Course	
Term	
Week	
Date	
Chapter. Topic	9. Dictionaries and Sets

9.1. Sets

Siva R Jasthi

Computer Science and Cybersecurity

Metropolitan State University

Outline

- Lists
- Tuples

are sequences.

We covered “Lists” and “Tuples” so far.

We will cover “Sets” today.

Lists vs Tuples vs Sets

	Lists	Tuples	Set
Ordered	✓	✓	✗
Indexed	✓	✓	✗
Add or Update items	✓	✗	✓
Can contain duplicates	✓	✓	✗
Uses	Square Brackets	Round Brackets	Curly Brackets
	[ ]	( )	{ }
Constructor	list() []	tuple() ()	set()

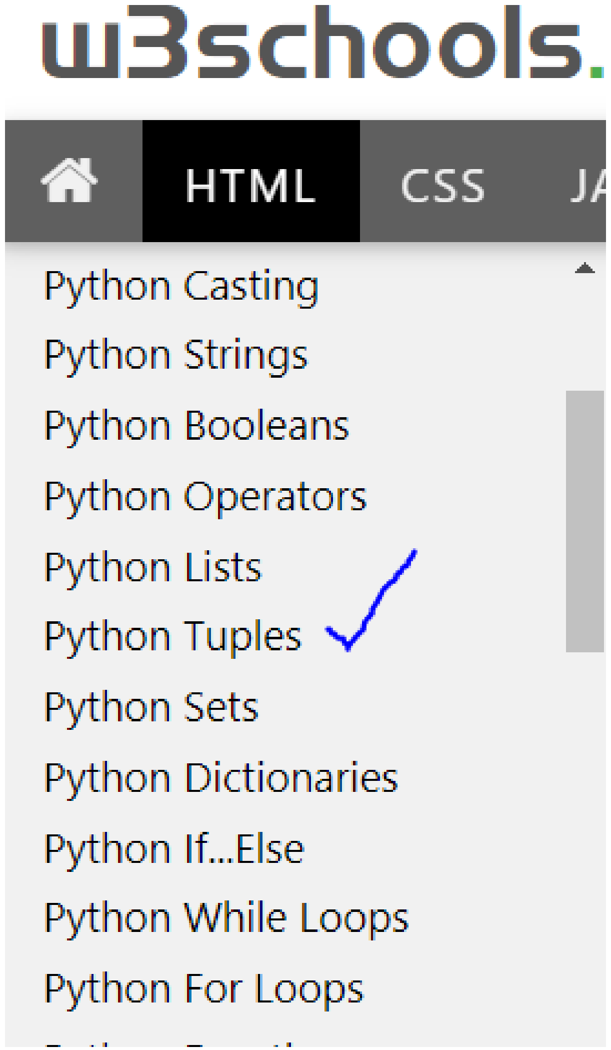
Sets

A **set** is a collection which is both **unordered** and **unindexed**.

Set can not contain duplicates.

Sets: An introduction
https://www.w3schools.com/python/python_sets.asp

Sets are written with curly brackets.

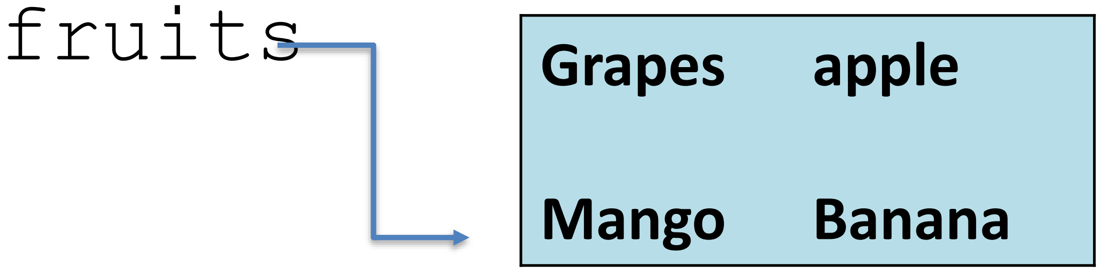


An example of a set

We use CURLY Brackets to indicate a Set.

```
fruits = {"apple", "banana", "grapes", "mango"}
```

You can visualize the set as follows:

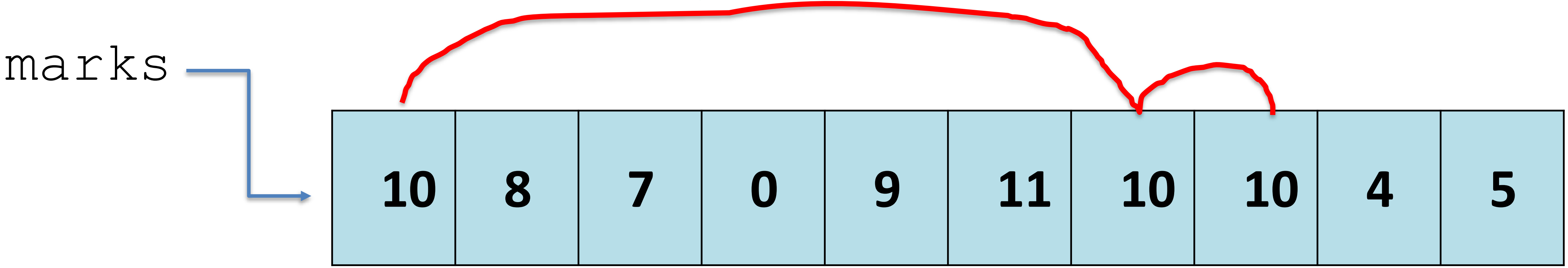


fruits

Grapes apple

Mango Banana

sets can not contain duplicates

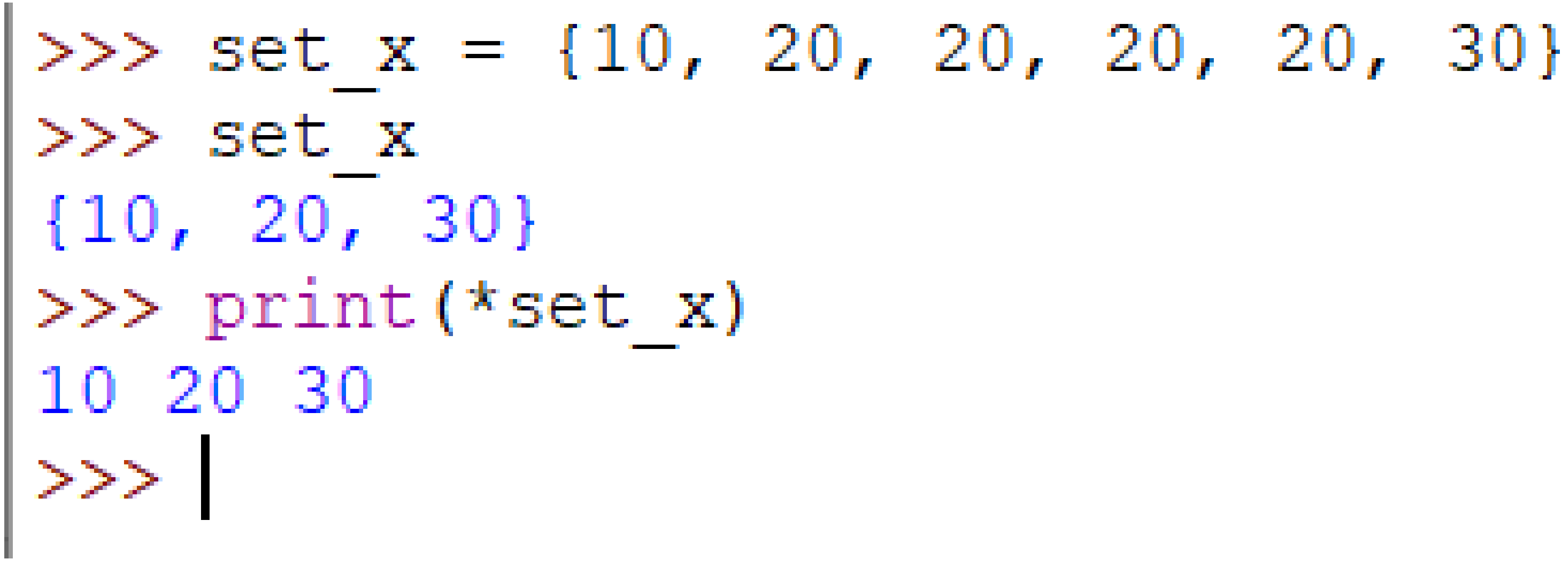


marks

10	8	7	0	9	11	10	10	4	5

We can not represent this collection as a “set”.

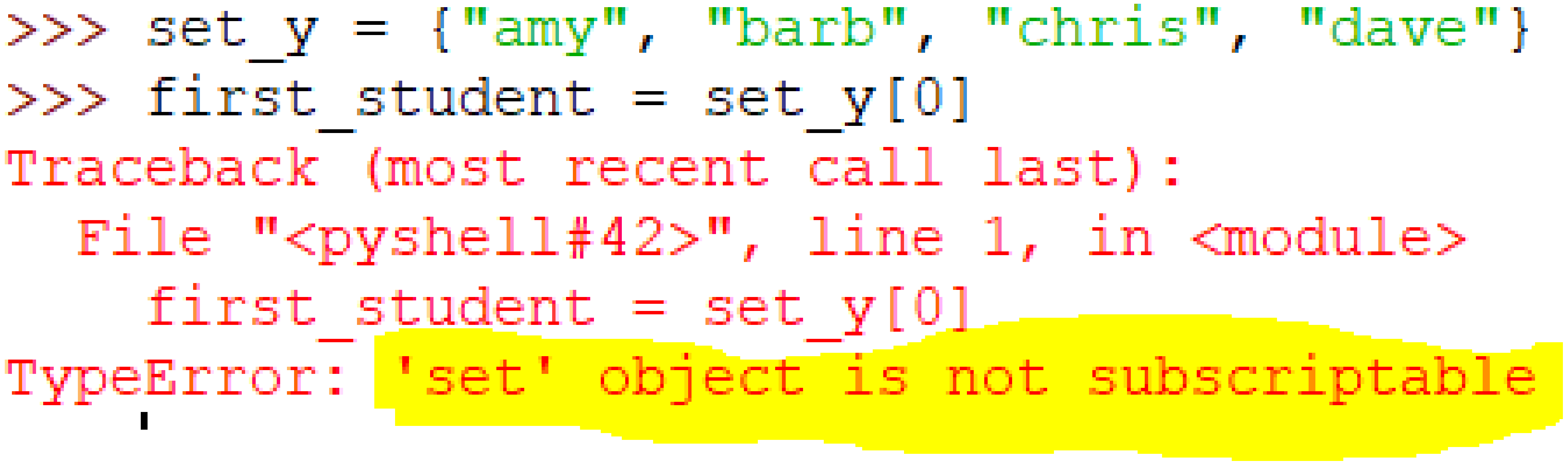
Duplicates will be automatically removed



```
>>> set_x = {10, 20, 20, 20, 20, 30}
>>> set_x
{10, 20, 30}
>>> print(*set_x)
10 20 30
>>> 
```

You can not reference items by index

Because sets are “unordered”.

