Jonny Gios - Unsplash

Loops and lists

Motivation - What does this code do?

```
x = 5
if x > 0:
   print(x)
   x = x - 1
print(“Done!”)
```

Motivation - What does this code do?

```
x = 5
if x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

Do we enter this if statement?

Motivation - What does this code do?

```
x = 5
if x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

Do we enter this if statement? Yep!

Motivation - What does this code do?

```
x = 5
if x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

Do we enter this if statement? Yep!

Output:

```
10
```

Motivation - What does this code do?

```
x = 5
if x > 0:
    print(x)
    x = x - 1
print("Done!")
```

Do we enter this if statement? Yep!

Output:

```
10
Done!
```

Motivation - What does this code do?

```
x = 5
if x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

Do we enter this if statement? Yep!

Output:

```
10
Done!
```

In the end, the value of x is 4

Making a *small change*

```
x = 5
if x > 0:
   print(x)
   x = x - 1
print(“Done!”)
```

Making a *small change*

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

Making a *small change*

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

if statements mean “if this condition is met, run the code inside one time”

Making a *small change*

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

if statements mean “if this condition is met, run the code inside one time”

While loops mean “keep running the code inside as long as the condition is still True”

Making a *small change*

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

if statements mean “if this condition is met, run the code inside one time”

While loops mean “keep running the code inside as long as the condition is still True”

Check the condition of the loop (here x > 0)
Execute the code inside
Go back to the top and check the condition again

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0?

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0? Yep!

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0? Yep!

Print 5, decrement x (x is now 4)

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0? Yep!
Print 5, decrement x (x is now 4)

Second time through the loop, is x > 0?

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0? Yep!
Print 5, decrement x (x is now 4)

Second time through the loop, is x > 0? Yep!

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0? Yep!

Print 5, decrement x (x is now 4)

Second time through the loop, is x > 0? Yep!

Print 4, decrement x (x is now 3)

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0? Yep!

Print 5, decrement x (x is now 4)

Second time through the loop, is x > 0? Yep!

Print 4, decrement x (x is now 3)

Third time through the loop, is x > 0?

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0? Yep!
Print 5, decrement x (x is now 4)

Second time through the loop, is x > 0? Yep!
Print 4, decrement x (x is now 3)

Third time through the loop, is x > 0? Yep!
Print 3, decrement x (x is now 2)

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0? Yep!
Print 5, decrement x (x is now 4)

Second time through the loop, is x > 0? Yep!
Print 4, decrement x (x is now 3)

Third time through the loop, is x > 0? Yep!
Print 3, decrement x (x is now 2)

Fourth time through the loop, is x > 0?

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0? Yep!
Print 5, decrement x (x is now 4)

Second time through the loop, is x > 0? Yep!
Print 4, decrement x (x is now 3)

Third time through the loop, is x > 0? Yep!
Print 3, decrement x (x is now 2)

Fourth time through the loop, is x > 0? Yep!
Print 2, decrement x (x is now 1)

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0? Yep!
Print 5, decrement x (x is now 4)

Second time through the loop, is x > 0? Yep!
Print 4, decrement x (x is now 3)

Third time through the loop, is x > 0? Yep!
Print 3, decrement x (x is now 2)

Fourth time through the loop, is x > 0? Yep!
Print 2, decrement x (x is now 1)

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0? Yep!

Print 5, decrement x (x is now 4)

Second time through the loop, is x > 0? Yep!

Print 4, decrement x (x is now 3)

Third time through the loop, is x > 0? Yep!

Print 3, decrement x (x is now 2)

Fourth time through the loop, is x > 0? Yep!

Print 2, decrement x (x is now 1)

Fifth time through the loop, is x > 0?

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0? Yep!

Print 5, decrement x (x is now 4)

Second time through the loop, is x > 0? Yep!

Print 4, decrement x (x is now 3)

Third time through the loop, is x > 0? Yep!

Print 3, decrement x (x is now 2)

Fourth time through the loop, is x > 0? Yep!

Print 2, decrement x (x is now 1)

Fifth time through the loop, is x > 0? Yep!

Print 1, decrement x (x is now 0)

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0? Yep!
Print 5, decrement x (x is now 4)

Second time through the loop, is x > 0? Yep!
Print 4, decrement x (x is now 3)

Third time through the loop, is x > 0? Yep!
Print 3, decrement x (x is now 2)

Fourth time through the loop, is x > 0? Yep!
Print 2, decrement x (x is now 1)

Fifth time through the loop, is x > 0? Yep!
Print 1, decrement x (x is now 0)

Sixth time through the loop, is x > 0?

