

J233

Coding for Journalists

LECTURER
Soo Oh

PROMPTS

Sign in to
<https://pollev.com/soooh>

Agenda

Homework review

Functions

- Review
- Tests with `assert`
- Lambda functions

BREAK

Data structures

Homework

Homework Review

Answers

Code style

ChatGPT

```
# Write a function that takes in a number and returns  
# the negative of that number.
```

```
def negative(n):  
    return -n
```

```
# Write a function in Python that converts degrees  
# Fahrenheit to degrees Celsius. (To convert  
# temperatures in degrees Fahrenheit to Celsius,  
# subtract 32 and multiply by 5/9.)
```

```
def f_to_c(f):  
    return (f - 32) * 5/9
```

Homework Review

Answers

Code style

ChatGPT

Don't put spaces after definitions (style thing).

```
def negative (n):  
    return n * -1
```

don't need this space

```
# This is better  
def negative(n):  
    return n * -1
```

Homework Review

Answers

Code style

ChatGPT

You don't always need so many parentheses (but it's not a bad thing).

```
# This is unnecessary!  
def f_to_c(f):  
    return (((f - 32) * 5)/9)
```

```
# This is fine  
def f_to_c(f):  
    return (f - 32) * 5/9
```

Homework Review

Answers

Code style

ChatGPT

Markdown

You got credit for using Markdown blocks. For future notebooks, include:

- Assignment name in title
- Your name somewhere at the top
- *Some* formatting for the questions, preferably using header styles

Homework Review

Answers

Code style

ChatGPT



You can create a simple Python function to return the negative of a given number like this:



python

Copy code

```
def get_negative(number):  
    return -number  
  
# Example usage:  
num = 5  
negative_num = get_negative(num)  
print(negative_num) # Output will be -5
```

In this function, `get_negative` takes a single argument `number` and returns its negation by simply multiplying it by -1.




Homework Review

Answers


Code style

ChatGPT



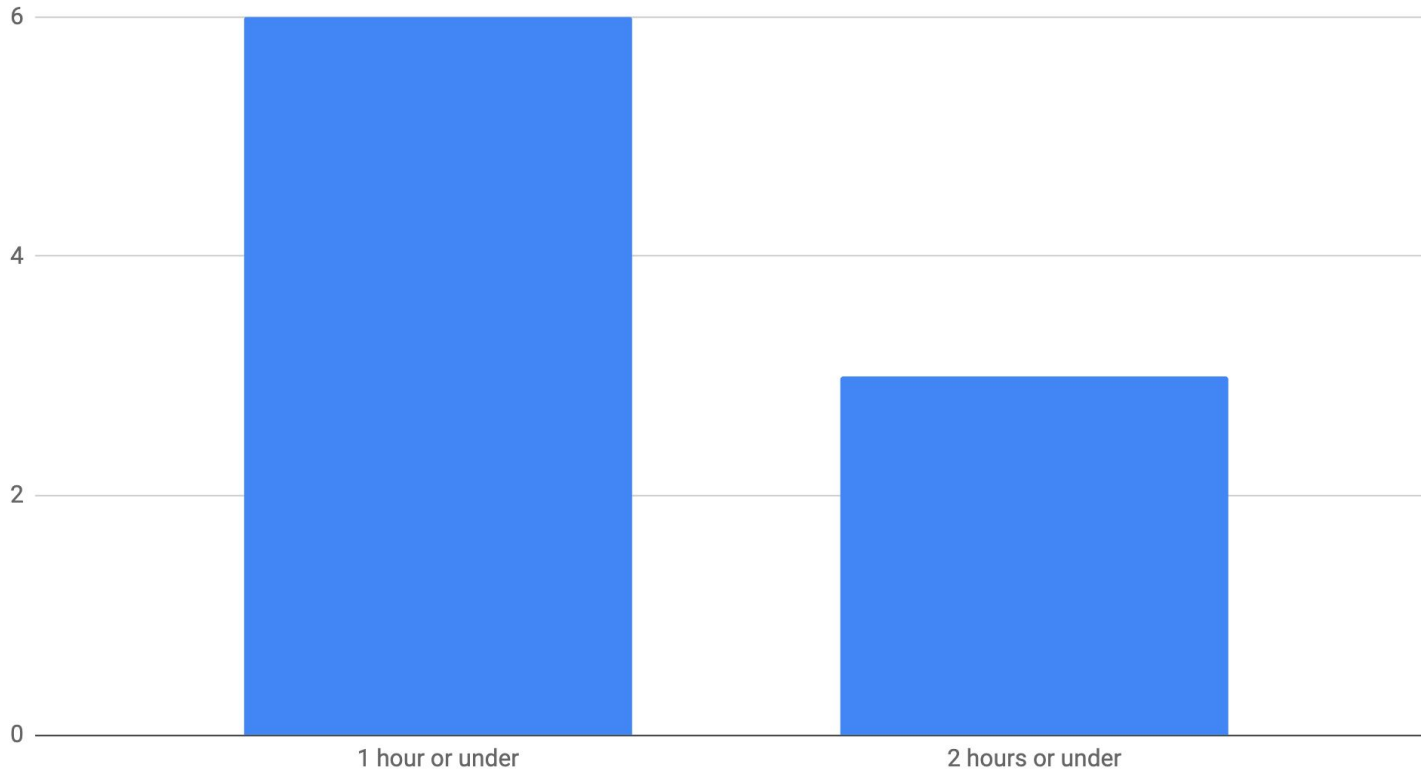
You can create a Python function to convert degrees Fahrenheit to degrees Celsius using the following formula:   