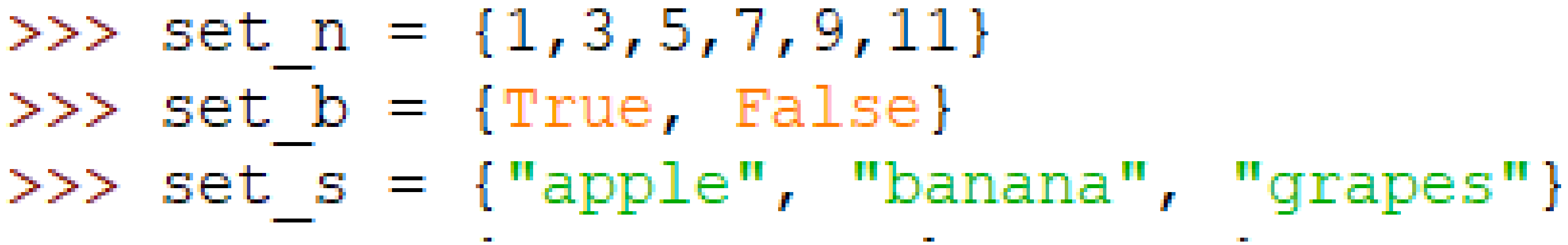


```
>>> set_y = {"amy", "barb", "chris", "dave"}
>>> first_student = set_y[0]
Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    first_student = set_y[0]
TypeError: 'set' object is not subscriptable
```

Sets support all “immutable” data types

Set of numbers, strings, booleans, objects, etc.

```
set_n = {1,3,5,7,9,11}

set_b = {True, False}

set_s = {"apple", "banana", "grapes"}
```



```
>>> set_n = {1,3,5,7,9,11}
>>> set_b = {True, False}
>>> set_s = {"apple", "banana", "grapes"}
```

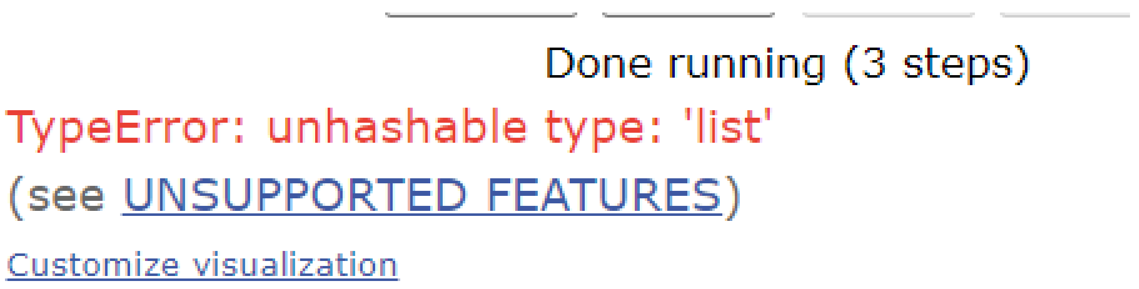
Here is a scenario: Set of Lists

Set of lists: Is this possible?

A = [1,2,3]

B = [2,3,4]

set_x = { A, B }



Done running (3 steps)

```
TypeError: unhashable type: 'list'
(see UNSUPPORTED FEATURES)
```

Assume that it is possible

set_x { [1,2,3], [2,3,4]}

B.insert(0,1)
B.remove(4)
B → 1,2,3

Set_x → [1,2,3], [1,2,3]
→ this violates the SET rule.

So, only the items that can not be changed
(in other words, the items that are immutable)
Can be added to the sets.

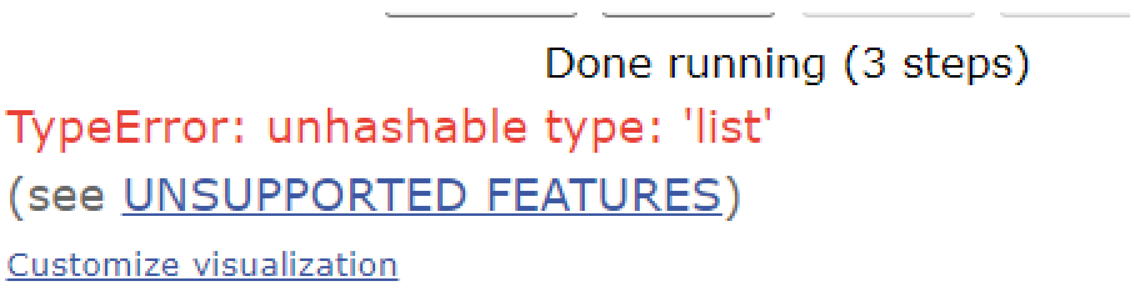
Here is a scenario: Set of Tuples

Set of lists: Is this possible?

A = (1,2,3)

B = (2,3,4)

set_x = { A, B }



```
Done running (3 steps)
TypeError: unhashable type: 'list'
(see UNSUPPORTED FEATURES)
```

Assume that it is possible

set_x { [1,2,3], [2,3,4]}

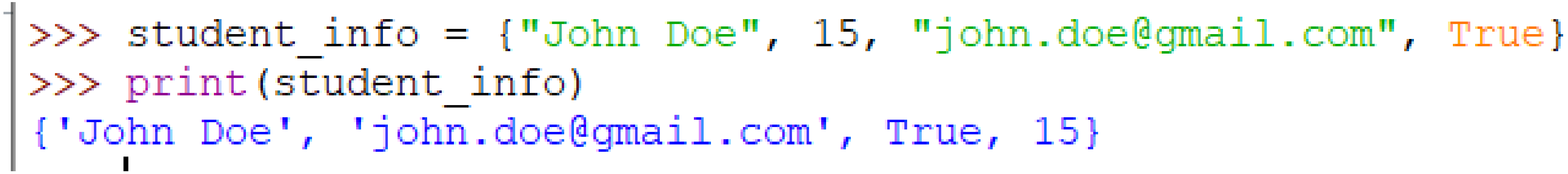
B.insert(0,1)
B.remove(4)
B → 1,2,3

Set_x → [1,2,3], [1,2,3]
→ this violates the SET rule.

So, only the items that can not be changed
(in other words, the items that are immutable)
Can be added to the sets.

A single set can contain multiple data types

```
// student name, age, email and is_student_active?
student_info = {"John Doe", 15, "john.doe@gmail.com, True}
```

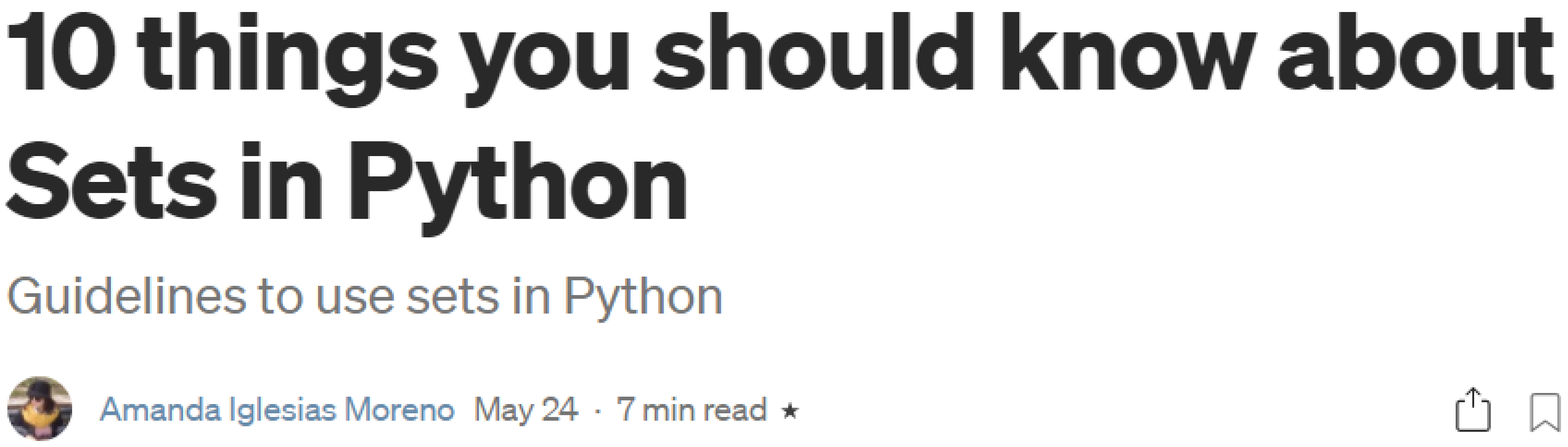


```
>>> student_info = {"John Doe", 15, "john.doe@gmail.com", True}
>>> print(student_info)
{'John Doe', 'john.doe@gmail.com', True, 15}
```

Please note that the order is not guaranteed in sets.

When to use sets?

You don’t want duplicate items.
You don’t care about order.
And you want to perform set operations like – union, intersection, difference, symmetric difference



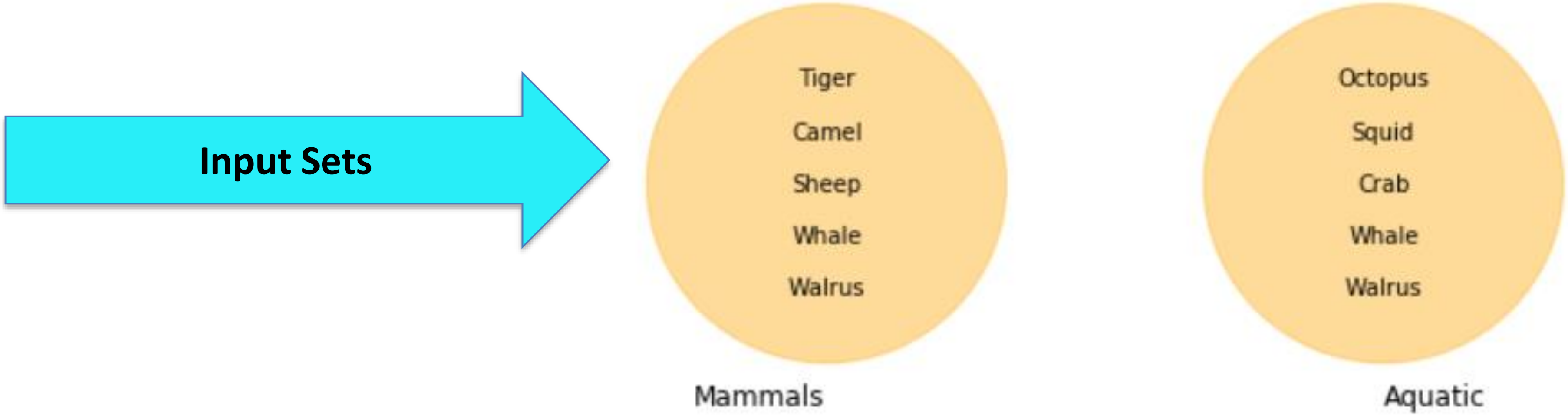
10 things you should know about Sets in Python

Guidelines to use sets in Python

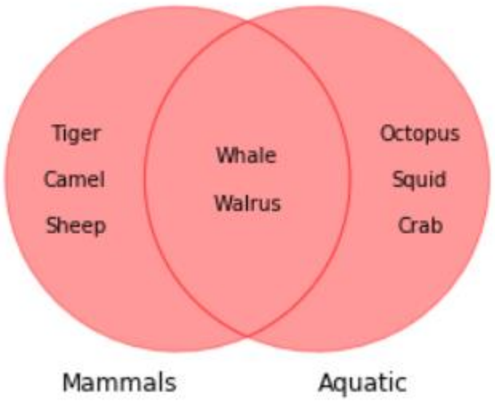
Amanda Iglesias Moreno May 24 · 7 min read ★

https://towardsdatascience.com/10-things-you-should-know-about-sets-in-python-9902828c0e80

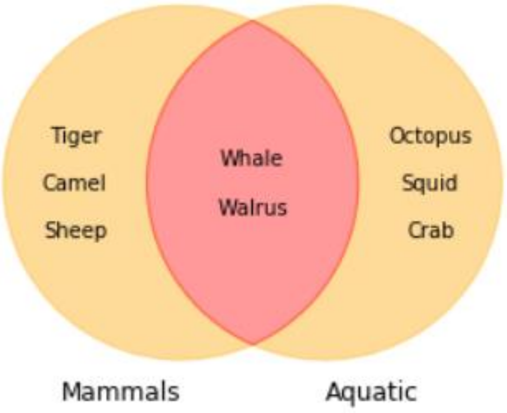
When to use sets?



