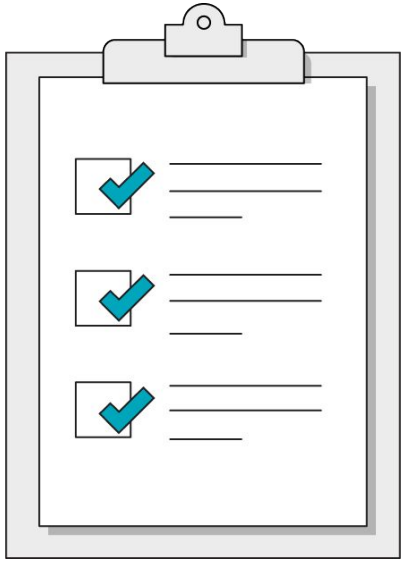
Conditionals

Our Learning Goals

- Define conditional statements in Python to create logic-driven programs.
- Use logical operators to enhance conditional statements.



What We'll Practice Today

This class is a **blended learning experience**. It connects to and reinforces topics that you encountered in the myGA pre-work.

We're going to return to topics covered in the pre-work and build upon them:

- **Comparison and logical operators**
- **Conditional statements**

Solo Exercise:

Jupyter Notebook Review

10 minutes

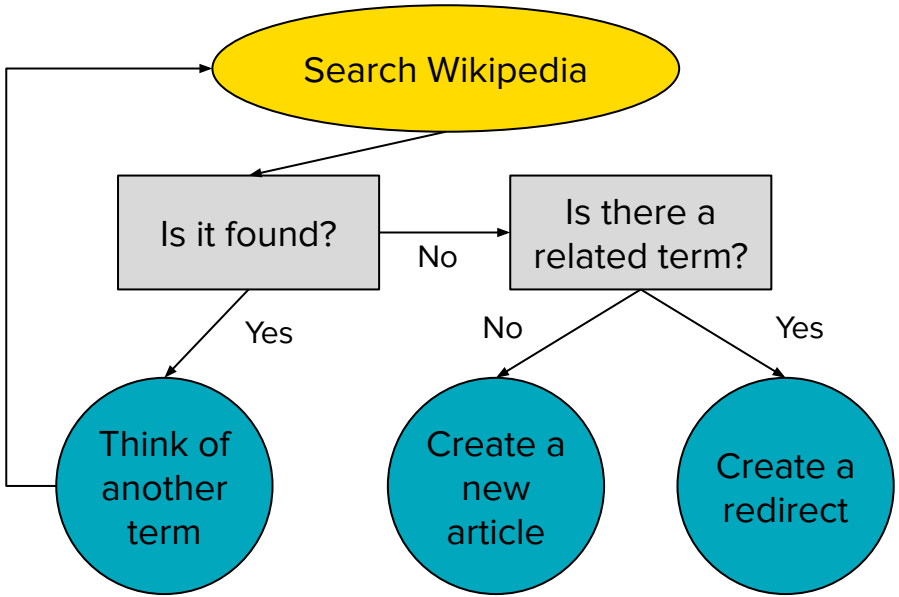
Let's dive right in and use what we learned in the pre-work! We want to understand where you are in your learning journey so that we can give the best possible experience in class.

Look over the exercises in today's Jupyter Notebook and attempt any that seem immediately doable to you.

Then, **rate your confidence level** on today's subjects from 1–5.

Why Use Conditional Statements?

Conditional logic allows you to create **programs that are vastly more complex**. Think of the decision-making process of giant flowcharts...



Conditionals

Comparison Operators

Comparison Operators

First, we'll need **comparison operators** — a set of operators that give you the ability to compare values and return a Boolean result (true or false).

Comparison Operator	Meaning
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equality
!=	Inequality

Conditional Statements

Conditionals use a Boolean value to decide whether or not to do something.

```
if value_one == value_two:
  print("The values are equal!")
  print(value_one)
```

Note the **code block** defined by indenting lines after a colon. This code block executes if the Boolean provided is **true**.