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0? Yep!
Print 5, decrement x (x is now 4)

Second time through the loop, is x > 0? Yep!
Print 4, decrement x (x is now 3)

Third time through the loop, is x > 0? Yep!
Print 3, decrement x (x is now 2)

Fourth time through the loop, is x > 0? Yep!
Print 2, decrement x (x is now 1)

Fifth time through the loop, is x > 0? Yep!
Print 1, decrement x (x is now 0)

Sixth time through the loop, is x > 0? No!

Following the flow

```
x = 5
while x > 0:
    print(x)
    x = x - 1
print(“Done!”)
```

First time through the loop, is x > 0? Yep!
Print 5, decrement x (x is now 4)

Second time through the loop, is x > 0? Yep!
Print 4, decrement x (x is now 3)

Third time through the loop, is x > 0? Yep!
Print 3, decrement x (x is now 2)

Fourth time through the loop, is x > 0? Yep!
Print 2, decrement x (x is now 1)

Fifth time through the loop, is x > 0? Yep!
Print 1, decrement x (x is now 0)

Sixth time through the loop, is x > 0? No!
Exit loop, print “Done!”

Why?

It can be *very* useful to repeat the same thing over and over again in code

Why?

It can be *very* useful to repeat the same thing over and over again in code

Examples?

Why?

It can be *very* useful to repeat the same thing over and over again in code

Examples?

- Counting
- Main loop of a game
- Asking for user input *until they get it right*
- Search

Thinking through it - What do these examples do?

A)

```
correct = False
x = -1
while not correct:
   x += 1
   correct = check_answer(x)
print(x)
```

B)

```
while True:
   print(“hi!”)
```

C)

```
command = input()
while command != “quit”:
   run_command(command)
```

Thinking through it - What do these examples do?

A)

```
correct = False
x = -1
while not correct:
   x += 1
   correct = check_answer(x)
print(x)
```

Thinking through it - What do these examples do?

A)

```
correct = False
x = -1
while not correct:
    x += 1
    correct = check_answer(x)
print(x)
```

Increments x until we hit a value that causes check_answer to return True

Note: check_answer is a custom function

Thinking through it - What do these examples do?

B)

```
while True:
   print(“hi!”)
```

Thinking through it - What do these examples do?

Prints “hi!” **FOREVER**

B)

```
while True:
   print(“hi!”)
```

Thinking through it - What do these examples do?

Prints “hi!” **FORLEVER**

This is an infinite loop!

B)

```
while True:
   print(“hi!”)
```

Thinking through it - What do these examples do?

Prints “hi!” **FORWARD**

This is an infinite loop!

B)

```
while True:
   print(“hi!”)
```

How do we stop code that is running forever?

Thinking through it - What do these examples do?

B)

```
while True:
   print(“hi!”)
```

Prints “hi!” **FOREVER**

This is an infinite loop!

How do we stop code that is running forever?

Click in terminal and press Ctrl + C

Thinking through it - What do these examples do?

B)

```
while True:
    print(“hi!”)
```

Prints “hi!” **FOREVER**

This is an infinite loop!

How do we stop code that is running forever?

Click in terminal and press Ctrl + C

Or “kill terminal” in VSCode with the trash can

Thinking through it - What do these examples do?

Keeps asking the user for input

C)

```
command = input()
while command != “quit”:
   run_command(command)
```

Thinking through it - What do these examples do?

Keeps asking the user for input

Does *something* with normal inputs

C)

```
command = input()
while command != “quit”:
   run_command(command)
```

Thinking through it - What do these examples do?

Keeps asking the user for input

Does *something* with normal inputs

Ends when it encounters “quit”

C)

```
command = input()
while command != “quit”:
   run_command(command)
```

Thinking through it - What do these examples do?

Keeps asking the user for input

Does *something* with normal inputs

Ends when it encounters “quit”

But I lied!

C)

```
command = input()
while command != “quit”:
    run_command(command)
```

Thinking through it - What do these examples do?

Keeps asking the user for input

Does *something* with normal inputs

Ends when it encounters “quit”

But I lied!

This is an infinite loop because we only pull input once!

C)

```
command = input()
while command != “quit”:
    run_command(command)
```

Thinking through it - What do these examples do?