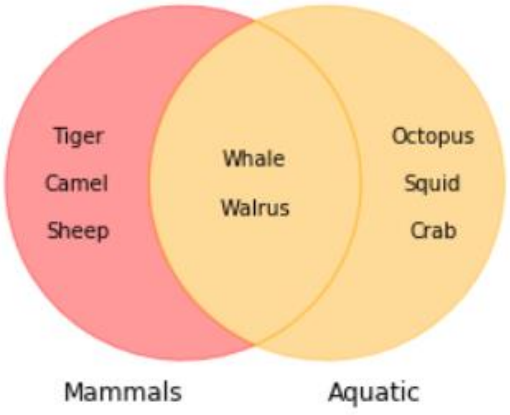
Union



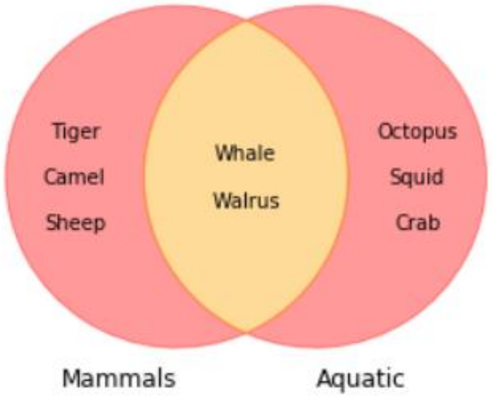
Intersection



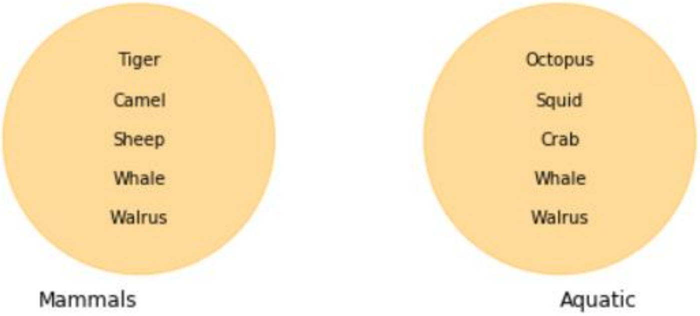
Difference (mammals-aquatic)



Symmetric Difference



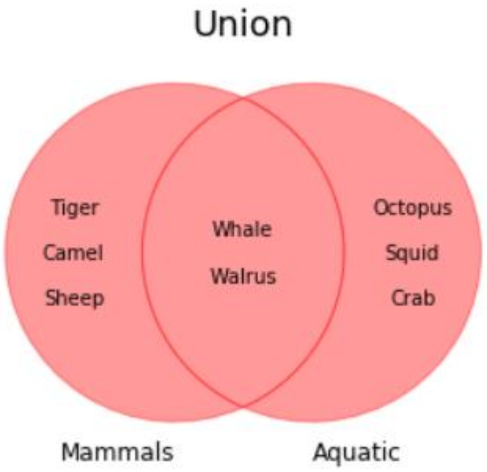
Set Operations: Union



a.union(b)

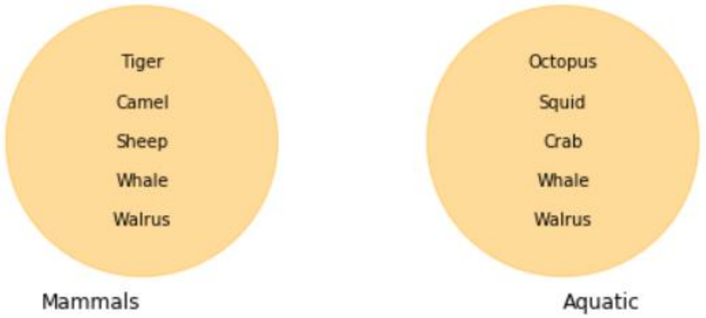
| operator

The **union** of two sets **A** and **B** is the set containing the elements that are in **A**, **B**, or **both**, and is denoted by **A U B**.



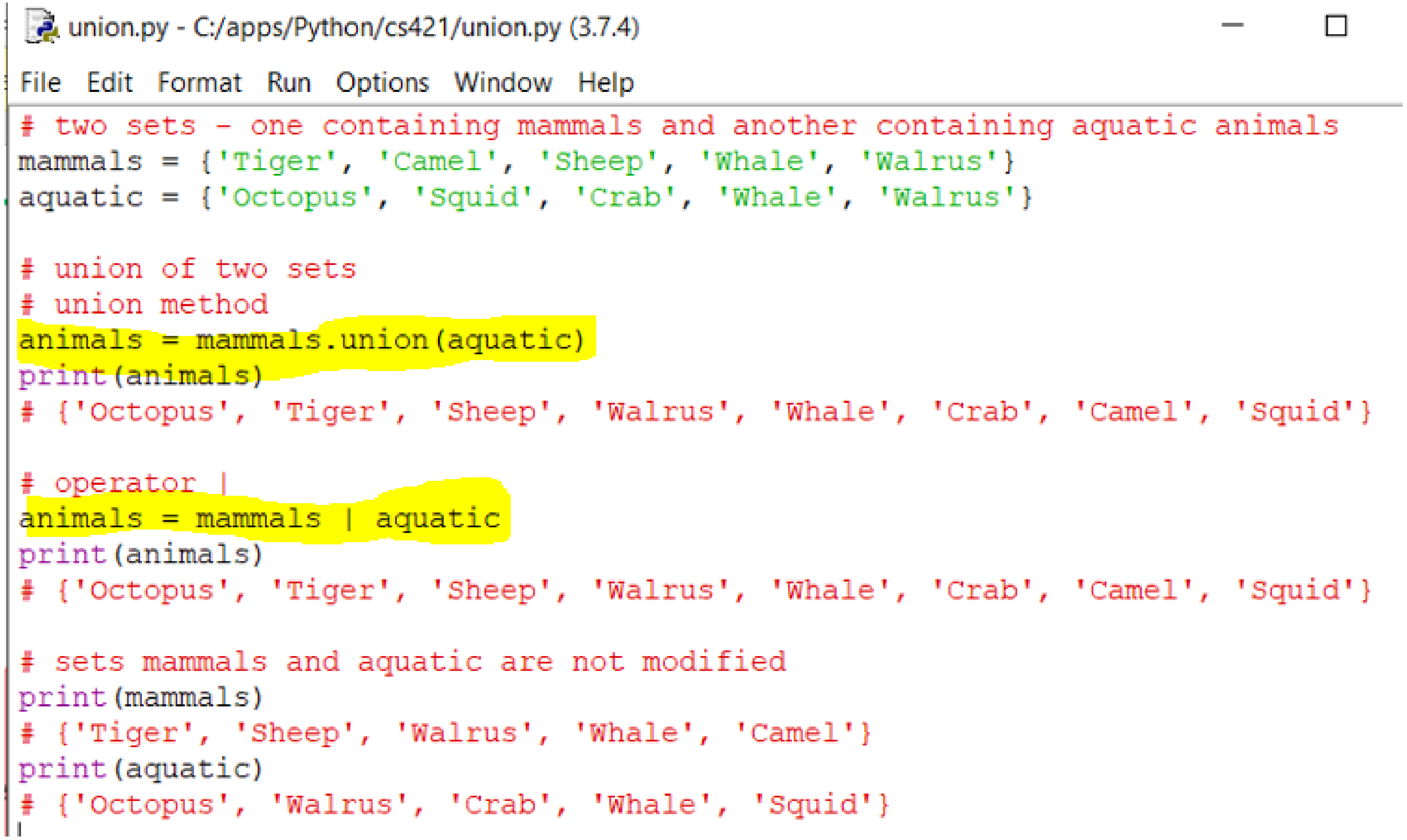
See union.py

Set Operations: Union

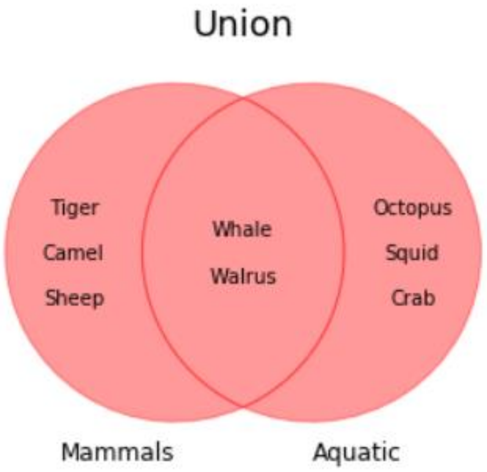


a.union(b)

| operator

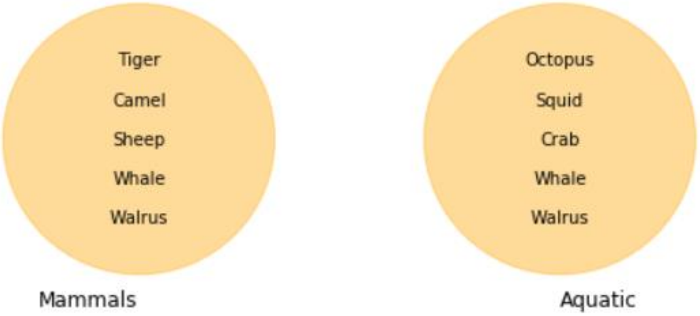

union.py - C:/apps/Python/cs421/union.py (3.7.4)
File Edit Format Run Options Window Help

```
# two sets - one containing mammals and another containing aquatic animals
mammals = {'Tiger', 'Camel', 'Sheep', 'Whale', 'Walrus'}
aquatic = {'Octopus', 'Squid', 'Crab', 'Whale', 'Walrus'}

# union of two sets
# union method
animals = mammals.union(aquatic)
print(animals)
# {'Octopus', 'Tiger', 'Sheep', 'Walrus', 'Whale', 'Crab', 'Camel', 'Squid'}

# operator |
animals = mammals | aquatic
print(animals)
# {'Octopus', 'Tiger', 'Sheep', 'Walrus', 'Whale', 'Crab', 'Camel', 'Squid'}

# sets mammals and aquatic are not modified
print(mammals)
# {'Tiger', 'Sheep', 'Walrus', 'Whale', 'Camel'}
print(aquatic)
# {'Octopus', 'Walrus', 'Crab', 'Whale', 'Squid'}
```



