PYTHON DATA SCIENCE TOOLBOX II

# Python Data Science Toolbox II

# You've learned:

- Writing custom functions
- Using custom functions in data science

# You'll learn:

- List comprehensions
  - Wrangle data to create other lists
- Iterators
  - You've encountered these before!
  - Rapidly iterate data science protocols and procedures over sets of objects

PYTHON DATA SCIENCE TOOLBOX II

# See you in the course!

PYTHON DATA SCIENCE TOOLBOX II

# Iterators in Pythonland

# Iterating with a for loop

- We can iterate over a list using a for loop

```
In [1]: employees = ['Nick', 'Lore', 'Hugo']

In [2]: for employee in employees:
   ...:     print(employee)
Nick
Lore
Hugo
```

# Iterating with a for loop

- We can iterate over a string using a for loop

```
In [1]: 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

```
In [1]: 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()

```
In [1]: word = 'Da'

In [2]: it = iter(word)

In [3]: next(it)
Out[3]: 'D'

In [4]: next(it)
Out[4]: 'a'

In [5]: next(it)
---------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-11-2cdb14c0d4d6> in <module>()
----> 1 next(it)
StopIteration:
```

# Iterating at once with *

```
In [1]: word = 'Data'

In [2]: it = iter(word)

In [3]: print(*it)
D a t a

In [4]: print(*it)
```

**No more values to go through!**

# Iterating over dictionaries

```
In [1]: pythonistas = {'hugo': 'bowne-anderson', 'francis':
'castro'}

In [2]: for key, value in pythonistas.items():
   ...:     print(key, value)
francis castro
hugo bowne-anderson
```

# Iterating over file connections

```
In [1]: file = open('file.txt')

In [2]: it = iter(file)

In [3]: print(next(it))
This is the first line.

In [4]: print(next(it))
This is the second line.
```

PYTHON DATA SCIENCE TOOLBOX II

# Let’s practice!

PYTHON DATA SCIENCE TOOLBOX II

# Playing with iterators

# Using enumerate()

```
In [1]: avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']

In [2]: e = enumerate(avengers)

In [3]: print(type(e))
<class 'enumerate'>

In [4]: e_list = list(e)

In [5]: print(e_list)
[(0, 'hawkeye'), (1, 'iron man'), (2, 'thor'), (3, 'quicksilver')]
```

# enumerate() and unpack

```
In [1]: avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']

In [2]: for index, value in enumerate(avengers):
   ....:     print(index, value)
0 hawkeye
1 iron man
2 thor
3 quicksilver

In [3]: for index, value in enumerate(avengers, start=10):
   ....:     print(index, value)
10 hakweye
11 iron man
12 thor
13 quicksilver
```

# Using zip()

```
In [1]: avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']

In [2]: names = ['barton', 'stark', 'odinson', 'maximoff']

In [3]: z = zip(avengers, names)

In [4]: print(type(z))
<class 'zip'>

In [5]: z_list = list(z)

In [6]: print(z_list)
[('hawkeye', 'barton'), ('iron man', 'stark'), ('thor',
'odinson'), ('quicksilver', 'maximoff')]
```

# zip() and unpack

```
In [1]: avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']

In [2]: names = ['barton', 'stark', 'odinson', 'maximoff']

In [3]: for z1, z2 in zip(avengers, names):
  ....:     print(z1, z2)
hawkeye barton
iron man stark
thor odinson
quicksilver maximoff
```

# Print zip with *

```
In [1]: avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']

In [2]: names = ['barton', 'stark', 'odinson', 'maximoff']

In [3]: z = zip(avengers, names)

In [4]: print(*z)
('hawkeye', 'barton') ('iron man', 'stark') ('thor', 'odinson')
('quicksilver', 'maximoff')
```

PYTHON DATA SCIENCE TOOLBOX II

# Let’s practice!

PYTHON DATA SCIENCE TOOLBOX II

# Using iterators for big data

# 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: `chunksize`

# Iterating over data

```
In [1]: import pandas as pd

In [2]: result = []

In [3]: for chunk in pd.read_csv('data.csv', chunksize=1000):
   ...:     result.append(sum(chunk['x']))

In [4]: total = sum(result)

In [5]: print(total)
4252532
```

# Iterating over data

```
In [1]: import pandas as pd

In [2]: total = 0

In [3]: for chunk in pd.read_csv('data.csv', chunksize=1000):
   ...:     total += sum(chunk['x'])

In [4]: print(total)
4252532
```

PYTHON DATA SCIENCE TOOLBOX II

# Let’s practice!

PYTHON DATA SCIENCE TOOLBOX II

# Congratulations!

# 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

PYTHON DATA SCIENCE TOOLBOX II

# See you in the next chapter!

PYTHON DATA SCIENCE TOOLBOX II

# List comprehensions

# Populate a list with a for loop

```
In [1]: nums = [12, 8, 21, 3, 16]

In [2]: new_nums = []

In [3]: for num in nums:
   ...:     new_nums.append(num + 1)

In [4]: print(new_nums)
[13, 9, 22, 4, 17]
```

