



Australian  
National  
University

# COMP1730/COMP6730

## Programming for Scientists

# Dictionaries and sets

# Announcements

- \* Homework 5 will be marked in the labs next week.
- \* Major assignment will be released early next week.
- \* Final examination details have been posted in Wattle.

# Lecture outline

- \* Mappings: the `dict` type.
- \* Sets: the `set` type.

# Mappings

- \* A *mapping* (a.k.a. *dictionary*) stores key–value pairs; each key stored in the mapping has exactly one value. A key may be any type of constant (immutable) value.
- \* Examples of use:
  - Storing a look-up index (e.g., a contact list).
  - Organising data with “complex” labels (like a multi-dimensional table).
  - Storing solutions to subproblems in a dynamic programming algorithm.



- \* What you can do with a mapping:
  - Create a new, empty mapping.
  - Store a value with a key.
  - Check if a given key stored in the mapping?
  - Look up the value stored for a given key.
  - Remove a key.
  - Enumerate keys, values, or key–value pairs.
  
- \* Key lookup is (amortised) constant time.

# python's dict type

- \* Create a new dictionary:

```
>>> adict = {}
```

```
>>> adict = dict()
```

```
>>> adict = { (2015, 12) : 33.4,  
              (2016, 6) : 148.3 }
```

```
>>> adict = { "be" : 2, "can" : 3 }
```

- Dictionary (and set!) literals are written with curly brackets ( { and } ).
- The literal can contain *key* : *value* pairs, which become the initial contents.

\* Key exists in dictionary:

```
>>> key in adict
```

\* Look-up and storing values:

```
>>> adict = { "be" : 2, "can" : 1 }
```

```
>>> adict["can"]
```

```
1
```

```
>>> adict["now"] = 2
```

```
>>> adict[3] = "yet"
```

- To index a value, write the key in square brackets after the dictionary expression.
- Assigning to a dictionary index expression adds or updates the key.

- \* `dict` is a mutable type.
  - Like lists, arrays.

- \* Keys must be *immutable* (\*).

```
>>> alist = [1, 0]
```

```
>>> adict = { alist : 2 }
```

```
TypeError: unhashable type: 'list'
```

- \* A dictionary can contain a mix of key types.
- \* Stored values can be of any type.





## \* Removing keys:

- `del adict[key]`

Removes *key* from *adict*.

- `adict.pop(key)`

Removes *key* from *adict* and returns the associated value.

- `adict.popitem()`

Removes an arbitrary (*key*, *value*) pair and returns it.

\* `del` and `pop` cause a runtime error if *key* is not in dictionary; `popitem` if it is empty.

# Iteration over dictionaries

- \* `adict.keys()`, `adict.values()`, and `adict.items()` return *views* of the keys, values and key–value pairs.
- \* Views are iterable, but *not* sequences.

```
for item in adict.items():  
    the_key = item[0]  
    the_value = item[1]  
    print(the_key, ':', the_value)
```

# Programming problem(s)

- \* Counting frequency of items:
  - words in a file (or web page);
  - (combinations of) values in a data table.
- \* Building a Markov model (over text, for example).
- \* Cross-referencing data tables with common keys.

# Sets

- \* A *set* is an unordered collection of (immutable) values without duplicates.
- \* Like a dictionary with only keys (no values).
- \* What you can do with a set:
  - Create a new set (empty or from an iterable).
  - Add or remove values.
  - Is a given element in the set? (membership).
  - Mathematical operators: union, intersection, difference (note: not complement!).
  - Enumerate values.

# python's set type

- \* Set literals are written with `{ . . }`, but with elements only, not key–value pairs:

```
>>> aset = { 1, 'c', (2.5, 'b') }
```

- \* `{ }` creates an empty dictionary, not a set!

- \* A set can be created from any iterable:

```
>>> aset = set("AGATGATT")
```

```
>>> aset
```

```
{'T', 'A', 'G'}
```

- No duplicate elements in the set.
- No order of elements in the set.

