

# 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

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

```
Nick  
Lore  
Hugo
```

# Iterating with a for loop

- We can iterate over a string using a for loop

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

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

# 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
```

# 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()`

# Iterating over iterables: next()

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

```
'D'
```

```
next(it)
```

```
'a'
```

```
next(it)
```

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

# Iterating at once with \*

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

```
D a t a
```

```
print(*it)
```

- No more values to go through!

# Iterating over dictionaries

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

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



# Iterating over file connections

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

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

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

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

# 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')]
```

# enumerate() and unpack

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

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

```
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')]
```

# zip() and unpack

```
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
```

# 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')
```



# 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`

# Iterating over data

```
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

# Iterating over data

```
import pandas as pd
total = 0

for chunk in pd.read_csv('data.csv', chunksize=1000):
    total += sum(chunk['x'])

print(total)
```

4252532

# 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)

# List comprehensions

PYTHON DATA SCIENCE TOOLBOX (PART 2)



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Populate a list with a for loop

```
nums = [12, 8, 21, 3, 16]

new_nums = []

for num in nums:
    new_nums.append(num + 1)

print(new_nums)
```

```
[13, 9, 22, 4, 17]
```

# A list comprehension

```
nums = [12, 8, 21, 3, 16]  
new_nums = [num + 1 for num in nums]  
print(new_nums)
```

```
[13, 9, 22, 4, 17]
```

# For loop and list comprehension syntax

```
new_nums = [num + 1 for num in nums]
```

```
for num in nums:  
    new_nums.append(num + 1)  
  
print(new_nums)
```

```
[13, 9, 22, 4, 17]
```

# List comprehension with range()

```
result = [num for num in range(11)]  
print(result)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

# List comprehensions

- Collapse for loops for building lists into a single line
- Components
  - Iterable
  - Iterator variable (represent members of iterable)
  - Output expression

# Nested loops (1)

```
pairs_1 = []  
  
for num1 in range(0, 2):  
    for num2 in range(6, 8):  
        pairs_1.append(num1, num2)  
  
print(pairs_1)
```

```
[(0, 6), (0, 7), (1, 6), (1, 7)]
```

- How to do this with a list comprehension?



# Nested loops (2)

```
pairs_2 = [(num1, num2) for num1 in range(0, 2) for num2 in range(6,  
print(pairs_2)
```

```
[(0, 6), (0, 7), (1, 6), (1, 7)]
```

- Tradeoff: readability

# Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

# Advanced comprehensions

PYTHON DATA SCIENCE TOOLBOX (PART 2)



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Conditionals in comprehensions

- Conditionals on the iterable

```
[num ** 2 for num in range(10) if num % 2 == 0]
```

```
[0, 4, 16, 36, 64]
```

- Python documentation on the `%` operator: The `%` (modulo) operator yields the remainder from the division of the first argument by the second.

```
5 % 2
```

```
1
```

```
6 % 2
```

```
0
```

# Conditionals in comprehensions

- Conditionals on the output expression

```
[num ** 2 if num % 2 == 0 else 0 for num in range(10)]
```

```
[0, 0, 4, 0, 16, 0, 36, 0, 64, 0]
```

# Dict comprehensions

- Create dictionaries
- Use curly braces `{}` instead of brackets `[]`

```
pos_neg = {num: -num for num in range(9)}  
print(pos_neg)
```

```
{0: 0, 1: -1, 2: -2, 3: -3, 4: -4, 5: -5, 6: -6, 7: -7, 8: -8}
```

```
print(type(pos_neg))
```

```
<class 'dict'>
```

# Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

# Introduction to generator expressions

PYTHON DATA SCIENCE TOOLBOX (PART 2)



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp



# Generator expressions

- Recall list comprehension

```
[2 * num for num in range(10)]
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

- Use `( )` instead of `[ ]`

```
(2 * num for num in range(10))
```

```
<generator object <genexpr> at 0x1046bf888>
```

# List comprehensions vs. generators

- List comprehension - returns a list
- Generators - returns a generator object
- Both can be iterated over

# Printing values from generators (1)

```
result = (num for num in range(6))  
for num in result:  
    print(num)
```

```
0  
1  
2  
3  
4  
5
```

```
result = (num for num in range(6))  
print(list(result))
```

```
[0, 1, 2, 3, 4, 5]
```

# Printing values from generators (2)

```
result = (num for num in range(6
```

- Lazy evaluation

```
print(next(result))
```

```
0
```

```
print(next(result))
```

```
1
```

```
print(next(result))
```

```
2
```

```
print(next(result))
```

```
3
```

```
print(next(result))
```

```
4
```

# Generators vs list comprehensions

IPython Shell

```
In [1]: [num for num in range(10**1000000)]
```

```
In [2]: |
```

# Generators vs list comprehensions

IPython Shell

```
In [1]: [num for num in range(10**1000000)]
```

```
In [2]: |
```

IPython Shell

```
In [1]: [num for num in range(10**1000000)]
```

```
In [2]: |
```

Your session has been disconnected.  
The performed operation was too  
resource-intensive.

