Sets in Python

Pai H. Chou
National Tsing Hua University

Outline

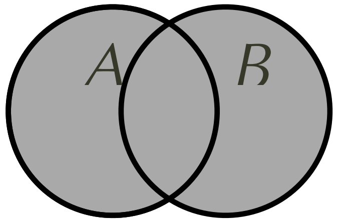
- Set data structures
- Set operators: `|`, `&`, `^`, `-`
- Set methods
  - mutation: add(), clear(), difference_update(), intersection_update(), pop(), remove(), symmetric_difference_update(), update()
  - nonmutation: copy(), difference(), discard(), intersection(), isdisjoint(), issubset(), issuperset(),symmetric_difference(), union(),
- set comprehension

Set data structure

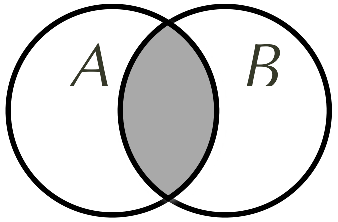
- Set review
- Mutable data structure
- Members of a set must be immutable! ("hashable")
  - number, str, tuple, bool
- "Unordered"
  - order may be respected but not guaranteed
- Iterable
  - may be used like a list in a for loop
- Unique values of members
  - a given value may appear at most once in a set

Review: Sets

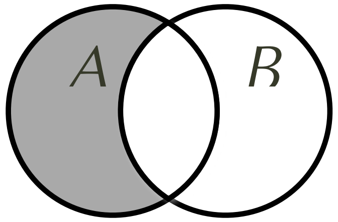
- Mathematical concept
  - unordered collection of data members
    e.g., {1, 2, 4, 8, 16}
- Operators:
  - union (聯集): $A \cup B$



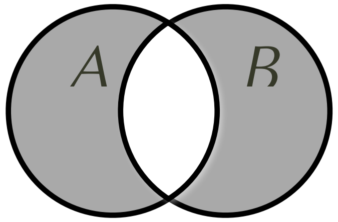
  - intersection (交集): $A \cap B$



  - difference: $A \setminus B$



  - symmetric-difference: $(A \cup B) \setminus (A \cap B)$



Constructing a set

- empty set
  - `s = set()`
  - `s = { }` won't work, because `{ }` defaults to `dict`!
- construct a set from a list or iterable
  - `s = set([1, 3, 5, 7])  # gives { 1, 3, 5, 7 }`
  - `s = set('hello') # unique { 'e','h','l','o' }`
- set literal
  - `s = { 1, 3, 5, 7 }`

Incorrect sets

- `s = {[1,2,3], [4,5,6]}`
  - ***not legal***! set members must be immutable even though sets themselves are mutable!
  - Solution: use tuples instead:
    `s = {(1,2,3), (4,5,6)}`
- `s = {{(1,2,3)}, {4,5}}`
  - ***not legal***, because set is mutable!
    => cannot have set of sets in Python, even though probably can be defined mathematically

set comprehension

- similar to list comprehension
  - { *expression* **for** *loopVar* **in** *iteration* }
- Difference: only unique values are kept

```
>>> {chr(65+i) for i in range(5)} # just like list comprehension
{'A', 'B', 'C', 'D', 'E'}
>>> {2**i for i in range(1, 11)} # powers of 2 up to 2^10
{2, 4, 8, 16, 32, 64, 128, 256, 512, 1024}
>>> [(chr(i), i) for i in range(65, 70)] # tuples of (char, code)
{('A', 65), ('B', 66), ('C', 67), ('D', 68), ('E', 69)}
```

