Every Boilermaker Engineer Codes: 101
Entry-Level Programming in Python

Lecture 09a

Dr. John H. Cole
<jhcole@purdue.edu>

PURDUE UNIVERSITY®

COLLEGE OF ENGINEERING

Spring 2021

Part I

Dictionaries

Table of Contents

Tables

DICTIONARY a table like object that maps *keys* to *values*

- are mutable
- dynamically sized

KEY any immutable object

- usually `ints` or `strings`
- can also use floats, tuples, booleans, NoneType etc.
- cannot use lists or dictionaries
- must be unique within each dictionary

VALUE any arbitrary object

Keys	Values
`2016`	`'Cubs'`
`'Texas'`	`'Austin'`
`0.5`	`'half'`
`False`	`[0, '', ()]`
`()`	`None`
`None`	`[]`

Syntax

```
d = {key1:value1, key2:value2, ...}
```

- enclosed in braces {}
- keys and values separated by colons ':'
- key-value pairs separated by commas ','

Terminal

```
>>> type({})
<class 'dict'>
>>> a = {'one': 1, 'two': 2, 'three': 3}
>>> a
{'one': 1, 'two': 2, 'three': 3}
>>> type(a)
<class 'dict'>
```

dict(args)

`dict(args)` returns a new dictionary from its arguments

accepts:

- keyword arguments
- iterables that produce pairs of values
- dictionaries

Terminal

```
>>> b = dict(one=1, two=2, three=3)
>>> c = dict([('two', 2), ('one', 1), ('three', 3)])
>>> d = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
>>> e = dict({'three': 3, 'one': 1, 'two': 2})
>>> a == b == c == d == e
True
```

Nesting Dictionaries

- dictionaries can be arbitrarily nested

Terminal

```
>>> a = { 'dict1': {'key_1': 'value_1'},
...       'dict2': {'key_2': 'value_2'}}
>>> b = {1: {'room': 'EE 170',   'time': '4:30'},
...      2: {'room': 'PHYS 203', 'time': '6:30'}}
```

d[key]

d[key] returns the value for key

- raises a KeyError if key does not exist

Terminal

```
>>> a = {'one':1, 'two':2, 'five':5}
>>> a['one']
1
>>> a['four']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'four'
>>>
```

d.get(key[, default])

`d.get(key)`	returns the value for key or None
`d.get(key, default)`	returns the value for key or `default`

- `default` defaults to None
- returns `default` if key does not exist

Terminal

```
>>> a = {'one':1, 'two':2, 'five':5}
>>> a.get('one')
1
>>> a.get('four')
>>> a.get('four', 4)
4
>>> a.get('five', 4)
5
>>>
```

d.pop(key[, default])

`d.pop(key)` remove and return the `value` for key

- raises a `KeyError` if key does not exist

`d.pop(key, default)` remove and return the `value` or `default`

- returns `default` if key does not exist

Terminal

```
>>> a = {'one':1,'two':2,'five':5}
>>> a.pop('four')
Traceback (most recent call last):
  ...
KeyError: 'four'
>>> a.pop('four', 4)
4
```

Terminal

```
>>> a.pop('five')
5
>>> a
{'one': 1, 'two': 2}
>>>
```

d.popitem()

`d.popitem()` remove and return the last key-value pair

- useful as a last in, first out (LIFO) stack
- raises a `KeyError` if the dictionary is empty
- return order not guaranteed in Python versions before 3.7

Terminal

```
>>> a = {'one':1,'two':2,'five':5}
>>> a.popitem()
('five', 5)
>>> a.popitem()
('two', 2)
>>> a.popitem()
('one', 1)
>>> a
{}
```

d[key] = value

`d[key] = value` set d[key] to value

- overwrites existing key values
- adds new key-value pairs if the key does not exist
- does not change order of existing keys

Terminal

```
>>> a = {'three':5}
>>> a['one'] = 1
>>> a['two'] = 2
>>> a
{'three': 5, 'one': 1, 'two': 2}
>>> a['three'] = 3
>>> a
{'three': 3, 'one': 1, 'two': 2}
```

d.update([other])

`d.update([other])` update `d` with key-value pairs from `other`

- overwrites existing key values
- adds new key-value pairs if the key does not exist
- accepts a dictionary, iterables of length 2, or keyword arguments

Terminal

```
>>> a = {}
>>> a.update({'one':1, 'two':2})
>>> a.update([('one', 1), ('two', 2)])
>>> a.update(one = 1, two = 2)
>>> a
{'one': 1, 'two': 2}
```

del

`del d[key]` removes `d[key]` from `d`

- raises a `KeyError` if `key` does not exist

Terminal

```
>>> a = {'one':1,'two':2,'five':5}
>>> a
{'one': 1, 'two': 2, 'five': 5}
>>> del a['five']
>>> a
{'one': 1, 'two': 2}
```

d.clear()

`d.clear()` removes all items from `d`

Terminal

```
>>> a = {'one':1,'two':2,'five':5}
>>> a
{'one': 1, 'two': 2, 'five': 5}
>>> a.clear()
>>> a
{}
```

Views

`d.keys()`	return a new view of keys of `d`
`d.values()`	return a new view of values of `d`
`d.items()`	return a new view of items (`(key, value)` pairs) of `d`

- changes to `d` are reflected in its views
- can be iterated over in loops
- support inclusion testing

