Lecture 2: Data Structures

Instructor: Jordan Schwartz

Slides adapted from Tony Liu

Logistics

- HW0 was due today at 9:45am (but you have a 24 hour grace period)
- HW1 is due two weeks from today
  - It covers last week's, this week's, and next week's lectures
- There's a post on Ed that tells you how to view (and ensure you get points for) attendance credit.

Grading - update

6 Homeworks: 55%

Final Project: 35% →

- Checkpoint: 5%
- Final Presentation: 15%
- Final Coding Deliverable: 15%

Attendance and Participation: 10%

One unexcused absence will be granted no questions asked.
Excused absences will be given for unavoidable conflicts, e.g. a job interview, illness.
There will be opportunities for extra credit throughout the course!

Official AI Usage Policy

While we understand that it's pretty impossible to avoid AI these days (looking at you Google search AI lol), and it's often times even harder to resist temptation... PLEASE TRY NOT TO USE AI TO COMPLETE YOUR ASSIGNMENTS!! The goal of this class is for YOU to learn Python. Chat GPT already knows it. It doesn't need the practice. And yes, I'm going to be quite transparent with you, I'm aware that much of what is assigned in this course could be completed using an LLM, there's just not much I can do to prevent you from using it other than appeal to why you are taking this course. If your goal is to learn how to apply an LLM to Python problems, then I'm not going to stop you, but I would like you to think about why that's your goal and if it will really benefit you in the long run. Don't waste opportunities to learn. So TLDR: I can't technically stop you from using an LLM to complete assignments, but I wish I could because it would promote better learning. Other than assignments and practice problems, I have no problems with using an LLM for learning support, such as searching the Python documentation.

(quick) Introductions pt. 2!

- Name
- Pronouns
- Is a taco a sandwich?

Questions and follow ups from last time?

Week 2 - today will be fast, buckle up

- Lists
- Tuples
- Sets
- Dictionaries
- Strings
- Sequences and iteration
- Comprehensions
- Argument passing
- is vs ==
- Positional and keyword arguments

Lists

- Ordered collections of any primitive or type of object
- Indicated by `[ ]`
- Mutable: the values in the object can change without changing the reference pointer to the object

[code demo]

Important list operations

- []
- list()
- .append(val)
- .extend(lst)
- .remove(val)
- .pop() # returns the last value in the list and removes it
- .index(val) # returns the index of the first occurrence of that value
- .sort()

Check in

```
>>> l = [3, 6, “hello”, [“world”], 9,
     “12”, [[“!”]], 15]
>>> l[Box S7 : Box S8 : Box S9]
[6, ['world'], '12']
>>> l[5][1]
Box S10
```

Range

A specific type of object that creates a **sequence** of numbers from the start number to the stop number by the skip number. (same rules as slicing)

```
>>> range(10)  # == range(0, 10, 1)
range(0, 10)
>>> for i in range(10):
...     print(i)
0
1
2
3
4
5
6
7
8
9
```

Tuples

- Ordered collections of any primitive or type of object
- Indicated by ( )
- Immutable: values in the object cannot be updated or changed

[code demo]

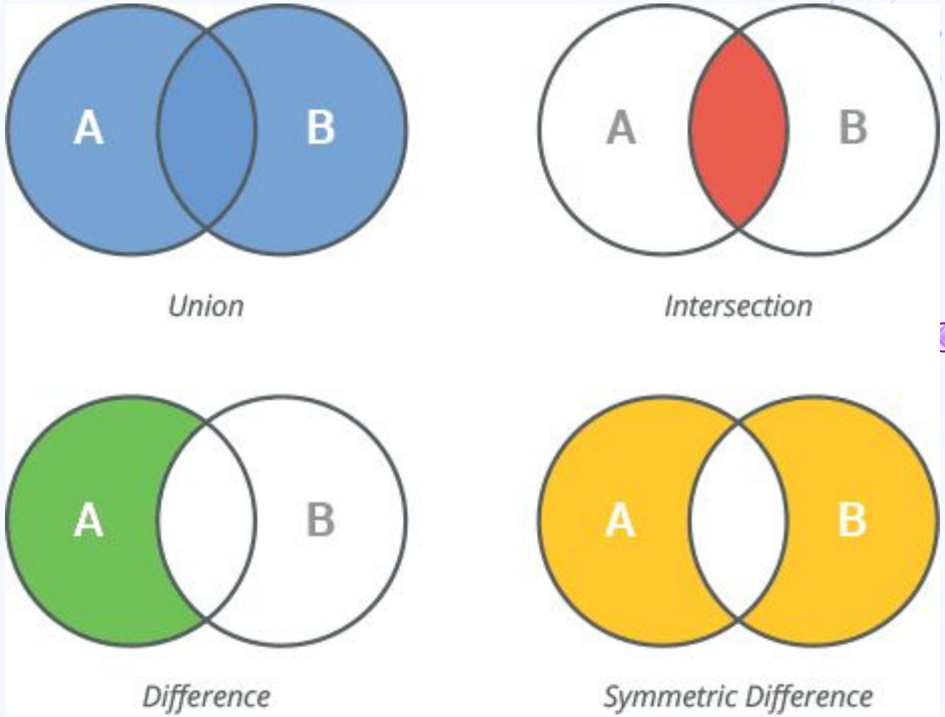
Sets

- **Unordered** collection of **unique** elements
- Initialized with `{ }` or `set()`
- `set_name.`**`add(val)`**
- `set_name.`**`remove(val)`**
- Note: you cannot initialize a new, empty set with `{}` because it will be recognized as a dictionary

