Counting made easy

DATA TYPES FOR DATA SCIENCE IN PYTHON

Jason Myers
Instructor

Exploring collections: Counter in Python

https://www.youtube.com/watch?v=QP7pwaRA-LA

datacamp

Collections Module

- Part of Standard Library
- Advanced data containers

Counter

- Special dictionary used for counting data, measuring frequency

```
from collections import Counter
nyc_eatery_count_by_types = Counter(nyc_eatery_types)
print(nyc_eatery_count_by_type)
```

```
Counter({'Mobile Food Truck': 114, 'Food Cart': 74, 'Snack Bar': 24,
'Specialty Cart': 18, 'Restaurant': 15, 'Fruit & Vegetable Cart': 4})
```

```
print(nyc_eatery_count_by_types['Restaurant'])
```

```
15
```

3. Counter
Counter is a powerful python object based on the dictionary object that accepts a list and counts the number of times a value is found within the elements of that list. Since it's based on a dictionary, you can use all the normal dictionary features. Here, I have an list named nyc_eatery_types that contains one column of data called type from a table about eateries in NYC parks. I create a new Counter based on that list and print it. You can see each type from list and the number of times it was found in the list. I can also see how many restaurants are in the counter by using Restaurant as the index and printing it. Counters also provide a wonderful way to find the high values they contain.

Counter to find the most common

- `.most_common()` method returns the counter values in descending order

```
print(nyc_eatery_count_by_types.most_common(3))
```

```
[('Mobile Food Truck', 114), ('Food Cart', 74), ('Snack Bar', 24)]
```

Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

Exploring collections: defaultdict in Python

https://www.youtube.com/watch?v=zTHtUm4AtcA

Dictionaries of unknown structure - defaultdict

DATA TYPES FOR DATA SCIENCE IN PYTHON

Jason Myers
Instructor

1. Dictionaries of unknown structure - defaultdict
Often, we'll be working with data were we don't know all the keys that will be used, but we want to store a complex structure under those keys. A good example is I want every key to have a list of values. I'd have to initialize every key with an empty list then add the values to the list. Here is an example of just that.

Dictionary Handling

```
for park_id, name in nyc_eateries_parks:
    if park_id not in eateries_by_park:
        eateries_by_park[park_id] = []
    eateries_by_park[park_id].append(name)
print(eateries_by_park['M010'])
```

```
{'MOHAMMAD MATIN','PRODUCTS CORP.', 'Loeb Boathouse Restaurant',
'Nandita Inc.', 'SALIM AHAMED', 'THE NY PICNIC COMPANY',
'THE NEW YORK PICNIC COMPANY, INC.', 'NANDITA, INC.',
'JANANI FOOD SERVICE, INC.'}
```

2. Dictionary Handling
You can I start by looping over a list of tuples with the park's id and the name of the eatery. Then I check to see if I have a list for that part already in my dictionary. If not I create an empty list. Next, I append the name of the eatery to the list for that park id.
Thankfully, collections provides an easier way using defaultdict.

Using defaultdict

The functionality of both dictionaries and defualtdict are almost same except for the fact that defualtdict never raises a KeyError. It provides a default value for the key that does not exists.

- Pass it a default type that every key will have even if it doesn't currently exist
- Works exactly like a dictionary

3. Using defaultdict
Defaultdict accepts a type that every value will default to if the key is not present in the dictionary. You can override that type by setting the key manually to a value of different type. Still working with the NYC Parks eatery data, I have a list of tuples that contain the park id and the name of an eatery. I want to create a list of eateries by park. I begin by importing defaultdict from collections. Next, I create a defaultdict that defaults to a list. Next I iterate over my data and unpack it into the park_id and name. I append each eatery name into list for each park id. Finally, I print the eateries in the park with the ID os M010, which if you are curious is Central Park.

```
from collections import defaultdict
eateries_by_park = defaultdict(list)
for park_id, name in nyc_eateries_parks:
    eateries_by_park[park_id].append(name)
print(eateries_by_park['M010'])
```

```
{'MOHAMMAD MATIN','PRODUCTS CORP.', 'Loeb Boathouse Restaurant',
'Nandita Inc.', 'SALIM AHAMED', 'THE NY PICNIC COMPANY',
'THE NEW YORK PICNIC COMPANY, INC.', 'NANDITA, INC.', ...}
```

defaultdict (cont.)

```
from collections import defaultdict
eatery_contact_types = defaultdict(int)
for eatery in nyc_eateries:
    if eatery.get('phone'):
        eatery_contact_types['phones'] += 1
    if eatery.get('website'):
        eatery_contact_types['websites'] += 1
print(eatery_contact_types)
```

```
defaultdict(<class 'int'>, {'phones': 28, 'websites': 31})
```

4. defaultdict (cont.)
It's also common to use a defaultdict as a type of counter for a list if dictionaries where we are counting multiple keys from those dictionaries. In our NYC park eateries, I was curious how many had a published phone number or a website. This time when creating my defaultdict, I tell it I want it to be an int. Then I look over my nyc_eateries data and add 1 to the phones key on my defaultdict if it has a phone number that is not None. Then I add 1 to the websites key if it has a website. Finally, I print my defaultdict to see the counts.

Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

Exploring collections: OrderedDict in Python

https://www.youtube.com/watch?v=op3tFHLbMas

Maintaining Dictionary Order with OrderedDict

DATA TYPES FOR DATA SCIENCE IN PYTHON

Jason Myers
Instructor

datacamp

Order in Python dictionaries

- Python version < 3.6 NOT ordered
- Python version > 3.6 ordered

Getting started with OrderedDict

```
from collections import OrderedDict
nyc_eatery_permits = OrderedDict()
for eatery in nyc_eateries:
    nyc_eatery_permits[eatery['end_date']] = eatery
print(list(nyc_eatery_permits.items())[:3]
```

```
('2029-04-28', {'name': 'Union Square Seasonal Cafe',
'location': 'Union Square Park', 'park_id': 'M089',
'start_date': '2014-04-29', 'end_date': '2029-04-28',
'description': None, 'permit_number': 'M89-SB-R', ...})
```

An OrderedDict is a dictionary subclass that remembers the order that keys were first inserted. The only difference between dict() and OrderedDict() is that: OrderedDict preserves the order in which the keys are inserted.

OrderedDict power feature

- `.popitem()` method returns items in reverse insertion order

```
print(nyc_eatery_permits.popitem())
```

```
('2029-04-28', {'name': 'Union Square Seasonal Cafe',
'location': 'Union Square Park', 'park_id': 'M089',
'start_date': '2014-04-29', 'end_date': '2029-04-28',
'description': None, 'permit_number': 'M89-SB-R', ...})
```

```
print(nyc_eatery_permits.popitem())
```

```
('2027-03-31', {'name': 'Dyckman Marina Restaurant',
'location': 'Dyckman Marina Restaurant', 'park_id': 'M028',
'start_date': '2012-04-01', 'end_date': '2027-03-31', ...})
```

OrderedDict power feature (2)

- You can use the `last=False` keyword argument to return the items in insertion order

```
print(nyc_eatery_permits.popitem(last=False))
```

```
('2012-12-07', {'name': 'Mapes Avenue Ballfields Mobile Food Truck',
'location': 'Prospect Avenue, E. 181st Street', 'park_id': 'X289',
'start_date': '2009-07-01', 'end_date': '2012-12-07',
'description': None, 'permit_number': 'X289-MT', 'phone': None,
'website': None, 'type_name': 'Mobile Food Truck'})
```

Python Tutorial: Namedtuple - When and why should you use namedtuples?

https://www.youtube.com/watch?v=GfxJYp9_nJA

Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON

namedtuple

DATA TYPES FOR DATA SCIENCE IN PYTHON

Jason Myers
Instructor

datacamp

What is a namedtuple?

- A tuple where each position (column) has a name
- Ensure each one has the same properties
- Alternative to a `pandas` DataFrame row

2. What is a namedtuple?
Python has another container called a namedtuple which is a tuple, but has names for each position of the tuple. This works well when you don't need the nested structure of a dictionary or desire each item to look identical, and don't want to add the overhead of a Pandas dataframe.

Creating a namedtuple

- Pass a name and a list of fields

```
from collections import namedtuple
Eatery = namedtuple('Eatery', ['name', 'location', 'park_id',
   ...: 'type_name'])
eateries = []
for eatery in nyc_eateries:
    details = Eatery(eatery['name'],
                     eatery['location'],
                     eatery['park_id'],
                     eatery['type_name'])
    eateries.append(details)
```

3. Creating a namedtuple
You create a namedtuple by passing a name for the tuple type and a list of field names. Let's begin by importing namedtuple from the collections module. Next we'll define our namedtuple. It's common practice to use Pascalcase (capitalizing each word) when naming namedtuples so hear I've used Eatery with a capital E for both the tuple name and the variable we stored the namedtuple as. Then we provide a list of fields we want on the nametuple. Now we can create our Eatery named tuples. I'm going to change our NYC Park eateries data into a list of namedtuples. I create an empty list then iterate over my nyc_eateries list creating an instance of my Eatery namedtuple by passing in the data from the loop as the arguments to my namedtuple.

Print the first element

```
print(eateries[0])
```

```
Eatery(name='Mapes Avenue Ballfields Mobile Food Truck',
location='Prospect Avenue, E. 181st Street',
park_id='X289', type_name='Mobile Food Truck')
```

Leveraging namedtuples

- Each field is available as an attribute of the namedtuple

```
for eatery in eateries[:3]:
    print(eatery.name)
    print(eatery.park_id)
    print(eatery.location)
```

An attribute is basically a named field or data storage location.

```
Mapes Avenue Ballfields Mobile Food Truck
X289
Prospect Avenue, E. 181st Street

Claremont Park Mobile Food Truck
X008
East 172 Street between Teller & Morris avenues ...
```

Let's practice!

DATA TYPES FOR DATA SCIENCE IN PYTHON