Loops

Today’s Learning Objectives

In this lesson, you will:

- Use loops to iterate over code blocks repeatedly.
- Differentiate between while and for loops.
- Use list comprehension to generate lists using loop syntax.

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:

- **For loops**
- **While loops**

Solo Exercise:

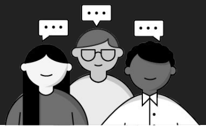
Jupyter Notebook Review

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.



Discussion:

Life Without Loops

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

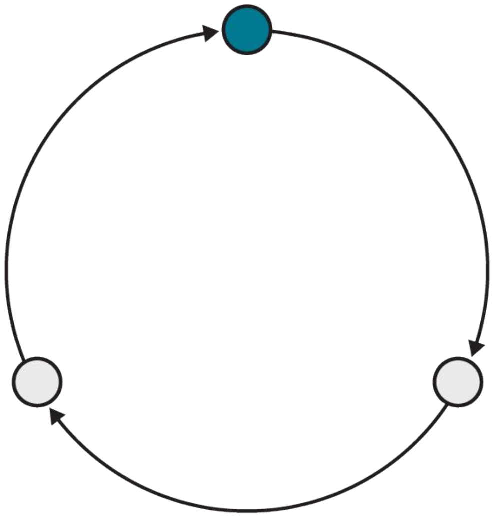
```
colors = ["red", "orange", "yellow"]

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?

- **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.



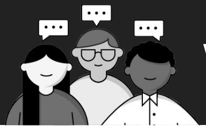
Loops

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

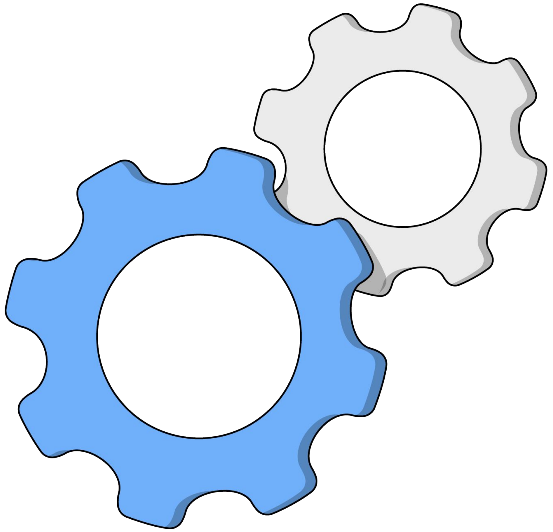
```
number = 1
while number > 0:
    number = number + 1
    print(number)
```

Solo Exercise:

4.1 While Loops

20 minutes

Use while loops to complete the challenges in Section 4.1 of the Jupyter Notebook.



Loops

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)
```

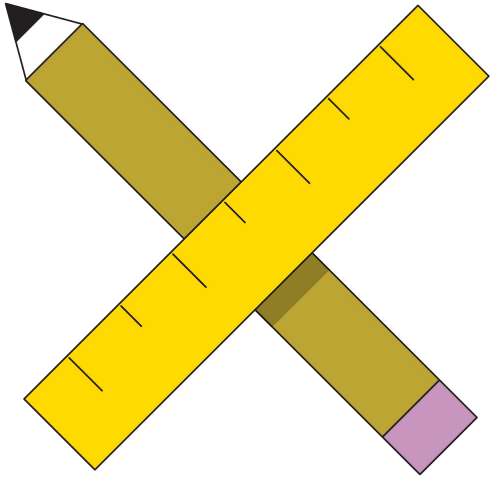
```
# color is more accurate

colors = ["red","yellow"]

for color in colors:

    print(color)
```

Solo Exercise:

4.2 For Loops

15 minutes

Let's practice some for loops by looping over a list of numbers in Section 4.2 of the workbook.



Using the range() Function

Because for loops require an iterable, it's 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)` `[0,1,2,3,4]`
Two parameters	A list including the first number, up to, but not including, the second number.	`range(5, 10)` `[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)` `[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]

list_length = len(numbers)

for i in range( len(list_name) ):

    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)
```

Nested Loops

Just as it’s possible to nest data structures inside of each other, it is also possible to put loops inside of other loops.

```
for num_one in range(1,13):
    for num_two in range(1,13):
        print(f"{num_one} times {num_two} = {num_one * num_two}")
```

For vs. While: Which One to Use?

While Loops		For 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.	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?

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.

List Comprehension

Here’s a neat trick for creating new lists using a loop-like syntax:

```
numbers = [1,2,3,4,5]

even_numbers = [ num * 2 for num in numbers ]

# even_numbers is [2,4,6,8,10]
```

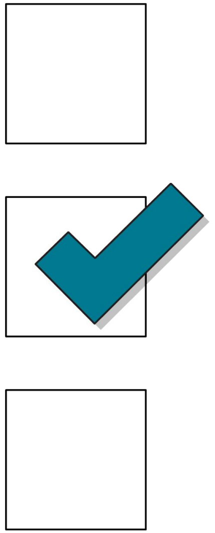
Solo Exercise:

4.3 List Comprehension

20 minutes

Shortcuts like list comprehension are what make experienced programmers fall in love with Python.

Let's try out some challenges using list comprehension instead of loops in Section 4.3 of the workbook.



Group Exercise:

4.4 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 4.4 will be tough, so be sure to lean on print statements to help navigate your way through these tasks one step at a time.

Loops

Wrapping Up

Recap

In today's class, we...

- Used loops to iterate over code blocks repeatedly.
- Differentiated between while and for loops.
- Used list comprehension to generate lists using loop syntax.

Looking Ahead

On your own:

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

Next Class:

Functions

Don’t Forget: Exit Tickets!