Restart Session

# Generators vs list comprehensions

IPython Shell

```
In [1]: (num for num in range(10**1000000))
```

```
Out[1]: <generator object <genexpr> at 0x7f8aca2601f8>
```

```
In [2]:
```

# Conditionals in generator expressions

```
even_nums = (num for num in range(10) if num % 2 == 0)  
print(list(even_nums))
```

```
[0, 2, 4, 6, 8]
```



# Generator functions

- Produces generator objects when called
- Defined like a regular function - `def`
- Yields a sequence of values instead of returning a single value
- Generates a value with `yield` keyword

# Build a generator function

- sequence.py

```
def num_sequence(n):  
    """Generate values from 0 to n."""  
    i = 0  
    while i < n:  
        yield i  
        i += 1
```

# Use a generator function

```
result = num_sequence(5)  
print(type(result))
```

```
<class 'generator'>
```

```
for item in result:  
    print(item)
```

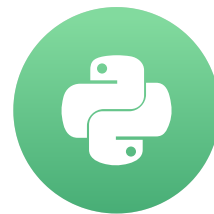
```
0  
1  
2  
3  
4
```

# Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

# Wrapping up comprehensions and generators.

PYTHON DATA SCIENCE TOOLBOX (PART 2)



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Re-cap: list comprehensions

- Basic

```
[output expression for iterator variable in iterable]
```

- Advanced

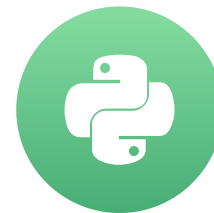
```
[output expression +  
conditional on output for iterator variable in iterable +  
conditional on iterable]
```

# Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

# Welcome to the case study!

PYTHON DATA SCIENCE TOOLBOX (PART 2)



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp



# World bank data

- Data on world economies for over half a century
- Indicators
  - Population
  - Electricity consumption
  - CO2 emissions
  - Literacy rates
  - Unemployment
  - Mortality rates

# Using zip()

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

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

```
print(list(z))
```

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

# Defining a function

- raise.py

```
def raise_both(value1, value2):  
    """Raise value1 to the power of value2  
    and vice versa."""  
    new_value1 = value1 ** value2  
    new_value2 = value2 ** value1  
    new_tuple = (new_value1, new_value2)  
    return new_tuple
```

# Re-cap: list comprehensions

## Basic

```
[output expression for iterator variable in iterable]
```

## Advanced

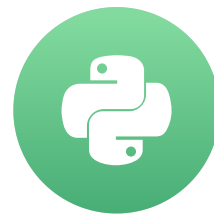
```
[output expression +  
conditional on output for iterator variable in iterable +  
conditional on iterable]
```

# Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

# Using Python generators for streaming data

PYTHON DATA SCIENCE TOOLBOX (PART 2)



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Generators for the large data limit

- Use a generator to load a file line by line
- Works on streaming data!
- Read and process the file until all lines are exhausted

# Build a generator function

- sequence.py

```
def num_sequence(n):  
    """Generate values from 0 to n."""  
    i = 0  
    while i < n:  
        yield i  
        i += 1
```



# Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

# Using pandas' read\_csv iterator for streaming data

PYTHON DATA SCIENCE TOOLBOX (PART 2)



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# Reading files in chunks

- Up next:
  - `read_csv()` function and `chunk_size` argument
  - Look at specific indicators in specific countries
  - Write a function to generalize tasks

# Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)

# Final thoughts

PYTHON DATA SCIENCE TOOLBOX (PART 2)



**Hugo Bowne-Anderson**  
Data Scientist at DataCamp

# You've applied your skills in:

- User-defined functions
- Iterators
- List comprehensions
- Generators

# Let's practice!

PYTHON DATA SCIENCE TOOLBOX (PART 2)