[Code demo]

Set Operations

- Union:
  - set1.union(set2)
  - set1 | set2
- Intersection:
  - set1.intersection(set2)
  - set1 & set2
- Difference:
  - set1.difference(set2)
  - set1 - set2
- Sym. Diff.:
  - set1.symmetric_difference(set2)
  - set1 ^ set2

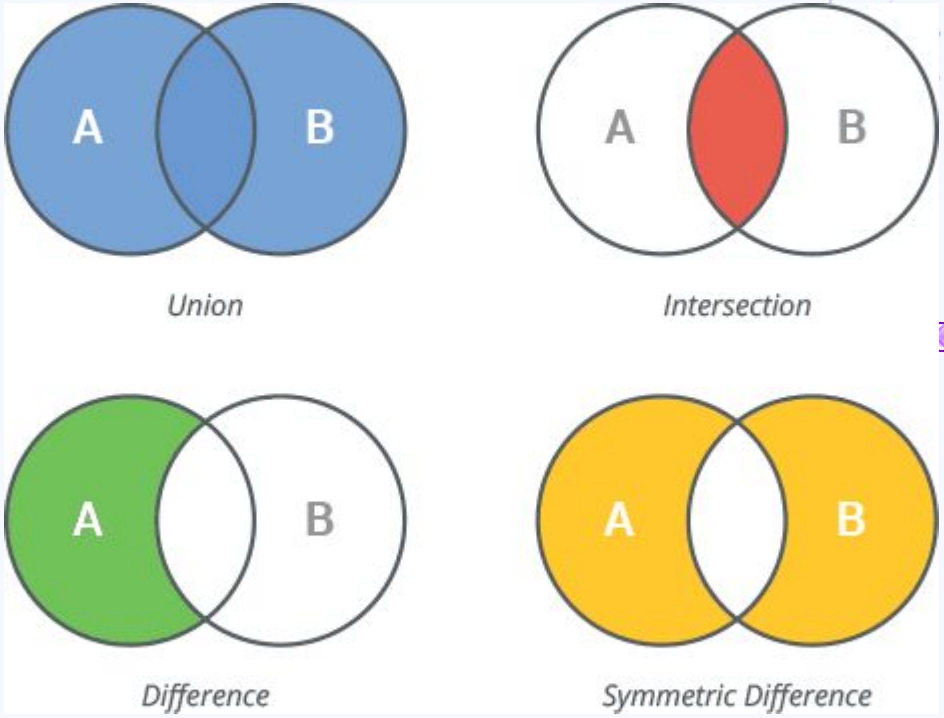


Check in - C12

s = {1, 2, 3, 4}
t = {3, 4, 5, 6}

Find the:

Union
Intersection
Difference → s - t
Symmetric difference



Check in - C12

s = {1, 2, 3, 4}
t = {3, 4, 5, 6}

Find the:

Union:
{1, 2, 3, 4, 5, 6}
Intersection:
{3, 4}

Difference → s - t:
{1, 2}
Symmetric difference:
{1, 2, 5, 6}

Dictionaries

- Mutable key-value mappings
- `{k: v}`
- Keys must be **immutable** (cannot be lists, sets, dicts)

[code demo]

Strings

- Immutable list of characters
- Can index and slice
- Can iterate over
- Marked by “ ” or ‘ ’

Quick aside: formatting strings

```
>>> var = 2
>>> f"string, {}, {var} and {var + 3}"
'string, , 2 and 5'

--------------------------------------------

>>> "string, {}, {var} and {var + 3}".format("A",
      var=2)
'string, A, 2 and 5'
```

Sequences and Iteration

- All sequences can be iterated over
- This means we go through each one of the elements in the sequence, one at a time, if the sequence is ordered, then in that order

Check in – C14

Using the techniques we've seen so far, write a function that takes in some number *n* and returns *True* if *n* is prime and *False* otherwise

(hints: %, range, loops, conditionals)