python

 Copy code

```
def fahrenheit_to_celsius(fahrenheit):  
    celsius = (fahrenheit - 32) * 5/9  
    return celsius  
  
# Example usage:  
fahrenheit = 98.6 # Replace with the Fahrenheit temperature you want to convert  
celsius = fahrenheit_to_celsius(fahrenheit)  
print(f"{fahrenheit} degrees Fahrenheit is equal to {celsius:.2f} degrees Celsius")
```


Week of 0911: students grouped by time spent outside of lecture and office hours



What questions do you have?

Functions and tests

Review

Tests with assert

Lambda functions

Anatomy of a function with no arguments

```
def print_something():  
    print('something')
```

```
print_something()
```

```
Out[]: 'something'
```

```
x = print_something()
```

```
Out[]: 'something'
```

```
print(x)
```

```
Out[]: None
```

We'll learn this later today!

Functions and tests

Review

Tests with assert

Lambda functions

4-space indent

Anatomy of a function with no arguments

The diagram illustrates the components of a Python function definition and its usage. It features a code snippet on a dark background with several callout boxes and red arrows pointing to specific parts of the code:

- def keyword**: Points to the `def` keyword in the function definition.
- function name (variable)**: Points to `print_something` in the function definition.
- parentheses**: Points to the empty parentheses `()` in the function definition.
- colon**: Points to the colon `:` at the end of the function definition line.
- 4-space indent**: Points to the four-space indentation of the function body line.
- call the function**: Points to the function call `print_something()` in the code snippet.

```
def print_something():  
    print('something')  
  
print_something()  
Out[]: 'something'  
  
x = print_something()  
Out[]: 'something'  
  
print(x)  
Out[]: None
```

Functions and tests

Review

Tests with assert

Lambda functions

Anatomy of a function with parameters

```
def even_number(num):  
    return num % 2 == 0
```

```
is_two_even = even_number(2)  
is_two_even  
Out[]: True
```

Functions and tests

Review

Tests with assert

Lambda functions

Anatomy of a function with parameters

def keyword

function name (variable)

parentheses with parameters

```
def even_number(num):  
    return num % 2 == 0
```

colon

use **return** to output a value

4-space indent

```
is_two_even = even_number(2)  
is_two_even  
Out[]: True
```

call function with argument (that's what's in between the parens)

Let's work on an example

Create a notebook for this lecture

Functions and tests

Review

Tests with assert

Lambda functions

```
# Write a function that calculates the total sales  
# price of an item with tax (10.25%)
```

Student version of slide

Functions and tests

Review

Tests with assert

Lambda functions

```
# Write a function that calculates the total sales  
# price of an item with tax (10.25%)
```

Student version of slide

```
assert total_price(10) == 11.025  
# In JupyterLab, there is no output  
  
# In a new cell, test a different assertion  
assert total_price(10) == 11  
Out[]: Error
```

Functions and tests

Review

Tests with assert

Lambda functions

Lambda functions are also known as **anonymous functions**.

You don't have to **def** a lambda function, but you must write it all in one line (can be limiting).

Functions and tests

Review

Tests with assert

Lambda functions

Student version of slide

```
def squared(n):  
    return n ** 2
```

```
squared(4)
```

```
Out[]:
```

```
# Lambdas (a.k.a. anonymous) functions can only be  
# written in one line (not multi-line like defined  
# functions)
```

```
squared = lambda n: n**2
```

```
squared(4)
```

```
Out[]:
```

```
exponents = lambda n, p: n**p
```

```
exponents(4, 3)
```

```
Out[]:
```

Functions and tests

Review

Tests with assert

Lambda functions

Anatomy of a lambda function with parameters

```
squared = lambda n: n**2
```

```
exponents = lambda n, p: n**p
```

Functions and tests

Review

Tests with assert

Lambda functions

Anatomy of a lambda function with parameters

```
squared = lambda n: n**2
```

```
exponents = lambda n, p: n**p
```

These lambda functions are named, but in the future, you will not always need to name them!

