BIOS6642

Dictionaries

Fuyong Xing

Department of Biostatistics and Informatics

Colorado School of Public Health

University of Colorado Anschutz Medical Campus

Outline

- **Dictionaries**
- Keyword arguments revisited
- Enumeration

Dictionaries

- A dictionary is a set of *key:value* pairs, representing a mapping from keys to values.
- Dictionaries are indexed by keys, which can be integers, floating-point numbers, strings or tuples that do not contain any mutable objects.
- An empty dictionary can be created by a pair of curly braces, *{}*, or the function, *dict*.
- Square brackets can be used to add items to an existing dictionary.
- Dictionaries are mutable.

Dictionaries

When using keys to look up corresponding values in a dictionary, an exception is raised if the key is not in the dictionary.

```
>>> d = {}  # Make an empty dictionary
>>> d
{}
>>> # Add an element
... d['Fred'] = 44
>>> d
{'Fred': 44}
>>> # Add another element
... d['Ella'] = 31
>>> d
{'Fred': 44, 'Ella': 31}
>>> d['Fred']
44
>>> d['Ella']
31
>>> d['Zach']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Zach'
```

Dictionaries

- Use the *in* operator to check whether a key is in the dictionary.

```
if 'Fred' in d:         # Check to see if 'Fred' is a valid key
    print(d['Fred'])    # Print the value associated with key 'Fred'
else:
    print('\'Fred\' is not a key in d')  # Warn user of missing key
```

Dictionaries

- Create a dictionary from two lists

```
>>> names = ['Fred', 'Ella', 'Owen', 'Zoe']
>>> numbers = [4174, 2287, 5003, 2012]
>>> names
['Fred', 'Ella', 'Owen', 'Zoe']
>>> numbers
[4174, 2287, 5003, 2012]
>>> d = dict(zip(names, numbers))
>>> d
{'Zoe': 2012, 'Owen': 5003, 'Fred': 4174, 'Ella': 2287}
```

Dictionaries

- Other ways to create new dictionaries
  - Place a comma-separated list of *key:value* pairs within the braces

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'jack': 4098, 'sape': 4139, 'guido': 4127}
>>> tel['jack']
4098
```

  - Use dictionary comprehensions

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

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

Dictionaries

- Other ways to create new dictionaries
  - Use keyword arguments

```
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

- Delete a pair of *key:value* using the *del* statement

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'jack': 4098, 'guido': 4127, 'irv': 4127}
```

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

Dictionaries

- Dictionary methods

Methods	Descriptions
keys()	Return a new view of the dictionary's keys.
values()	Return a new view of the dictionary's values.
items()	Return a new view of the dictionary's items
get(key[, default])	Return the value for key if key is in the dictionary, else default. If default is not given, it defaults to *None.*
pop(key[, default])	If key is in the dictionary, remove it and return its value, else return default. If default is not given and key is not in the dictionary, a KeyError is raised.

https://docs.python.org/3/library/stdtypes.html#mapping-types-dict

Dictionaries

- Dictionary methods

Codes

```
d = {'Fred': 44, 'Ella': 39, 'Owen': 40, 'Zoe': 41}
for k in d.keys():
    print(k, end=' ')
print()
```

Output

```
Fred Ella Owen Zoe
```

Codes

```
d = {'Fred': 44, 'Ella': 39, 'Owen': 40, 'Zoe': 41}
for v in d.values():
    print(v, end=' ')
print()
```

Output

```
44 39 40 41
```

Dictionaries

- Dictionary methods

Codes

```
d = {'Fred': 44, 'Ella': 39, 'Owen': 40, 'Zoe': 41}
for k, v in d.items():
    print(k, v)
```

Output

```
Fred 44
Ella 39
Owen 40
Zoe 41
```

Codes

```
d = {'Fred': 44, 'Ella': 39, 'Owen': 40, 'Zoe': 41}
print(d.get('Ella'))
print(d.get('Zach', 0))
```

Output

```
39
0
```

Dictionaries

- Dictionary methods

Codes

```
d = {'Fred': 44, 'Ella': 39, 'Owen': 40, 'Zoe': 41}
print(d.pop('Ella'))
print(d)
```

Output

```
39
{'Fred': 44, 'Owen': 40, 'Zoe': 41}
```

Codes

```
d = {'Fred': 44, 'Ella': 39, 'Owen': 40, 'Zoe': 41}
print(d.pop('Zach'))
```

Output

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-20-6361323cdd43> in <module>()
      1 d = {'Fred': 44, 'Ella': 39, 'Owen': 40, 'Zoe': 41}
----> 2 print(d.pop('Zach'))

KeyError: 'Zach'
```

Dictionaries

Example: translate Spanish to English words

Codes

```
translator = {'uno':'one',
    'dos':'two',
    'tres':'three',
    'cuatro':'four',
    'cinco':'five',
    'seis':'six',
    'siete':'seven',
    'ocho':'eight'}
word = '*'
while word != '': # Loop until user presses return by itself
    # Obtain word from the user
    word = input('Enter Spanish word:')
    if word in translator:
        print(translator[word])
    else:
        print('???') # Unknown word
```

Output

```
Enter Spanish word:dos
two
Enter Spanish word:
???
```

Dictionaries

- Example: count the frequencies of characters in a string

Codes

```
def count(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c] = 1
        else:
            d[c] += 1
    return d

d = count('parrot')
for c in d:
    print(c, d[c])
```

Output

```
p 1
a 1
r 2
o 1
t 1
```

Dictionaries

- Example: count the frequencies of characters in a string

Codes

```
def count(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c] = 1
        else:
            d[c] += 1
    return d

d = count('parrot')
for c in sorted(d.keys()):
    print(c, d[c])
```

Output

```
a 1
o 1
p 1
r 2
t 1
```

Outline

- Dictionaries
- **Keyword arguments revisited**
- Enumeration

Keyword arguments revisited

- The caller can pass its actual parameters in any order using **keyword arguments**.

