CSE231 - Lab 09

Nested Dictionaries, Sets

A Quick Refresher on Dictionaries

List

index	value
0	"Eggs"
1	"Milk"
2	"Cheese"
3	"Yogurt"
4	"Butter"
5	"More Cheese"

Dictionary

key	value
'Eggs'	2.59
'Milk'	3.19
'Cheese'	4.80
'Yogurt'	1.35
'Butter'	2.59
'More Cheese'	6.19

Dictionaries define a set of key-value pairs, where accessing the dictionary at a particular key yields an associated value. Keys can be any *immutable* object, whereas values can be a *mutable or immutable* object.

Adding to a Dictionary

```
D = {}

D[‘key’] = ‘value’  # D : {‘key’: ‘value’}

D[2] = [1, 2, 3]  # D : {‘key’: ‘value’, 2: [1, 2, 3]}

D[‘key’] += ‘d’  # D : {‘key’: ‘valued’, 2: [1, 2, 3]}

D[2].append(4)  # D : {‘key’: ‘valued’, 2: [1, 2, 3, 4]}

D[3].append(4)  # KeyError; 3 is not a key in D
```

Adding to a Dictionary (Typical Use)

```
D = {}

for item in something:

    if item in D:

        D[item] += 1  # values could be lists, ints, floats,

    else:             # etc. You'd of course use the

        D[item] = 1   # corresponding appendation method
```

Accessing a Nested Dictionary

```
D = {1: {‘name’: ‘Eva’, ‘age’: 21, ‘gender’: ‘Female’},
     2: {‘name’: ‘Wungus’, ‘age’: 22, ‘gender’: ‘Unknown’}}
print(D[1])  # {‘name’: ‘Eva’, ‘age’: 21, ‘gender’: ‘Female’}
print(D[1][‘name’])  # ‘Eva’
print(D[2][‘age’])  # 22
D[1][‘age’] = 22
print(D[1][‘age’])  # 22
```

Nesting Dictionaries

Like lists and tuples, dictionaries can be nested. You can have a dictionary whose values are *other* dictionaries. Remember that keys in a dictionary have to be immutable, thus keys *cannot* be dictionaries.

Having a nested dictionary can be a bit confusing, so hopefully an example might help. I'll be showing this on my code editor since PythonTutor isn't compatible with external files.

Sets

The last basic data structure, we've finally made it! Sets are an idea taken from mathematics, most of you should know about them, but if not, we'll briefly go over the essentials.

Sets are a collection of *unique*, *unordered* values. Meaning that the two following sets are considered *equivalent*:

A = {1, 2, 3, 3, 4, 5} ... $\{ x \mid 1 \leq x \leq 5, x \in \mathbb{Z} \}$

B = {5, 4, 4, 4, 3, 2, 1, 1} ... $\{ x \mid 1 \leq x \leq 5, x \in \mathbb{Z} \}$

"$\{ x \mid 1 \leq x \leq 5, x \in \mathbb{Z} \}$" can be read as: "x such that ('x | ') x is inclusively bounded between 1 and 5 ('$1 \leq x \leq 5$'), and is within the set of all integers ('$x \in \mathbb{Z}$')."

Sets (cont.)

How do we translate this to Python? In much the same way, actually.

```
A = {1, 2, 3, 3, 4, 5}

B = {5, 4, 4, 4, 3, 2, 1, 1}

print(A == B)     # True

print(A)     # {1, 2, 3, 4, 5}

print(B)     # {1, 2, 3, 4, 5}

print(len(A))     # 5
```

(The order of values is pseudo-random when printed, Mimir test cases obviously won't check if your sets are in a particular order)

Set Initialization

```
D = {}    # curly braces with nothing = empty dictionary

S = set()    # recommended empty set initialization

S = {20, 5, 10}    # initialization of a set with values

S = set(“abcabbcd”)    # can convert from iterables

print(S)    # {‘b’, ‘d’, ‘a’, ‘c’}

S = {x for x in “abracadabra” if x not in “abc”}

print(S)    # {‘r’, ‘d’}, set comprehension!
```

Adding/Discarding

```
S = set()

S.add(100)      # S: {100}, adds a given element

S.discard(100) # S: set(), discards a given element

S.remove(100)  # KeyError

# .remove() removes an object from the set, but raises

# KeyError if the object doesn’t exist (.discard() does not)

# .pop() is alike .discard(), but returns the removed item
```

Set Operations

Like in mathematics, sets have a lot of unique operations. If you already know about mathematical sets, you may recognize these guys: ⊆, ⊂, ∪, ∩, etc.. All of these operations have a Python equivalent.

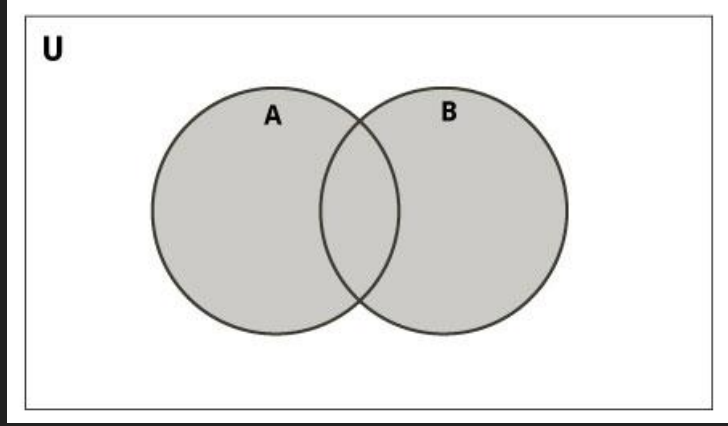
In addition to this, you can of course use `len()` to determine the number of elements within the set (| | in mathematics, also known as the “cardinality” when talking about sets), and use the `` `in` `` keyword to determine if an element exists within your set (∈ in mathematics).