Functions and tests

Review

Tests with assert

Lambda functions

Anatomy of a lambda function with parameters

```
squared = lambda n: n**2
```

```
exponents = lambda n, p: n**p
```

keyword `lambda`

Functions and tests

Review

Tests with assert

Lambda functions

Anatomy of a lambda function with parameters

```
squared = lambda n: n**2
```

```
exponents = lambda n, p: n**p
```

parameters, separated by a comma if you have more than one

Functions and tests

Review

Tests with assert

Lambda functions

Anatomy of a lambda function with parameters

```
squared = lambda r: n**2
```

colon

```
exponents = lambda n, p: n**p
```


Functions and tests

Review

Tests with assert

Lambda functions

Anatomy of a lambda function with parameters

```
squared = lambda n: n**2
```

what the function
returns, without using
the **return** keyword

```
exponents = lambda n, p: n**p
```

What questions do you have?

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

Data structures are... structures to hold data.

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
university = 'Berkeley'
```

```
len(university)
```

```
Out[]:
```

```
university[0]
```

```
Out[]:
```

```
university[8]
```

```
Out[]:
```

```
university[-1]
```

```
Out[]:
```

```
university[:2]
```

```
Out[]:
```

Student version of slide

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
university = 'Berkeley'
```

```
university[5:]
```

```
Out[ ]:
```

```
university[:-2]
```

```
Out[ ]:
```

```
university[2:4]
```

```
Out[ ]:
```

Student version of slide

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# just like int() and float(),  
# we also have str()  
# converts elements into strings
```

```
str(432)
```

```
Out[]:
```

```
str(432.0)
```

```
Out[]:
```

Student version of slide

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

A **list** is an ordered collection of elements.

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
j233 = ['Saumya', 'Simmerdeep', 'Nadia', 'Wendy',  
        'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey']
```

```
len(j233)
```

```
Out[]:
```

Student version of slide

```
j233[0]
```

```
Out[]:
```

```
j233[9]
```

```
Out[]:
```


Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
j233 = ['Saumya', 'Simmerdeep', 'Nadia', 'Wendy',  
        'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey']
```

```
j233[-1]
```

```
Out[]:
```

Student version of slide

```
j233[2:6]
```

```
Out[]:
```

```
# list method: list.sort()
```

```
j233.sort()
```

```
j233
```

```
Out[]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
j233 = ['Saumya', 'Simmerdeep', 'Nadia', 'Wendy',  
        'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey']
```

```
# list method: in
```

```
'Soo' in j233
```

```
Out[ ]:
```

Student version of slide

```
# list method: not in
```

```
'Soo' not in j233
```

```
Out[ ]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
random_numbers = [3, 0, 10, -2, 10, 7, -2]
random_numbers.sort()
random_numbers
Out[]:
```

Student version of slide

a list can be made of different types

but it's not ideal

```
random_numbers_plus_bear = [3, 0, 10, -2, 10, 7,
                             -2, 'bear']
random_numbers_plus_bear.sort()
Out[]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
j233 = ['Saumya', 'Simmerdeep', 'Nadia', 'Wendy',  
'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey',  
'Soo']  
lecturers = ['Soo', 'Yoli']
```

Student version of slide

```
# combine 2 lists  
everyone = j233 + lecturers  
everyone  
Out[ ]:
```

What questions do you have?

Data structures

string indexing

list

set

dict

methods

↷ None

↷ mutable vs. immutable

tuples

loops

A **set** is an unordered collection with no duplicate elements.

Data structures

string indexing

list

set

dict

methods

↷ None

↷ mutable vs. immutable

tuples

loops

everyone

Out[]:

Student version of slide

```
# new function: set()
```

```
everyone_set = set(everyone)
```

```
everyone_set
```

```
Out[]:
```

How is set different from list? (No limit to responses)

Nobody has responded yet.

Hang tight! Responses are coming in.



Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

everyone

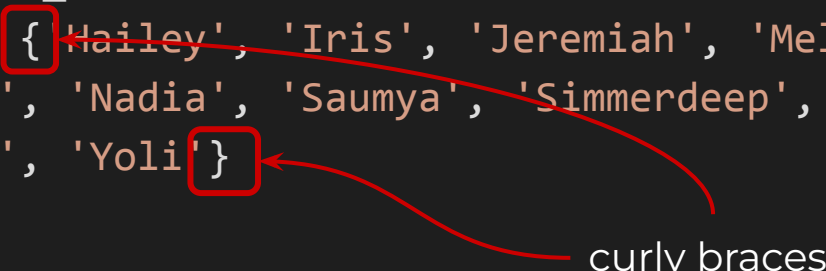
```
Out[]: ['Saumya', 'Simmerdeep', 'Nadia', 'Wendy',  
'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey',  
'Soo', 'Soo', 'Yoli']
```

```
# new function: set()
```

```
everyone_set = set(everyone)
```

```
everyone_set
```

```
Out[]: {'Hailey', 'Iris', 'Jeremiah', 'Melanie',  
'Mitzi', 'Nadia', 'Saumya', 'Simmerdeep', 'Soo',  
'Wendy', 'Yoli'}
```



curly braces

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

everyone

```
Out[]: ['Saumya', 'Simmerdeep', 'Nadia', 'Wendy',  
'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey',  
'Soo', 'Soo', 'Yoli']
```