Codes

```
def process(a, b, c):
    print('a =', a, ' b =', b, ' c =', c)



x = 14
process(1, 2, 3)
process(a=10, b=20, c=30)
process(b=200, c=300, a=100)
process(c=3000, a=1000, b=2000)
process(10000, c=30000, b=20000)
```

Keyword arguments revisited

- The caller can pass its actual parameters in any order using **keyword arguments**.

Codes

```
a = 1   b = 2   c = 3
a = 10   b = 20   c = 30
a = 100   b = 200   c = 300
a = 1000   b = 2000   c = 3000
a = 10000   b = 20000   c = 30000
```

Keyword arguments revisited

- We can define a function to require keyword parameters by prefixing a formal parameter with two asterisks (**).

```
>>> def process(**args):
...     for arg in args:
...         print(arg, '-->', args[arg])
...     print('args =', args)
...
>>> process(num=5, x='Hello', value=True, zz=100)
num --> 5
zz --> 100
value --> True
x --> Hello
args = {'num': 5, 'zz': 100, 'value': True, 'x': 'Hello'}
```

args is a dictionary.

Keyword arguments revisited

- Pass a dictionary to a function that expects multiple positional parameters

Codes

```
def f(a, b, c):
    print('a =', a, ' b =', b, ' c =', c)


f(1, 2, 3)                          # Pass three parameters
dict = {}
dict['b'] = 22
dict['a'] = 11
dict['c'] = 33
f(**dict)
f(**{'a':10, 'b':20, 'c':30})    # Pass a dictionary
```

Output

```
a = 1  b = 2  c = 3
a = 11  b = 22  c = 33
a = 10  b = 20  c = 30
```

Keyword arguments revisited

- A function may have mixed parameters that represent regular positional arguments, variable-length arguments, and keyword arguments.

Codes

```
def cheeseshop(kind, *arguments, **keywords):
    print("-- Do you have any", kind, "?")
    print("-- I'm sorry, we're all out of", kind)
    for arg in arguments:
        print(arg)
    print("-" * 40)
    for kw in keywords:
        print(kw, ":", keywords[kw])
cheeseshop("Limburger", "It's very runny, sir.",
           "It's really very, VERY runny, sir.",
           shopkeeper="Michael Palin",
           client="John Cleese",
           sketch="Cheese Shop Sketch")
```

Keyword arguments revisited

- A function may have mixed parameters that represent regular positional arguments, variable-length arguments, and keyword arguments.

Output

```
-- Do you have any Limburger ?
-- I'm sorry, we're all out of Limburger
It's very runny, sir.
It's really very, VERY runny, sir.
----------------------------------------
shopkeeper : Michael Palin
client : John Cleese
sketch : Cheese Shop Sketch
```

Outline

- Dictionaries
- Keyword arguments revisited
- **Enumeration**

Enumeration

- The following program prints the elements of a list, *lst*, and their indices.

```
for i in range(len(lst)):
    print(i, lst[i])
```

- The program above can be re-written using the *enumerate* function.

```
for i, elem in enumerate(lst):
    print(i, elem)
```

Enumeration

- The *enumerate* function returns an enumerate object that produces tuples. Each tuple pairs an index with its associated element.

```
>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']
>>> list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
>>> list(enumerate(seasons, start=1))
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

Enumeration

- The *enumerate* function can accept a list, a tuple, a dictionary or a set as input.

Codes

```
lst = [10, 20, 30, 40, 50]
t = 100, 200, 300, 400, 500
d = {"A": 4, "B": 18, "C": 0, "D": 3}
s = {1000, 2000, 3000, 4000, 5000}
print(lst)
print(t)
print(d)
print(s)
for x in enumerate(lst):
    print(x, end=" ")
print()
for x in enumerate(t):
    print(x, end=" ")
print()
for x in enumerate(d):
    print(x, end=" ")
print()
for x in enumerate(s):
    print(x, end=" ")
print()
```

Enumeration

- The *enumerate* function can accept a list, a tuple, a dictionary or a set as input.

Output

```
[10, 20, 30, 40, 50]
(100, 200, 300, 400, 500)
{'D': 3, 'C': 0, 'A': 4, 'B': 18}
{5000, 4000, 3000, 2000, 1000}
(0, 10) (1, 20) (2, 30) (3, 40) (4, 50)
(0, 100) (1, 200) (2, 300) (3, 400) (4, 500)
(0, 'D') (1, 'C') (2, 'A') (3, 'B')
(0, 5000) (1, 4000) (2, 3000) (3, 2000) (4, 1000)
```

Readings

- Think Python, chapter 11

References

- *Think Python: How to Think Like a Computer Scientist*, 2nd edition, 2015, by Allen Downey
- *Python Cookbook*, 3rd edition, by David Beazley and Brian K. Jones, 2013
- *Python for Data Analysis*, 2nd edition, by Wes McKinney, 2018
- *Fundamentals of Python Programming,* by Richard L. Halterman