Keeps asking the user for input

Does *something* with normal inputs

Ends when it encounters “quit”

But I lied!

This is an infinite loop because we
only pull input once!

C)

```
command = input()
while command != “quit”:
    run_command(command)
    command = input()
```

Live coding: Average of N numbers

Imagine I want to take the average of some numbers

Live coding: Average of N numbers

Imagine I want to take the average of some numbers

How can I accept an arbitrary number of inputs from the user?

New keywords

`break` - stop the loop

`continue` - stop this iteration, jump back to top of loop and try again

What does this code do?

```
counter = 0
while counter < 5:
    counter += 1
    if counter == 3:
        break
    print(counter)
```

What does this code do?

```
counter = 0
while counter < 5:
    counter += 1
    if counter == 3:
        continue
    print(counter)
```

What does this code do?

```
counter = 0
while counter < 5:
    if counter == 3:
        continue
    counter += 1
    print(counter)
```

Iterating over a string

```
my_str = input()
```

Iterating over a string

```
my_str = input()
idx = 0
```

Iterating over a string

```
my_str = input()
idx = 0
while idx < len(my_str):
```

Iterating over a string

```
my_str = input()
idx = 0
while idx < len(my_str):
    print(my_str[idx])
    idx += 1
```

Reminder: while loops

Reminder: while loops

Keep looping until condition is False (or we break out)

For loops

For loops

We iterate over a sequence so often, we have a loop that does exactly that!

For loops

We iterate over a sequence so often, we have a loop that does exactly that!

Syntax:

```
for variable in container:
   # do stuff!
# outside of loop
```

For loops

We iterate over a sequence so often, we have a loop that does exactly that!

Syntax:

This is the variable we are looping through (e.g., a string)

```
for variable in container:
    # do stuff!
# outside of loop
```

For loops

We iterate over a sequence so often, we have a loop that does exactly that!

New variable to hold the current item (we choose the var name)

This is the variable we are looping through (e.g., a string)

Syntax:

```
for variable in container:
    # do stuff!
# outside of loop
```

For loops

We iterate over a sequence so often, we have a loop that does exactly that!

New variable to hold the current item (we choose the var name)

This is the variable we are looping through (e.g., a string)

Syntax:

```
for variable in container:
    # do stuff!
# outside of loop
```

Body of the loop executes once for each item in container

Comparing our loops

Comparing our loops

while

```
my_str = input()
idx = 0
while idx < len(my_str):
    print(my_str[idx])
    idx += 1
```

Comparing our loops

while

```
my_str = input()
idx = 0
while idx < len(my_str):
    print(my_str[idx])
    idx += 1
```

for

```
my_str = input()
for symbol in my_str:
    print(symbol)
```

What’s the point?

- You can build a for and a while loop to do the same thing
    - But usually one is cleaner than the other for your scenario
        - e.g., you do not need to increment variables in a for loop
- General rules:
    - Iterating over a container -> For loop
    - You know the number of loops right before looping -> For loop
    - Looping an unknown number of times -> While loop

`range`

Remember wanting to count from 1 to 100 with a while loop?

`range`

Remember wanting to count from 1 to 100 with a while loop?

For loops can do this easily using the `range` function

`range`

Remember wanting to count from 1 to 100 with a while loop?

For loops can do this easily using the `range` function

```
range(n)
```

`range`

Remember wanting to count from 1 to 100 with a while loop?

For loops can do this easily using the `range` function

`range(n)` -> go from 0 to n - 1

`range`

Remember wanting to count from 1 to 100 with a while loop?

For loops can do this easily using the `range` function

`range(n)` -> go from 0 to n - 1

`range(n, k)`

`range`

Remember wanting to count from 1 to 100 with a while loop?

For loops can do this easily using the `range` function

`range(n)` -> go from 0 to n - 1

`range(n, k)` -> go from n (inclusive) to k - 1

`range`

Remember wanting to count from 1 to 100 with a while loop?

For loops can do this easily using the `range` function

`range(n)` -> go from 0 to n - 1

`range(n, k)` -> go from n (inclusive) to k - 1

`range(n, k, z)`

`range`

Remember wanting to count from 1 to 100 with a while loop?

For loops can do this easily using the `range` function

`range(n)` -> go from 0 to n - 1

`range(n, k)` -> go from n (inclusive) to k - 1

`range(n, k, z)` -> go from n to k (don’t include k), step by z

`range`

Remember wanting to count from 1 to 100 with a while loop?