```
# new function: set()
```

```
everyone_set = set(everyone)
```

```
everyone_set
```

```
Out[]: {'Hailey', 'Iris', 'Jeremiah', 'Melanie',  
'Mitzi', 'Nadia', 'Saumya', 'Simmerdeep', 'Soo',  
'Wendy', 'Yoli'}
```

```
# 'Soo' doesn't repeat
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
random_set = set([5, 3, 43343, -3, 443, 94, -93])
random_set
Out[]: {-93, -3, 3, 5, 94, 443, 43343}
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
random_set = set([5, 3, 43343, -3, 443, 94, -93])
random_set
Out[]: {-93, -3, 3, 5, 94, 443, 43343}
# note that it doesn't actually sort!
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
random_set = set([5, 3, 43343, -3, 443, 94, -93])
```

```
random_set
```

```
Out[]: {-93, -3, 3, 5, 94, 443, 43343}
```

```
# note that it doesn't actually sort!
```

```
random_set[0]
```

```
Out[]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
random_set = set([5, 3, 43343, -3, 443, 94, -93])
```

```
random_set
```

```
Out[]: {-93, -3, 3, 5, 94, 443, 43343}
```

```
# note that it doesn't actually sort!
```

```
random_set[0]
```

```
Out[]: Error
```

```
# Can't be indexed because it's not ordered
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
random_set = set([5, 3, 43343, -3, 443, 94, -93])
```

```
random_set
```

```
Out[]: {-93, -3, 3, 5, 94, 443, 43343}
```

```
# note that it doesn't actually sort!
```

```
random_set[0]
```

```
Out[]: Error
```

```
# Can't be indexed because it's not ordered
```

```
set([5, 3, 43343, -3, 443, 94, -93, 'dog'])
```

```
Out[]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
random_set = set([5, 3, 43343, -3, 443, 94, -93])
```

```
random_set
```

```
Out[]: {-93, -3, 3, 5, 94, 443, 43343}
```

```
# note that it doesn't actually sort!
```

```
random_set[0]
```

```
Out[]: Error
```

```
# Can't be indexed because it's not ordered
```

```
set([5, 3, 43343, -3, 443, 94, -93, 'dog'])
```

```
Out[]: {-3, -93, 3, 43343, 443, 5, 94, 'dog'}
```

```
# This mixed 'sorted' set will print alpha
```

```
# Remember: it's not actually sorted
```


Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
j233 = {'Saumya', 'Simmerdeep', 'Nadia', 'Wendy',  
'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey',  
'Soo'}  
lecturers = {'Soo', 'Yoli'}
```

Set operations

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
j233 = {'Saumya', 'Simmerdeep', 'Nadia', 'Wendy',  
        'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey',  
        'Soo'}  
lecturers = {'Soo', 'Yoli'}
```

& (intersection)

What exists in both sets?

```
j233 & lecturers  
Out[ ]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
j233 = {'Saumya', 'Simmerdeep', 'Nadia', 'Wendy',  
        'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey',  
        'Soo'}  
lecturers = {'Soo', 'Yoli'}
```

& (intersection)

What exists in both sets?

```
j233 & lecturers  
Out[: {'Soo'}
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
j233 = {'Saumya', 'Simmerdeep', 'Nadia', 'Wendy',  
        'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey',  
        'Soo'}  
lecturers = {'Soo', 'Yoli'}
```

| (union)

What's the combination of the set?

```
j233 | lecturers  
Out[ ]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
j233 = {'Saumya', 'Simmerdeep', 'Nadia', 'Wendy',  
'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey',  
'Soo'}  
lecturers = {'Soo', 'Yoli'}
```

| (union)

What's the combination of the set?

```
j233 | lecturers  
Out[:] {'Hailey', 'Iris', 'Jeremiah', 'Melanie',  
'Mitzi', 'Nadia', 'Saumya', 'Simmerdeep', 'Soo',  
'Wendy', 'Yoli'}
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
j233 = {'Saumya', 'Simmerdeep', 'Nadia', 'Wendy',  
'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey',  
'Soo'}  
lecturers = {'Soo', 'Yoli'}
```

– (difference)

```
# What's unique to lecturers
```

```
lecturers - j233
```

```
Out[ ]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
j233 = {'Saumya', 'Simmerdeep', 'Nadia', 'Wendy',  
        'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey',  
        'Soo'}  
lecturers = {'Soo', 'Yoli'}
```

– (difference)

```
# What's unique to lecturers  
lecturers - j233  
Out[]: {'Yoli'}
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
j233 = {'Saumya', 'Simmerdeep', 'Nadia', 'Wendy',
        'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey',
        'Soo'}
lecturers = {'Soo', 'Yoli'}
```

– (difference)

```
# What's unique to lecturers
```

```
lecturers - j233
```

```
Out[]: {'Yoli'}
```

```
# What's unique to j233
```

```
j233 - lecturers
```

```
Out[]:
```


Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
j233 = {'Saumya', 'Simmerdeep', 'Nadia', 'Wendy',
        'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey',
        'Soo'}
lecturers = {'Soo', 'Yoli'}
```

