

Structures: Sets and Tuples

**CPSC 217: Introduction to Computer Science for Multidisciplinary
Studies I**
July 2024

Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

June 4, 2024

Copyright © 2024



Tuples?

What is a Tuple?

- A collection of values
 - Values
 - May all have the same type, or
 - May have different types
 - Each item is referred to as an element
 - Each element has an index (ORDERED)
 - Unique integer identifying its position in the tuple
 - A tuple is one type of data structure
 - A mechanism for organizing related data

Main thing to remember!

- Similar to lists, but
 - length cannot be changed
 - **Items cannot be modified (immutable)**
 - () empty tuple, (3,) length one tuple

```
aTuple = (1, "ICT", 3.14)
```

Tuples

- Like a list, a tuple is a sequence type that its elements can be of any other type
- Support many of the same operations as lists
- Unlike lists, tuples are used to store data that should not be changed.
- Format
 - `<tuple name> = (<value 1>, <value 2>, ... , <value n>)`
- Example

```
student = ('Marc', 123456789, 9.5)
print (student)
print (student[1])
#student[2] = 10
```



```
('Marc', 123456789, 9.5)
123456789
```

TypeError: 'tuple' object does not support item assignment

Tuple

- Format:

<list name> = (<value 1>, <value 2>, ... , <value n>)

Examples:

nums = (10.0, 9.0, 8.5, 5.0, 7.5)

letters = ('a', 'b', 'c', 'd', 'e', 'f', 'g')

names = ('Marc', 'Jim', 'Ken')

mixed = (1.0, 1, "this", True)

By defining the tuple memory is allocated for it

names = (x,) → Singleton tuple of one time

Regular brackets () without comma are interpreted as empty tuple

Tuple operations

Operations	Example	Description
Indexing	<code>name[i]</code>	Access item by index
Slicing	<code>name[start:end:step]</code>	Get sub-tuple
Concatenation	<code>names1+names2</code>	Join two tuples into larger tuple
Update tuple	Immutable	Use slicing to get sub-tuple Use concatenation to get larger tuple
Length	<code>len(name)</code>	Get length of tuple
Repetition	<code>name*x</code>	Multiply to get tuple with int x copies of its contents in order
Membership	<code>n in name</code>	Boolean if item is in tuple at base level
Loop	<code>for x in name : print(x)</code>	Iterate through each item in tuple
Index	<code>name.index("Carl")</code>	Returns first index of item "Carl" in tuple name

Tuple

- In effect when we return multiple values from a function we are using tuples

- The same

```
def foo():
```

```
    return x,y
```

```
def foo():
```

```
    return (x,y)
```

- A number of common languages don't have tuples a structure like tuples, and are limited to returning a single pointer of data.

Packing/Unpacking

- You can define a tuple without brackets. Python will interpret variables/expressions separated by commas.

```
x = 1,2
```

```
print(x) -> (1,2)
```

```
print(x[0]) -> 1
```

```
print(x[1]) -> 2
```

```
a,b = x
```

```
print(a) -> 1
```

```
print(b) -> 2
```

The process seen here is generally called packing, and unpacking

What is a Set?

- A collection of values
 - Values
 - May all have the same type, or
 - May have different types
 - Each item is referred to as an element
 - ~~Each element has an index UNORDERED~~
 - ~~Unique integer identifying its position in the list~~
 - A set is one type of data structure
 - A mechanism for organizing related data

What is a Set?

- A set contains only immutable types
- A set only contains **unique!!!** elements
- A collection of values
 - Values
 - May all have the same type, or
 - May have different types
 - Each item is referred to as an element
 - ~~Each element has an index UNORDERED~~
 - ~~Unique integer identifying its position in the list~~
 - A set is one type of data structure
 - A mechanism for organizing related data

Set

- Unlike a list/tuple, a set is unordered
- The functions for a set are very different (we can't index/slice)
- Unlike tuples, sets can change.
- Format
 - `<set name> = {<value>, <value>, ... , <value>}`
- Example
 - `names = {"Albert", "Brian", "Carl"}`

Set

- Format:

`<set name> = {<value 1>, <value 2>, ... , <value n>}`

Examples:

