

for loops.

Week 3 | Lecture 2 (3.2.1)

While waiting for class to start:

Download and open the Jupyter Notebook (.ipynb) for Lecture 3.2.1

You may also use this lecture's JupyterHub link instead (although opening it locally is encouraged).

Upcoming:

- Reflection 3 released Friday @ 11 AM
- Lab 3 deadline this Friday @ 11 PM
- PRA (Lab) on Friday @ 2PM this week (ONLINE)
- **Midterm** - May 31 in lecture @ 9:10 AM

if nothing else, write `#cleancode`

Today's Content

- Lecture 3.2.1
 - **for** loops
- Lecture 3.2.2
 - **for** loops on indices, nested loops

Mentimeter Checkpoint

- Join at www.menti.com:
 - Code: **9493 9270**
- Link
 - <https://www.menti.com/bldrkshx7mcz>



Looping (Iterating)

- Looping means repeating something over and over until a particular condition is satisfied.

Email ←

**Send
Promotional
Email**

Looping



**List of
Customers**

Looping (Iterating)

- Looping means repeating something over and over until a particular condition is satisfied.

Yes/No



Does the
Tweet
contain
#cleancode

Looping

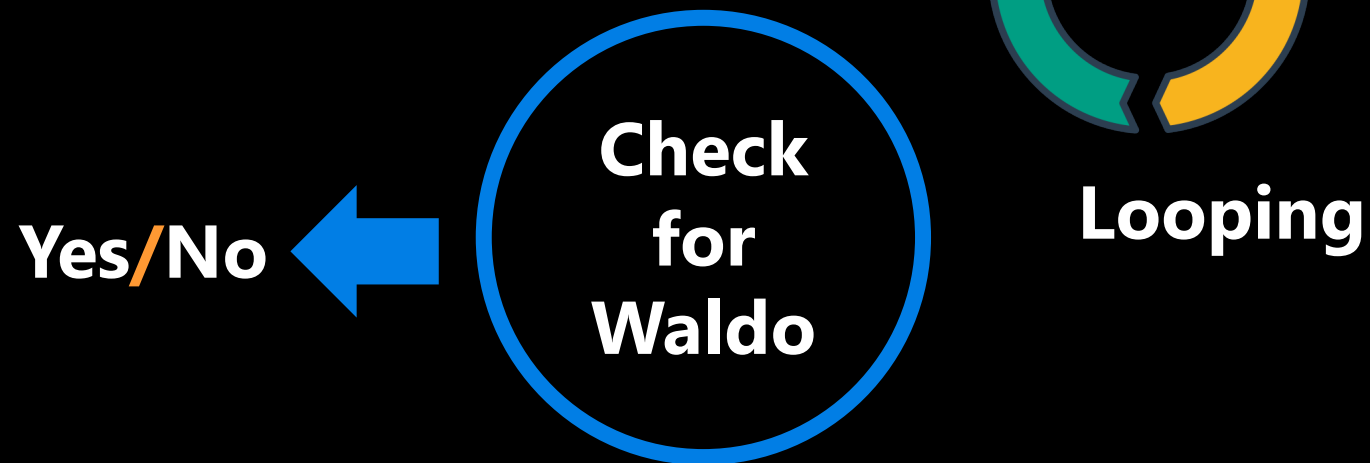


List of
Tweets



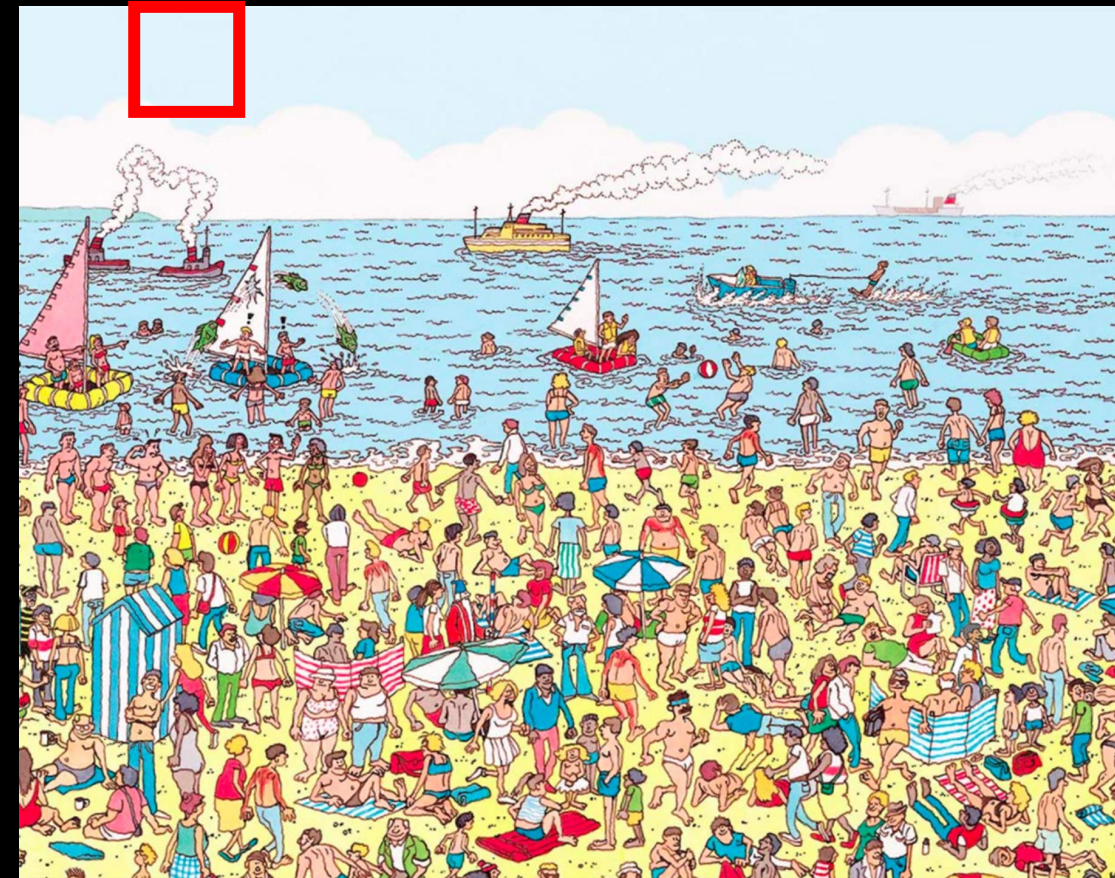
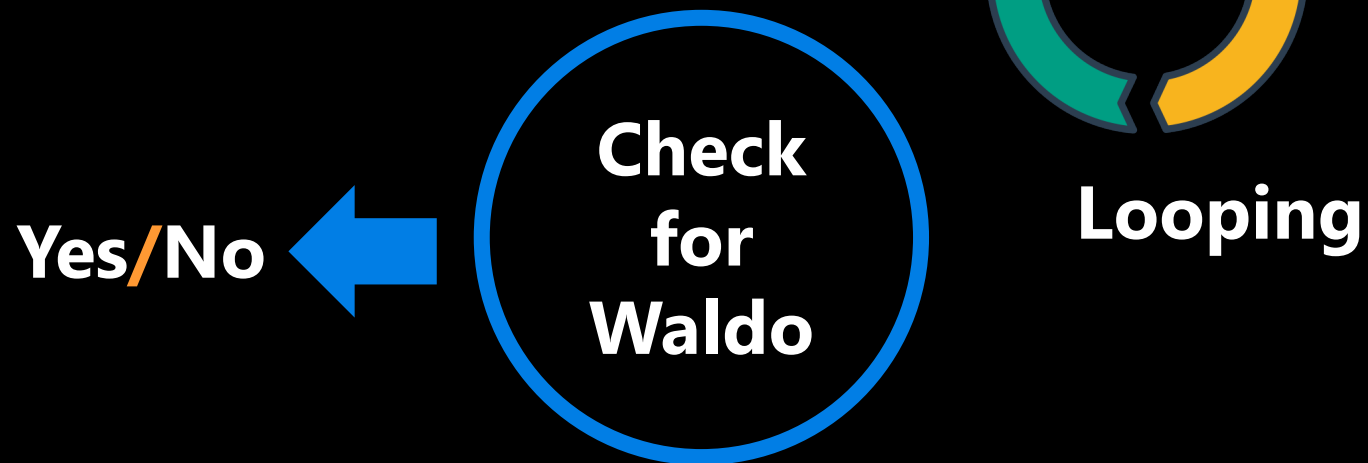
Looping (Iterating)

- Looping means repeating something over and over until a particular condition is satisfied.



