[301] Dictionary Nesting

Tyler Caraza-Harter

Learning Objectives Today

More dictionary operations

- len, in, for loop
- d.keys(), d.values()
- defaults for get and pop, defaultdict

makes coding more convenient

Learning Objectives Today

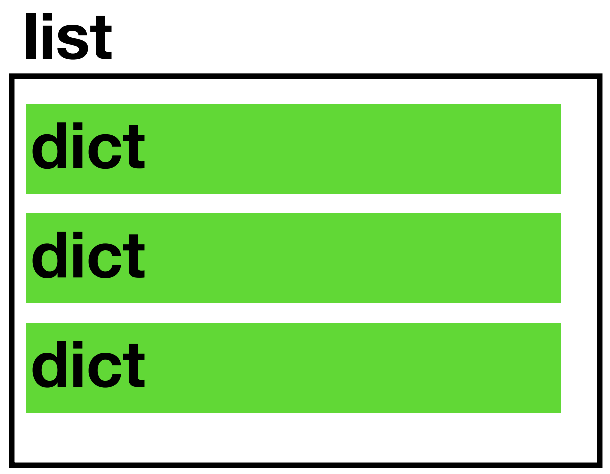
More dictionary operations

- len, in, for loop
- d.keys(), d.values()
- defaults for get and pop, defaultdict

makes coding more convenient

Syntax for nesting (dicts inside dicts, etc)

- indexing/lookup
- step-by-step resolution

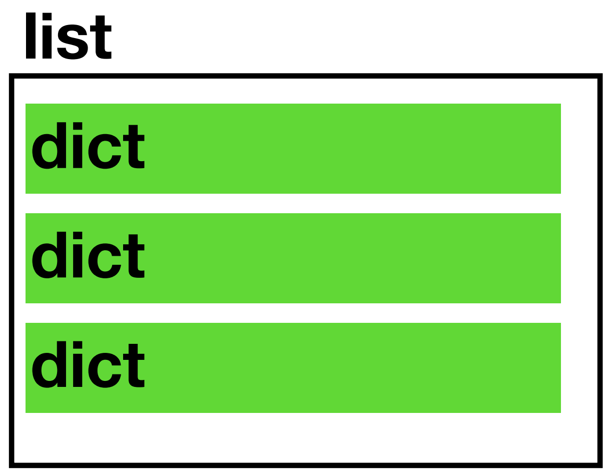


Learning Objectives Today

More dictionary operations

- len, in, for loop
- d.keys(), d.values()
- defaults for get and pop, defaultdict

makes coding more convenient



Syntax for nesting (dicts inside dicts, etc)

- indexing/lookup
- step-by-step resolution

Understand common use cases for nesting

- transition probabilities with Markov chains (dict in dict)
- binning/bucketing (list in dict)
- a more convenient table representation (dict in list)

one of the most common data analysis tasks

we'll generate random English-like texts

Today's Outline

More Dictionary Ops

Probabilities Tables

Markov Chains

Default Dictionaries

Binning

Table Representation

Creation of Empty Dict

Non-empty dict:

```
d = {“a”: “alpha”, “b”: “beta”}
```

Empty dict (way 1):

```
d = {}
```

Empty dict (way 2):

```
d = dict()
```

Creation of Empty Dict

Non-empty dict:

```
d = {“a”: “alpha”, “b”: “beta”}
```

Empty dict (way 1):

```
d = {}
```

Empty dict (way 2):

```
d = dict()
```

Similar for Lists

```
empty_list_1 = []
empty_list_2 = list()
```

len, in, for

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}


print(len(num_words))


print(1 in num_words)


print(“one” in num_words)


for x in num_words:
    print(x)
```

len, in, for

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}

print(len(num_words))  →  4

print(1 in num_words)

print(“one” in num_words)

for x in num_words:
    print(x)
```

len, in, for

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}
```

```
print(len(num_words))
```

4

```
print(1 in num_words)
```

True

```
print(“one” in num_words)
```

```
for x in num_words:
    print(x)
```

len, in, for

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}
```

```
print(len(num_words))
```