For loops can do this easily using the `range` function

`range(n)` -> go from 0 to n - 1

`range(n, k)` -> go from n (inclusive) to k - 1

`range(n, k, z)` -> go from n to k (don’t include k), step by z

Can use a negative z to go backward!

range example - What will it print?

```
for i in range(10):
   print(i)
```

range example - What will it print?

```
for i in range(10):
   print(i)
```

Prints 0 through 9 (does not print 10)

range example 2 - What will it print?

```
username = “lou1960”
for i in range(len(username)):
   print(f”{i}, {username[i]}”)
```

range example 2 - What will it print?

```
username = “lou1960”
for i in range(len(username)):
   print(f”{i}, {username[i]}”)
```

```
0, l
1, o
2, u
3, 1
4, 9
5, 6
7, 0
```

range example 3 - What will it print?

```
username = “lou1960”
for i in range(len(username)):
   if username[i].isdigit():
      print(f”First digit detected at index {i}”)
      break
```

range example 3 - What will it print?

```
username = “lou1960”
for i in range(len(username)):
    if username[i].isdigit():
        print(f”First digit detected at index {i}”)
        break
```

Looks character by character to find a digit, the prints and breaks

range example 3 - What will it print?

```
username = “lou1960”
for i in range(len(username)):
    if username[i].isdigit():
        print(f”First digit detected at index {i}”)
        break
```

Looks character by character to find a digit, the prints and breaks

Here, output would be: `First digit detected at index 3`

Going beyond strings

We can use for loops to loop through containers/sequences

Going beyond strings

We can use for loops to loop through containers/sequences

- So far, we’ve only talked about strings, but there are others!

Going beyond strings

We can use for loops to loop through containers/sequences

So far, we’ve only talked about strings, but there are others!

What is a string?

Going beyond strings

We can use for loops to loop through containers/sequences

So far, we’ve only talked about strings, but there are others!

What is a string?

A sequence of characters

Going beyond strings

We can use for loops to loop through containers/sequences

So far, we’ve only talked about strings, but there are others!

What is a string?

A sequence of characters

A <u>**list**</u> is a sequence of *values*

Going beyond strings

We can use for loops to loop through containers/sequences

So far, we've only talked about strings, but there are others!

What is a string?

A sequence of characters

A **list** is a sequence of *values*

More options than just characters

Lists

Strings use “ “ or ‘ ’ to denote their boundaries

Lists

Strings use “ “ or ‘ ’ to denote their boundaries

Lists use [and]

Lists

Strings use “ “ or ‘ ’ to denote their boundaries

Lists use [and]

```
example_list = [5, 2, “banana”, True]
```

Lists

Strings use “ “ or ‘ ’ to denote their boundaries

Lists use [and]

```
example_list = [5, 2, “banana”, True]
```

Why do we need commas here but not in strings?

Lists

Strings use “ “ or ‘ ’ to denote their boundaries

Lists use [and]

```
example_list = [5, 2, “banana”, True]
```

Why do we need commas here but not in strings?

Strings are *always* a sequence of characters

Lists

Strings use “ “ or ‘ ’ to denote their boundaries

Lists use [and]

```
example_list = [5, 2, “banana”, True]
```

Why do we need commas here but not in strings?

Strings are *always* a sequence of characters

Lists need commas to separate values (e.g., 52 vs 5,2)

Lists

```
example_list = [5, 2, “banana”, True]
```

Lists *can* hold different types at the same time (ints, strings, bools above)

Lists

```
example_list = [5, 2, “banana”, True]
```

Lists *can* hold different types at the same time (ints, strings, bools above)

But often, we store one type (e.g., a list of ints)

Accessing lists

```
example_list = [5, 2, “banana”, True]
```

Accessing lists

```
example_list = [5, 2, “banana”, True]
```

How can we access the first element in the list?

Accessing lists

```
example_list = [5, 2, “banana”, True]
```

How can we access the first element in the list?

```
    example_list[0]
```

Accessing lists

```
example_list = [5, 2, “banana”, True]
```

How can we access the first element in the list?

```
    example_list[0]
```

We can access elements in all the same ways as accessing characters in a string! :D

Accessing lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

How can I access the 87?
How can I access the 98?
What does `grades[3]` return?
How can I slice to return the last three grades?
How can I fetch the number of grades?
What does grades[0:-1:2] return?

Accessing lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

How can I access the 87? `grades[2]`

How can I access the 98?

What does `grades[3]` return?

