General Assembly

# Intro to Python

GENERAL ASSEMBLY

# Our Goals for Today

- Write Python scripts that use conditionals and loops.
- Begin building Python skills through code challenges.

Intro to Python

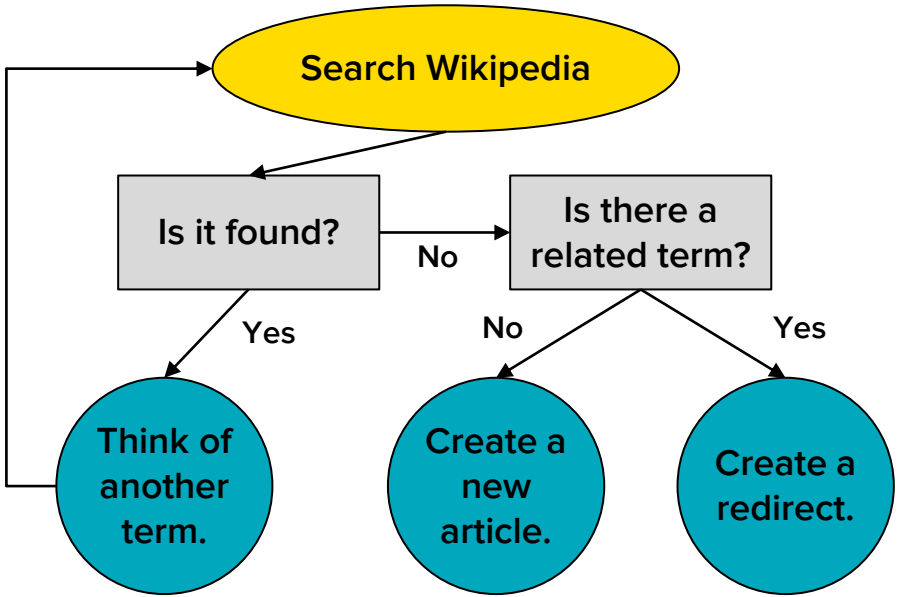
# Control Flow

Control Flow

# Conditionals

# Why use conditional statements?

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



# 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 determine 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

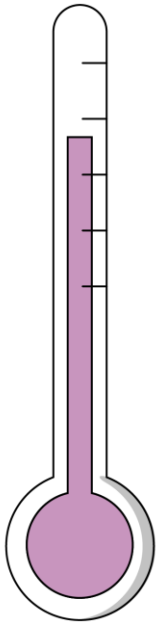
```
if player_one_score == player_two_score:
  print("We have a tie")
elif player_one_score > player_two_score:
  print("Player one is victorious!")
else:
  print("Player two has triumphed!")
```

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:

# 11. 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 11 of the workbook.

Control Flow

# Logic Operators

# Nesting Conditionals

You can include a conditional **inside of another** conditional 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?

Trivia!

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`

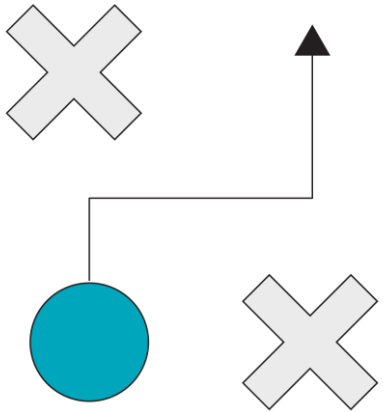
Solo Exercise:

# 12. Complex Conditions

20 Minutes

Some complex conditional challenges await you in Section 12 of the Jupyter Notebook.

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





Control Flow

# Loops

Discussion:

# Life Without Loops

Imagine that we have a list of colors, and we want to print each one:

```
colors = ["red", "orange", "yellow"]

print(colors[0])

print(colors[1])

print(colors[2])
```

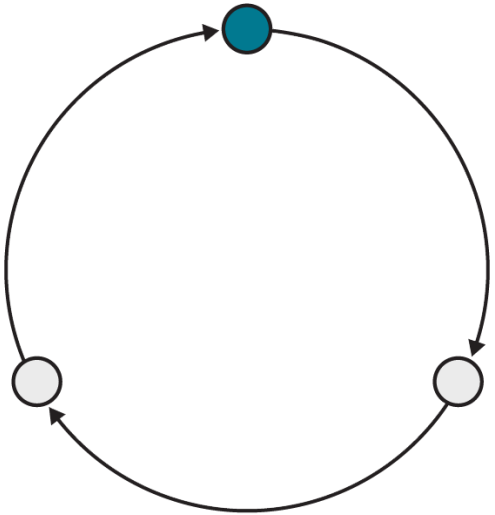
**Does this seem sustainable?**

**Loop:** A control flow statement allowing for the repeated execution of a code block until a specific condition is reached.

# Why Loops Matter

- **Loops** take advantage of what computers do best — evaluate instructions across organized sets of data very quickly.
- Computers excel when working in isolated patterns, which is exactly how a loop works.
- You can avoid needlessly copying or re-typing code by repeating it in a loop.