`nums = {10.0, 9.0, 8.5, 5.0, 7.5}`

`letters = {'a', 'b', 'c', 'd', 'e', 'f', 'g'}`

`names = {'Marc', 'Jim', 'Ken'}`

`mixed = {1.0, 1, "this", True}`

By defining the set memory is allocated for it

`names = set()` → Only way to declare an empty set

`{}` -> is interpreted as a empty dictionary

Set operations

Operations	Example	Description
Unique	<code>x = {1,1,1,2,2,2,2}</code>	<code>x = {1,2}</code>
Membership	<code>n in name</code>	Boolean if item is in set at base level
Concatenation	<code>names1+names2</code>	Join two sets into larger set
Update set	<code>add(item)</code> <code>update(set)</code> <code>remove(item)</code> <code>discard(item)</code> <code>pop()</code>	no change if duplicate add all items from other set error if no item no error random remove
Length	<code>len(name)</code>	Get length of set
Repetition	<code>name*x</code>	Multiply to get set with int x copies of its contents in order
Loop	<code>for x in name :</code> <code> print(x)</code>	Iterate through each item in set

Sets

- Why do we use sets?
 - Natural uniqueness can make some things quick (we can skip membership checks)
 - Sets are rather common in many pure mathematics, logic, philosophy, and computer science (especially AI)
 - Where have you seen sets visualized (Venn Diagrams!)

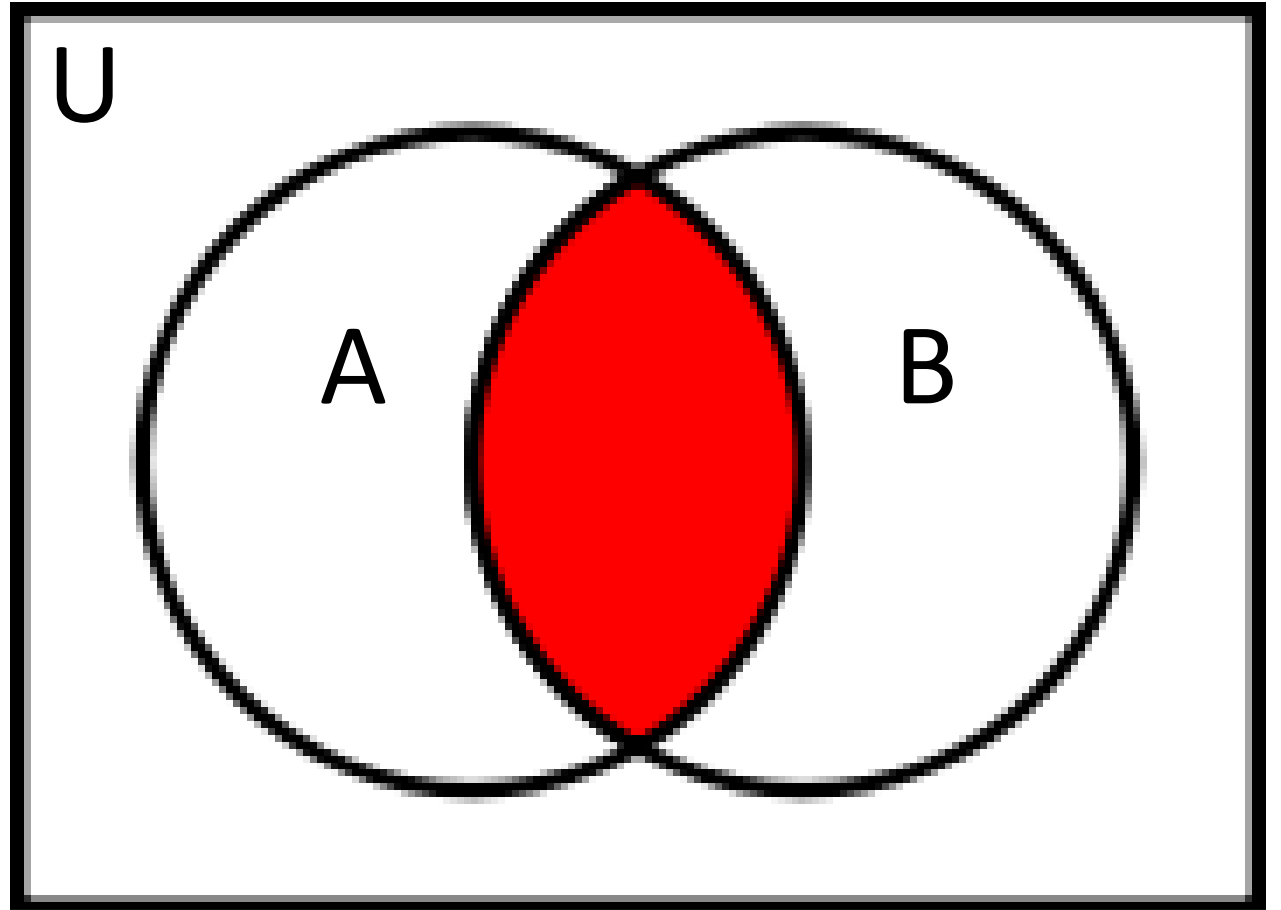
Intersection (and)

