15-110 Principles of Computing – F19

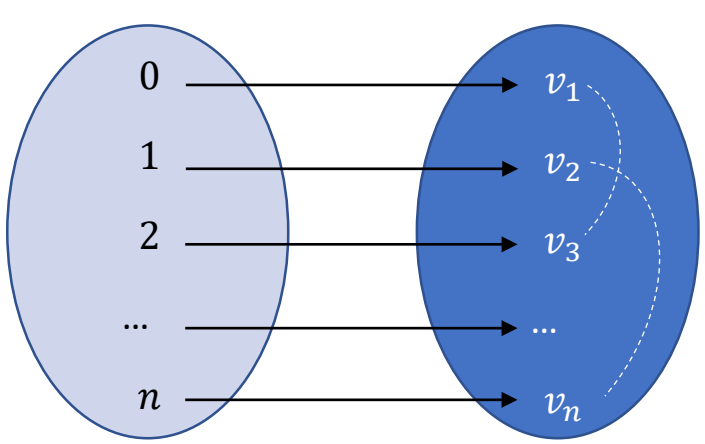
Lecture 21:
Sets

Teacher:
Gianni A. Di Caro

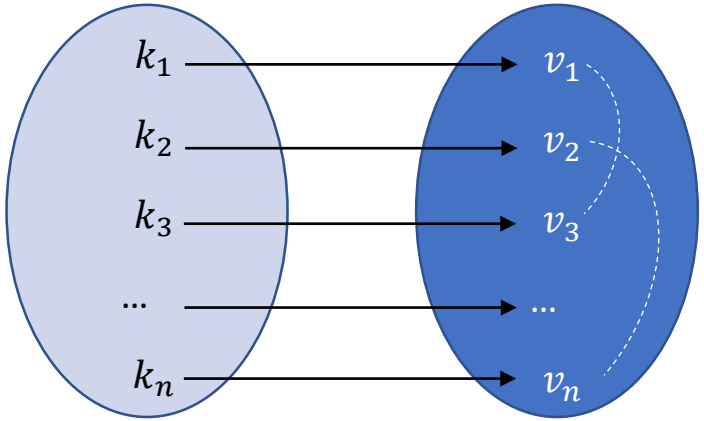
جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Sets: unordered, unhashed, non-scalar data structures

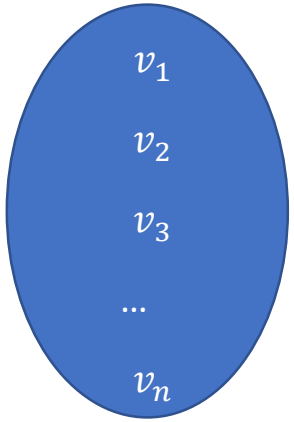
- **Set data structure**: unordered collection of items where every element is unique (no duplicates) and must be immutable
- The set itself is mutable: elements can be inserted and removed, aliases between sets can be created



Sequences
(Lists, tuples)



Dictionaries



Sets

Creation of a set

- Use of { } brackets with literals:

```
new_set = {1, 2, 3, 4.5, 5.3, True, (1,2), 'Hello'}
print(new_set, type(new_set))
```

✓ Any mixed collection of immutable types is admitted

```
new_set = {1, 2, 3, [1,2]}
```
→ Error, the list [1,2] is a mutable object

✓ The set variable itself is mutable

Creation of a set: aliasing (=) and cloning (copy() method)

- ✓ **Aliasing** (same content, same identity):

```
A = {1, 2, 3}
B = A
B.add(11)
print(A,B)   #Output: {1,2,3,11} {1,2,3,11}
```

- ✓ **Cloning** (shallow copy):

```
A = {1, 2, 3}
B = A.copy()
B.add(11)
print(A,B)   #Output: {1,2,3} {1,2,3,11}
```