# Set operators

`elem in aset`

membership ( $e \in A$ )

`aset.issubset(bset)`

subset ( $A \subseteq B$ )

`aset | bset`

union ( $A \cup B$ )

`aset & bset`

intersection ( $A \cap B$ )

`aset - bset`

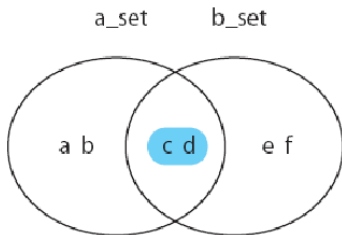
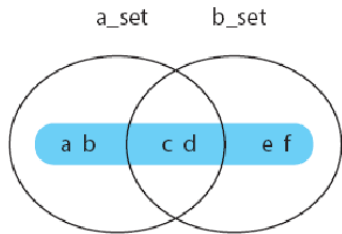
difference ( $A \setminus B, A - B$ )

`aset ^ bset`

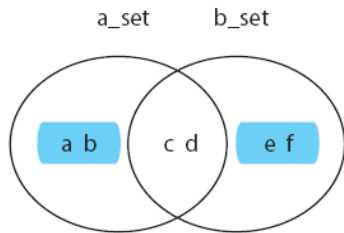
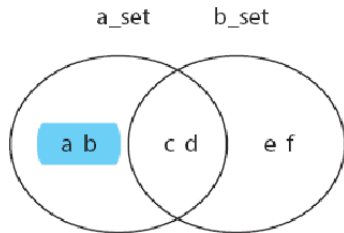
symmetric difference

- \* Set operators return a new result set, and do not modify the operands.
- \* Also exist as methods (`aset.union(bset)`, `aset.intersection(bset)`, etc).

- \* The union of `a_set` and `b_set` is the set of all elements that are in `a_set`, in `b_set`, or in both.
- \* The intersection of `a_set` and `b_set` is the set of elements that are in both `a_set` and `b_set`.

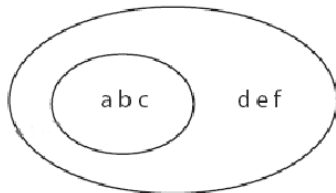


- \* The difference of  $a\_set$  and  $b\_set$  is the set of elements in  $a\_set$  that are not in  $b\_set$ .
- \* The symmetric difference of  $a\_set$  and  $b\_set$  is the set of elements that are in either but not in both.





- ★  $a\_set$  is a subset of  $b\_set$  iff every element in  $a\_set$  is also in  $b\_set$ .
- ★  $A \subseteq B$  iff  $A \cap B = A$ .

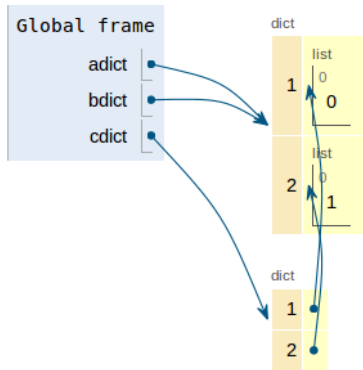


(Image from Punch & Enbody)

# Copying

- \* Dictionaries and sets are mutable objects.
- \* Like lists, dictionaries and sets store *references* to values.
- \* `dict.copy()` and `set.copy()` create a *shallow* copy of the dictionary or set.
  - New dictionary / set, but containing references to the same values.
  - Dictionary keys and set elements are immutable, so shared references do not matter.
  - Values stored in a dictionary can be mutable.

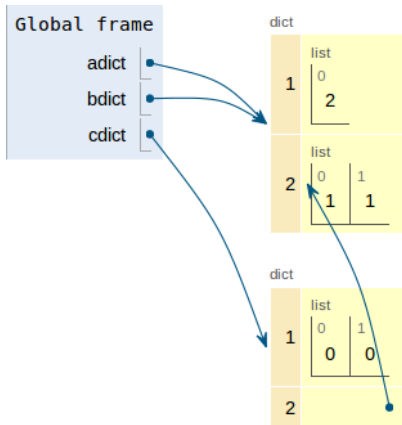
```
adict = {1:[0],2:[1]}  
bdict = adict  
cdict = adict.copy()  
bdict[1] = [2]  
cdict[1] = [0, 0]  
adict[2].append(1)
```



```

adict = {1:[0],2:[1]}
bdict = adict
cdict = adict.copy()
bdict[1] = [2]
cdict[1] = [0, 0]
adict[2].append(1)

```



# Programming problem: Frequent words

- \* Read in a text file.
- \* Print the words in order from most common to least common.