Dictionaries and Sets

- dict
- set
- frozenset
- set/dict comprehensions

Dictionaries (type dict)

$$\{\, key_1 \texttt{:}\ value_1 \texttt{,}\ \ \ldots \texttt{,}\ \ key_k \texttt{:}\ value_k \,\}$$

- Stores a mutable set of (key, value) pairs, denoted *items*, with distinct keys
- Constructing empty set: `dict()` or `{}`
- *dict* [*key*] lookup for key in dictionary, and returns associated value. Key must be present otherwise a KeyError is raised.
- *dict* [*key*] = *value* assigns value to *key*, overriding exising value if present.

key	value
'a'	7
'foo'	'42nd'
5	29
'5'	44
5.5	False
False	True

distinct keys,
i.e. not ”==”

https://docs.python.org/3/tutorial/datastructures.html#dictionaries

Dictionaries (type dict)

Python shell

```
> d = {'a': 42, 'b': 57}
> d
| {'a': 42, 'b': 57}

> d.keys()
| dict_keys(['a', 'b'])

> list(d.keys())
| ['a', 'b']

> d.items()
| dict_items([('a', 42), ('b', 57)])

> list(d.items())
| [('a', 42), ('b', 57)]
```

```
> for key in d:
          print(key)
| a
| b

> for key, val in d.items():
          print("Key", key, "has value", val)
| Key a has value 42
| Key b has value 57

> {5: 'a', 5.0: 'b'}
| {5: 'b'}
```

⚠

Python shell

```
> surname = dict(zip(['Donald', 'Mickey', 'Scrooge'], ['Duck', 'Mouse', 'McDuck']))
> surname['Mickey']
| 'Mouse'
```

Dictionary operation	Description
len(d)	Items in dictionary
d[key]	Lookup key
d[key] = value	Update value of key
del d[key]	Deletes an existing key
key in d	Key membership
key not in d	Key non-membership
clear()	Remove all items
copy()	Shallow copy
get(key)	d[key] if key in dictionary, otherwise `None`
items()	*View* of the dictionaries items
keys()	*View* of the dictionaries keys
values()	*View* of the dictionaries values
pop(key)	Remove key and return previous value
popitem()	Remove and return an arbitrary item
update()	Update key/value pairs from another dictionary

https://docs.python.org/3.6/library/stdtypes.html#mapping-types-dict

Order returned by `list(d.keys())` ?

The Python (3.6.4) Tutorial
5.5 Dictionaries

"Performing list(d.keys()) on a dictionary returns a list of all the keys used in the dictionary, in **arbitrary order** (if you want it sorted, just use sorted(d.keys()) instead)."

docs.python.org/3/tutorial/datastructures.html

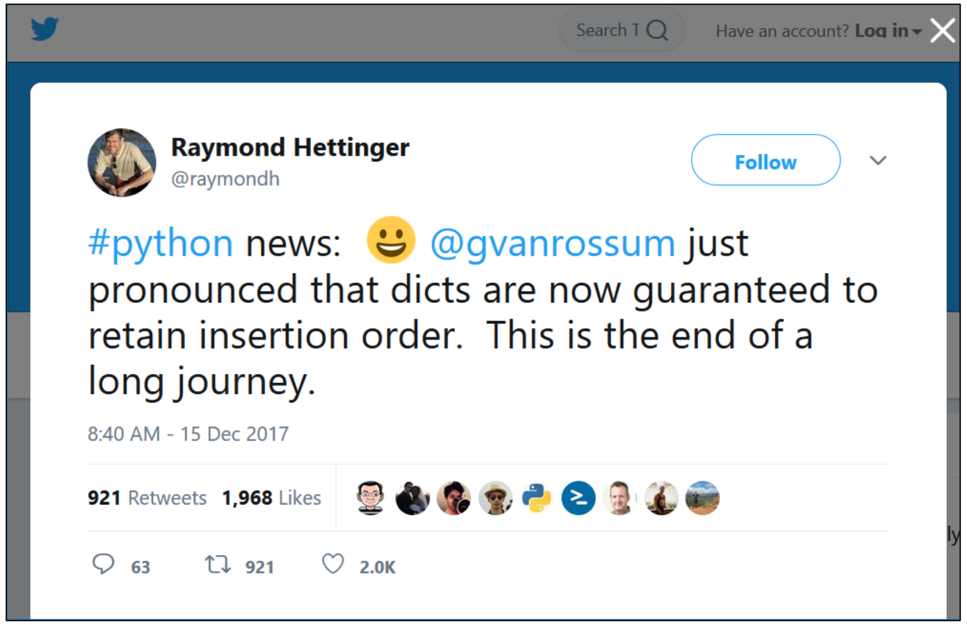
The Python (3.6.4) Standard Library
4.10.1. Dictionary view objects

"Keys and values are iterated over in an **arbitrary order** which is non-random, varies across Python implementations, and depends on the dictionary's history of insertions and deletions."

docs.python.org/3/library/stdtypes.html

Python 3.6.4 shell

```
> d = {'d': 1, 'c': 2, 'b': 3, 'a':4}
> d['x'] = 5    # new key at end
> d['c'] = 6    # overwrite value
> del d['b']    # remove key 'b'
> d['b'] = 7    # reinsert key 'b' at end
> d
| {'d': 1, 'c': 6, 'a': 4, 'x': 5, 'b': 7}
```



Raymond Hettinger
@raymondh

#python news: 😃 @gvanrossum just pronounced that dicts are now guaranteed to retain insertion order. This is the end of a long journey.