Creation of a set

- Use of built-in function `set()` to create a new set from an *iterable* (string, list/tuple, dictionary, set): each element of the iterable object defines an element of the set, **duplicates are discarded**

```
l = [1, 2, 3, 4.5, 5.3, True, (1,2)]     # set from a generic list
new_set = set(l)
print(new_set, type(new_set))


numbers = {1: 'p', 2: 'p', 3:'p', 4:'r', 5:'p', 6:'r'}

new_set = set(numbers)                   # set from a dictionary using the keys, set is {1, 2, 3, 4, 5, 6}
new_set = set(numbers.values())          # set from a dictionary using the values, set is {'p','r'}
new_set = set("apple")                   # set from a string, new_set is {'e', 'a', 'l', 'p'}
new_set = set(["apple"])                 # set from a list of strings, new_set is {'apple'}
empty_set = set()                        # empty set
```

Changing a set: add(), update()

- ✓ No indexing, no hashing make sense in sets

- Add single elements with method `add()`:

```
my_set = {1,3,5}
my_set.add(2)
my_set.add(1)
```

→ no error is thrown but no duplicate element is inserted since 1 in already in set

- Add multiple elements with method `update(iterable)` that takes as input iterables (strings, tuples/lists, dictionaries, sets) and add each element into the set, no duplicates are inserted

```
my_set = {1,3,5}
my_set.update([2,3,4])          # my_set now contains {1,2,3,4,5}
my_set.update({2,2,2})          # my_set is left unchanged, it still contains {1,2,3,4,5}
my_set.update({'c':1, 'b':2})   # my_set now contains {1, 2, 3, 4, 5, 'b', 'c'}
my_set.update("apple")          # my_set now contains {1, 2, 3, 4, 5, 'b', 'c', 'e', 'a', 'l', 'p'}
```

Changing a set: discard(), remove ()

- Remove single element, no error is thrown if the element isn't there: `discard()`

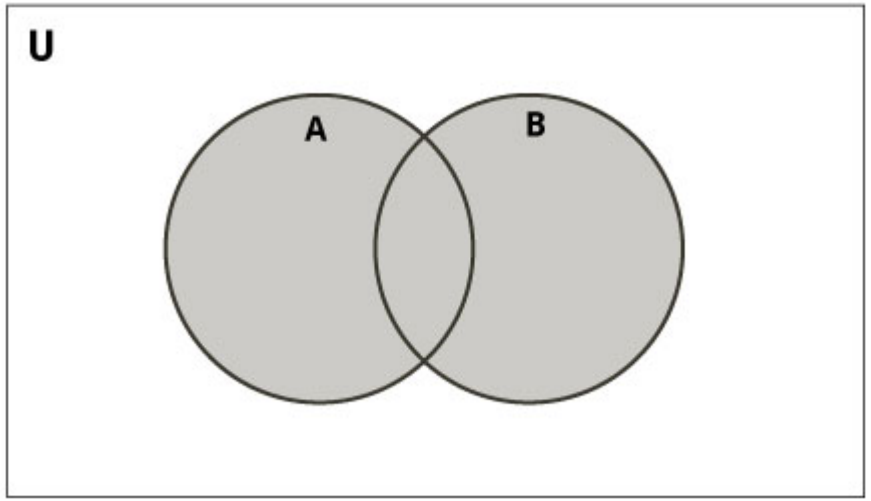
```
my_set = {1,3,5}
my_set.discard(3)    # my_set is now {1,5}
my_set.discard(4)    → no error is thrown, set stays unchanged
```

- Remove single element, an error is thrown if the element isn't there: `remove()`

```
my_set = {1,3,5}
my_set.remove(3)    # my_set is now {1,5}
my_set.remove(4)    → an error is thrown!
```

Operations with sets: Union

- Sets are mathematical objects, and we can perform the usual mathematical operations on them: **union, intersection, difference, symmetric difference**



