Digital World (2018)

Week 5, S1: Dictionaries; Modularity

Chris Poskitt

SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

Refresher: from **lists to dictionaries**

- a list maps indices to values

- a dictionary maps (immutable) keys to values

d =

‘Rowan Atkinson’	→	*‘+44 1356 345867’*
‘John Cleese’	→	*‘+44 1904 534534’*
‘Cersei Lannister’	→	*‘+353 112 112’*
‘Chris Poskitt’	→	*‘+65 8888 8888’*

d[‘key’] = value   d.keys()   d.values()   d.items()

d.copy()   copy.deepcopy(d)   x in d.keys()   x not in d.values()

Refresher: **lists** vs. **dictionaries**

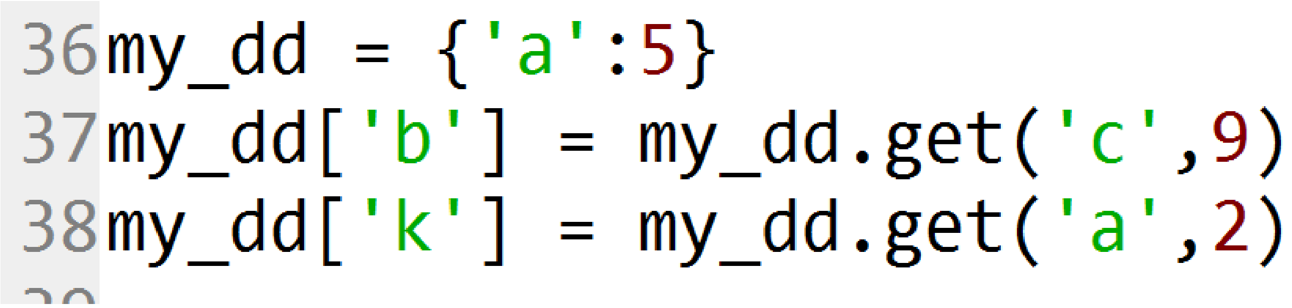
b.socrative.com, POSKITT5665

Which of the following **is** a difference between lists and dictionaries?

- A. List elements cannot be mutable, but dictionary values can be mutable
- B. Assigning to an index that does not exist in a list is an error, but assigning a value to a key that does not exist in a dictionary is not
- C. A list can contain a dictionary as one of its elements, but a dictionary cannot contain a list as one of its values
- D. There is a `dict` constructor that creates a dictionary from a suitable object, but there is no `list` constructor that similarly creates lists

Dictionaries: **get** function

b.socrative.com, POSKITT5665



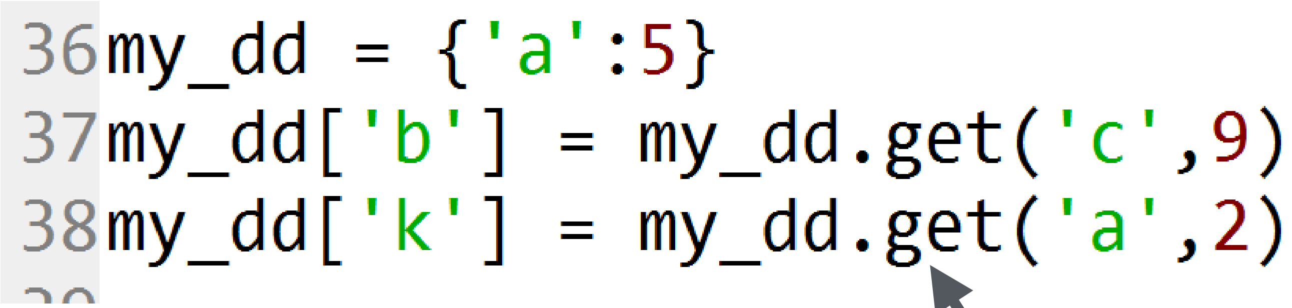
```
my_dd = {'a':5}
my_dd['b'] = my_dd.get('c',9)
my_dd['k'] = my_dd.get('a',2)
```

After this code, my_dd is

A. {'a': 5, 'b': 9, 'k': 2}
B. {'a': 5, 'b': 9, 'k': 5}
C. {'a': 5, 'b': 5, 'k': 2}
D. {'a': 5, 'b': 5, 'k': 5}
E. Error

Dictionaries: **get** function

b.socrative.com, POSKITT5665



```
my_dd = {'a':5}
my_dd['b'] = my_dd.get('c',9)
my_dd['k'] = my_dd.get('a',2)
```

if there is an "a" key, return its value; if not, return 2

After this code, my_dd is

A. {'a': 5, 'b': 9, 'k': 2}
B. {'a': 5, 'b': 9, 'k': 5}
C. {'a': 5, 'b': 5, 'k': 2}
D. {'a': 5, 'b': 5, 'k': 5}
E. Error

What is the **get** function good for?