8:40 AM - 15 Dec 2017

921 Retweets 1,968 Likes

63 921 2.0K

Raymond Hettinger @ Twitter

See also Raymond's talk @ PyCon 2017
Modern Python Dictionaries
A confluence of a dozen great ideas

Dictionary comprehension

- Similarly to creating a list using list comprehension, one can create a set of key-value pairs:

{*key* : *value* `for` *variable* `in` *list*}

Python shell

```
> names = ['Mickey', 'Donald', 'Scrooge']
> dict(enumerate(names, start=1))
| {1: 'Mickey', 2: 'Donald', 3: 'Scrooge'}
> {name: idx for idx, name in enumerate(names, start=1)}
| {'Donald': 2, 'Mickey': 1, 'Scrooge': 3}
```

Sets (set and frozenset)

$$\{\, value_1 ,\ \ \ldots ,\ \ value_k \,\}$$

- Values of type `set` represent mutable sets, where "==" elements only appear once
- Do **not** support: indexing, slicing
- `frozenset` is an immutable version of `set`

Python shell

```
> S = {2,5,'a','c'}
> T = {3,4,5,'c'}
> S | T
| {2, 3, 4, 5, 'a', 'c'}
> S & T
| {5, 'c'}
```

```
> S ^ T
| {2, 3, 4, 'a'}
> S - T
| {2, 'a'}
> {4, 5, 5.0, 5.1}
| {4, 5, 5.1}
```

⚠

Operation	Description
S | T	Set union
S & T	Set intersection
S – T	Set difference
S ^ T	Symmetric difference
set()	Empty set ({} = empty dictionary)
set(L)	Create set from list
x in S	Membership
x not in S	Non-membership
S.isdisjoint(T)	Disjoint sets
s <= T	Subset
S < T	Proper subset
S >= T	Superset
S > T	Proper superset
len(S)	Size of S

https://docs.python.org/3/tutorial/datastructures.html#sets
https://docs.python.org/3.6/library/stdtypes.html#set-types-set-frozenset

Question – What value has the expression ?

```
sorted( { 5, 5.5, 5.0, '5' } )
```

a) `{'5', 5, 5.0, 5.5}`

b) `{5, 5.5}`

c) `['5', 5, 5.0, 5.5]`

d) `['5', 5, 5.5]`

☹ e) TypeError

f) Don’t know

Set comprehension

- Similarly to creating a list using list comprehension, one can create a set of values (also using nested for- and if-statements):

  `{`*value* `for` *variable* `in` *list*`}`

- A value occurring multiple times as *value* will only be included once

primes_set.py

```
n = 101
not_primes = {m for f in range(2, n) for m in range(2*f, n, f)}
primes = set(range(2, n)) - not_primes
```

Python shell

```
> L = ['a', 'b', 'c']
> {(x,(y,z)) for x in L for y in L for z in L if x != y and y != z and z != x}
| {('a',('b','c')),('a',('c','b')),('b',('a','c')),...,('c',('b','a'))}
```