`else` Statements

You will often want to have an **`else`** statement immediately after the **`if`** statement. This will trigger when the **`if`** comparison turns out to be **`false`**.

```
if value_one == value_two:
  print("The values are equal!")
else:
  print("The values are not equal!")
}
```

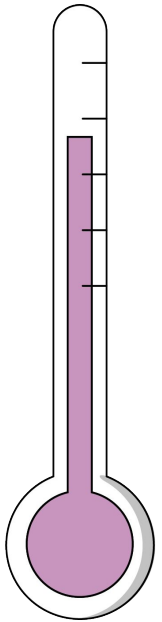
Multiple Conditions

```
if player_one_score == player_two_score:
  print("We have a tie")
elif player_one_score > player_two_score:
  print("Player one is victorious!")
else:
  print("Player two has triumphed!")
```


Discussion:

What Happens in This Code?

```
temperature = 0;
if temperature = 100:
  print("Leave your pets indoors for safety")
else:
  print("The weather is OK")
```



Careful! Equals Aren't All Equal...

When you use "=", that is an **assignment operator**.

If you try to use "= "instead of "==" in a comparison statement, you will get a syntax error! Be sure to use the **equality operator** instead:

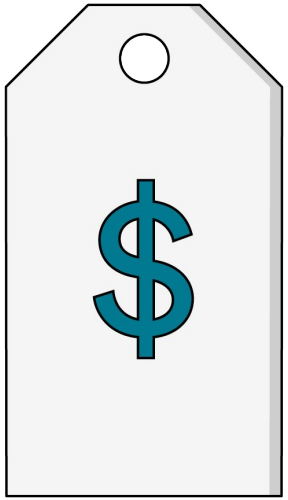
```
temperature = 0;
if temperature == 100:
  print("Leave your pets indoors for safety")
```

Partner Exercise:

3.1 Price Conditions

15 minutes

Now that our programs are able to make some decisions, it's time to test out our conditional logic with some price-based questions in Section 3.1 of the workbook.



Conditionals

Logic Operators

Nesting Conditionals

You can include a conditional **inside of another** if you want to check multiple conditions:

```
product_is_good = True;
bank_account = 500;

if product_is_good == True:
    if bank_account > 100:
        print("Let's buy this product!")
```

Multiple Conditions, One Statement

You can check to see if two conditions are met in one statement with a **logic operator**:

```
product_is_good = True;
bank_account = 500;

if product_is_good == True and bank_account > 100:
  print("Let's buy this product!")
```

Logic Operators

Logic operators allow us to combine multiple conditions together. For very complex conditionals, you can put several conditions in parentheses to evaluate them as a single expression.

Operator	Description
and	Evaluates to **`true`** only if all combined values are true.
or	Evaluates to **`true`** if any of the combined values are true.
not / !	Reverses the Boolean result of whatever follows it.

Pro Tip: Condensing Conditionals

Like we saw in the previous slide, not all conditionals need a comparison statement — especially if the values being tested are already Booleans.

Thus, you will rarely see a comparison with "**`== true`**" or "**`!== false`**".

Instead, you can use the following pattern:

```
if product_is_good and not product_is_too_expensive:
  # do something
```

True or False?

1. `15 > 10 and 25 > 30`
2. `len("apples") > 5 or len("orange") < 2`
3. `( 5 > 10 and 10 > 15 ) or "orange" == "orange"`
4. `"bananas" not in ["oranges", "apples"] and len("bananas") < 10`
5. `2 + 2 != 4 and 2 + 2 == 5`

Making Conditionals Easier

- `print()` is your friend.
  - Leave yourself messages to figure out what path was taken by a conditional.
  - If a conditional gives you mysterious results, log the values and data types.
  - You can also print the Boolean result of a comparison as a sanity check.
- Avoid chains that are far too long for one line.
  - You can nest conditionals within other conditions for complex logic.

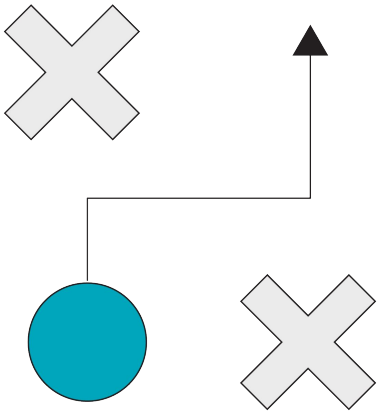
Solo Exercise:

3.2 Complex Conditions

60 minutes

Some complex conditional challenges await you in Section 3.2 of the Jupyter Notebook.

You will need to use our new friends, *logical operators*, to complete these challenges.



Conditionals

Wrapping Up

Recap

In today's class, we...

- Defined conditional statements in Python to create logic-driven programs.
- Used logical operators to enhance conditional statements.

Looking Ahead

On your own:

- Ensure that you've completed the Python pre-work and pre-work quiz.

Next Class:

Loops

Don’t Forget: Exit Tickets!


GA