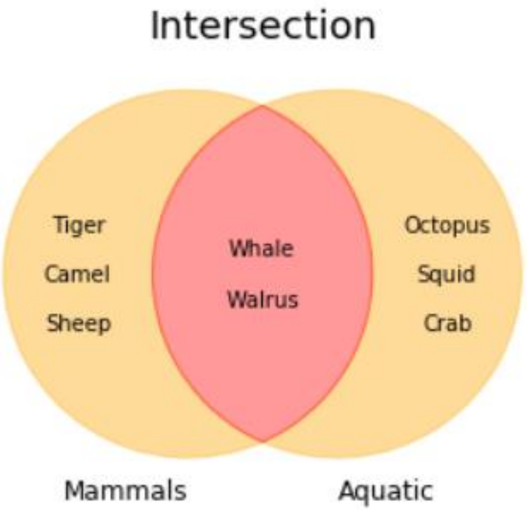
See union.py

Set Operations: Intersection



a.intersection (b)

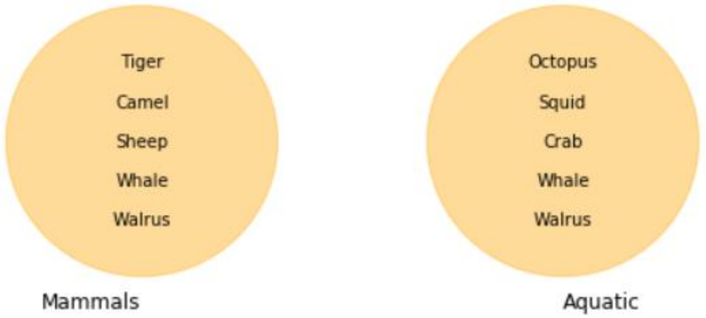
& operator



The **intersection** of two sets **A** and **B** is the set containing the elements that are common to both sets and is denoted by **A ∩ B**.

See intersection.py

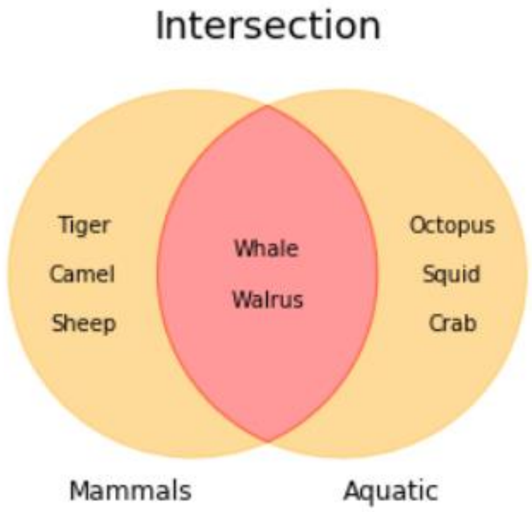
Set Operations: Intersection

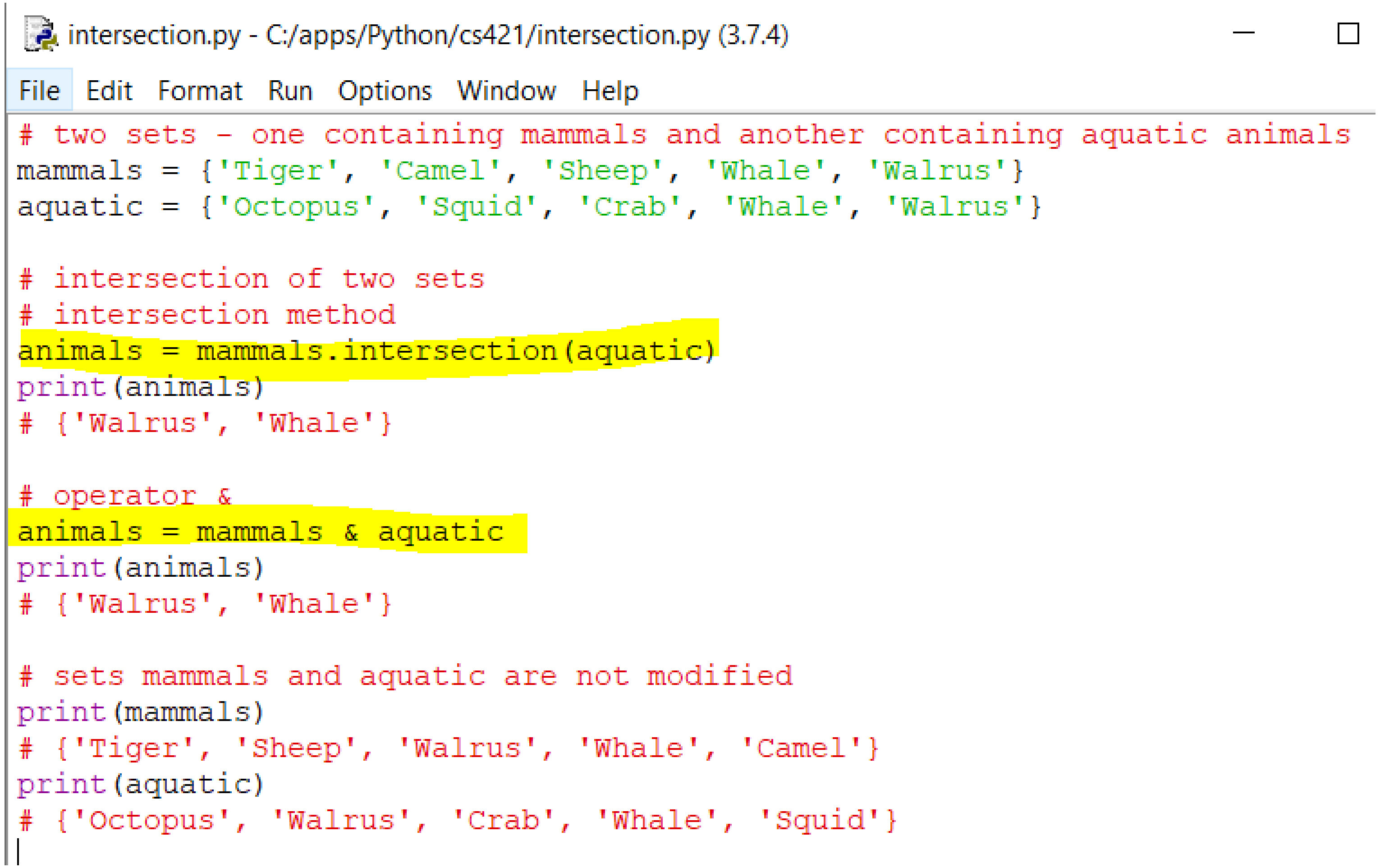


PYTHON PROGRAMMING
by SIVA JASTHI

a.intersection (b)

& operator

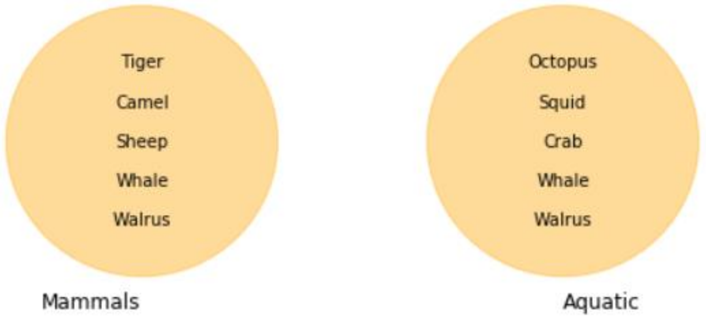



intersection.py - C:/apps/Python/cs421/intersection.py (3.7.4)
File Edit Format Run Options Window Help

```
# two sets - one containing mammals and another containing aquatic animals
mammals = {'Tiger', 'Camel', 'Sheep', 'Whale', 'Walrus'}
aquatic = {'Octopus', 'Squid', 'Crab', 'Whale', 'Walrus'}

# intersection of two sets
# intersection method
animals = mammals.intersection(aquatic)
print(animals)
# {'Walrus', 'Whale'}

# operator &
animals = mammals & aquatic
print(animals)
# {'Walrus', 'Whale'}

# sets mammals and aquatic are not modified
print(mammals)
# {'Tiger', 'Sheep', 'Walrus', 'Whale', 'Camel'}
print(aquatic)
# {'Octopus', 'Walrus', 'Crab', 'Whale', 'Squid'}
```

See union.py

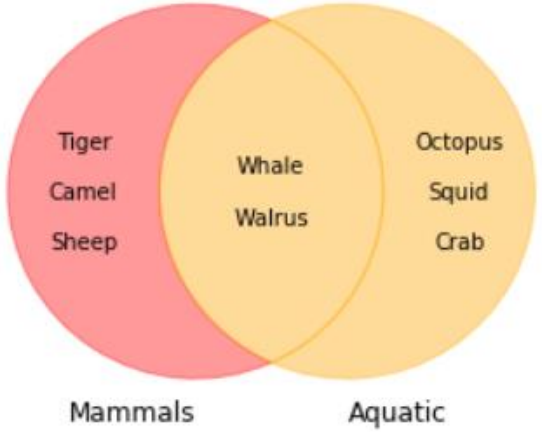
Set Operations: Difference



a.difference (b)

- operator

Difference (mammals-aquatic)



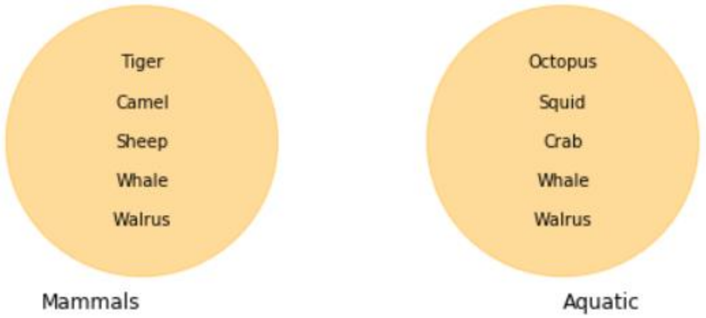
The difference of two sets **A** and **B** is the set of all elements of set **A** that are not contained in set **B** and is denoted by **A-B**.