Set Notation

$A \cap B$

Python

`A & B`



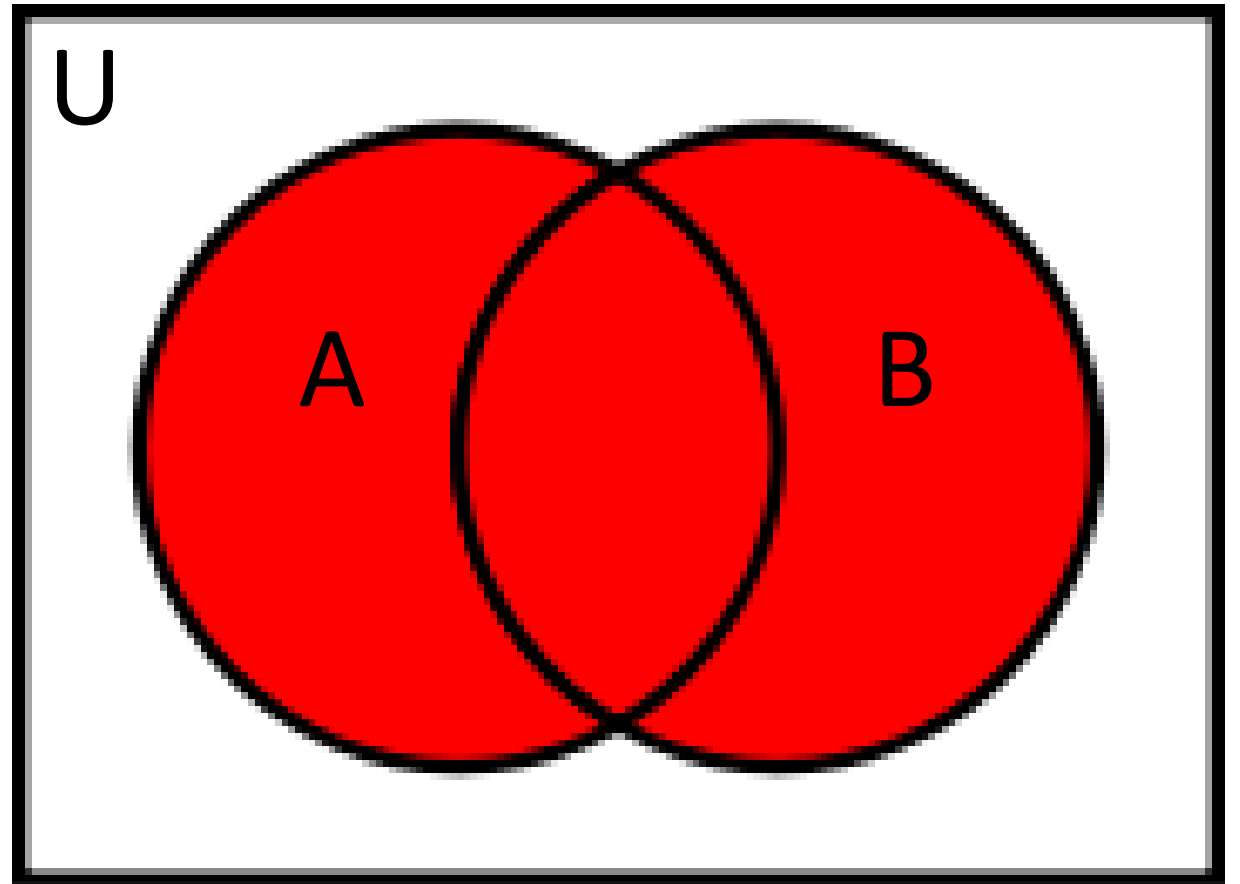
Union (or)