How can I slice to return the last three grades?

How can I fetch the number of grades?

What does grades[0:-1:2] return?

Accessing lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

How can I access the 87? `grades[2]`

How can I access the 98? `grades[-1]`

What does `grades[3]` return?

How can I slice to return the last three grades?

How can I fetch the number of grades?

What does grades[0:-1:2] return?

Accessing lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

How can I access the 87? `grades[2]`

How can I access the 98? `grades[-1]`

What does `grades[3]` return? 92

How can I slice to return the last three grades?

How can I fetch the number of grades?

What does grades[0:-1:2] return?

Accessing lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

How can I access the 87? `grades[2]`

How can I access the 98? `grades[-1]`

What does `grades[3]` return? 92

How can I slice to return the last three grades? `grades[-3:]`

How can I fetch the number of grades?

What does grades[0:-1:2] return?

Accessing lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

How can I access the 87? `grades[2]`

How can I access the 98? `grades[-1]`

What does `grades[3]` return? 92

How can I slice to return the last three grades? `grades[-3:]`

How can I fetch the number of grades? `len(grades)`

What does grades[0:-1:2] return?

Accessing lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

How can I access the 87? `grades[2]`

How can I access the 98? `grades[-1]`

What does `grades[3]` return? 92

How can I slice to return the last three grades? `grades[-3:]`

How can I fetch the number of grades? `len(grades)`

What does grades[0:-1:2] return? `[100, 87, 90]`

Accessing lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

What does this line do?

```
grades[1] = 99
```

Accessing lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

What does this line do?

```
grades[1] = 99
```

It updates grades! `grades = [100, 99, 87, 92, 90, 75, 98]`

Accessing lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

What does this line do?

```
grades[1] = 99
```

It updates grades! `grades = [100, 99, 87, 92, 90, 75, 98]`

Strings are **immutable**, so this would fail.

Accessing lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

What does this line do?

```
grades[1] = 99
```

It updates grades! `grades = [100, 99, 87, 92, 90, 75, 98]`

Strings are **immutable**, so this would fail.
Lists are **mutable**, this is totally fine!

Recap (section 40)

What is the difference between
`for` and `while`?

What is a list?

What does range do?

Modifying lists

Modifying lists

We can create an empty list:

```
my_list = []
```

Modifying lists

We can create an empty list:

```
my_list = []
```

We can add to lists with `.append()`

Modifying lists

We can create an empty list:

```
my_list = []
```

We can add to lists with `.append()`

```
my_list.append(5)
```

Modifying lists

We can create an empty list:

```
my_list = []
```

We can add to lists with `.append()`

`my_list.append(5) -> my_list` is now `[5]`

Modifying lists

We can create an empty list:

```
my_list = []
```

We can add to lists with `.append()`

`my_list.append(5) -> my_list` is now `[5]`

Some list methods modify the list “in place”

Modifying lists

We can create an empty list:

`my_list = []`

We can add to lists with `.append()`

`my_list.append(5) -> my_list` is now `[5]`

Some list methods modify the list “in place”

We are <u>NOT</u> doing my_list = my_list.append(5) that would break!

Modifying lists

```
grades = [100, 95, 87, 92, 90, 75, 98, 75]
```

What do you think this does?

```
grades.remove(75)
```

Modifying lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

What do you think this does?

```
grades.remove(75)
```

It removes the first instance of 75 in the list

Modifying lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

What do you think this does?

```
grades.remove(75)
```

It removes the first instance of 75 in the list

So now grades is `[100, 95, 87, 92, 90, 98]`

Modifying lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

What do you think this does?

```
grades.remove(75)
```

It removes the first instance of 75 in the list

So now grades is `[100, 95, 87, 92, 90, 98]`

What happens if we do this?

```
grades.remove(0)
```

Modifying lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

What do you think this does?

```
grades.remove(75)
```

It removes the first instance of 75 in the list

So now grades is `[100, 95, 87, 92, 90, 98]`

What happens if we do this?

`grades.remove(0)` Error! 0 not in list

Modifying lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

We can also pop items out of the list!

```
grades.pop(0)
```

Modifying lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

We can also pop items out of the list!

```
grades.pop(0)
```

This removes the element at that index, and returns it

Modifying lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

We can also pop items out of the list!

```
grades.pop(0)
```

This removes the element at that index, and returns it

So `grades.pop(0)` returned 100, and grades is now:

```
[95, 87, 92, 90, 75, 98]
```

Modifying lists

```
grades = [100, 95, 87, 92, 90, 75, 98]
```

We can also pop items out of the list!

```
grades.pop(0)
```

This removes the element at that index, and returns it

So `grades.pop(0)` returned 100, and grades is now:
`[95, 87, 92, 90, 75, 98]`

`list.pop()` (no argument) pops the last element in list

Recap (section 30)

What is the difference between
`for` and `while`?

