10:
    # In this case, x can either be greater than 5 or less than negative 5, and y must
    # always be greater than 10

if x > 5 or x < -5 and y > 10:
    # Without parentheses, the x < -5 and y > 10 conditions go together and both must b
    # true, unless x > 5, in which case the whole statement is true
```

Sometimes, these can get a little confusing, so it is a good idea to use parentheses in complicated expressions, to make it clear what you intend to happen.

II. TESTING MULTIPLE SIMILAR CONDITIONS

It is common to want to test the same variable for two conditions, or to test two variables against the same condition. In these cases, you may be tempted to leave part out, but this will not work, anytime you use operators like `>` or `==`, you must provide both a left and a right hand side for it.

```
# To check if x is either 5 or 6, you might be tempted to write:
if x == 5 or 6:
    # INCORRECT: This does not work!

# You must write them both out:
if x == 5 or x == 6:
    # This is the correct way to check
```

```
# To check if either first_name or last_name is Scott you might be tempted to write:
if first_name or last_name == "Scott":
    # INCORRECT: This does not work!

# You must write them both out:
if first_name == "Scott" or last_name == "Scott":
    # This is the correct way to check
```

III. TRUE AND FALSE VALUES

In addition to comparing against other variables, you can also set a variable itself to be a **`True`** or **`False`**. To this point, we have learned about the variable types of int, float, and string. Variables that are either true or false are a new data type called Boolean variables (named after a mathematician, George Boole).

You can set a boolean variable directly, such as **`is_hot = True`** or **`is_hot = False`**. Notice that you must always use a capital T on True and a capital F on False.

Because these variables are already either true or false, you don't need to compare them against another value. For example:

```
is_hot = True

# You can check the value of the variable directly
if is_hot:
    print("It's hot")

# It works, but is redundant (and therefore bad practice) to check if it is True:
if is_hot == True:
    print("It's hot")
```

You can check if a boolean variable is not true with the **`not`** keyword:

```
is_hot = True

# Using the "not" keyword
if not is_hot:
    print("It is not hot")

# It works, but is generally avoided to check if it is false:
if is_hot == False:
    print("It is not hot")
```

Boolean variables are typically named with words that indicate that it will only have two values, and help you understand what true means. For example, if you want to have a boolean variable that determines whether someone is an adult, you would typically choose a name of **`is_adult`** rather than **`age`** or even **`adult`**.

Common boolean variable names are **`is_xxx`**, **`has_xxx`**, **`can_xxx`**, etc.