Set Notation

$A \cup B$

Python

$A | B$



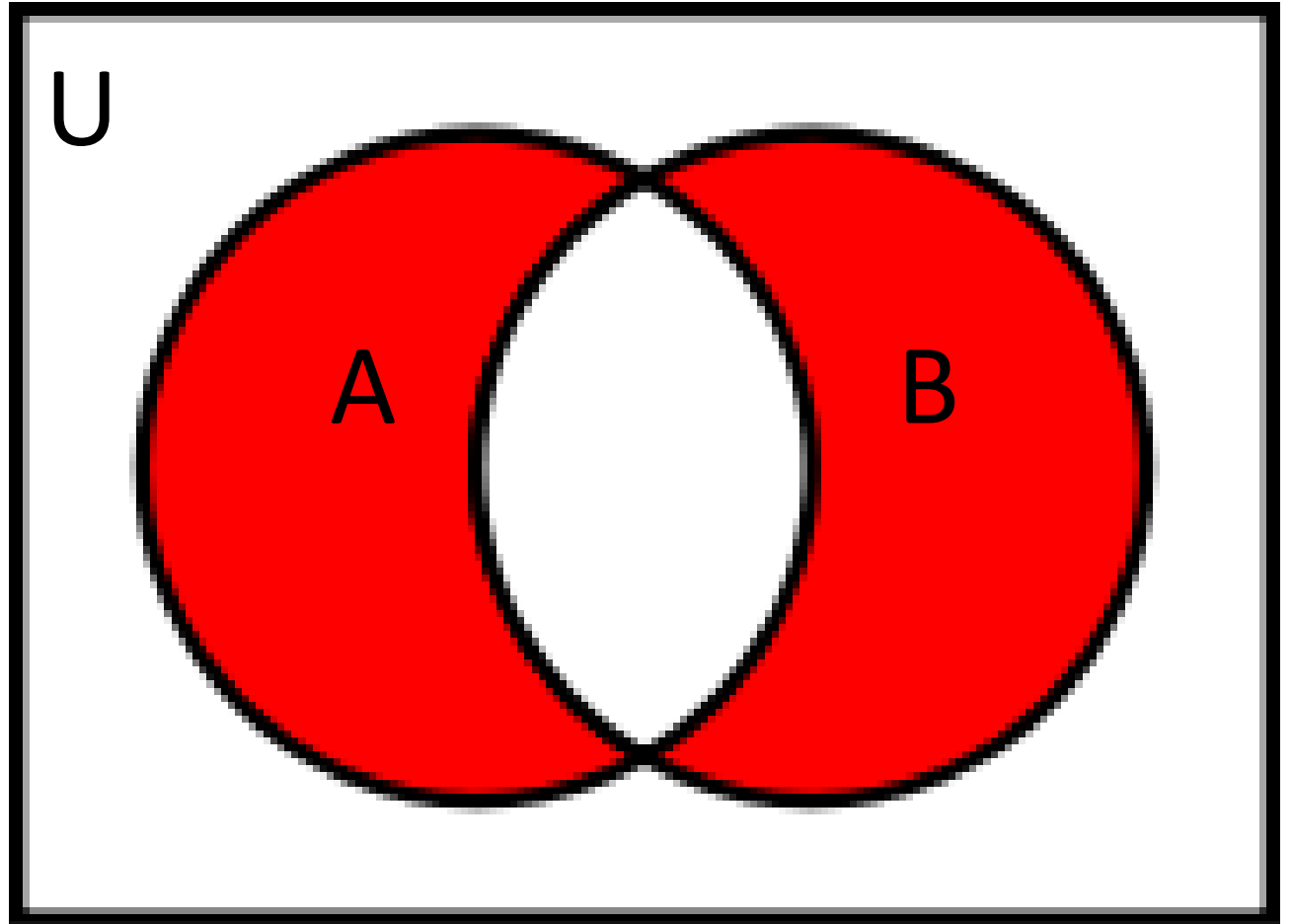
Symmetric Difference (not and)