Comprehensions

Comprehensions

- A loop in one line!
- **Returns a new list**
- [expression for value in sequence optional if condition]

```
squares = []
for i in range(10):
    squares.append(i ** 2)
```

```
squares = [i ** 2 for i in range(10)]
```

```
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

References and Arguments

Argument Passing

C and Java use **pass by value**
When a…

- **Primitive** is passed: a copy of the actual value is made for the method’s use. Manipulation of that parameter does NOT affect the original variable in the calling environment
- **Object** is passed: the variable that is passed in as a parameter contains a *reference* to the object on the heap. This *reference* is then copied for the method

Argument Passing

C and Java use **pass by value**

- Objects:
  - If the parameter in the method modifies the original object on the heap using the *reference* passed in, then the original object is modified.
  - If the method tries to assign a new object to the formal parameter variable, it only changes the local copy of the reference. The original *reference* variable in the calling method remains unchanged and still points to the initial object.

Argument Passing

Python is **pass by assignment**

- Immutable objects are treated the same as **pass by value**
  - Copies of the original value are made
  - Modification inside the method do not change the outer variable
- Mutable objects are treated the same as **pass by reference**
  - A reference to the original object are passed in
  - Modifications to the object inside the method **do affect** the original object

Let's Visualize

[Example of primitive]

[Example with mutable list]

[Swapping example]

Equality

Equality: `is` vs `==`

is

- Checks if two objects are *the same object* or *have the same reference pointer*
- Will always return the obvious value for primitives, but is meant for objects
- **identity**

[Code demo]

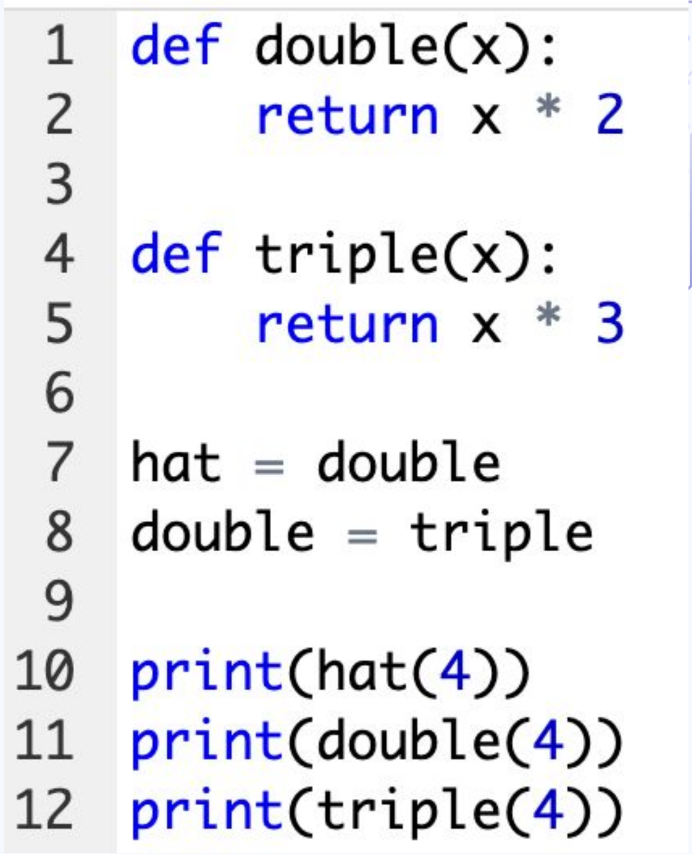
==

- Checks whether two objects are “equal” as defined by the object type
- Checks if two objects have the same values
- Meant for primitives OR comparing the inner values of an object
- **equality**

Check in – L11

What is the output of running this code?

What line could you add such that triple multiplies its input by 2 and returns that value?



```
def double(x):
    return x * 2

def triple(x):
    return x * 3

hat = double
double = triple

print(hat(4))
print(double(4))
print(triple(4))
```

Positional and Keyword Arguments

[code demo]

🔑: function arguments can be referred to via position, or by keyword
🔑: can even provide optional arguments and default values

Positional unpacking

* explicitly unpacks a sequence in a function call **or** definition

[code demo]

Keyword unpacking

** explicitly unpacks a dictionary in a function call **or** definition

[code demo]

Check in

```
d = {'a': [2, 3], 'b': [3, 4], 'c': [4, 5]}
```

M1

```
def foo(bar, **barbar):
    for k, v in barbar.items():
        print(bar[k][0] + v[1])
foo(d, **d)
```

M2

```
def foo2(bar, a, b, c):
    for k, v in bar.items():
        if a == k:
            print(a[0] + v[1])
        elif b == k:
            print(b[0] + v[1])
        else:
            print(c[0] + v[1])
foo2(d, **d)
```

M3

```
def foo3(bar, a, b, c):
    for k, v in bar.items():
        if a == v:
            print(a[0] + v[1])
        elif b == v:
            print(b[0] + v[1])
        else:
            print(c[0] + v[1])
foo3(d, **d)
```

M4

```
def foo4(**bar, barbar):
    for k, v in bar:
        print(barbar[k][0] + v[1])
foo4(d, **d)
```

A
```
5
7
9
```

B
```
7
8
9
```

C Error

Thanks!

Do you have any questions?

youremail@freepik.com

+34 654 321 432

yourwebsite.com

slidesgo