– (difference)

```
# What's unique to lecturers
```

```
lecturers - j233
```

```
Out[]: {'Yoli'}
```

```
# What's unique to j233
```

```
j233 - lecturers
```

```
Out[]: {'Hailey', 'Iris', 'Jeremiah', 'Melanie',
        'Mitzi', 'Nadia', 'Saumya', 'Simmerdeep', 'Wendy'}
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
j233 = {'Saumya', 'Simmerdeep', 'Nadia', 'Wendy',  
        'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey',  
        'Soo'}  
lecturers = {'Soo', 'Yoli'}
```

\wedge (symmetric difference)

What exists uniquely in each set?

```
j233 ^ lecturers  
Out[ ]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
j233 = {'Saumya', 'Simmerdeep', 'Nadia', 'Wendy',  
        'Mitzi', 'Iris', 'Jeremiah', 'Melanie', 'Hailey',  
        'Soo'}  
lecturers = {'Soo', 'Yoli'}
```

\wedge (symmetric difference)

What exists uniquely in each set?

```
j233 ^ lecturers  
Out[:] {'Hailey', 'Iris', 'Jeremiah', 'Melanie',  
        'Mitzi', 'Nadia', 'Saumya', 'Simmerdeep', 'Wendy',  
        'Yoli'}
```

What questions do you have?

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

A **dict** (dictionary) is kind of like a **set** with **key: value** pairs.

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

colon between
key and value

commas after
every key: value
pair except the
last pair

wrapped in curly braces

Data structures

string indexing

list

set

dict

methods

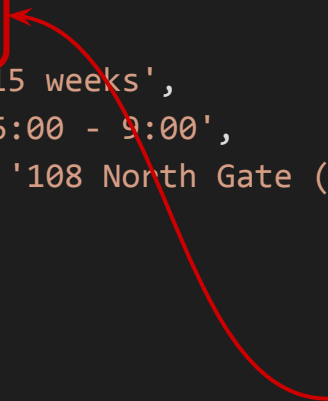
↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'units': 5,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```



Think of dict as a set
(using curly braces) with
attributes. For example,
you can't have two of the
same attributes.

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

```
course['department']
```

```
Out[ ]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

```
course['department']
```

```
Out[]: 'JOURN'
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

```
course['department']
```

```
Out[]: 'JOURN'
```

```
course['instructor']
```

```
Out[]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

```
course['department']
```

```
Out[]: 'JOURN'
```

```
course['instructor']
```

```
Out[]: 'Soo Oh'
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

```
list(course)
```

```
Out[]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

```
list(course)
```

```
Out[]: ['department', 'number', 'title', 'term', 'instructor',  
        'units', 'length', 'time', 'location']
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

```
'department' in course
```

```
Out[]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

```
'department' in course
```

```
Out[]: True
```


Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

```
'department' in course
```

```
Out[]: True
```

```
'section' in course
```

```
Out[]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

```
'department' in course
```

```
Out[]: True
```

```
'section' in course
```

```
Out[]: False
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

```
course['location']
```

```
Out[ ]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

```
course['location']  
Out[]: '108 North Gate (Lower News)'
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

```
course['location']  
Out[]: '108 North Gate (Lower News)'
```

```
# Change the value of 'location' to 'Online via Zoom'
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
course = {  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': '108 North Gate (Lower News)'  
}
```

```
course['location']  
Out[]: '108 North Gate (Lower News)'
```

```
# Change the value of 'location' to 'Online via Zoom'  
course['location'] = 'Online via Zoom'
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

course

Out[]:

```
{  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': 'Online via Zoom'  
}
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

course

Out[]:

```
{  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': 'Online via Zoom'  
}
```

Add a new key:value pair ('section': '001')

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

course

Out[]:

```
{  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': 'Online via Zoom'  
}
```

Add a new key:value pair ('section': '001')

```
course['section'] = '001'
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

course

Out[]:

```
{  
    'department': 'JOURN',  
    'number': '223',  
    'title': 'Coding for Journalists',  
    'term': 'Fall 2023',  
    'instructor': 'Soo Oh',  
    'units': 3,  
    'length': '15 weeks',  
    'time': 'M 6:00 - 9:00',  
    'location': 'Online via Zoom',  
    'section': '001'  
}
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Creating empty structures
```

```
empty_list = []
```

```
# is the same as
```

```
empty_list = list()
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Creating empty structures
```

```
empty_list = []
```

```
# is the same as
```

```
empty_list = list()
```

```
empty_dict = {}
```

```
# is the same as
```

```
empty_dict = dict()
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Creating empty structures
```

```
empty_list = []
```

```
# is the same as
```

```
empty_list = list()
```

```
empty_dict = {}
```

```
# is the same as
```

```
empty_dict = dict()
```

```
empty_set = set()
```

```
# only one way :(
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
colors_set = {'green', 'blue', 'red', 'yellow'}
```

```
# earlier we learned about set(list) and list(dict)
```

```
colors_list = list(colors_set)
```

```
colors_list
```

```
Out[]:
```

Student version of slide

```
colors_list.remove('red')
```

```
colors_list
```

```
Out[]:
```

```
colors_set_redux = set(colors_list)
```

```
colors_set_redux
```

```
Out[]:
```

```
colors_set
```

```
Out[]:
```

Data structures

string indexing

list

set

dict

methods

↪ **None**

↪ mutable vs. immutable

tuples

loops

A brief detour into **None**

How do you represent nothing?