Set Notation

$$A \triangle B$$

Python

$$A \wedge B$$



Complement Difference

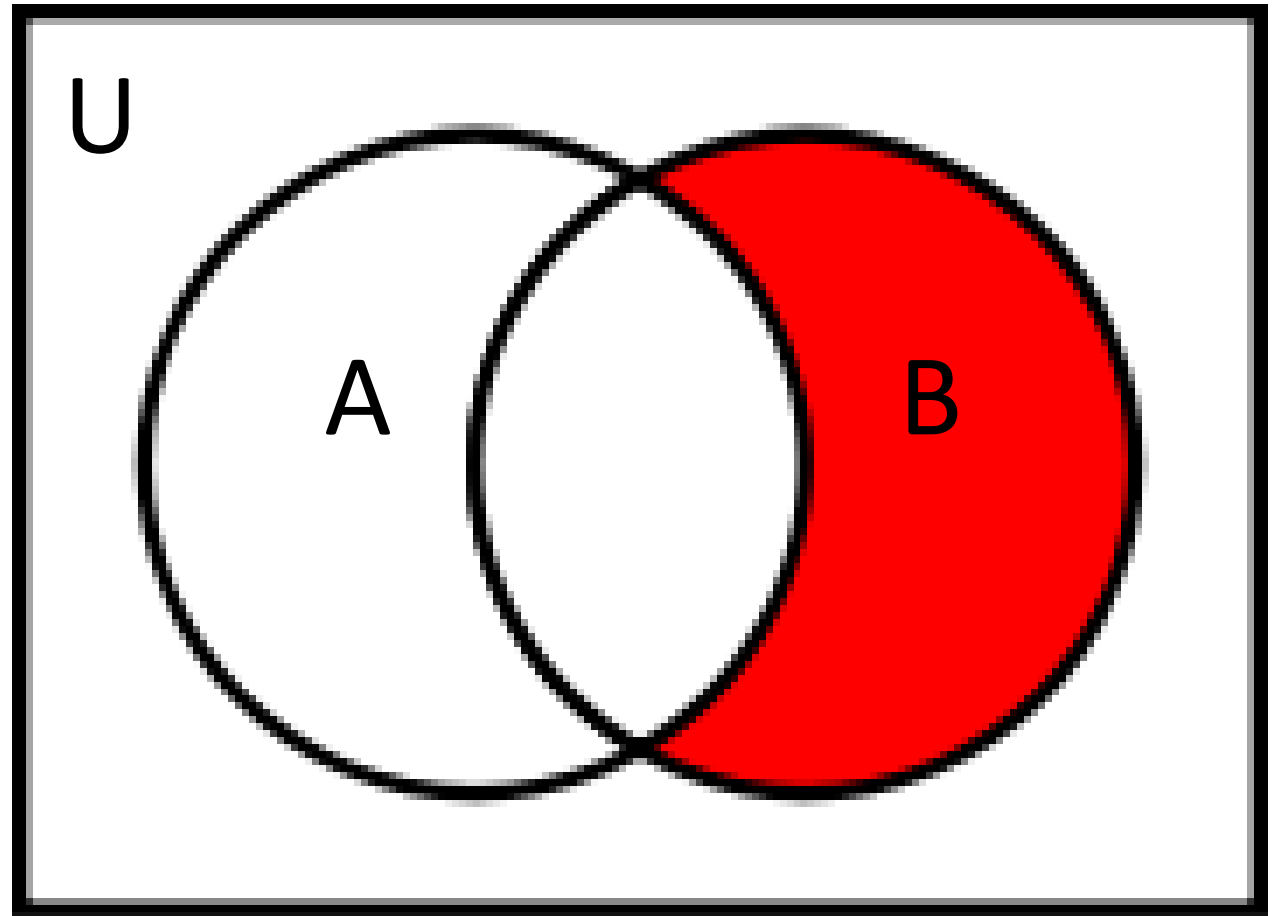
Set Notation

$B \setminus A$

$A^c \cap B$

Python

$B - A$



Set questions

Operations	Example	Description
Is disjoint	<code>x.isdisjoint(y)</code>	True if neither x,y share an element
Is subset	<code>x.issubset(y)</code> OR <code>x <= y</code>	True if all elements in x are in y
Is superset	<code>x.issuperset(y)</code> OR <code>x >= y</code>	True if all in elements in y are in x
Equal	<code>x==y</code>	True if all elements in x are in y, all elements in y are in x
Not equal	<code>x != y</code>	True if at least one element is not in both x and y
Proper subset	<code>x < y</code>	<code>x <= y</code> and <code>x != y</code>
Proper superset	<code>x > y</code>	<code>x >= y</code> and <code>x != y</code>

Onward to ... dictionaries.

Jonathan Hudson
jwhudson@ucalgary.ca
<https://cspages.ucalgary.ca/~jwhudson/>



UNIVERSITY OF
CALGARY