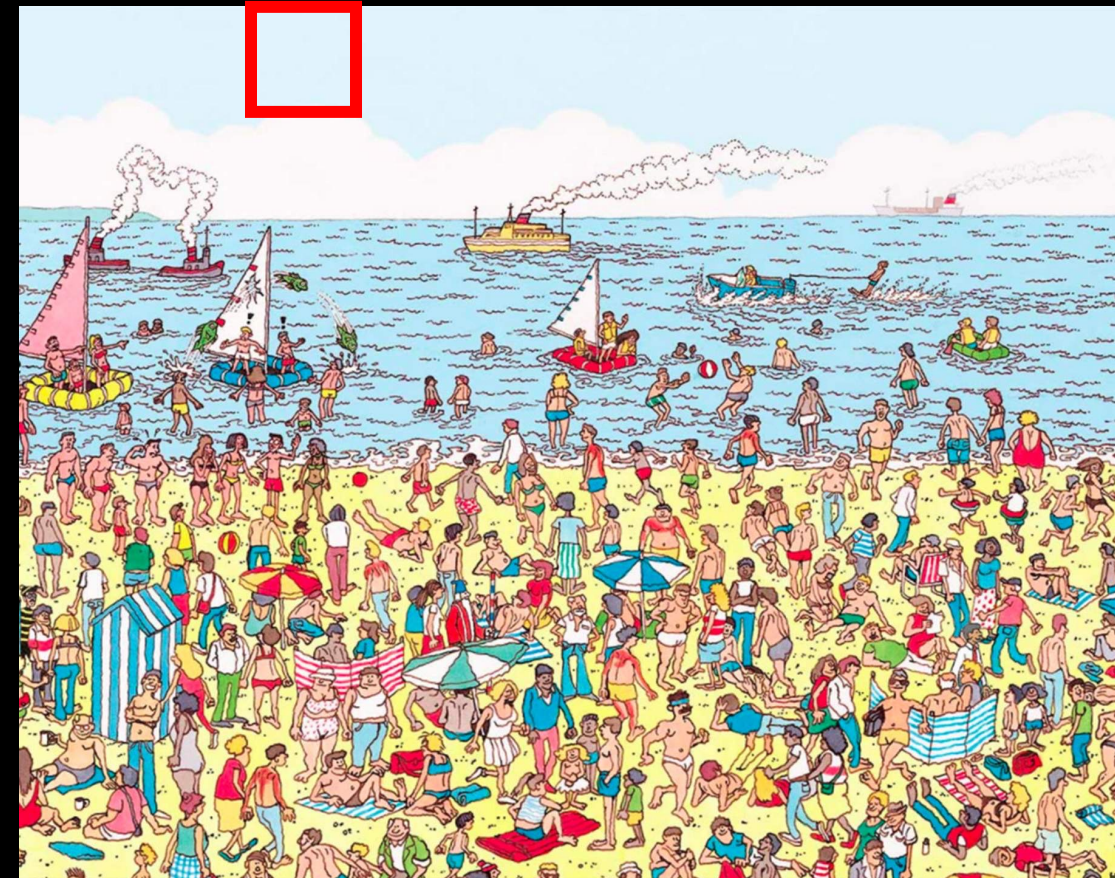
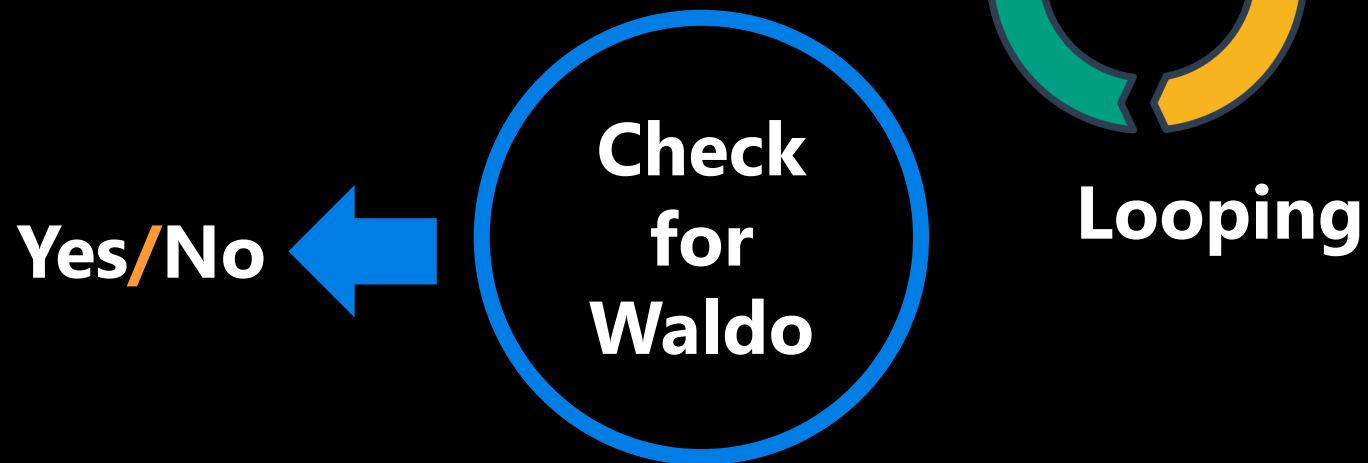
Looping (Iterating)

- Looping means repeating something over and over until a particular condition is satisfied.