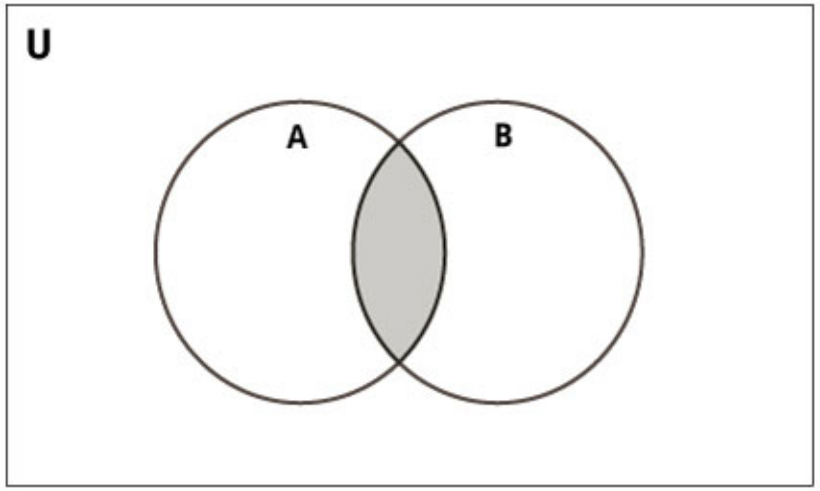
- **Union**, operator |
- Union method: C = `A.union(B)`

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
C = A.union(B)

print(A | B, C == (A|B))
#Output: {1, 2, 3, 4, 5, 6, 7, 8}  True
```

- In-place union (update / modify A):
  - `A |= B`
  - `A.update(B)` (returns None)

Operations with sets: Intersection



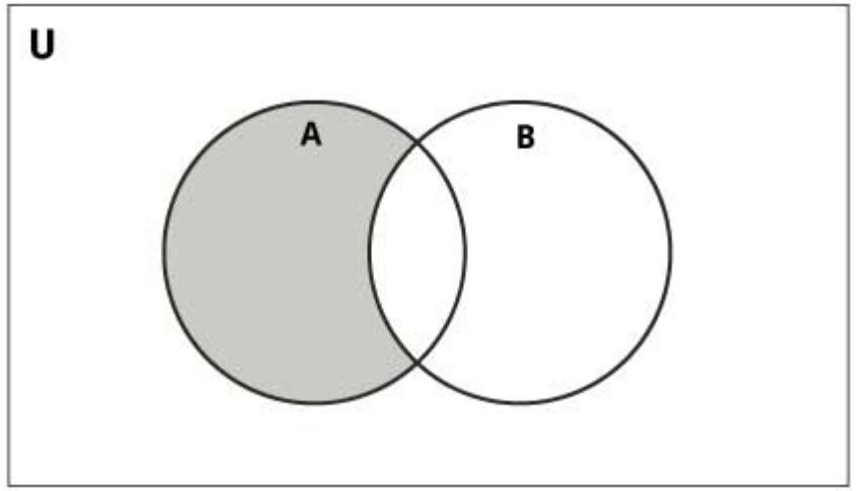
- **Intersection**, operator `&`
- Intersection, method: A.intersection(B)

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
C = A.intersection(B)

print(A & B, C == (A&B))
#Output: {4, 5} True
```

- In-place intersection (update / modify `A`):
  - `A.intersection_update(B)` (returns `None`)
  - `A &= B`

Operations with sets: Difference



- **Difference**, operator –
- Difference method: A.difference (B)

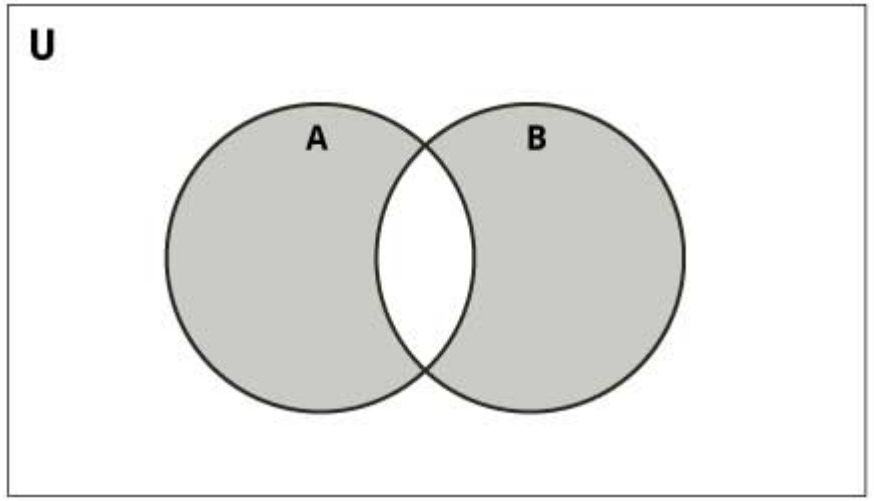
```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
C = A.difference (B)

print(A - B, C == (A-B))
#Output: {1, 2, 3} True

print(B - A, C == (B-A))
#Output: {6, 7, 8} False
```