Note: **A-B** is not same as **B-A**

See difference.py

Set Operations: Difference

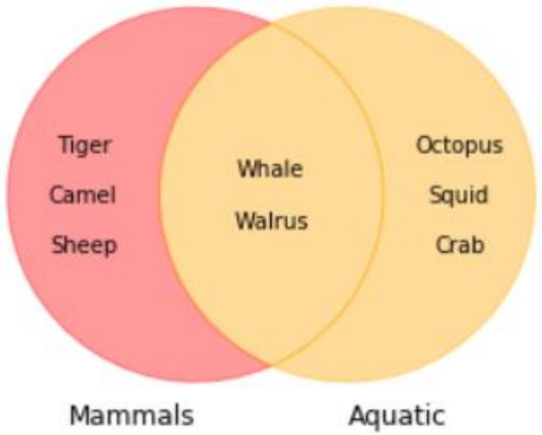
PYTHON PROGRAMMING by SIVA JASTHI

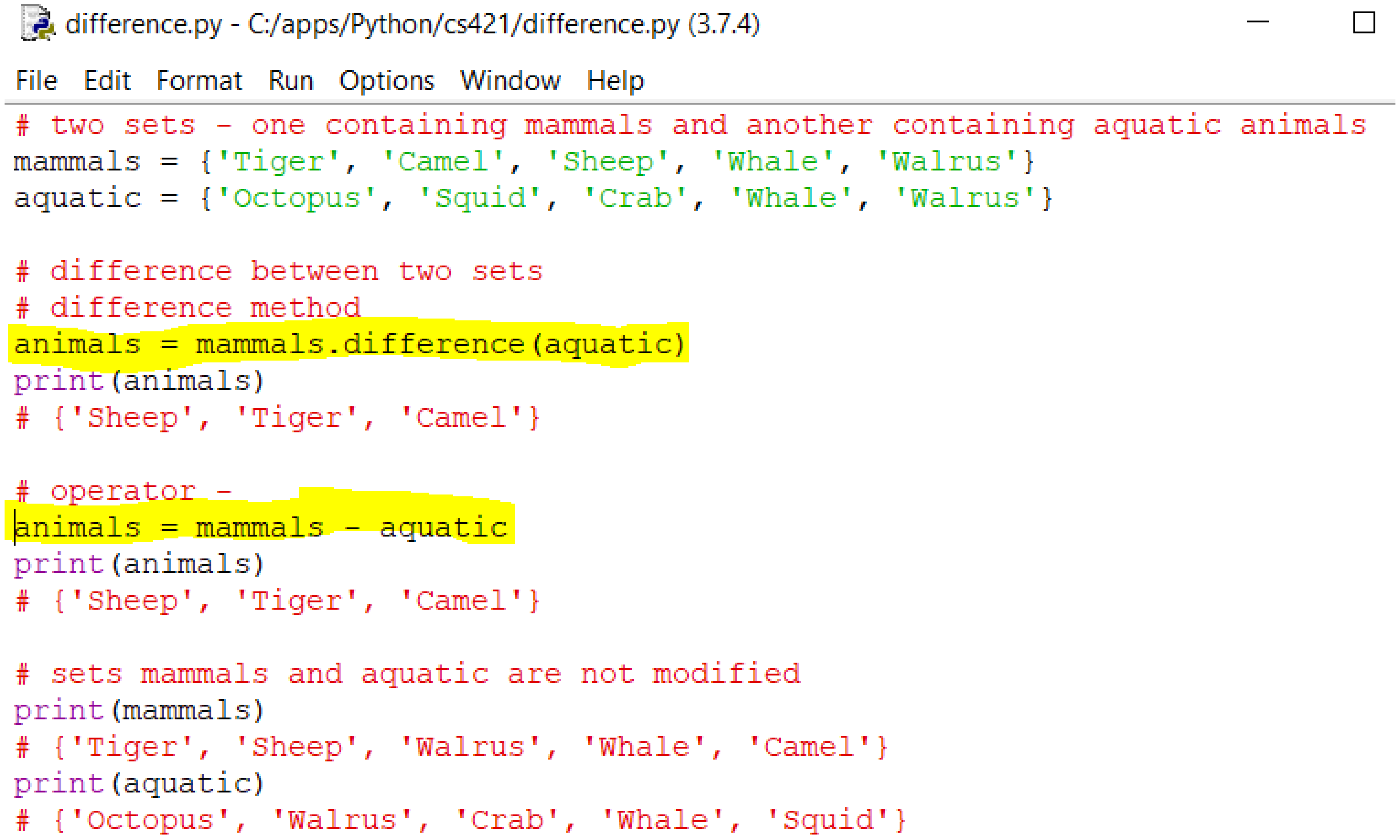


a.difference (b)

- operator

Difference (mammals-aquatic)



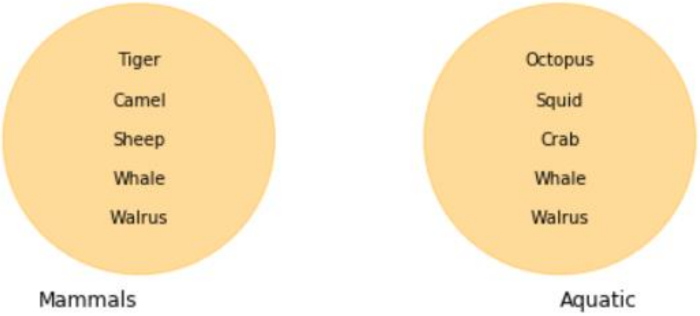


difference.py - C:/apps/Python/cs421/difference.py (3.7.4)
File Edit Format Run Options Window Help

```python
# two sets - one containing mammals and another containing aquatic animals
mammals = {'Tiger', 'Camel', 'Sheep', 'Whale', 'Walrus'}
aquatic = {'Octopus', 'Squid', 'Crab', 'Whale', 'Walrus'}

# difference between two sets
# difference method
animals = mammals.difference(aquatic)
print(animals)
# {'Sheep', 'Tiger', 'Camel'}

# operator -
animals = mammals - aquatic
print(animals)
# {'Sheep', 'Tiger', 'Camel'}

# sets mammals and aquatic are not modified
print(mammals)
# {'Tiger', 'Sheep', 'Walrus', 'Whale', 'Camel'}
print(aquatic)
# {'Octopus', 'Walrus', 'Crab', 'Whale', 'Squid'}
```

See difference.py

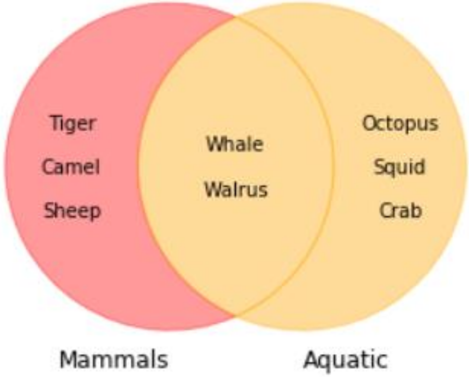
Set Operations: Difference



mammals.difference(aquatic)

mammals - aquatic

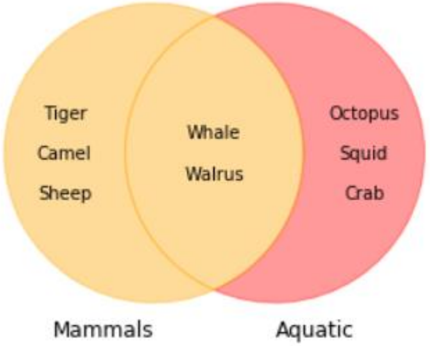
Difference (mammals-aquatic)



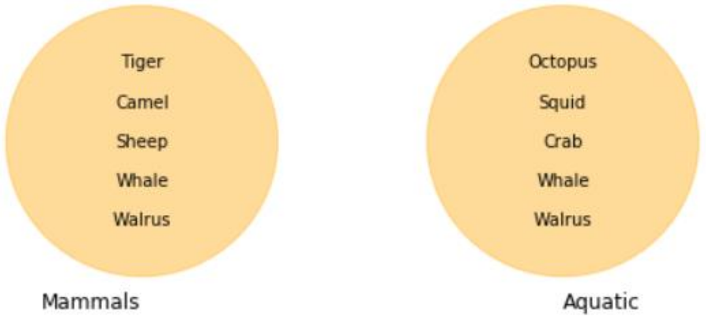
aquatic.difference(mammals)

aquatic - mammals

Difference (aquatic-mammals)

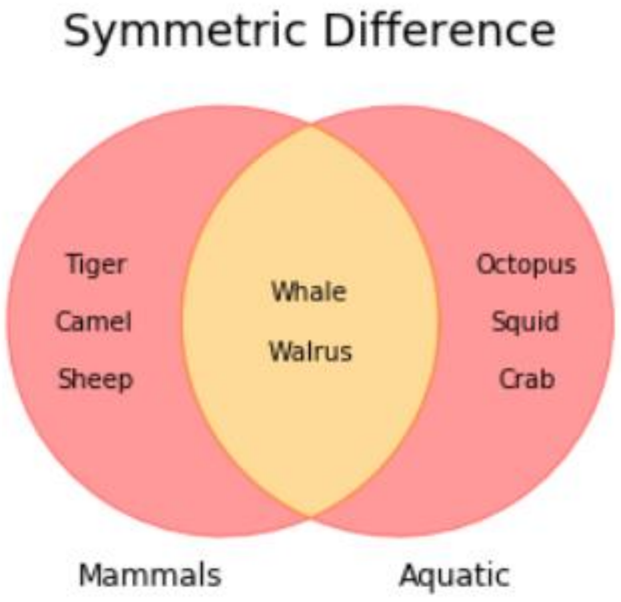


Set Operations: Symmetric Difference



a.symmetric_difference (b)

^ operator

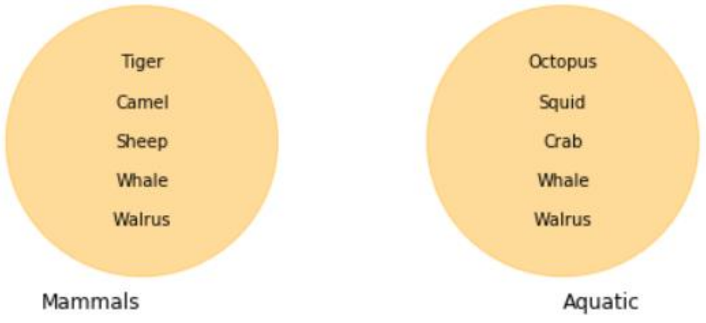


See symmetric_difference.py

The **symmetric difference** of two sets **A** and **B** is the set of elements that are in either of the sets **A** and **B**, but not in both, and is denoted by **A △ B.**