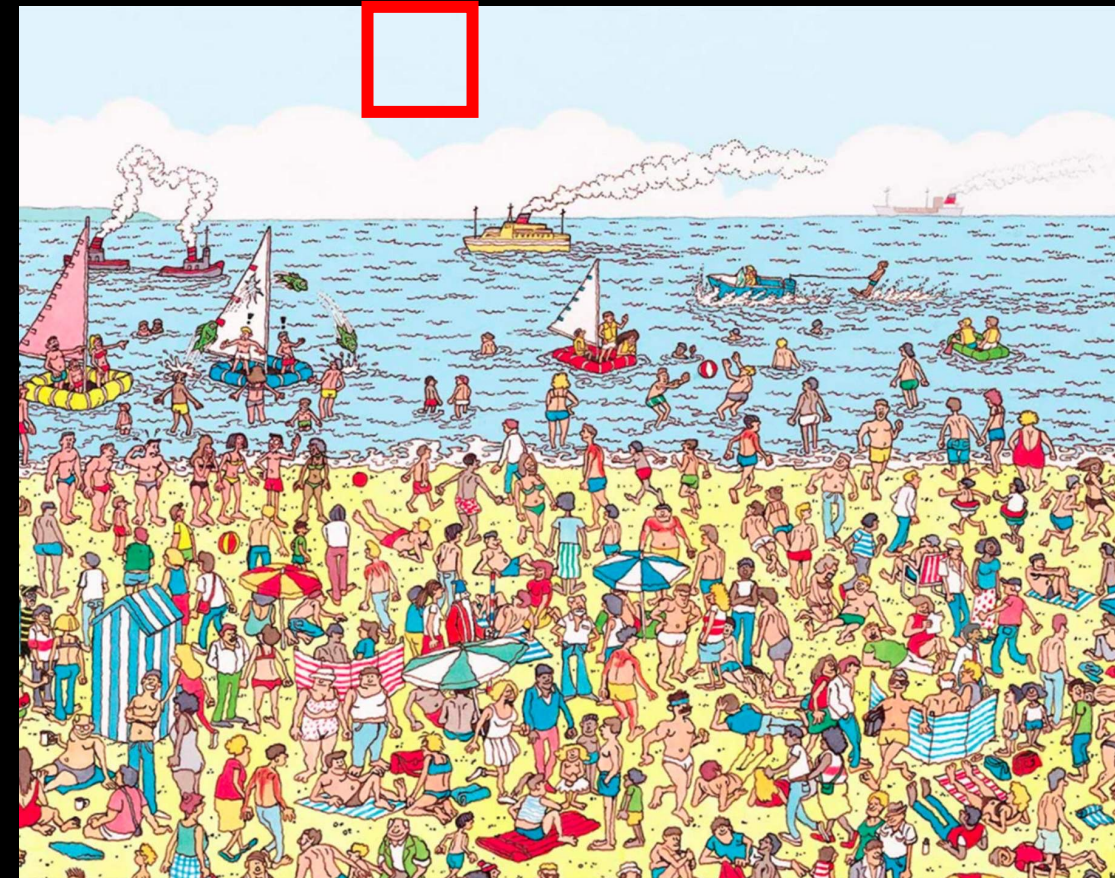
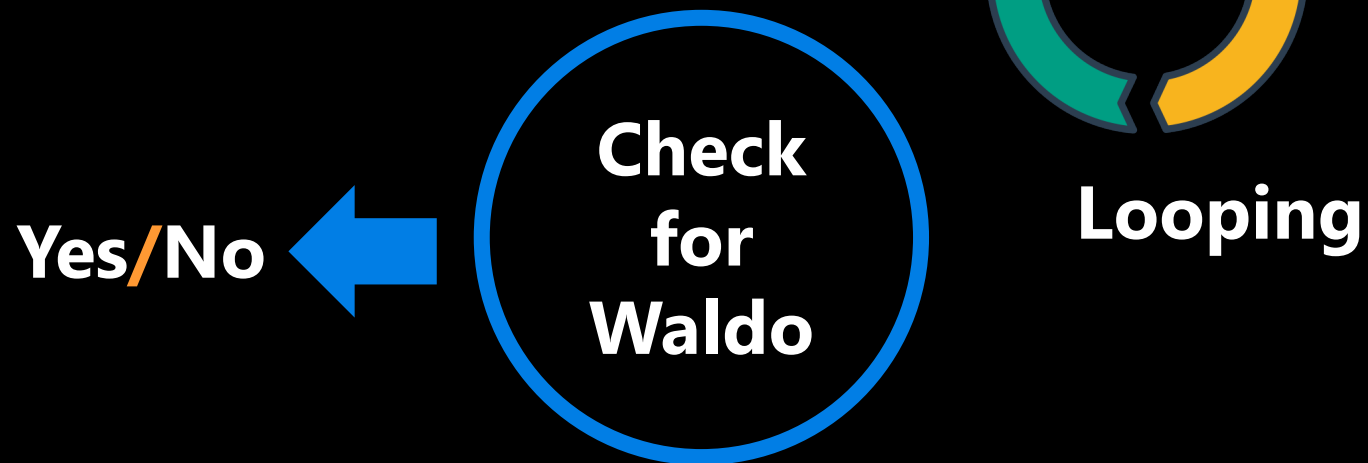
Looping (Iterating)

- Looping means repeating something over and over until a particular condition is satisfied.