Hash, equality, and immutability

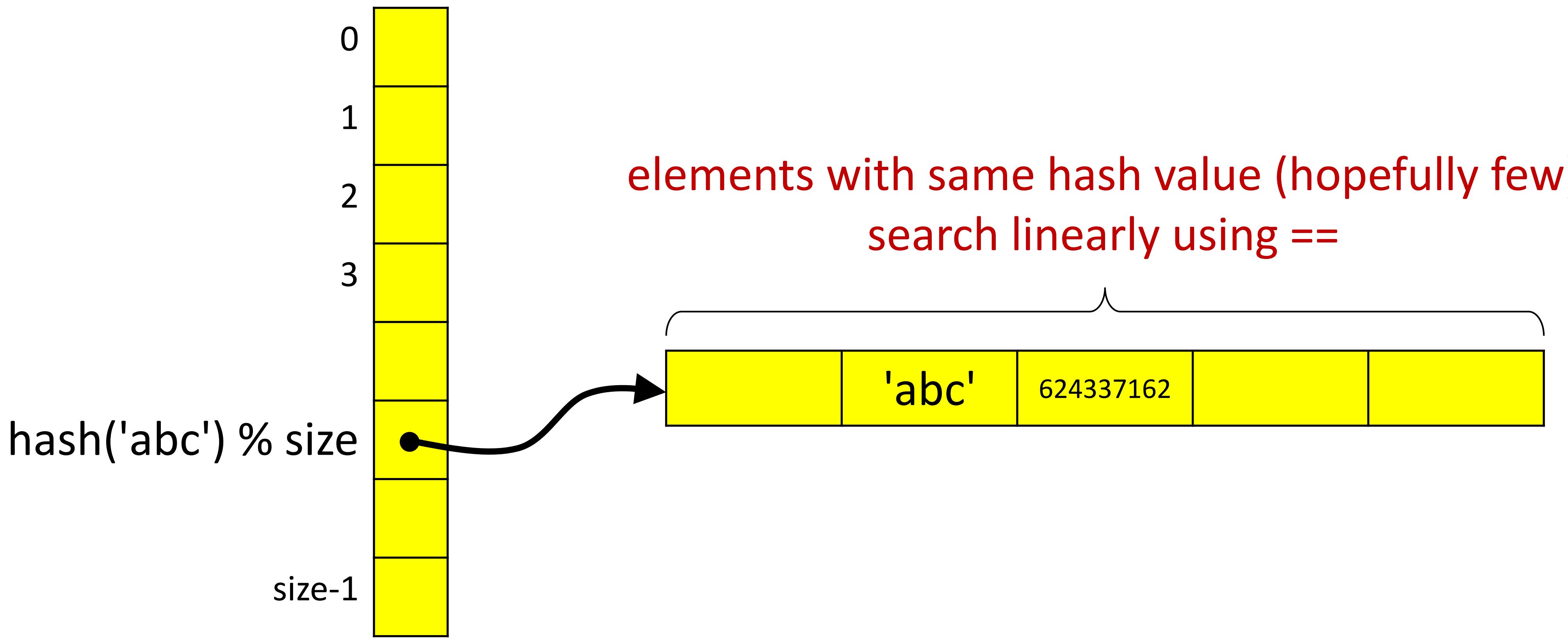
- Keys for dictionaries and sets must be *hashable*, i.e. have a `__hash__()` method returning an integer that does not change over their lifetime and an `__eq__()` method to check for equality with "==".

`'abc'.__hash__()` could e.g. return `624337162`

`(624337162).__hash__()` would also return `624337162`

- All built-in immutable types are hashable. In particular tuples of immutable values are hashable. I.e. trees represented by nested tuples like `((('a'),'b'),('c',('d','e')))` can be used as dictionary keys or stored in a set.

Skecth of internal set implementation



Raymond Hettinger,
Modern Python Dictionaries
- A confluence of a dozen great ideas

(Simple) functions

- You can define your own functions using:

  `def` *function-name* (*var*$_1$, ..., *var*$_k$):

  body code

- If the body code executes

  `return` *expression*

  the result of *expression* will be returned by the function. If expression is omitted or the body code terminates without performing `return`, then `None` is returned.

- When *calling* a function *name* (*value*,..., *value*$_k$) body code is executed with *var*$_i$=*value*$_i$

Python shell

```
> def sum3(x, y, z):
      return x+y+z

> sum3(1, 2, 3)
| 6
> sum3(5, 7, 9)
| 21

> def powers(L, power):
      P = [x**power for x in L]
      return P

> powers([2,3,4], 3)
| [8, 27, 64]
```

Question – What tuple is printed ?

```
def even(x):
    if x % 2 == 0:
        return True
    else:
        return False

print((even(7), even(6)))
```

a) `(False, False)`
b) [✓ `(False, True)`]
c) `(True, False)`
d) `(True, True)`
e) Don’t know

Handin 3 & 4 – Triplet distance (Dobson, 1975)

```
((('A','F'),'B'),('D',('C','E')))

              (a)

               /\
              /  \
             /    \
            /      \
           /        \
          /          \
         /\          /\
        /  \        /  \
       /    \      /    \
      /\    'B'  'D'    /\
     /  \              /  \
   'A'  'F'          'C'  'E'
```

```
(((('D','A'),'B'),'F'),('C','E'))

                  (b)

                   /\
                  /  \
                 /    \
                /      \
               /        \
              /          \
             /\          /\
            /  \        /  \
           /    \     'C'  'E'
          /      \
         /\     'F'
        /  \
       /    \
      /\    'B'
     /  \
   'D'  'A'
```

How similar are the trees?

arxiv.org/abs/1706.10284

Handin 3 & 4 – Triplet distance (Dobson, 1975)

Consider all $\binom{n}{3}$ subsets of size three, and count how many do not have identical substructure (topology) in the two trees.

```
     /\
    /  \
   /    \
 'D'    /\
       /  \
     'A'  'F'

('D',('A','F'))
```

canonical ordering

```
             (a)

     anchor *
           / \
          /   \
         /     \
        /       \
       /         \
      /\         /\
     /  \       /  \
    /    \     /    \
   /\    'B' 'D'    /\
  /  \             /  \
'A'  'F'         'C'  'E'
```

{'A','D','F'}

```
                 (b)

                /\
               /  \
              /    \
             /      \
            /        \
           /          \
   anchor *           /\
         / \         /  \
        /   \      'C'  'E'
       /     \
      /\     'F'
     /  \
    /    \
   /\    'B'
  /  \
'D'  'A'
```

```
     /\
    /  \
   /    \
 'F'    /\
       /  \
     'A'  'D'

('F',('A','D'))
```

arxiv.org/abs/1706.10284