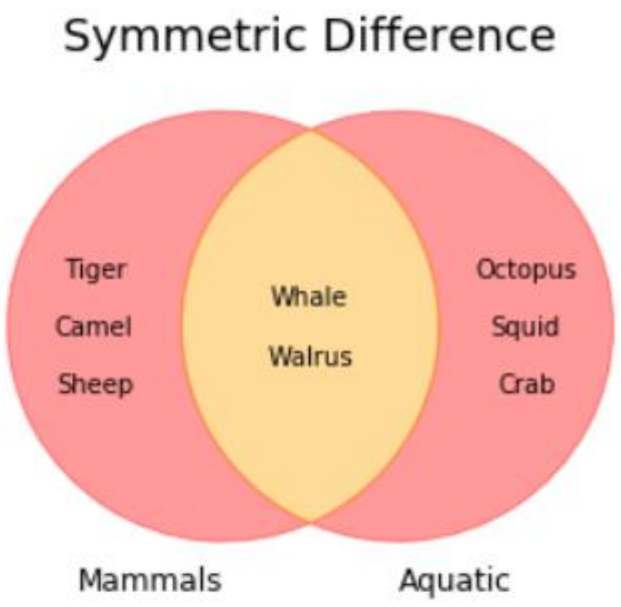
Set Operations: Symmetric Difference

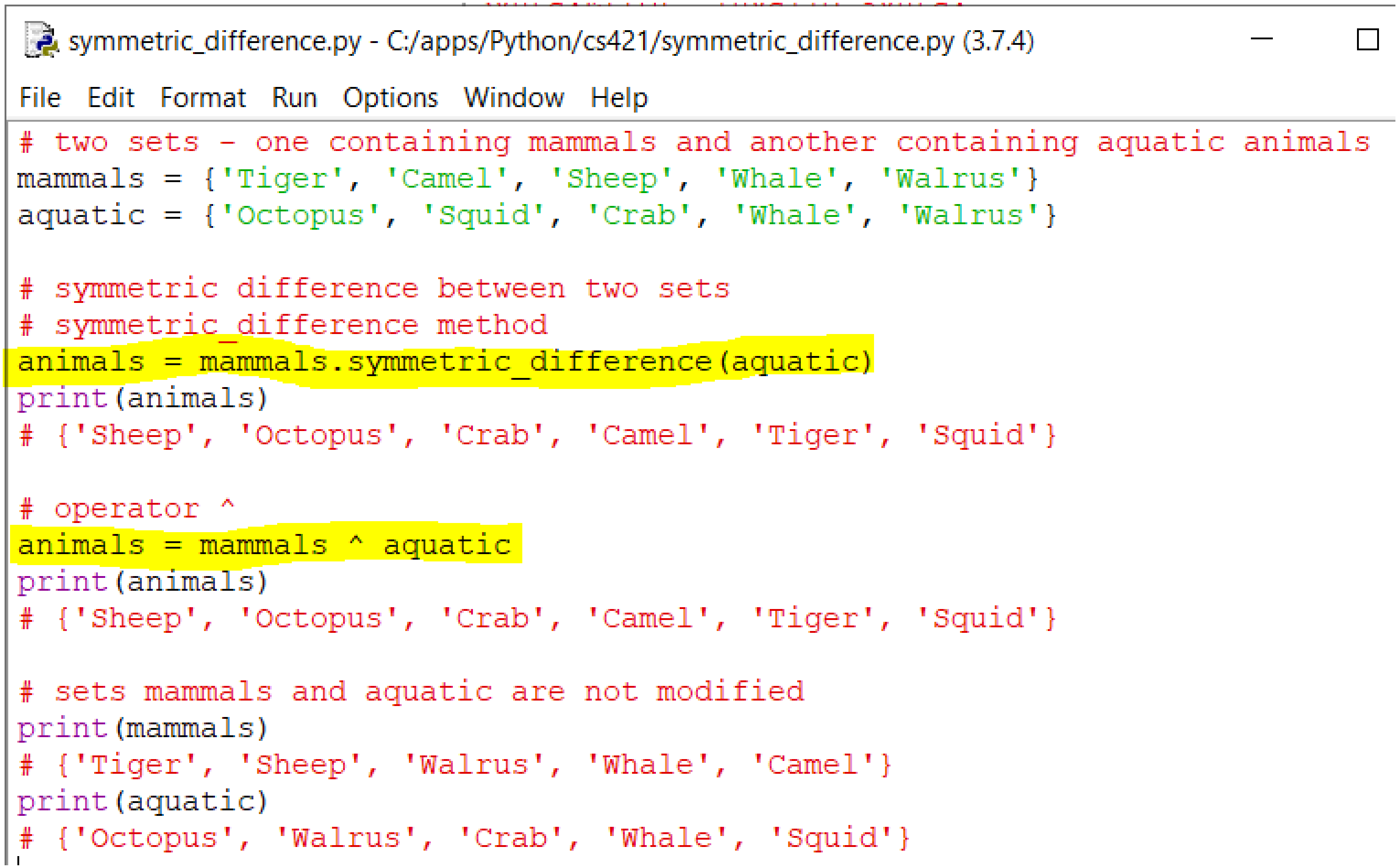
PYTHON PROGRAMMING by SIVA JASTHI



a.symmetric_difference (b)

^ operator



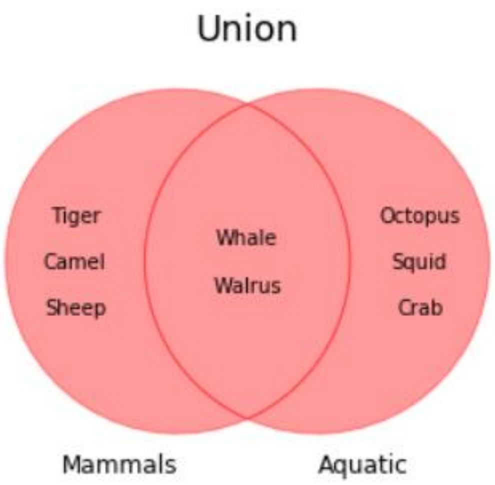

symmetric_difference.py - C:/apps/Python/cs421/symmetric_difference.py (3.7.4)
File Edit Format Run Options Window Help

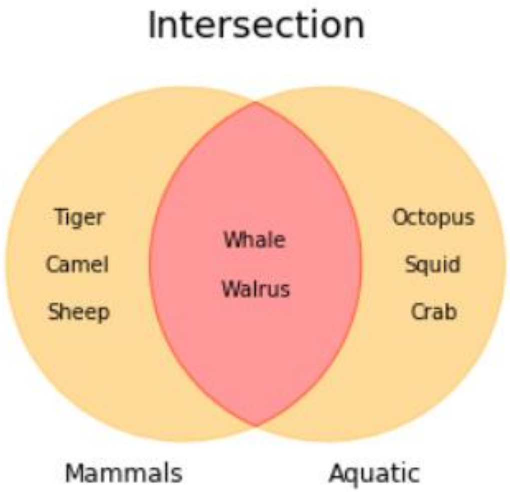
```python
# two sets - one containing mammals and another containing aquatic animals
mammals = {'Tiger', 'Camel', 'Sheep', 'Whale', 'Walrus'}
aquatic = {'Octopus', 'Squid', 'Crab', 'Whale', 'Walrus'}

# symmetric difference between two sets
# symmetric_difference method
animals = mammals.symmetric_difference(aquatic)
print(animals)
# {'Sheep', 'Octopus', 'Crab', 'Camel', 'Tiger', 'Squid'}

# operator ^
animals = mammals ^ aquatic
print(animals)
# {'Sheep', 'Octopus', 'Crab', 'Camel', 'Tiger', 'Squid'}

# sets mammals and aquatic are not modified
print(mammals)
# {'Tiger', 'Sheep', 'Walrus', 'Whale', 'Camel'}
print(aquatic)
# {'Octopus', 'Walrus', 'Crab', 'Whale', 'Squid'}
```

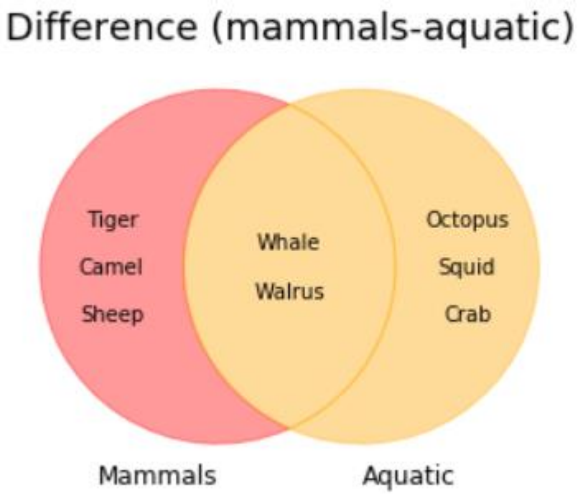
See symmetric_difference.py

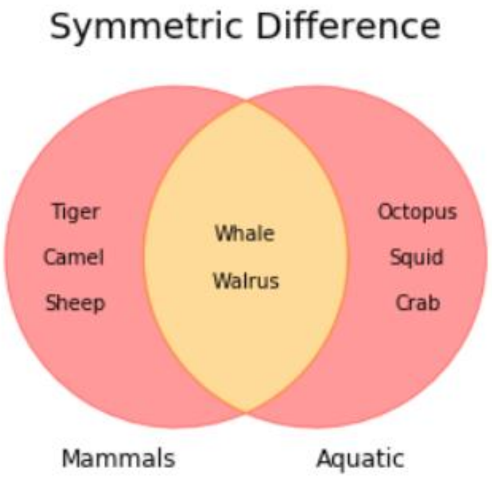
Set Operations: Summary

Tiger
Camel
Sheep
Whale
Walrus
Mammals
Octopus
Squid
Crab
Whale
Walrus
Aquatic


Union


Intersection


Difference (mammals-aquatic)


Symmetric Difference

union
|

intersection
&

difference
-

symmetric_
difference
^

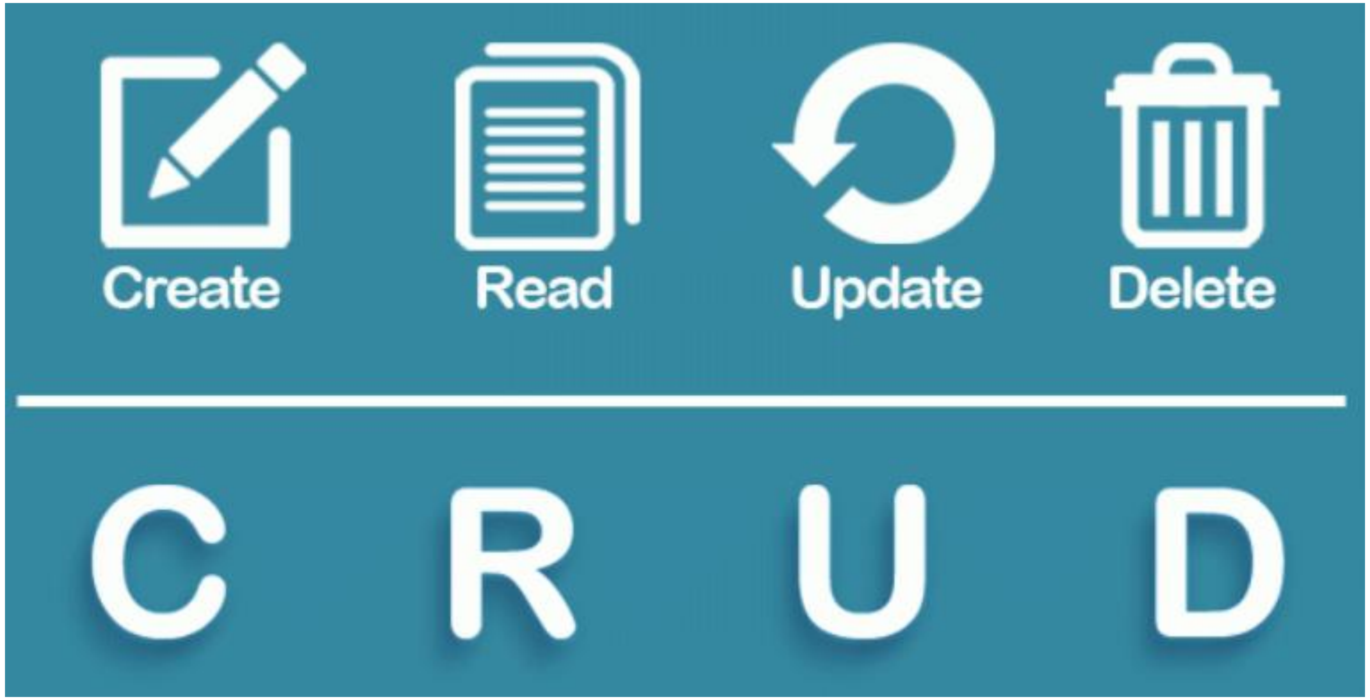
CRUD of Sets

C – Create (Add, Insert, Append, Extend, Copy)