Data structures

string indexing

list

set

dict

methods

↪ **None**

↪ mutable vs. immutable

tuples

loops

A brief detour into **None**

How do you represent nothing?

store	apples	bananas	kiwis
Store A	52	9	27
Store B	2		100
Store C	0	53	4

Data structures

string indexing

list

set

dict

methods

↪ **None**

↪ mutable vs. immutable

tuples

loops

A brief detour into **None**

How do you represent nothing?

store	apples	bananas	kiwis
Store A	52	9	27
Store B	2		100
Store C	0	53	4

Data structures

string indexing

list

set

dict

methods

↷ **None**

↷ mutable vs. immutable

tuples

loops

```
store_b = {  
    'store': 'Store B',  
    'apples': 2,  
    'bananas': None,  
    'kiwis': 100  
}
```

```
store_b['store']  
Out[]:
```

```
store_b['apples']  
Out[]:
```

```
store_b['bananas']  
Out[]:
```

```
type(store_b['bananas'])  
Out[]:
```

Student version of slide

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
random_list = [2, 34, 3]
random_list.extend([-4, 35, 16])
random_list
Out[]:
```

```
sorted(random_list)
Out[]:
```

```
random_list
Out[]:
```

```
sorted_random_list = random_list.sort()
type(sorted_random_list)
Out[]:
```

```
sorted_random_list = sorted(random_list)
```

Student version of slide

Data structures

string indexing

list

set

dict

methods

↪ **None**

↪ mutable vs. immutable

tuples

loops

From the [Python documentation](#):

“You might have noticed that methods like **insert**, **remove** or **sort** that only modify the list have no return value printed – they return the default **None**.

This is a design principle for all **mutable** data structures in Python.”

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# strings are immutable
```

```
my_name = 'Soo'
```

```
my_name[0]
```

```
Out[ ]:
```

```
my_name[0] = 's'
```

```
Out[ ]:
```

```
# But you can change the variable
```

```
my_name = 'Oh'
```

```
my_name
```

```
Out[ ]:
```

```
# Lists are mutable
```

```
my_names = ['Soo', 'Oh']
```

```
my_names[0] = 'soo'
```

```
my_names
```

```
Out[ ]:
```

Student version of slide

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
store_a = {'store': 'Store A', 'apples': 52, 'bananas': 9, 'kiwis': 27}
store_b = {'store': 'Store B', 'apples': 2, 'bananas': None, 'kiwis': 100}
store_c = {'store': 'Store C', 'apples': 0, 'bananas': 53, 'kiwis': 4}
```

```
inventory = [store_a, store_b]
```

```
# New method: .append()
inventory.append(store_c)
inventory
Out[ ]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
store_a = {'store': 'Store A', 'apples': 52, 'bananas': 9, 'kiwis': 27}
store_b = {'store': 'Store B', 'apples': 2, 'bananas': None, 'kiwis': 100}
store_c = {'store': 'Store C', 'apples': 0, 'bananas': 53, 'kiwis': 4}
```

```
inventory = [store_a, store_b]
```

```
# New method: .append()
```

```
inventory.append(store_c)
```

```
inventory
```

```
Out[ ]:
```

```
[
    {'store': 'Store A', 'apples': 52, 'bananas': 9, 'kiwis': 27},
    {'store': 'Store B', 'apples': 2, 'bananas': None, 'kiwis': 100},
    {'store': 'Store C', 'apples': 0, 'bananas': 53, 'kiwis': 4}
]
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
store_a = {'store': 'Store A', 'apples': 52, 'bananas': 9, 'kiwis': 27}
store_b = {'store': 'Store B', 'apples': 2, 'bananas': None, 'kiwis': 100}
store_c = {'store': 'Store C', 'apples': 0, 'bananas': 53, 'kiwis': 4}
```