For each of the operators in Python, there is also a corresponding method function as you’ll see.

Set Operators Table

Name	Operator	Example Usage	Description
Union	|	`A | B`	All elements from both sets
Intersection	&	`A & B`	All elements shared between both sets
Difference	-	`A - B`	All elements in A that are not in B
Symmetric Difference	^	`A ^ B`	All elements unique to both sets

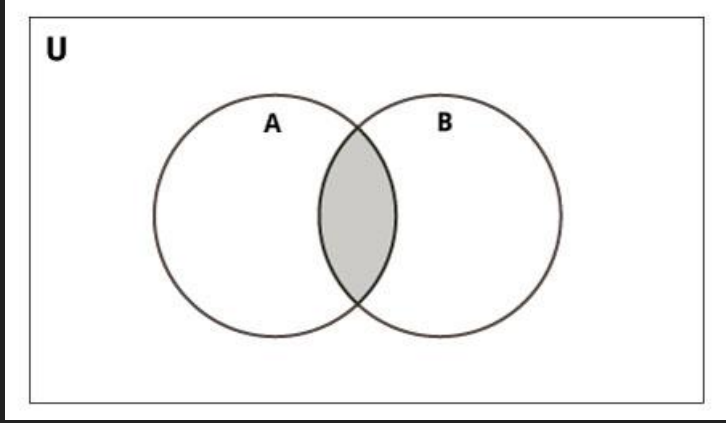
Union




```
A = {1, 2}

B = {3, 4}

print(A | B)     # {1, 2, 3, 4}

print(A.union(B))     # {1, 2, 3, 4}
```

All elements, within both sets. Equivalent to `U` in mathematics.

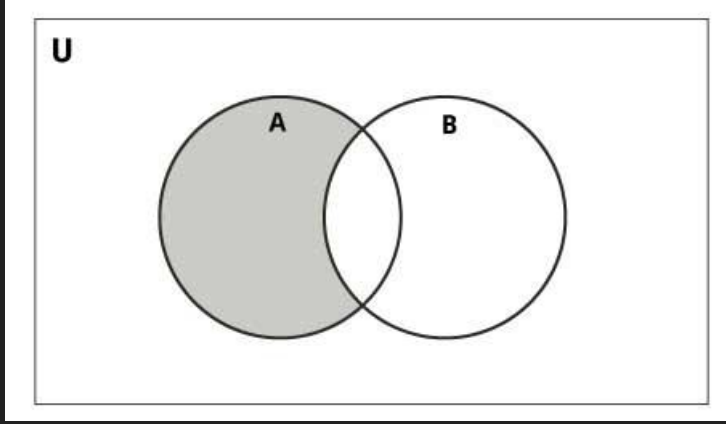
Intersection




```
A = {1, 2, 3}

B = {2, 3, 4}

print(A & B)    # {2, 3}

print(A.intersection(B))    # {2, 3}
```

All elements ***shared*** between both sets. Equivalent to ∩ in mathematics.

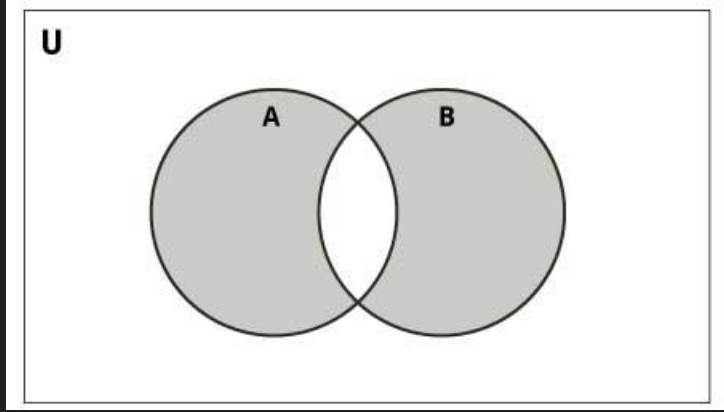
Difference




```
A = {1, 2, 3}

B = {2, 3, 4}

print(A - B)     # {1}

print(A.difference(B))     # {1}
```

Elements ***in A but not in B***. Equivalent to − in mathematics.

Symmetric Difference




```
A = {1, 2, 3}

B = {2, 3, 4}

print(A ^ B)    # {1, 4}

print(A.symmetric_difference(B))    # {1, 4}
```

All elements ***unique*** to both sets. Equivalent to ⊖ in mathematics.

Membership Tests

The last few operators compatible with sets are the comparison operators; they test the membership relations between two sets, and also have method function equivalents.

```
A = {1, 2, 3}

B = {1, 2}

print(B <= A)  # True

print(B.issubset(A))  # True
```

Tests whether every element in B is in A (whether B is a subset). You can also use the < operator, to test for a proper subset, i.e., `(B <= A) and (B != A)`

Membership Tests

The last few operators compatible with sets are the comparison operators; they test the membership relations between two sets, and also have method function equivalents.

```
A = {1, 2, 3}

B = {1, 2}

print(B >= A)  # False

print(B.issuperset(A))  # False
```

Tests whether every element in A is in B (whether A is a subset). You can also use the `>` operator, to test for a proper subset, i.e., `(B >= A) and (B != A)`