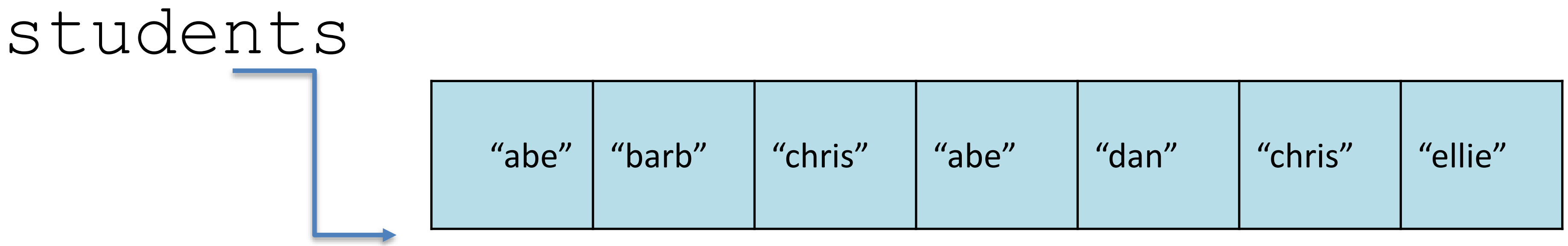
R – Read (Query, Traversal, Find, Search)

U – Update (Modify, Change, Edit)

D – Delete (Remove, Empty)

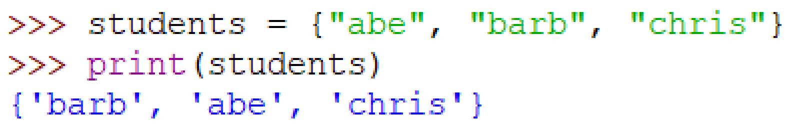


Creating a set

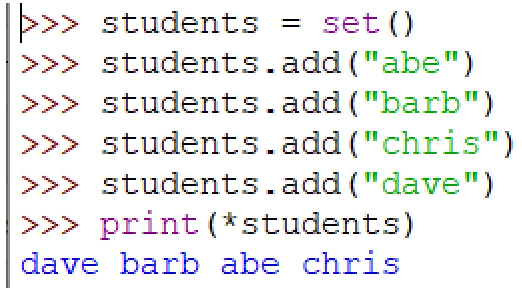

students

“abe”	“barb”	“chris”	“abe”	“dan”	“chris”	“ellie”

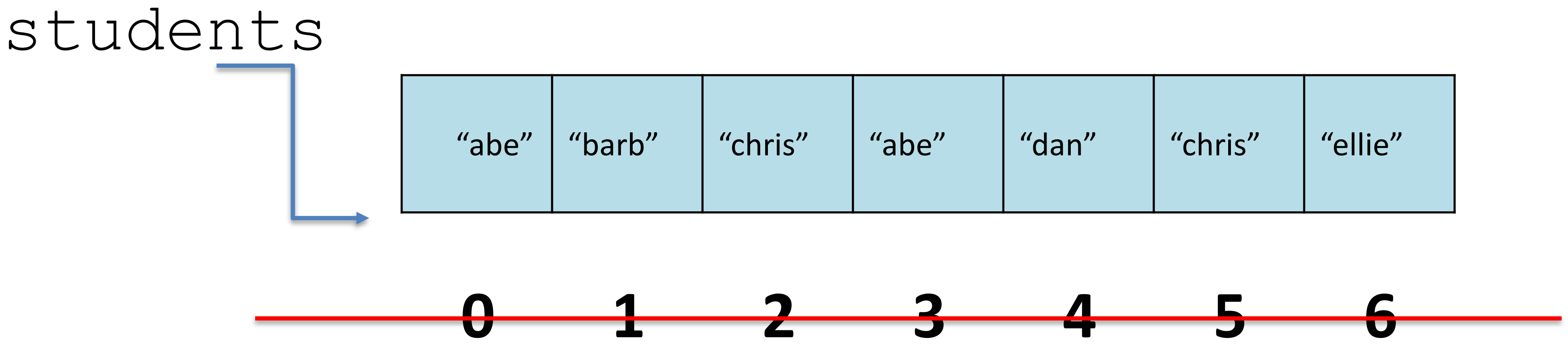
# set of students
students = {"abe", "barb", "chris“}


```
>>> students = {"abe", "barb", "chris"}
>>> print(students)
{'barb', 'abe', 'chris'}
```

#create an empty set
# then start adding the students
students = set( )
students.add(“abe”)
Studensts.add(“barb”)


```
>>> students = set()
>>> students.add("abe")
>>> students.add("barb")
>>> students.add("chris")
>>> students.add("dave")
>>> print(*students)
dave barb abe chris
```

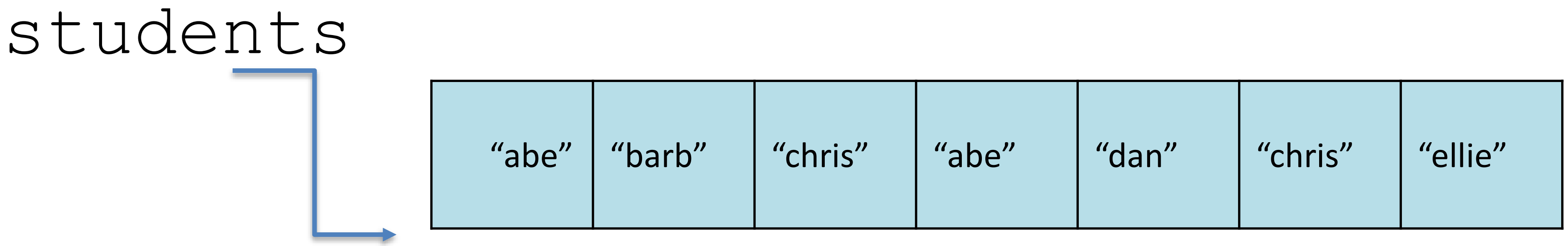
Reading/Accessing an item from a set



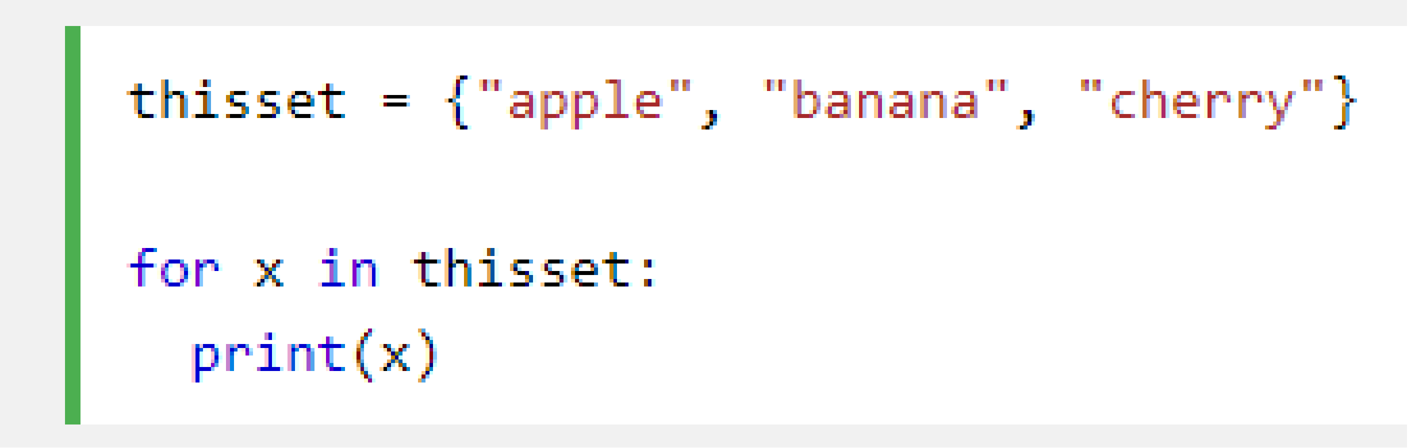
We can not use **subscript** notation to access an element.

NOT VALID

Reading all items / Iterating the set

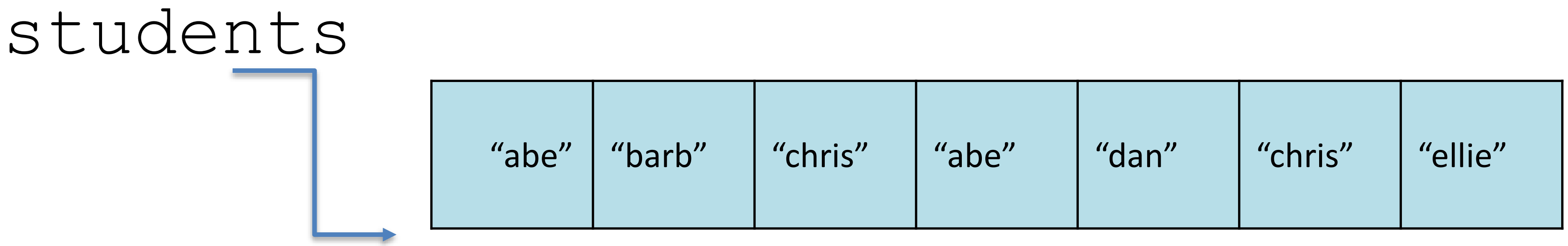


There are many ways to iterate over a set
https://www.geeksforgeeks.org/iterate-over-a-set-in-python/



```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

Checking for the memberships? in

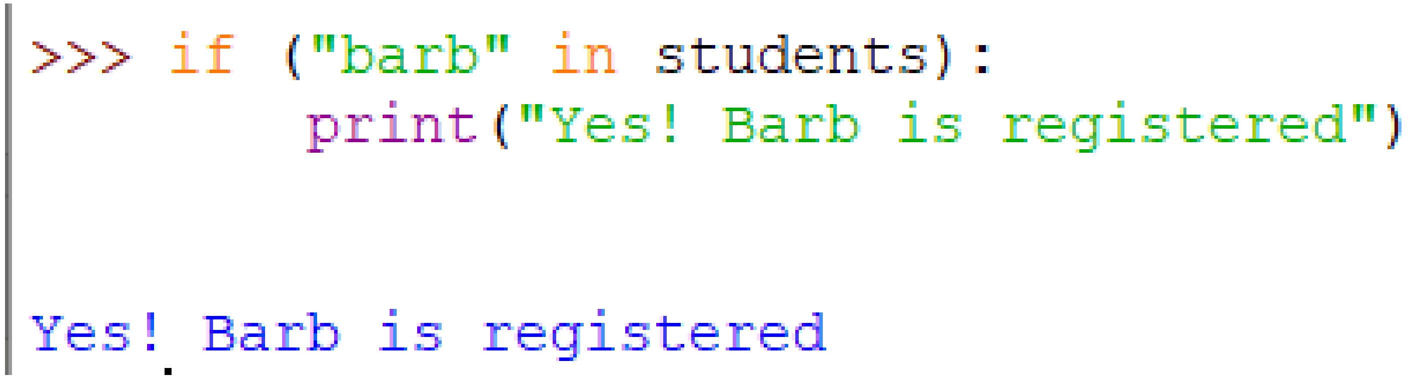


students

“abe”	“barb”	“chris”	“abe”	“dan”	“chris”	“ellie”

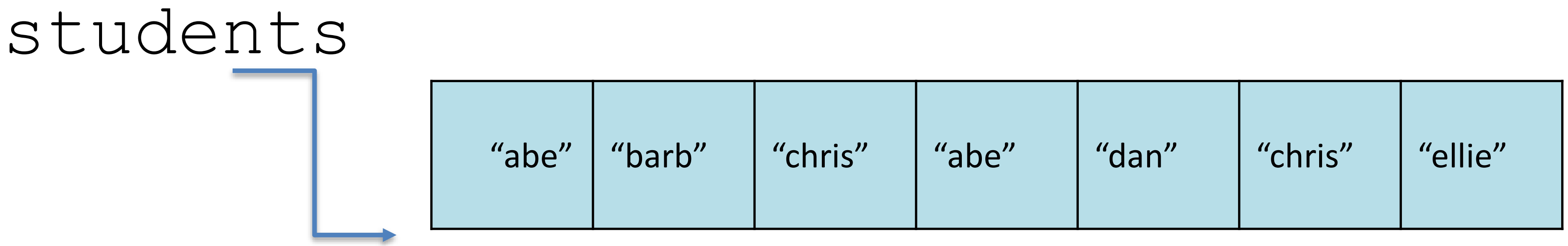
Is “barb” in the set?

"barb" in students