```
inventory_a_b = [store_a, store_b]
inventory_all_stores = inventory_a_b + [ store_c ]
```


Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
store_a = {'store': 'Store A', 'apples': 52, 'bananas': 9, 'kiwis': 27}
store_b = {'store': 'Store B', 'apples': 2, 'bananas': None, 'kiwis': 100}
store_c = {'store': 'Store C', 'apples': 0, 'bananas': 53, 'kiwis': 4}
```

```
inventory_a_b = [store_a, store_b]
inventory_all_stores = inventory_a_b + [ store_c ]
```

```
inventory_a_b
```

```
Out[ ]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
store_a = {'store': 'Store A', 'apples': 52, 'bananas': 9, 'kiwis': 27}
store_b = {'store': 'Store B', 'apples': 2, 'bananas': None, 'kiwis': 100}
store_c = {'store': 'Store C', 'apples': 0, 'bananas': 53, 'kiwis': 4}
```

```
inventory_a_b = [store_a, store_b]
inventory_all_stores = inventory_a_b + [ store_c ]
```

```
inventory_a_b
```

```
Out[ ]:
```

```
[
    {'store': 'Store A', 'apples': 52, 'bananas': 9, 'kiwis': 27},
    {'store': 'Store B', 'apples': 2, 'bananas': None, 'kiwis': 100}
]
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
store_a = {'store': 'Store A', 'apples': 52, 'bananas': 9, 'kiwis': 27}
store_b = {'store': 'Store B', 'apples': 2, 'bananas': None, 'kiwis': 100}
store_c = {'store': 'Store C', 'apples': 0, 'bananas': 53, 'kiwis': 4}
```

```
inventory_a_b = [store_a, store_b]
inventory_all_stores = inventory_a_b + [ store_c ]
```

```
inventory_a_b
```

```
Out[ ]:
```

```
[
    {'store': 'Store A', 'apples': 52, 'bananas': 9, 'kiwis': 27},
    {'store': 'Store B', 'apples': 2, 'bananas': None, 'kiwis': 100}
]
```

```
inventory_all_stores
```

```
Out[ ]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
store_a = {'store': 'Store A', 'apples': 52, 'bananas': 9, 'kiwis': 27}
store_b = {'store': 'Store B', 'apples': 2, 'bananas': None, 'kiwis': 100}
store_c = {'store': 'Store C', 'apples': 0, 'bananas': 53, 'kiwis': 4}
```

```
inventory_a_b = [store_a, store_b]
inventory_all_stores = inventory_a_b + [ store_c ]
```

```
inventory_a_b
```

```
Out[]:
```

```
[
    {'store': 'Store A', 'apples': 52, 'bananas': 9, 'kiwis': 27},
    {'store': 'Store B', 'apples': 2, 'bananas': None, 'kiwis': 100}
]
```

```
inventory_all_stores
```

```
Out[]:
```

```
[
    {'store': 'Store A', 'apples': 52, 'bananas': 9, 'kiwis': 27},
    {'store': 'Store B', 'apples': 2, 'bananas': None, 'kiwis': 100},
    {'store': 'Store C', 'apples': 0, 'bananas': 53, 'kiwis': 4}
]
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

There are many built-in methods for variable types (including strings) and data structures:

- Optional reading:
<https://docs.python.org/3/library/stdtypes.html>

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

Tuples are similar to lists but they're **immutable**.

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
grades = ('A', 'B', 'C', 'D', 'F')
```

```
# tuples operate like lists; many of the  
# same methods apply
```

```
grades[0]
```

```
Out[ ]:
```

Student version of slide

```
grades[1]
```

```
Out[ ]:
```

```
len(grades)
```

```
Out[ ]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
grades = ('A', 'B', 'C', 'D', 'F')
```

```
# But elements inside tuples cannot change
```

```
# so methods like .extend() or .append() will not work
```

```
grades[0] = 'A+'
```

```
Out[]:
```

Student version of slide

```
# You can redefine the whole variable though
```

```
grades = ['A', 'B', 'C', 'D', 'F']
```

```
print(grades)
```

```
Out[]:
```


Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

A brief intro to loops...

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Loop through a list  
for n in range(0, 5):  
    print(n, end=' ')  
Out[ ]:
```

range() is a built-in
Python function

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Loop through a list
for n in range(0, 5):
    print(n, end=' ')
Out[]: 0 1 2 3 4
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Loop through a list
```

```
for n in range(0, 5):  
    print(n, end=' ')
```

```
Out[]: 0 1 2 3 4
```

```
vegetables = ['asparagus', 'onion', 'salad greens', 'radishes']
```

```
for vegetable in vegetables:  
    print(vegetable, end=' ')
```

```
Out[]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Loop through a list
```

```
for n in range(0, 5):  
    print(n, end=' ')
```

```
Out[]: 0 1 2 3 4
```

```
vegetables = ['asparagus', 'onion', 'salad greens', 'radishes']
```

```
for vegetable in vegetables:  
    print(vegetable, end=' ')
```

```
Out[]: asparagus onion salad greens radishes
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Loop through a List
```

```
for n in range(0, 5):  
    print(n, end=' ')
```

```
Out[]: 0 1 2 3 4
```

```
vegetables = ['asparagus', 'onion', 'salad greens', 'radishes']
```

```
for vegetable in vegetables:  
    print(vegetable, end=' ')
```

```
Out[]: asparagus onion salad greens radishes
```

```
for vegetable in vegetables:  
    index = vegetables.index(vegetable)  
    print(f'{index} {vegetable}')
```

```
Out[]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Loop through a List
```

```
for n in range(0, 5):  
    print(n, end=' ')
```

```
Out[]: 0 1 2 3 4
```

```
vegetables = ['asparagus', 'onion', 'salad greens', 'radishes']
```

```
for vegetable in vegetables:  
    print(vegetable, end=' ')
```

```
Out[]: asparagus onion salad greens radishes
```

```
for vegetable in vegetables:  
    index = vegetables.index(vegetable)  
    print(f'{index} {vegetable}')
```

```
Out[]:
```

```
0 asparagus
```

```
1 onion
```

```
2 salad greens
```

```
3 radishes
```

```
# does the same thing as
```

```
for index, vegetable in enumerate(vegetables):  
    print(f'{index} {vegetable}')
```

```
# (and you get to save a line!)
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Loop through a List
```

```
for n in range(0, 5):  
    print(n, end=' ')
```

```
Out[]: 0 1 2 3 4
```

```
vegetables = ['asparagus', 'onion', 'salad greens', 'radishes']
```

```
for vegetable in vegetables:  
    print(vegetable, end=' ')
```

```
Out[]: asparagus onion salad greens radishes
```

```
for vegetable in vegetables:  
    index = vegetables.index(vegetable)  
    print(f'{index} {vegetable}')
```

```
Out[]:
```

```
0 asparagus
```

```
1 onion
```

```
2 salad greens
```

```
3 radishes
```

```
# does the same thing as
```

```
for index, vegetable in enumerate(vegetables):  
    print(f'{index} {vegetable}')
```

```
# (and you get to save a line!)
```


Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Loop through a List
```

```
for n in range(0, 5):  
    print(n, end=' ')
```

```
Out[]: 0 1 2 3 4
```

```
vegetables = ['asparagus', 'onion', 'salad greens', 'radishes']
```

```
for vegetable in vegetables:  
    print(vegetable, end=' ')
```

```
Out[]: asparagus onion salad greens radishes
```

```
for vegetable in vegetables:  
    index = vegetables.index(vegetable)  
    print(f'{index} {vegetable}')
```

```
Out[]:
```

```
0 asparagus  
1 onion  
2 salad greens  
3 radishes
```

doesn't print with
quote marks in
notebooks

```
# does the same thing as
```

```
for index, vegetable in enumerate(vegetables):  
    print(f'{index} {vegetable}')
```

```
# (and you get to save a line!)
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Loop through a set the same way
```

```
grocery_list = {'bread', 'cheese', 'tomatoes', 'bread'}
```

```
grocery_list
```

```
Out[ ]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Loop through a set the same way
```

```
grocery_list = {'bread', 'cheese', 'tomatoes', 'bread'}
```

```
grocery_list
```

```
Out[: {'bread', 'cheese', 'tomatoes'}
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Loop through a set the same way
```

```
grocery_list = {'bread', 'cheese', 'tomatoes', 'bread'}
```

```
grocery_list
```

```
Out[]: {'bread', 'cheese', 'tomatoes'}
```

```
for item in grocery_list:
```

```
    print(item)
```

```
Out[]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Loop through a set the same way
```

```
grocery_list = {'bread', 'cheese', 'tomatoes', 'bread'}
```

```
grocery_list
```

```
Out[]: {'bread', 'cheese', 'tomatoes'}
```

```
for item in grocery_list:
```

```
    print(item)
```

```
Out[]:
```

```
tomatoes
```

```
cheese
```

```
bread
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Loop through a dict
```

```
store_a = {'store': 'Store A', 'apples': 52, 'bananas': 9,  
           'kiwis': 27}
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Loop through a dict
```

```
store_a = {'store': 'Store A', 'apples': 52, 'bananas': 9,  
           'kiwis': 27}
```

```
for key, value in store_a.items():  
    print(f'{ key } --- { value }')
```

```
Out[]:
```

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

```
# Loop through a dict
```

```
store_a = {'store': 'Store A', 'apples': 52, 'bananas': 9,  
           'kiwis': 27}
```

```
for key, value in store_a.items():  
    print(f'{ key } --- { value }')
```

```
Out[ ]:
```

```
store --- Store A
```

```
apples --- 52
```

```
bananas --- 9
```

```
kiwis --- 27
```


Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

Let's practice writing a **for** loop.

Write a **for** loop that goes through every number in the list **[1, 2, 3, 4, 5]** and prints out the square of each number.

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

Let's practice writing a **for** loop.

Write a **for** loop that goes through every number in the list **[1, 2, 3, 4, 5]** and prints out the square of each number.

Write the previous **for** loop as a function that takes in any list of numbers and prints out each number's square.

Data structures

string indexing

list

set

dict

methods

↔ None

↔ mutable vs. immutable

tuples

loops

Let's practice writing a **for** loop.

Write a **for** loop that goes through every number in the list **[1, 2, 3, 4, 5]** and prints out the square of each number.

Write the previous **for** loop as a function that takes in any list of numbers and prints out each number's square.

Write the previous **for** loop function that does the same thing, but this time, it should also take in an exponent.

Homework

<https://journ233.github.io>