Introduction to iterators

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Hugo Bowne-Anderson
Data Scientist at DataCamp

Iterating with a for loop

- We can iterate over a list using a for loop

```python
employees = ['Nick', 'Lore', 'Hugo']
for employee in employees:
    print(employee)
```

```
Nick
Lore
Hugo
```

2. Iterating with a for loop
There's no reason to be scared of iterators because you have actually been working with them for some time now! When you use a for loop to print out each element of a list, you're iterating over the list.

Iterating with a for loop

- We can iterate over a string using a for loop

```python
for letter in 'DataCamp':
    print(letter)
```

```
D
a
t
a
C
a
m
p
```

3. Iterating with a for loop
You can also use a for loop to iterate over characters in a string such as you see here. You can also use a for loop to iterate a over

Iterating with a for loop

- We can iterate over a range object using a for loop

```
for i in range(4):
    print(i)
```

```
0
1
2
3
```

4. Iterating with a for loop
a sequence of numbers produced by a special range object. The reason that we can loop over such objects is that they are special objects

Iterators vs. iterables

- Iterable
  - Examples: lists, strings, dictionaries, file connections
  - An object with an associated `iter()` method
  - Applying `iter()` to an iterable creates an iterator
- Iterator
  - Produces next value with `next()`

5. Iterators vs. iterables
called iterables: lists, strings and range objects are all iterables, as are many other Python objects, such as dictionaries and file connections! The actual definition of an iterable is an object that has an associated iter method. Once this iter method is applied to an iterable, an iterator object is created. Under the hood, this is actually what a for loop is doing: it takes an iterable, creates the associated iterator object, and iterates over it! An iterator is defined as an object that as an associated next method that produces the consecutive values. To create an iterator from an iterable,

Iterating over iterables: next()

6. Iterating over iterables: next()
all we need to do is use the function iter and pass it the iterable. Once we have the iterator defined, we pass it to the function next and this returns the first value. Calling next again on the iterator returns the next value until there are no values left to return and then it throws us a StopIteration error.

```python
word = 'Da'
it = iter(word)
next(it)
```

```
'D'
```

```python
next(it)
```

```
'a'
```

```python
next(it)
```

```
StopIteration                      Traceback (most recent call last)
<ipython-input-11-2cdb14c0d4d6> in <module>()
-> 1 next(it)
StopIteration:
```

Iterating at once with *

```python
word = 'Data'
it = iter(word)
print(*it)
```

```
D a t a
```

```python
print(*it)
```

- No more values to go through!

7. Iterating at once with *
You can also print all values of an iterator in one fell swoop using the star operator, referred to as the splat operator in some circles. This star operator unpacks all elements of an iterator or an iterable. **Be warned, however, once you do so, you cannot do it again as there are no more values to iterate through! We would have to redefine our iterator to do so.**

Iterating over dictionaries

```python
pythonistas = {'hugo': 'bowne-anderson', 'francis': 'castro'}
for key, value in pythonistas.items():
    print(key, value)
```

```
francis castro
hugo bowne-anderson
```

8. Iterating over dictionaries
We mentioned before that dictionaries and file connections are iterables as well. To iterate over the key-value pairs of a Python dictionary, we need to unpack them by applying the items method to the dictionary as you can see here.

Iterating over file connections

```python
file = open('file.txt')
it = iter(file)
print(next(it))
```

```
This is the first line.
```

```python
print(next(it))
```

```
This is the second line.
```

9. Iterating over file connections
With respect to file connections, here you can see how to use the iter and next methods to return the lines from a file, file dot txt.

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Playing with iterators

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Hugo Bowne-Anderson
Data Scientist at DataCamp

Using enumerate()

```
avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']
e = enumerate(avengers)
print(type(e))
```

```
<class 'enumerate'>
```

```
e_list = list(e)
print(e_list)
```

```
[(0, 'hawkeye'), (1, 'iron man'), (2, 'thor'), (3, 'quicksilver')]
```

2. Using enumerate()
enumerate is a function that takes any iterable as argument, such as a list, and returns a special enumerate object, which consists of pairs containing the elements of the original iterable, along with their index within the iterable. **We can use the function list** to turn this enumerate object into a list of tuples and print it to see what it contains. The enumerate object itself

enumerate() and unpack

```python
avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']
for index, value in enumerate(avengers):
    print(index, value)
```

```
0 hawkeye
1 iron man
2 thor
3 quicksilver
```