- In-place difference (update / modify `A`):
  - `A.difference_update(B)` (returns `None`)
  - `A -= B`

Operations with sets: Symmetric difference



- **Symmetric difference**, operator ^
- Difference method: A.symmetric_difference (B)

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
C = A.symmetric_difference(B)

print(A ^ B, C == (A^B))
#Output: {1, 2, 3, 6, 7, 8} True
```

- In-place difference (update / modify `A`):
  - `A.symmetric_difference_update(B)` (returns `None`)
  - `A ^= B`

(Regular) Operations with sets: membership, iteration

- **Membership:** `in` operator

```
my_set = set("apple")      # my_set is {'e', 'a', 'l', 'p'}
in_set = 'a' in my_set     # in_set is True
in_set = 2 in my_set       # in_set is False
```

- **Iteration:**

```
my_set = set("apple")
for letter in my_set:          # for loop with letter taking the values in {'e', 'a', 'l', 'p'}
    print(letter)
```

Relational operators

- **Equality:** `==, !=`

```
A == B     # returns True if the two sets are equal in content, False otherwise
A != B     # returns True if the two sets have some differences, False otherwise
```

- **Subset of:** `<=, A.issubset(B)`

```
A <= B             # returns True if A is a subset of B, False otherwise
A.issubset(B)      # returns True if A is a subset of B, False otherwise
```

- **Superset of:** `>=, A.issuperset (B)`

```
A >= B             # returns True if A is a superset of B, False otherwise
A.issuperset(B)    # returns True if A is a superset of B, False otherwise
```

- **Strictly subset or superset:** `>, <`

```
A > B              # returns True if A >= B and A != B, False otherwise
A < B              # returns True if A <= B and A != B, False otherwise
```

List of all methods for sets

Method	Description
add()	Adds an element to the set
clear()	Removes all elements from the set
copy()	Returns a copy of the set
difference()	Returns the difference of two or more sets as a new set
difference_update()	Removes all elements of another set from this set
discard()	Removes an element from the set if it is a member. (Do nothing if the element is not in set)
intersection()	Returns the intersection of two sets as a new set
intersection_update()	Updates the set with the intersection of itself and another
isdisjoint()	Returns `True` if two sets have a null intersection

Use the help() function to get complete descriptions inline

https://docs.python.org/3/library/stdtypes.html - set-types-set-frozenset

List of all methods for sets

issubset()	Returns `True` if another set contains this set
issuperset()	Returns `True` if this set contains another set
pop()	Removes and returns an arbitary set element. Raise `KeyError` if the set is empty
remove()	Removes an element from the set. If the element is not a member, raise a `KeyError`
symmetric_difference()	Returns the symmetric difference of two sets as a new set
symmetric_difference_update()	Updates a set with the symmetric difference of itself and another
union()	Returns the union of sets in a new set
update()	Updates the set with the union of itself and others

Use the help() function to get complete descriptions inline

https://docs.python.org/3/library/stdtypes.html - set-types-set-frozenset

Useful built-in functions that can be used with sets

Function	Description
all()	Return `True` if all elements of the set are true (or if the set is empty).
any()	Return `True` if any element of the set is true. If the set is empty, return `False.`
enumerate()	Return an enumerate object. It contains the index and value of all the items of set as a pair.
len()	Return the length (the number of items) in the set.
max()	Return the largest item in the set.
min()	Return the smallest item in the set.
sorted()	Return a new sorted list from elements in the set(does not sort the set itself).
sum()	Retrun the sum of all elements in the set.

Practice

Task 1.9 (5 points) Implement the function `intersect_three(s1, s2, s3)` that takes three sets and returns the intersection of all three (i.e., a set with all elements that are in `s1` and `s2` and `s3`).

Task 1.7 (5 points) Implement the function `unique_values(d)` that returns a sorted list containing all unique values that occur in the dictionary `d`.
For example, `unique_values({'a': 2, 'b': 2, 'c': 1})` should return the list `[1,2]`.