```
>>> if ("barb" in students):
        print("Yes! Barb is registered")


Yes! Barb is registered
```

Updating the set (adding items)



students

“abe”	“barb”	“chris”	“abe”	“dan”	“chris”	“ellie”

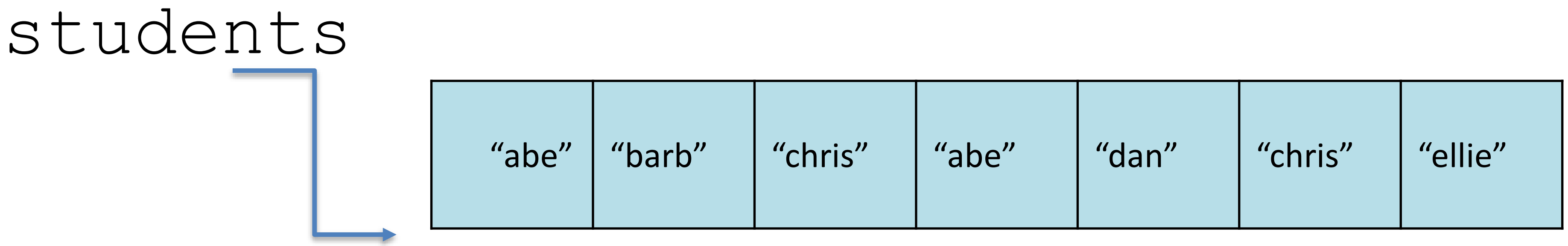
You can add elements to the sets in two ways.

[1] Adding just one element
students.add(“Fiona”)

[2] Adding one set_y to another set_x
set_x.update(set_y)

set_y can be a LIST, TUPLE, LIST or DICTIONARY

Deleting items from the set



students

“abe”	“barb”	“chris”	“abe”	“dan”	“chris”	“ellie”

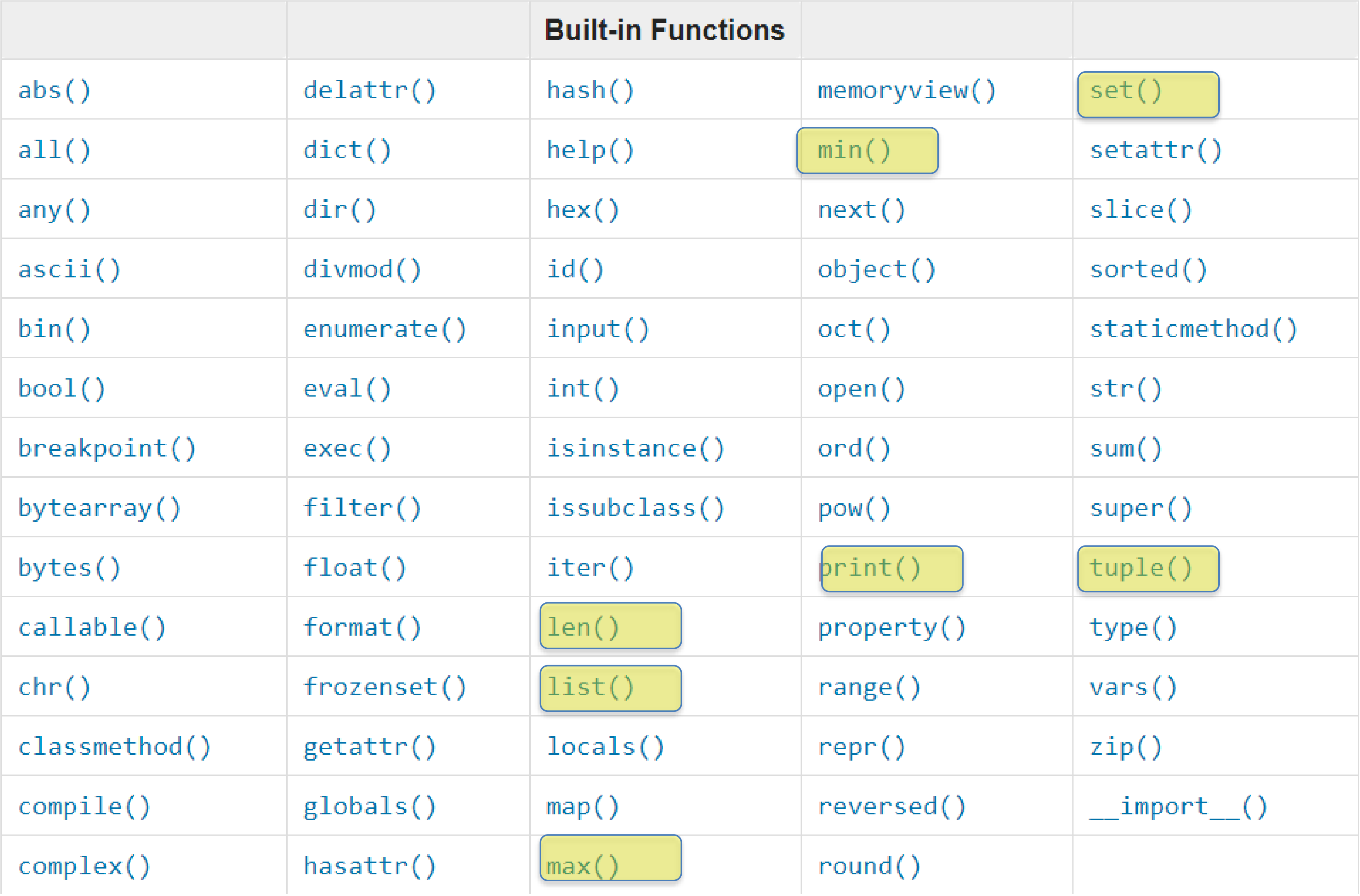
You can delete elements from the set in three ways.

[1] remove(x)
X is removed. If x doesn’t exist in the set, you will get a KEYERROR.

[2] discard(x)
X is removed if it exists. It is not an error if it doesn’t exist.

[3] pop( )
pop( ) removes a random element (we can not predict which element).
And **returns the element removed**.

Python’s Built-in Functions



		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

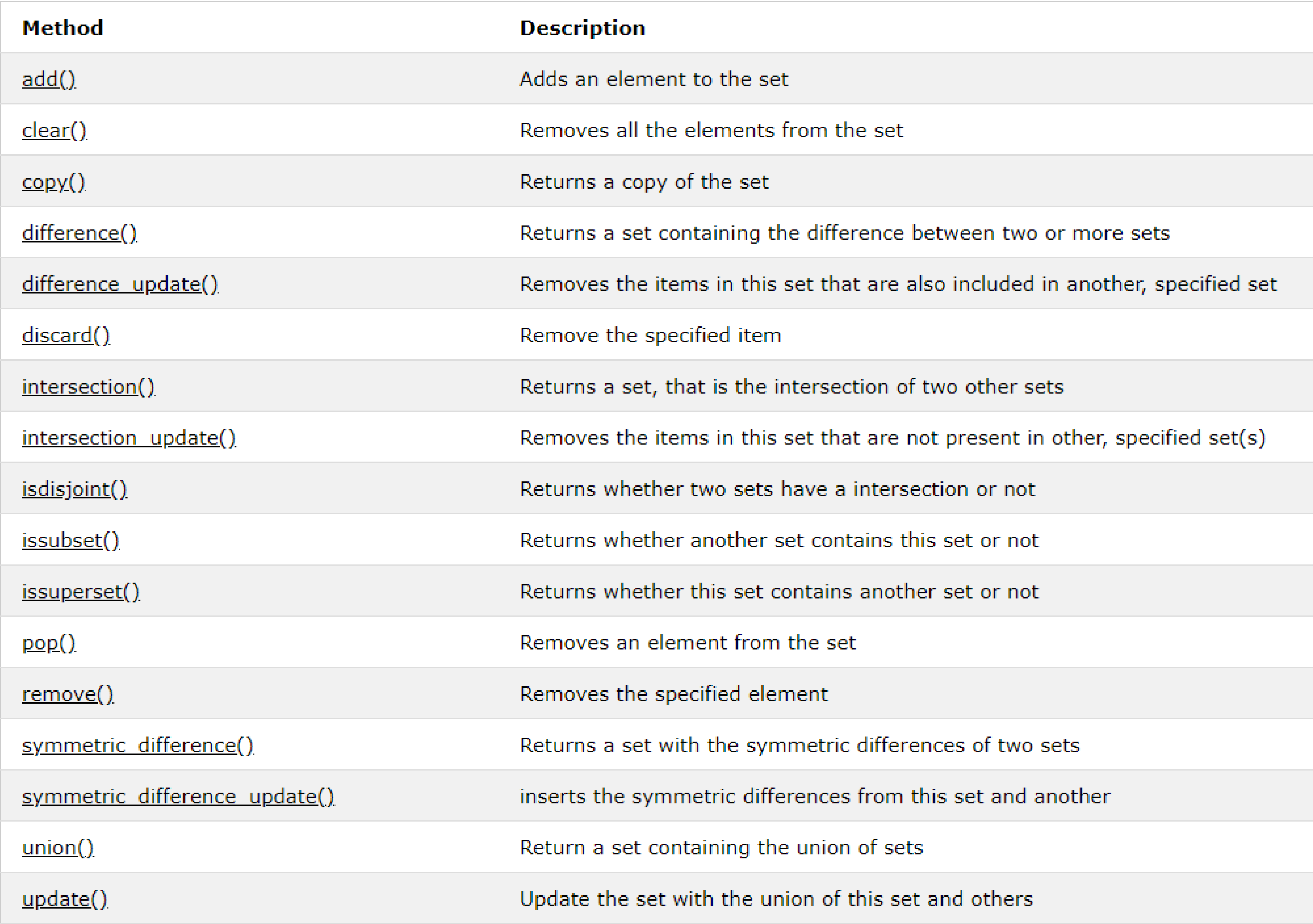
Some functions are valid for sets.

I highlighted some.

Can you find other functions that are valid on tuples?

https://docs.python.org/3/library/functions.html

set methods



Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

https://www.w3schools.com/python/python_ref_set.asp

Set summary

Unordered collection.
Does not contain duplicates.

Supports all data types
A given set can also contain items of different data types.

Very efficient in performing the set operations

* union
* intersection
* difference
* symmetric difference

Lists vs Tuples vs Sets

	Lists	Tuples	Set
Ordered	✓	✓	✗
Indexed	✓	✓	✗
Add or Update items	✓	✗	✓
Can contain duplicates	✓	✓	✗
Uses	Square Brackets	Round Brackets	Curly Brackets
	[ ]	( )	{ }