Terminal

```
>>> a = dict(one=1,
...          two=2)
>>> k = a.keys()
>>> v = a.values()
>>> i = a.items()
>>>
```

Terminal

```
>>> k
dict_keys(['one', 'two'])
>>> v
dict_values([1, 2])
>>> i
dict_items([('one', 1), ('two', 2)])
```

Views – continued

`d.keys()`	return a new view of keys of `d`
`d.values()`	return a new view of values of `d`
`d.items()`	return a new view of items (`(key, value)` pairs) of `d`

- changes to `d` are reflected in its views
- can be iterated over in loops
- support inclusion testing

Terminal

```
>>> a
{'one': 1, 'two': 2}
>>> del a['one']
>>> a
{'two': 2}
>>>
```

Terminal

```
>>> k
dict_keys(['two'])
>>> v
dict_values([2])
>>> i
dict_items([('two', 2)])
```

Sorting

- views can be sorted()

Terminal

```
>>>  a = {'one': 1, 'two': 2, 'five': 5, 'three': 3}
>>> sorted(a)
['five', 'one', 'three', 'two']
>>> sorted(a.keys())
['five', 'one', 'three', 'two']
>>> sorted(a.values())
[1, 2, 3, 5]
>>> sorted(a.items())
[('five', 5), ('one', 1), ('three', 3), ('two', 2)]
```

Reversing

- views can be `reversed()` (in Python 3.8+)
- returns an iterator that traverses elements in reverse order

Terminal

```
>>> a
{'one': 1, 'two': 2, 'five': 5, 'three': 3}
>>> list(reversed(a))
['three', 'five', 'two', 'one']
>>> list(reversed(a.keys()))
['three', 'five', 'two', 'one']
>>> list(reversed(a.values()))
[3, 5, 2, 1]
>>> list(reversed(a.items()))
[('three', 3), ('five', 5), ('two', 2), ('one', 1)]
```

list(d) AND tuple(d)

`list(d)`	returns all the keys in d as a list
`tuple(d)`	returns all the keys in d as a tuple

- values are not returned
- can call `list()` or `tuple()` on a views too

Terminal

```
>>> a = dict(one=1, two=2, five=5, three=3)
>>> list(a.keys())
['one', 'two', 'five', 'three']
>>> tuple(a.keys())
('one', 'two', 'five', 'three')
>>> list(a.values())
[1, 2, 5, 3]
>>> tuple(a.items())
(('one', 1), ('two', 2), ('five', 5), ('three', 3))
```

iter(d)

`iter(d)` returns an iterator over the keys in d

- is a shortcut for `iter(d.keys())`
- can call `iter()` on a views too

Terminal

```
>>> a = dict([('one', 1), ('two', 2), ('five', 5)])
>>> a
{'one': 1, 'two': 2, 'five': 5}
>>> i = iter(a)
>>> next(i)
'one'
>>> next(i)
'two'
>>> next(i)
'five'
```

d.copy()

d.copy() returns a *shallow* copy of d

Terminal

```
>>> a = dict(one=1, two=2, five=5)
>>> b = a.copy()
>>> a
{'one': 1, 'two': 2, 'five': 5}
>>> b
{'one': 1, 'two': 2, 'five': 5}
>>> a.popitem()
('five', 5)
>>> a
'one': 1, 'two': 2
>>> b
'one': 1, 'two': 2, 'five': 5
```

Terminal

```
>>> c = dict(l=[1,2,5])
>>> d = c.copy()
>>> c
{'l': [1, 2, 5]}
>>> d
{'l': [1, 2, 5]}
>>> c['l'].pop()
5
>>> c
{'l': [1, 2]}
>>> d
{'l': [1, 2]}
```

len(d)

`len(d)` returns the number of items in `d`

Terminal

```
>>> a = dict(one=1, two=2, five=5)
>>> len(a)
3
```

Inclusion

`key in d` returns True if d has the key key

`key not in d` returns False if d has the key key

Terminal

```
>>> a = dict(one=1, two=2, five=5)
>>> 'five' in a
True
>>> 'five' not in a
False
>>> 'three' in a
False
>>> 'three' not in a
True
```

Comparison

`a == b` returns True if `a` and `b` have the same key-value pairs

- key-value pair ordering does not matter
- other comparisons (e.g. <, <=, >=, >) raise a `TypeError`

Terminal

```
>>> a = dict(one=1, two=2)
>>> b = dict(one=1, five=5)
>>> c = dict(two=2, one=1)
>>> a == b
False
>>> a == c
True
>>> id(a) == id(c)
False
```

Key Loops

- `for` loop, iterates over keys by default

Terminal

```
>>> a = dict(one=1,
             two=2,
             five=5)
>>> for k in a:
...     print(k)
...
one
two
five
```

Terminal

```
>>> a = dict(one=1,
             two=2,
             five=5)
>>> for k in a.keys():
...     print(k)
...
one
two
five
```

Value Loops

- use a values view to iterate over values

Terminal

```
>>> a = dict(one=1, two=2, five=5)
>>> for v in a.values():
...     print(v)
...
1
2
5
```

Item Loops

- use an items view to iterate over key-value pairs

Terminal

```
>>> a = dict(one=1, two=2, five=5)
>>> for k,v in a.items():
...     print(k,v)
...
one 1
two 2
five 5
```

Thanks for watching!