  - the expression must construct an immutable data structure

set comprehension with a condition

- add **if** condition after in
    - [*expression* **for** *loopVar* **in** *iteration* **if** *cond*]

```
>>> {chr(i) for i in range(65, 65+26)}    # all uppercase letters
{'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'}
>>> {chr(i) for i in range(65, 65+26) \
...  if chr(i) not in ['A','E','I','O','U']} # non-vowel subset
{'B', 'C', 'D', 'F', 'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P', 'Q',
'R', 'S', 'T', 'V', 'W', 'X', 'Y', 'Z'}
```

```
>>> {i*(i+1) for i in range(11)}
{0, 2, 6, 12, 20, 30, 42, 56, 72, 90, 110}
>>> {i*(i+1) for i in range(1, 11) if i*(i+1)%3==0}
{6, 12, 30, 42, 72, 90}                # filter for multiples of 3
```

Set operators

- – (set subtraction) -- analogous to A & (~B)
- | (union) -- analogous to bitwise OR
- & (intersection) -- analogous to bitwise AND
- ^ (exclusive-union) analogous to bitwise XOR

Python	Math	Meaning	Example
A - B	$A \setminus B$	set subtract	{1,2} \ {2, 3} = {1}
A | B	$A \cup B$	union	{1, 2} ∪ {2, 3} = {1,2,3}
A & B	$A \cap B$	intersection	{1,2} ∩ {2, 3} = {2}
A ^ B	$(A \cup B) \setminus (A \cap B)$	exclusive union	{1,2} ^ {2, 3} = {1, 3}

Set comparison operators

Python	Math	Meaning	Example
A > B	$A \supset B$	superset of (超集合)	$\{1, 2, 3\} > \{2, 3\}$
A >= B	$A \supseteq B$	superset or equal	$\{1, 2, 3\} >= \{1, 2, 3\}$
A < B	$A \subset B$	subset of (子集合)	$\{1,2\} < \{1, 2, 3\}$
A <= B	$A \subseteq B$	subset or equal	$\{1, 2, 3\} <= \{1, 2, 3\}$
A == B	$A = B$	equal (same values)	$\{1, 2, 3\} = \{1, 2, 3\}$
A != B	$A \neq B$	not equal (not same values)	$\{1, 3\} \neq \{2, 3\}$

Example set operators

```
>>> S = set(range(4))
>>> S
{0, 1, 2, 3}
>>> S | {'A'} # union
{0, 1, 2, 3, 'A'}
>>> S - {2, 7}
{0, 1, 3}
>>> S & {'y', 'z'}
set()
>>> S ^ {3, 4, 5}
{0, 1, 2, 4, 5}
>>> S | {3, 4, 5}
{0, 1, 2, 3, 4, 5}
```

```
>>> S
{0, 1, 2, 3}
>>> S >= {2, 3} # check superset
True
>>> S >= {2, 7} # check superset
False
>>> S == {3, 2, 2, 1, 0, 3}
True
>>> S < {3, 2, 1, 0}
False
>>> S <= {1, 2, 3, 0}
True
```

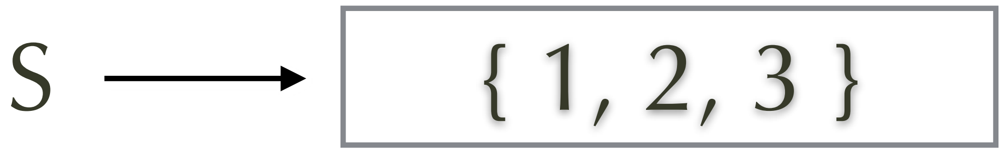
Mutation Methods of set class

- S.method(args) where S is a set

method	Explanation	equivalent to
`S.add(e)`	add *e* to the set	`S |= {e}`
`S.clear()`	clear elements from set	`S -= S`
`S.pop()`	remove arbitrary element	`S` -= some random e
`S.remove(e)`	remove element *e*, error if *e* is not in `S`	`S -= {e}`
`S.discard(e)`	remove element *e*, no error in all cases	`S -= {e}`
`S.update(T)`	update S with union w/ *T*	`S |= T`

- But... these methods modify the set *S* itself, not just compute and return a new set