4

```
print(1 in num_words)
```

True

```
print(“one” in num_words)
```

False

(it is only checking keys, not vals)

```
for x in num_words:
    print(x)
```

len, in, for

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}
```

```
print(len(num_words))
```

4

```
print(1 in num_words)
```

True

```
print(“one” in num_words)
```

False

(it is only checking keys, not vals)

```
for x in num_words:
    print(x)
```

2
1
0
3

(for iterates over keys, not vals)

(note there is no order here)

len, in, for

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}
```

```
print(len(num_words))
```

4

```
print(1 in num_words)
```

True

```
print(“one” in num_words)
```

False

(it is only checking keys, not vals)

```
for x in num_words:
    print(x, num_words[x])
```

2 two
1 one
0 zero
3 three

you can iterate over values by combining a for loop with lookup

Extracting keys and values

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}


print(type(num_words.keys()))


print(type(num_words.values()))
```

Extracting keys and values

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}
```

```
print(type(num_words.keys()))
```

<class 'dict_keys'>

```
print(type(num_words.values()))
```

<class 'dict_values'>

don’t worry about these new types, because we can force them to be lists

Extracting keys and values

```
num_words = {0:“zero”, 1:”one”, 2:”two”, 3:”three”}
```

```
print(type(num_words.keys()))
```

<class 'dict_keys'>

```
print(type(num_words.values()))
```

<class 'dict_values'>

```
print(list(num_words.keys()))
```

[3, 1, 2, 0]

```
print(list(num_words.values()))
```

[“one”, “two”,
“zero”, “three”]

Defaults with get and pop

```
suffix = {1:“st”, 2:”nd”, 3:”rd”}


suffix.pop(0) # delete fails, because no key 0


suffix[4] # lookup fails because no key 4


suffix.get(4, “th”) # returns “th” because no key 4
```

specify a default if
key cannot be found

Defaults with get and pop

```
suffix = {1:“st”, 2:”nd”, 3:”rd”}
```

specify a default if
key cannot be found

```
suffix.pop(0) # delete fails, because no key 0

suffix[4] # lookup fails because no key 4

suffix.get(4, “th”) # returns “th” because no key 4
```

specify a default if
key cannot be found

Defaults with get and pop

```
suffix = {1:“st”, 2:”nd”, 3:”rd”}
```

specify a default if
key cannot be found

```
suffix.pop(0, “th”) # returns “th” because no key 0

suffix[4] # lookup fails because no key 4

suffix.get(4, “th”) # returns “th” because no key 4
```

specify a default if
key cannot be found

Defaults with get and pop

```
suffix = {1:“st”, 2:”nd”, 3:”rd”}


for num in range(6):
    print(str(num) + suffix.get(num, “th”))
```

Defaults with get and pop

```
suffix = {1:“st”, 2:”nd”, 3:”rd”}


for num in range(6):
    print(str(num) + suffix.get(num, “th”))
```

```
0th
1st
2nd
3rd
4th
5th
```

Today's Outline

More Dictionary Ops

Probabilities Tables

Markov Chains

Default Dictionaries

Binning

Table Representation

Demo 1: Letter Frequency

Goal: if we randomly pick a word in a text, what is the probability that it will be a given letter?

Input:

- Plaintext of book (from Project Gutenberg)
- A letter

Output:

- The portion of letters in the text that are that letter

Example:

```
prompt> python goldbug.py
text: AAAAABBCCC
A: 50%
B: 20%
C: 30%
```

Today's Outline

More Dictionary Ops

Probabilities Tables

Markov Chains

Default Dictionaries

Binning

Table Representation

Today's Outline

More Dictionary Ops

Probabilities Tables

Markov Chains

Default Dictionaries

Binning

Table Representation

Today's Outline

More Dictionary Ops

Probabilities Tables

Markov Chains

Default Dictionaries

Binning

Table Representation

Today's Outline

More Dictionary Ops

Probabilities Tables

Markov Chains

Default Dictionaries

Binning

Table Representation