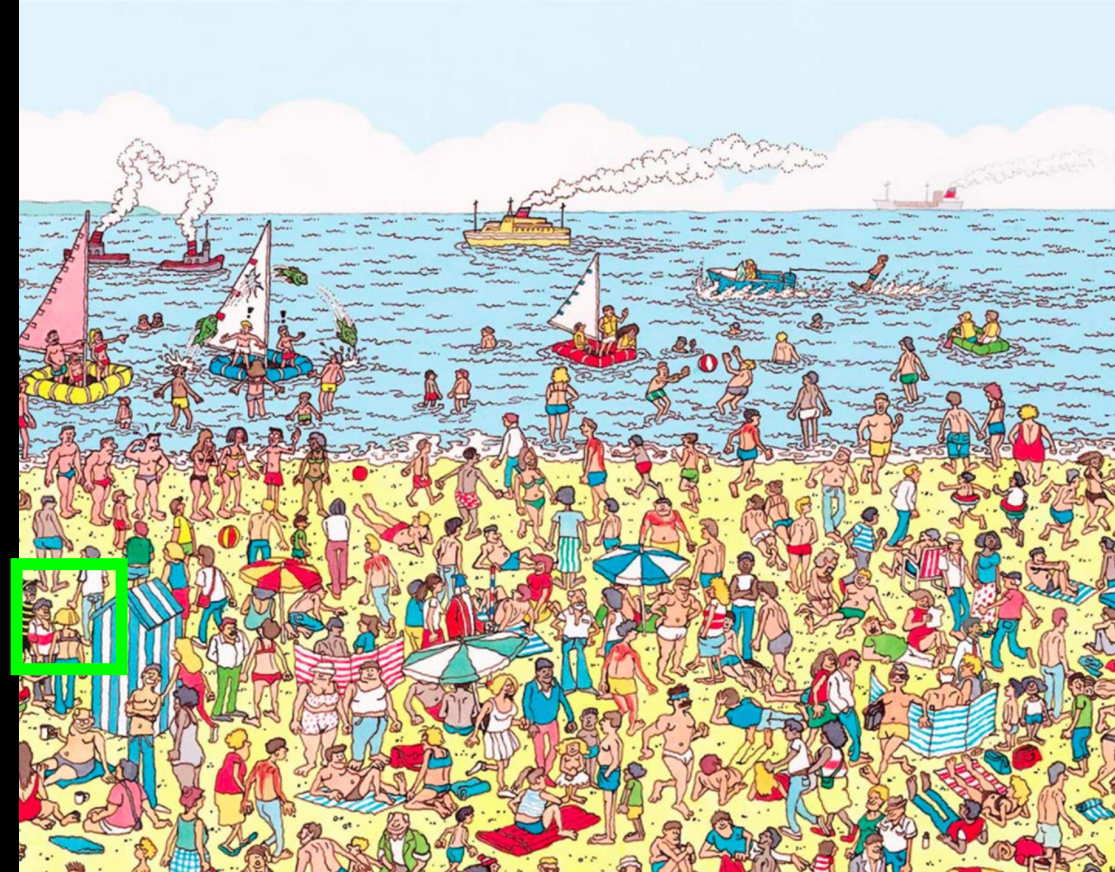
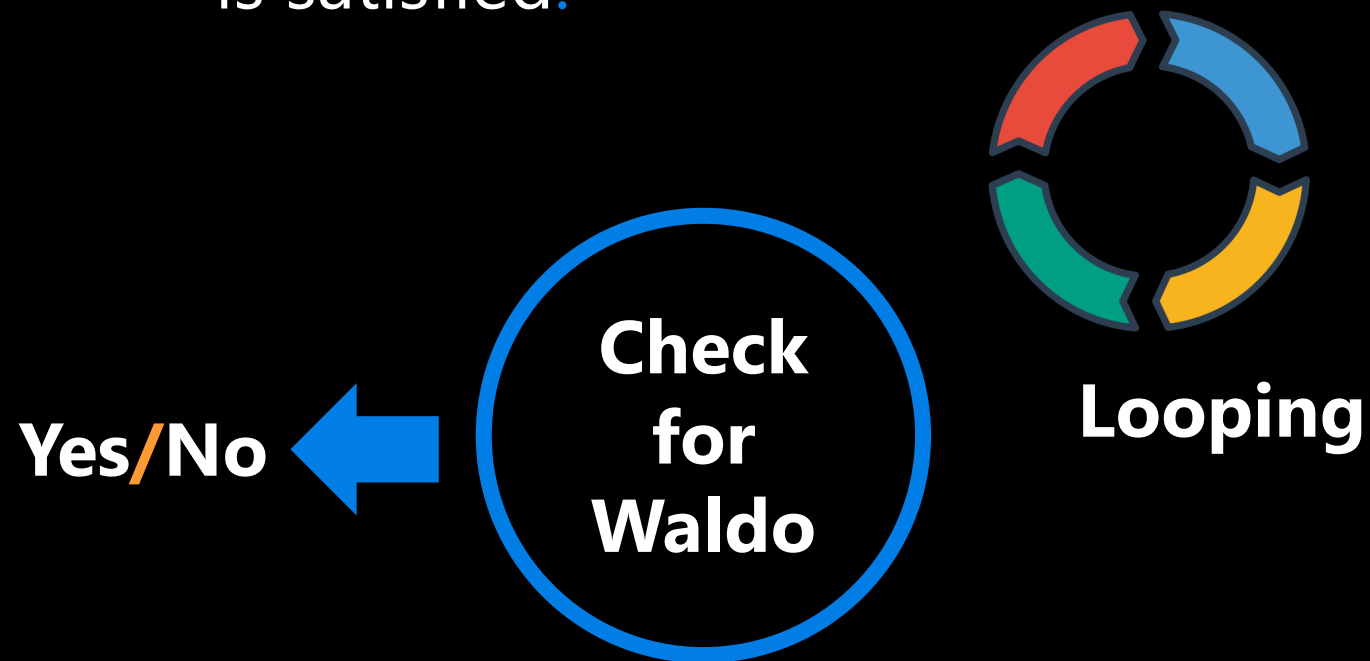
Looping (Iterating)

- Looping means repeating something over and over until a particular condition is satisfied.



Looping (Iterating)

- Looping means repeating something over and over until a particular condition is satisfied.



for loops

- There are several ways to repeat a block of code.
- We've already seen **while** loops and this week, we'll discuss **for** loops.
- Do Something = block of code we want to execute.

```
while expression:  
    do something.
```

```
for item in iterable:  
    do something.
```

for loops

- A **for** loop starts with the keyword **for**.

```
name = 'Sebastian'
```

```
for character in name:  
    print(character)
```


for loops

- Next, we provide the name of one of more variables.
- We have called the variable `character`, but you can call it whatever you like as long as it follows rules for naming a variable.

```
name = 'Sebastian'
```

```
for character in name:  
    print(character)
```

```
for item1, item2 in iterable:  
    do something.
```

for loops

- Our variable `character` will be bound to each of the items in the sequence in turn.

```
name = 'Sebastian'
```

```
for character in name:  
    print(character)
```

for loops

- Specify what the values are in.
- What is the iterable?
- An iterable is an object that can be iterated over.
- Strings are iterable (we know these from last week).
- Lists (**next week**) are iterable.

```
name = 'Sebastian'
```

```
for character in name:  
    print(character)
```

for loops

- As with the `while` loop, the `for` loop statement ends with a colon.
- This is how Python knows you are going to create a new block of code.

```
name = 'Sebastian'
```