Difference between in-place modification and assignment

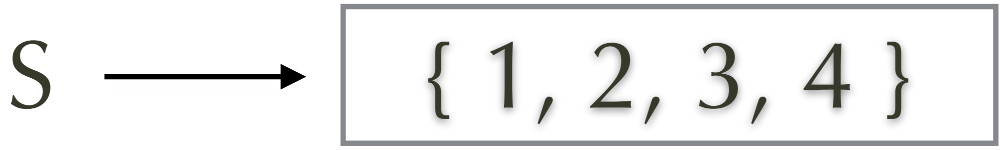


original set

- `S.add(4)`

  same as

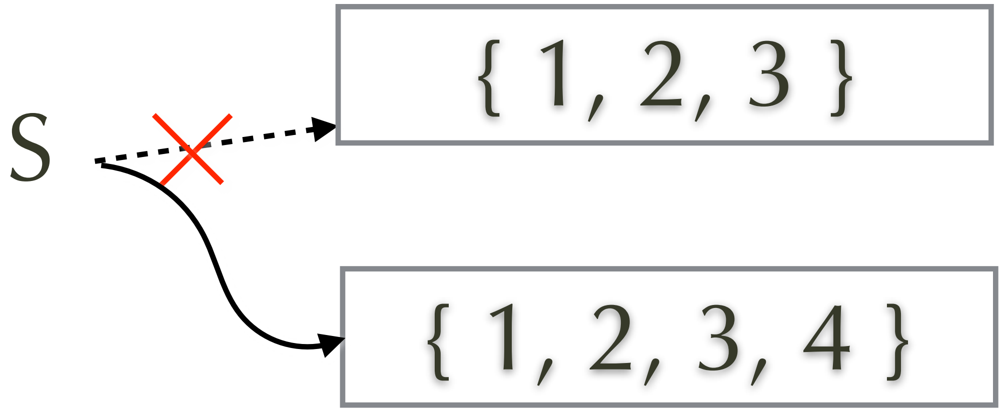
  `S |= {4}`



modified set

same set identity as original,
but modified

- `S = S|{4}`



original set
unmodified

S refers to a newly created set

Mutation vs Non-Mutation Methods of set class

- S and T are sets

Non-Mutation	equivalent to
S.union(T)	S | T
S.difference(T)	S - T
S.symmetric_difference(T)	S ^ T
S.intersection(T)	S & T

Mutation	equivalent to
S.update(T)	S |= T
S.difference_update(T)	S -= T
S.symmetric_difference_update(T)	S ^= T
S.intersection_update(T)	S &= T

Example set methods

```
>>> S = set(range(4))
>>> S
{0, 1, 2, 3}
>>> S.add('A')
>>> S   # S |= {'A'}
{0, 1, 2, 3, 'A'}
>>> S.pop()
0   # your answer may vary!
>>> S
{1, 2, 3, 'A'}
>>> S.update({'y', 'z'})
>>> S  # S |= {'y','z'}
{1, 2, 3, 'y', 'z', 'A'}
>>> S.update({'y', 'z'})
>>> S
{1, 2, 3, 'y', 'z', 'A'}
```

```
>>> S.issuperset({2, 3}) # S >= {2, 3}
True
>>> S.issuperset({2, 7}) # S >= {2, 7}
False
>>> S.remove('A') # S -= {'A'}
>>> S       # but error if 'A' not in S
{1, 2, 3, 'y', 'z'}
>>> S.union({'A', 'B'}) # S | {'A','B'}
{1, 2, 3, 'y', 'z', 'A', 'B'}
>>> S  # S is not modified!
{1, 2, 3, 'y', 'z'}
>>> S.discard('y') # S -= {'y'}
>>> S
{1, 2, 3, 'z'}
>>> S.discard('y') #do nothing if not in
>>> S.remove('y') # error if not in set
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'y'
```