3. enumerate() and unpack
is also an iterable and we can loop over it while unpacking its elements using the clause for index, value in enumerate(avengers). It is the default behavior of enumerate to begin indexing at 0. However, you can alter this with a second argument, start, which you can see here.

```python
for index, value in enumerate(avengers, start=10):
    print(index, value)
```

```
10 hawkeye
11 iron man
12 thor
13 quicksilver
```

Using zip()

```
avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']
names = ['barton', 'stark', 'odinson', 'maximoff']
z = zip(avengers, names)
print(type(z))
```

```
<class 'zip'>
```

```
z_list = list(z)
print(z_list)
```

```
[('hawkeye', 'barton'), ('iron man', 'stark'),
('thor', 'odinson'), ('quicksilver', 'maximoff')]
```

4. Using zip()
Now let's move on to zip, which accepts an arbitrary number of iterables and returns an iterator of tuples. Here we have two lists, one of the avengers, the other of their names. Zipping them together creates a zip object which is an iterator of tuples. We can turn this zip object into a list and print the list. The first element is a tuple containing the first elements of each list that was zipped. The second element is a tuple containing the second elements of each list that was zipped and so on.

zip() and unpack

```python
avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']
names = ['barton', 'stark', 'odinson', 'maximoff']
for z1, z2 in zip(avengers, names):
    print(z1, z2)
```

```
hawkeye barton
iron man stark
thor odinson
quicksilver maximoff
```

5. zip() and unpack
Alternatively, we could use a for loop to iterate over the zip object and print the tuples.
We could also
6. Print zip with *
have used the splat operator to print all the elements!

Print zip with *

```
avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']
names = ['barton', 'stark', 'odinson', 'maximoff']
z = zip(avengers, names)
print(*z)
```

```
('hawkeye', 'barton') ('iron man', 'stark')
('thor', 'odinson') ('quicksilver', 'maximoff')
```

6. Print zip with *
have used the splat operator to print all the elements!

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Using iterators to load large files into memory

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Hugo Bowne-Anderson
Data Scientist at DataCamp

Loading data in chunks

- There can be too much data to hold in memory
- Solution: load data in chunks!
- Pandas function: `read_csv()`
    - Specify the chunk: `chunk_size`

2. Loading data in chunks
dealing with large amounts of data. Let's say that you are pulling data from a file, database or API and there's so much of it, just so much data, that you can't hold it in memory. One solution is to load the data in chunks, perform the desired operation or operations on each chuck, store the result, discard the chunk and then load the next chunk; this sounds like a place where an iterator could be useful! To surmount this challenge, we are going to use the pandas function read_csv, which provides a wonderful option whereby you can load data in chunks and iterate over them. All we need to do is to specify the chunk using the argument yep, you guessed it: chunk_size. As with much of what we do in Data Science, this is best illustrated by an example.

Iterating over data

```python
import pandas as pd
result = []
for chunk in pd.read_csv('data.csv', chunksize=1000):
    result.append(sum(chunk['x']))
total = sum(result)
print(total)
```

```
4252532
```

3. Iterating over data

Let's say that we have a csv with a column called 'x' of numbers and I want to compute the sum of all the numbers in that column. However, the file is too large to store in memory. We first import pandas and then initialize an empty list result to hold the result of each iteration. We then use the read_csv function, utilizing the argument chunk_size, setting it to the size of the chunks I want to read in. In this example, we use a chunk size of 1,000. You can play around with it. The object created by the read_csv call is an iterable so I can can iterate over it, using a for loop, in which each chunk will be a DataFrame. Within the for loop, that is, on each iteration, we compute the sum of the column of interest and we append it to the list result. Once this is executed, we can take the sum of the list result and this gives us our total sum of the column of interest. Iterators to the rescue!

Iterating over data

Do practice!

```python
import pandas as pd
total = 0
for chunk in pd.read_csv('data.csv', chunksize=1000):
    total += sum(chunk['x'])
print(total)
```

```
4252532
```

4. Iterating over data
Also note that we need not have used a list to store each result - we could have initialized total to zero before iterating over the file and added each sum during the iteration procedure, as you see here. Now things get really cool: you're going to use an iterator to load Twitter data in chunks and perform a similar computation that you did in the prequel to this course.

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Congratulations!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

Hugo Bowne-Anderson
Data Scientist at DataCamp

What's next?

- List comprehensions and generators
- List comprehensions:
  - Create lists from other lists, DataFrame columns, etc.
  - Single line of code
  - More efficient than using a for loop

Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)