- suppose we want to represent a sparse matrix

What is the **get** function good for?

- suppose we want to represent a sparse matrix

```
M = [ [0, 0, 0, 1, 0],
      [0, 0, 0, 0, 0],
      [0, 2, 0, 0, 0],
      [0, 0, 0, 0, 0],
      [0, 0, 0, 3, 0] ]
```

What is the **get** function good for?

- suppose we want to represent a sparse matrix

```
M = [ [0, 0, 0, 1, 0],
      [0, 0, 0, 0, 0],
      [0, 2, 0, 0, 0],
      [0, 0, 0, 0, 0],
      [0, 0, 0, 3, 0] ]
```

```
M[row_idx][col_idx]
```

What is the **get** function good for?

- suppose we want to represent a sparse matrix

```
M = [ [0, 0, 0, 1, 0],
      [0, 0, 0, 0, 0],
      [0, 2, 0, 0, 0],
      [0, 0, 0, 0, 0],
      [0, 0, 0, 3, 0] ]
```

```
M[row_idx][col_idx]
```

M =

(0, 3) ⟶ *1*
(2,1) ⟶ *2*
(4,3) ⟶ *3*

What is the **get** function good for?

- suppose we want to represent a sparse matrix

```
M = [ [0, 0, 0, 1, 0],
      [0, 0, 0, 0, 0],
      [0, 2, 0, 0, 0],
      [0, 0, 0, 0, 0],
      [0, 0, 0, 3, 0] ]
```

```
M[row_idx][col_idx]
```

M =

$(0, 3) \longrightarrow 1$

$(2, 1) \longrightarrow 2$

$(4, 3) \longrightarrow 3$

```
M.get((row_idx, col_idx), 0)
```

Building more complex programs

Building more complex programs

- a complex problem may initially seem overwhelming

Building more complex programs

- a complex problem may initially seem overwhelming

- stop! identify simpler, self-contained subproblems

  => *separate subproblems into their own functions*

Building more complex programs

- a complex problem may initially seem overwhelming

- stop! identify simpler, self-contained subproblems

  => *separate subproblems into their own functions*

- implement the overall problem by calling the functions for those subproblems

  => *more readable; more manageable; more maintainable*

Building more complex programs

- a complex problem may initially seem overwhelming

- stop! identify simpler, self-contained subproblems

  => *separate subproblems into their own functions*

- implement the overall problem by calling the functions for those subproblems

  => *more readable; more manageable; more maintainable*

⚠ modularity: degree by which complex program can be separated and recombined

Libraries provide you with *modular* code

Libraries provide you with *modular* code

```
import random
```

Libraries provide you with *modular* code

```
import random
```

```
from math import sqrt
```

Libraries provide you with *modular* code

```
import random
```

```
from math import sqrt
```

```
from math import *
```

Libraries provide you with *modular* code

```
import random
```

```
from math import sqrt
```

```
from math import *
```

⚠ *you can write your own*

```
import yourlib
```

Question CS1, aka *SUTDCraps*

- in round one, roll two standard dice

  => if the sum is 2, 3, or 12 (“craps”), you lose
  => if the sum is 7 or 11 (“natural”), you win
  => if the sum is another value X, you get X points; next round

- in every subsequent round, roll two standard dice

  => if the sum is 7, you lose
  => if the sum is equal to your X points, you win
  => if the sum is another value, go to the next round

Question CS1, aka *SUTDCraps*

- in round one, roll two standard dice

    => if the sum is 2, 3, or 12 ("craps"), you lose
    => if the sum is 7 or 11 ("natural"), you win
    => if the sum is another value X, you get X points; next round

- in every subsequent round, roll two standard dice

    => if the sum is 7, you lose
    => if the sum is equal to your X points, you win
    => if the sum is another value, go to the next round

! *make life easier now*: "factor out" smaller tasks into separate functions
make life easier in the future: maximise re-use of those functions

Summary

- the get function can define a default value for when a dictionary key does not exist

- solve complex problems by dividing them into smaller tasks

- function composition leads to more readable code

  => *create separate functions for smaller, self-contained tasks*

- re-using functions leads to more maintainable code

  => *"updating* ***once*** *updates it* ***everywhere****"*