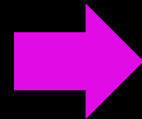
```
for character in name:  
    print(character)
```


for loops

- Indenting four spaces tells Python what lines of code are in that block you want to repeated.

```
name = 'Sebastian'
```

```
for character in name:  
    print(character)
```



Indent

for loops

- What output should we get?

```
name = 'Sebastian'
```

```
for character in name:  
    print(character)
```

for loops

- What output should we get?

```
name = 'Sebastian'
```

```
for character in name:  
    print(character)
```

Output:

```
S  
e  
b  
a  
s  
t  
i  
a  
n
```

for loops

- What output should we get?

```
name = 'Sebastian'
```

```
for character in name:  
    print(character)
```

```
print(character)
```


for loops

- What output should we get?

```
name = 'Sebastian'
```

```
for character in name:  
    print(character)
```

```
print(character)
```

character holds
the value of the last
item in the iterable.



Output:

S
e
b
a
s
t
i
a
n
n

for loops

- Let's try it ourselves.

**Open your
notebook**

Click Link:

1. Your first for loop

for vs while

- You've learned about **for** loops and **while** loops, but when should you use them?
- Firstly, all **for** loops can be written as **while** loops, and vice-versa.
- You should use a **for** loop when you know how many times the loop should run.
- If you want the loop to break based on a condition (do this until...) you should use a **while** loop.

for vs while

- Problem: You have had your **DNA** sequenced and each of your chromosomes is represented by a string of nucleotides: adenine (A), thymine (T), guanine (G), and cytosine (C).
- **chrome_4 = ATGGGCAA**
- Create a function to count the number of occurrences of a nucleotides.

```
my_func(chrome_4, 'A')  
>>> 3
```

**Open your
notebook**

Click Link:
2. while vs for

while

```
i = 0
counter = 0
while i < len(chrome_4):
    if chrome_4[i] == 'A':
        counter += 1
    i += 1
```

for

```
counter = 0
for character in chrome_4:
    if character == 'A':
        counter += 1
```

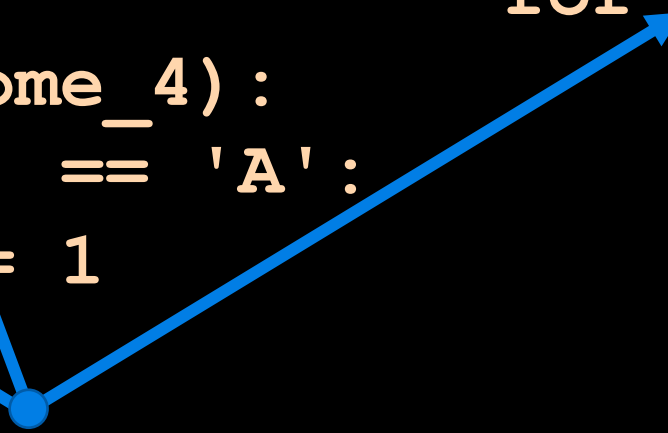
- Differences

while

```
i = 0
counter = 0
while i < len(chrome_4):
    if chrome_4[i] == 'A':
        counter += 1
    i += 1
```

for

```
counter = 0
for character in chrome_4:
    if character == 'A':
        counter += 1
```

- 
- Differences
 - In the **while** loop, the loop variable (**i**) was the index of each character, while in the for loop the loop variable (**character**) is the value of each character.
 - No indexing [**i**] required in the for loop.

while

```
i = 0
counter = 0
while i < len(chrome_4):
    if chrome_4[i] == 'A':
        counter += 1
    i += 1
```

for

```
counter = 0
for character in chrome_4:
    if character == 'A':
        counter += 1
```



■ Differences

- We do not have to worry about how long the string is (e.g., use `len()`) because the **for** loop will go through every character of the string exactly once.

while

```
i = 0
counter = 0
while i < len(chrome_4):
    if chrome_4[i] == 'A':
        counter += 1
    i += 1
```

for

```
counter = 0
for character in chrome_4:
    if character == 'A':
        counter += 1
```



■ Differences

- We do not have to worry about incrementing the loop variable (`i += 1`) as the `for` loop takes care of this.

while

```
i = 0
counter = 0
while i < len(chrome_4):
    if chrome_4[i] == 'A':
        counter += 1
    i += 1
```

6 lines

for

```
counter = 0
for character in chrome_4:
    if character == 'A':
        counter += 1
```