What is a list?

What does this code do?

```
my_list = []
my_list.append(3)
my_list.append(5)
my_list.append(7)
print(my_list)
my_list.pop(1)
print(my_list)
my_list.remove(3)
print(my_list)
```

List functions and methods

```
len(list1)
list1 + list2
min(list1)
max(list1)
sum(list1)
list.index(value)
list.count(value)
```

List functions and methods

`len(list)` Returns length of the list (# of items)

```
list1 + list2
min(list)
max(list)
sum(list)
list.index(value)
list.count(value)
```

List functions and methods

`len(list)`	Returns length of the list (# of items)
`list1 + list2`	Concatenates lists, produces a new list
`min(list)`	
`max(list)`	
`sum(list)`	
`list.index(value)`	
`list.count(value)`	

List functions and methods

`len(list)`	Returns length of the list (# of items)
`list1 + list2`	Concatenates lists, produces a new list
`min(list)`	Returns minimum value in list
`max(list)`	Returns minimum value in list
`sum(list)`	
`list.index(value)`	
`list.count(value)`	

List functions and methods

`len(list)`	Returns length of the list (# of items)
`list1 + list2`	Concatenates lists, produces a new list
`min(list)`	Returns minimum value in list
`max(list)`	Returns minimum value in list
`sum(list)`	Adds all items together (numbers only)
`list.index(value)`	
`list.count(value)`	

List functions and methods

`len(list)`	Returns length of the list (# of items)
`list1 + list2`	Concatenates lists, produces a new list
`min(list)`	Returns minimum value in list
`max(list)`	Returns minimum value in list
`sum(list)`	Adds all items together (numbers only)
`list.index(value)`	Returns index of first match of val, error if not found
`list.count(value)`	

List functions and methods

`len(list)`	Returns length of the list (# of items)
`list1 + list2`	Concatenates lists, produces a new list
`min(list)`	Returns minimum value in list
`max(list)`	Returns minimum value in list
`sum(list)`	Adds all items together (numbers only)
`list.index(value)`	Returns index of first match of val, error if not found
`list.count(value)`	Counts number of occurrences of val

Checking membership

We can use the `in` operator to check membership

Checking membership

We can use the `in` operator to check membership

```
if 100 not in grades:
   print('No perfect grades? :(')
if 0 in grades:
   print('uh oh')
```

Lists and loops

You can use a for loop on lists just like strings:

```
grades = [100, 95, 87, 92, 25, 90, 75, 98, 12]

for score in grades:
    if score < 60:
        print(f"Failing grade detected: {score}")
```

An incredibly useful method

Imagine you have a string:

```
dest = "4.37 ly"
```

And you want the two separate parts.

An incredibly useful method

Imagine you have a string:

```
dest = "4.37 ly"
```

And you want the two separate parts.

Strings have a `.split()` method that will split into parts:

An incredibly useful method

Imagine you have a string:

```
dest = "4.37 ly"
```

And you want the two separate parts.

Strings have a `.split()` method that will split into parts:

`dest.split()` returns the list `["4.37", "ly"]`

An incredibly useful method

Imagine you have a string:

```
dest = "4.37 ly"
```

And you want the two separate parts.

Strings have a `.split()` method that will split into parts:

`dest.split()` returns the list `["4.37", "ly"]`

An incredibly useful method

Imagine you have a string:

```
dest = "4.37 ly"
```

And you want the two separate parts.

Strings have a `.split()` method that will split into parts:

`dest.split()` returns the list `["4.37", "ly"]`

You can also specify a different delimiter:

```
"this<AH>is<AH>a<AH>test".split("<AH>")
```

`enumerate`

Not sure if you want to use a range in your for loop or not?

`enumerate`

Not sure if you want to use a range in your for loop or not?

`enumerate` is cheat code: just do both!

`enumerate`

Not sure if you want to use a range in your for loop or not?

`enumerate` is cheat code: just do both!

```
my_str = "test"
for index, symbol in enumerate(my_str):
   print(f"Character at index {index}: {symbol}")
```