Control Flow

# `while` Loops

# `while` Loop

A `while` loop is very straightforward. It continues executing while the condition you've given it is still true. Once the condition is false, it stops!

```
number = 0
while number < 10:
    number = number + 1
    print(number)
```

Discussion:

# What will this code do?

```
number = 1
while number > 0:
    number = number + 1
    print(number)
```

# Caution! Infinite loops ahead.

Be careful with loops! If the condition is never broken, you'll have an unending loop that will devour your computer's processing power.

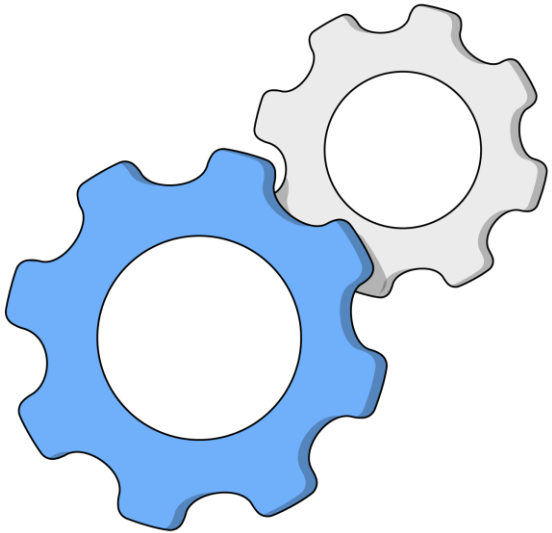
```
number = 1
while number > 0:
    number = number + 1
    print(number)
```

Solo Exercise:

# 13. While Loops

15 Minutes

Use `while` loops to complete the challenges in Section 13 of the Jupyter Notebook.



Control Flow

# For Loops

# For Loop

A for loop is geared more toward iterating over lists or collections of data:

1. Define a variable to act as our iterator.
2. Provide the iterable value, typically a list.

```
for number in [0,1,2,3,4,5]:
  print(number)
# outputs 0,1,2,3,4,5
```

# The Naming of Things

Remember, clear naming and syntax are one of Python's great advantages. Don't spoil it by naming your iterator something vague — be descriptive!

```
# x is not descriptive!
colors = ["red","yellow"]
for x in colors:
        print(x)
```

```
# color is more accurate
colors = ["red","yellow"]
for color in colors:
        print(color)
```

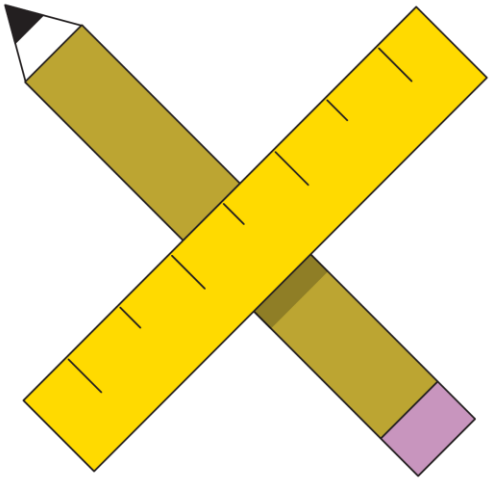
 

Solo Exercise:

# 14. For Loops

10 Minutes

Let's practice some **for loops** by looping over a list of numbers in Section 14 of the workbook.



# Using the range() Function

Because for loops require an iterable, it is common to use the range() function to automatically generate a list of numbers over which to iterate.

```
for number in range(6):
  print(number)
# outputs 0,1,2,3,4,5
# why isn't 6 included in the output?
```

# The Many Faces of `range()`

| | Explanation | Example |
|---|---|---|
| **One parameter** | A list from zero up to, but not including, the given number. | `range(5)`<br>`[0,1,2,3,4]` |
| **Two parameters** | A list including the first number, up to, but not including, the second number. | `range(5, 10)`<br>`[5,6,7,8,9]` |
| **Three parameters** | A list from the first number up to, but not including, the second number, increasing (or "striding") by the third number. | `range(1,10,2)`<br>`[1,3,5,7,9]` |

# Looping With Indices

If you want to actually modify a list while looping, you must use indices:

```
numbers = [1,2,3,4]

for i in range( len(numbers) ):

        numbers[i] = numbers[i] * 2
```

# Looping Over Dictionaries

Dictionaries have an `.items()` method to help loop through keys and values.

```
author = { "first_name": "Kazuo", "last_name": "Ishiguro" }

for key, value in author.items():

        print(value)
```

# For vs. While: Which Loop to Use

## While Loops

Great for instances when you have no way of knowing how many times you need to iterate.

Useful for things like random chance or eliminating specific items of an undetermined amount.

vs.

## For Loops

The most common loops, used when you know exactly how many times you need to iterate.

If you have a list and need to look through every item, use a for loop.

# For or While?

Trivia!

1. Look through a list to find the largest number.
2. Flip a coin until heads shows up three times in a row.
3. Print every value in a list.
4. Animate a game screen until the player's health is zero.
5. Check the weather every five seconds until it rains.

Group Exercise:

# 15. Hold on to Your Loops

30 Minutes

Things can start getting complicated now that we have loops, conditionals, *and* data structures!

The challenges in Section 15 will be tough, so be sure to lean on print statements to help navigate your way through these tasks one step at a time.

# Questions?


GA