4 lines

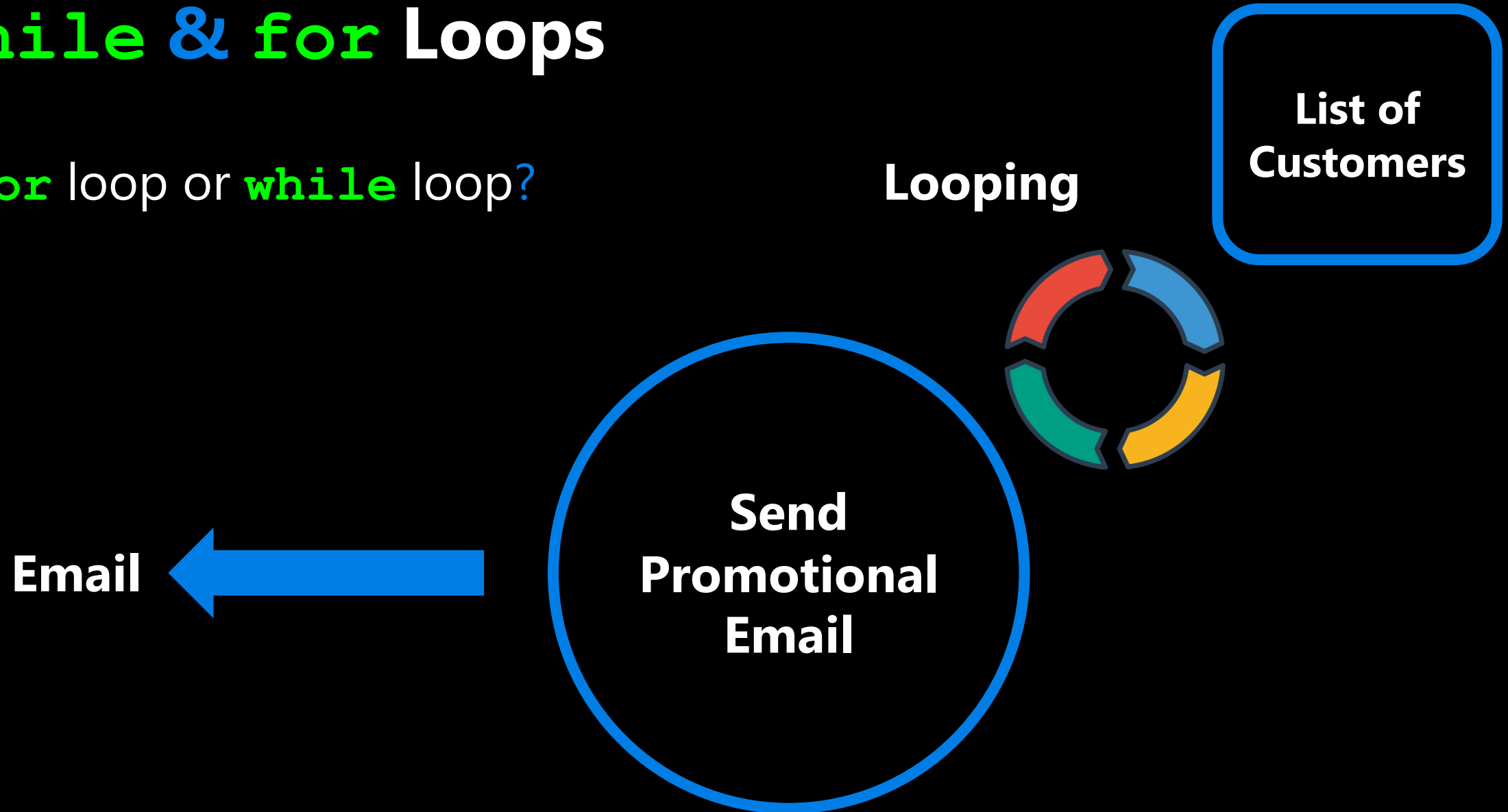
- Differences
- The for loop is **MUCH** easier to read and therefore, desirable when writing code for large collaborative projects.
- #cleancode

for vs while

- You should use a **for** loop when you know how many times the loop should run.
- If you want the loop to break based on a condition (do this until....) you should use a **while** loop.

while & for Loops

- **for** loop or **while** loop?



while & for Loops

- **for** loop or **while** loop?