# A list comprehension

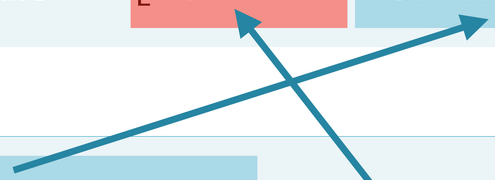
```
In [1]: nums = [12, 8, 21, 3, 16]

In [2]: new_nums = [num + 1 for num in nums]

In [3]: print(new_nums)
[13, 9, 22, 4, 17]
```

# For loop and list comprehension syntax

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



```
In [2]: for num in nums:
   ...:     new_nums.append(num + 1)

In [3]: print(new_nums)
[13, 9, 22, 4, 17]
```

# List comprehension with range()

```
In [1]: result = [num for num in range(11)]

In [2]: 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)

```
In [1]: pairs_1 = []

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

In [3]: print(pairs_1)
[(0, 6), (0, 7), (1, 6), (1, 7)]
```

- How to do this with a list comprehension?

# Nested loops (2)

```
In [1]: pairs_2 = [(num1, num2) for num1 in range(0, 2) for num2
in range(6, 8)]

In [2]: print(pairs_2)
[(0, 6), (0, 7), (1, 6), (1, 7)]
```

- Tradeoff: readability

PYTHON DATA SCIENCE TOOLBOX II

# Let’s practice!

PYTHON DATA SCIENCE TOOLBOX II

# Advanced comprehensions

# Conditionals in comprehensions

- Conditionals on the iterable

```
In [1]: [num ** 2 for num in range(10) if num % 2 == 0]
Out[1]: [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.

```
In [1]: 5 % 2
Out[1]: 1

In [2]: 6 % 2
Out[2]: 0
```

# Conditionals in comprehensions

- Conditionals on the output expression

```
In [2]: [num ** 2 if num % 2 == 0 else 0 for num in range(10)]
Out[2]: [0, 0, 4, 0, 16, 0, 36, 0, 64, 0]
```

# Dict comprehensions

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

```
In [1]: pos_neg = {num: -num for num in range(9)}

In [2]: print(pos_neg)
{0: 0, 1: -1, 2: -2, 3: -3, 4: -4, 5: -5, 6: -6, 7: -7, 8: -8}

In [3]: print(type(pos_neg))
<class 'dict'>
```

PYTHON DATA SCIENCE TOOLBOX II

# Let’s practice!

PYTHON DATA SCIENCE TOOLBOX II

# Introduction to generators

# Generator expressions

- Recall list comprehension

```
In [1]: [2 * num for num in range(10)]
Out[1]: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

- Use ( ) instead of [ ]

```
In [2]: (2 * num for num in range(10))
Out[2]: <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)

```
In [1]: result = (num for num in range(6))

In [2]: for num in result:
  ....:     print(num)
0
1
2
3
4
5
```

```
In [1]: result = (num for num in range(6))

In [2]: print(list(result))
[0, 1, 2, 3, 4, 5]
```

# Printing values from generators (2)

```
In [1]: result = (num for num in range(6))

In [2]: print(next(result))
0

In [3]: print(next(result))
1

In [4]: print(next(result))
2

In [5]: print(next(result))
3

In [6]: print(next(result))
4
```

**Lazy evaluation**

# 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

```
In [1]: even_nums = (num for num in range(10) if num % 2 == 0)

In [2]: 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

```
In [1]: result = num_sequence(5)

In [2]: print(type(result))
<class 'generator'>

In [3]: for item in result:
   ....:     print(item)
0
1
2
3
4
```

PYTHON DATA SCIENCE TOOLBOX II

# Let’s practice!

PYTHON DATA SCIENCE TOOLBOX II

# Wrap-up: comprehensions

# 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]
```

PYTHON DATA SCIENCE TOOLBOX II

# Let’s practice!

PYTHON DATA SCIENCE TOOLBOX II

# Welcome to the Case Study!

# World bank data

- Data on world economies for over half a century
- Indicators
  - Population
  - Electricity consumption
  - $CO_2$ emissions
  - Literacy rates
  - Unemployment

# Using zip()

```
In [1]: avengers = ['hawkeye', 'iron man', 'thor', 'quicksilver']

In [2]: names = ['barton', 'stark', 'odinson', 'maximoff']

In [3]: z = zip(avengers, names)

In [4]: print(type(z))
<class 'zip'>

In [5]: 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
```

**Function header**

**Function body**

# 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]
```

PYTHON DATA SCIENCE TOOLBOX II

# Let’s practice!

PYTHON DATA SCIENCE TOOLBOX II

# Using Python generators for streaming data

# 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
```

PYTHON DATA SCIENCE TOOLBOX II

# Let’s practice!

PYTHON DATA SCIENCE TOOLBOX II

# Using iterators for streaming data

# Reading files in chunks

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

PYTHON DATA SCIENCE TOOLBOX II

# Let’s practice!

PYTHON DATA SCIENCE TOOLBOX II

# Final thoughts

# You've applied your skills in:

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

PYTHON DATA SCIENCE TOOLBOX II

# Good job and keep coding!