Looping

List of
Tweets



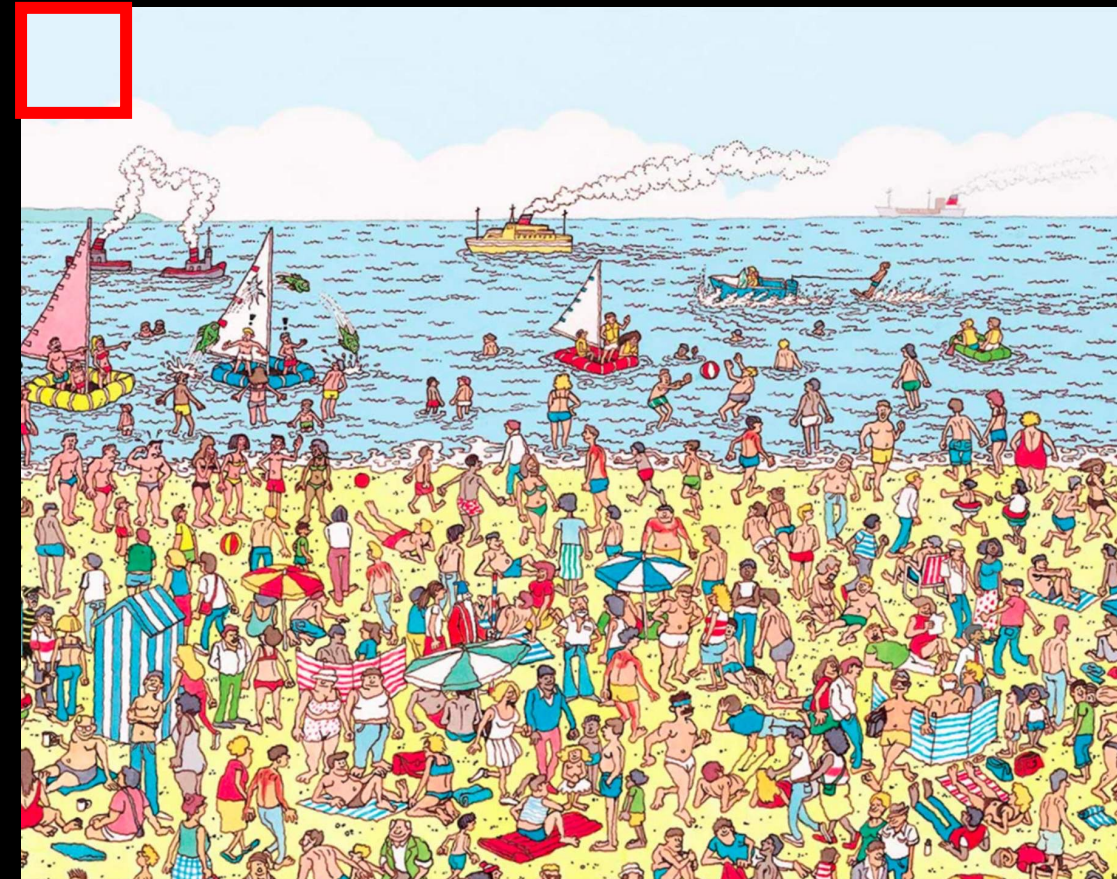
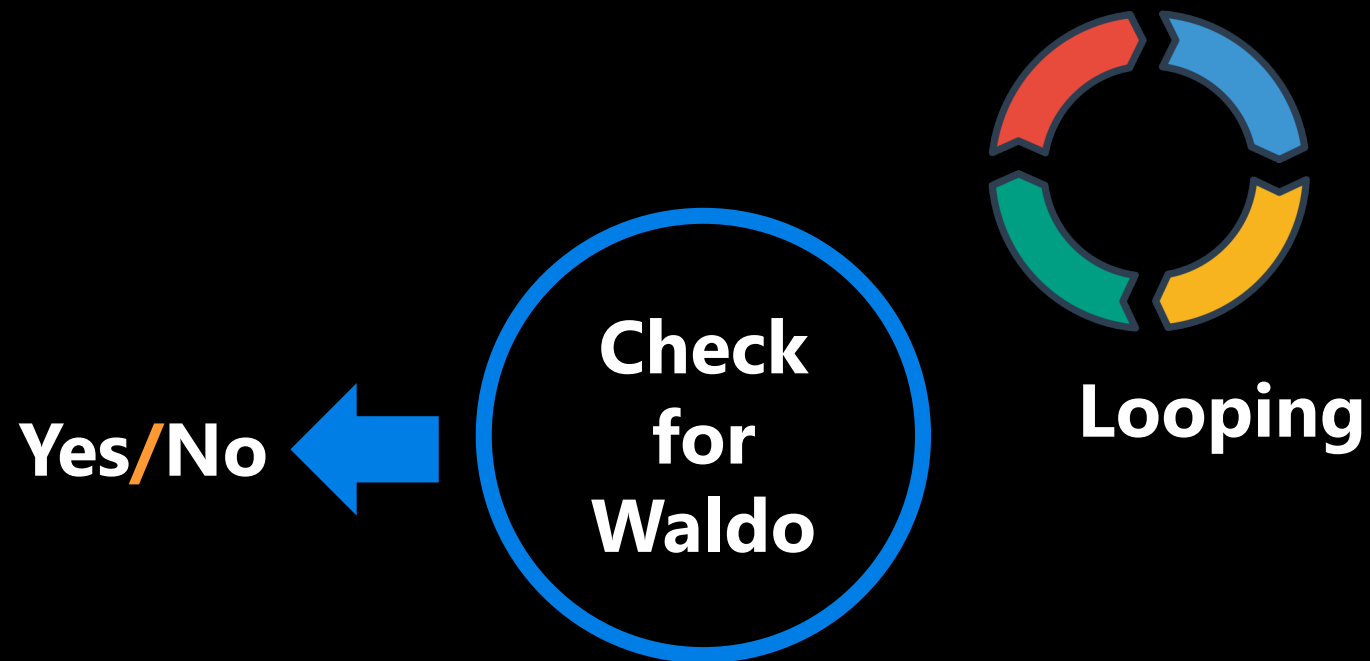
Yes/No



Does the
Tweet
contain
#cleancode

while & for Loops

- for loop or while loop?



Breakout Session 1

Vowels: a, e, i, o, u.

- Write a function that takes in a string and returns the number of vowels in the string. (use **for** loop)
- Test 1
 - `count_vowels('Happy Anniversary!')`
 - 5
- Test 2
 - `count_vowels('xyz')`
 - 0

Open your notebook

Click Link:

3. Breakout Session 1

Breakout Session 2

- Write a function to return the unique separators in a string of integer codes. (use **for** loop)
- The string only contains integers and separators.
- Test 1
 - `find_seperators('23,613-23;2:45')`
 - `',';-;:'`
- Test 2
 - `find_seperators('613-555-3224')`
 - `'_'`

**Open your
notebook**

Click Link:

4. Breakout Session 2

Lecture Recap

Practice!

- **for** loops.
- Looping over strings.
- When to use a **for** loop over a **while** loop.

for loops.

Week 3 | Lecture 2 (3.2.1)

if nothing else, write `#cleancode`