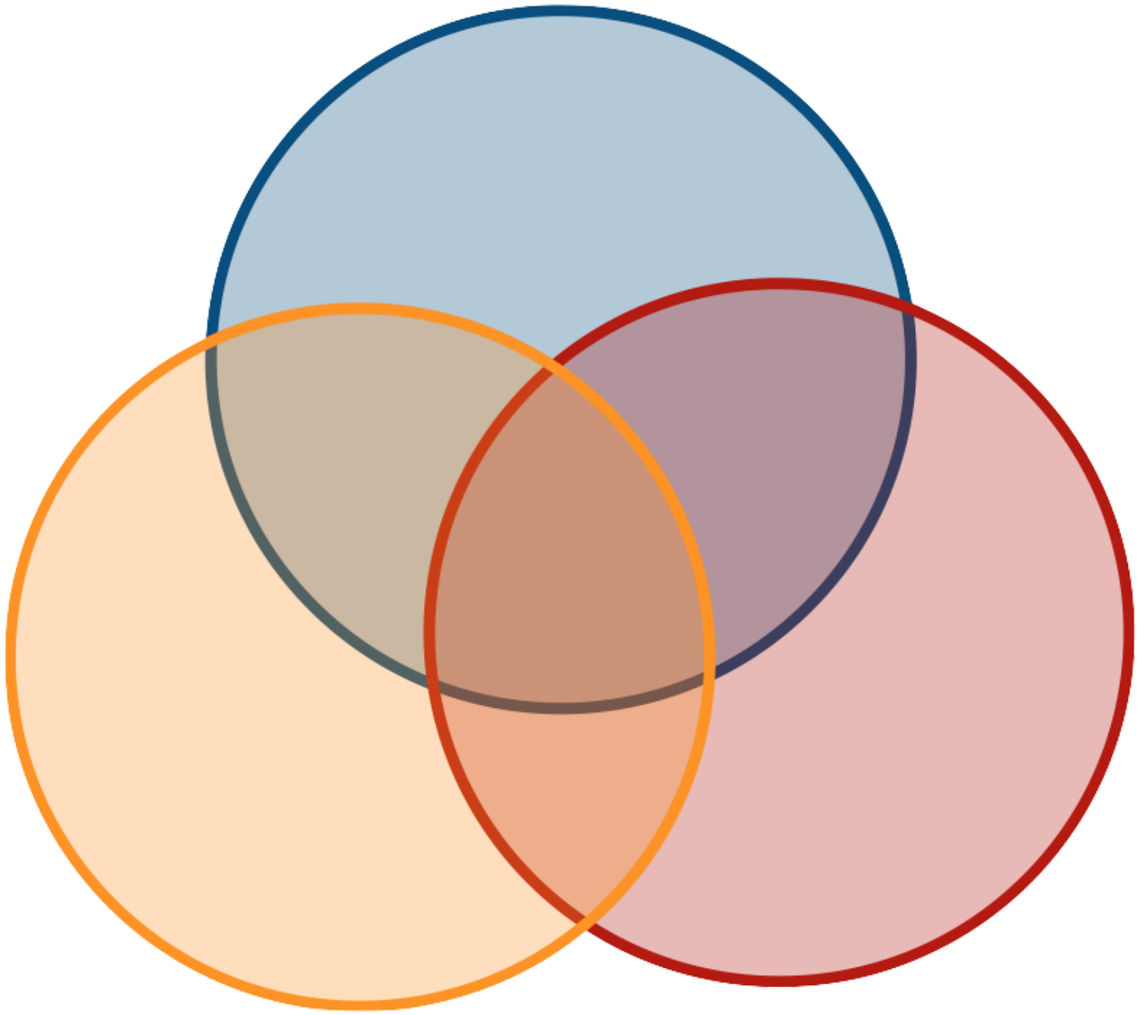
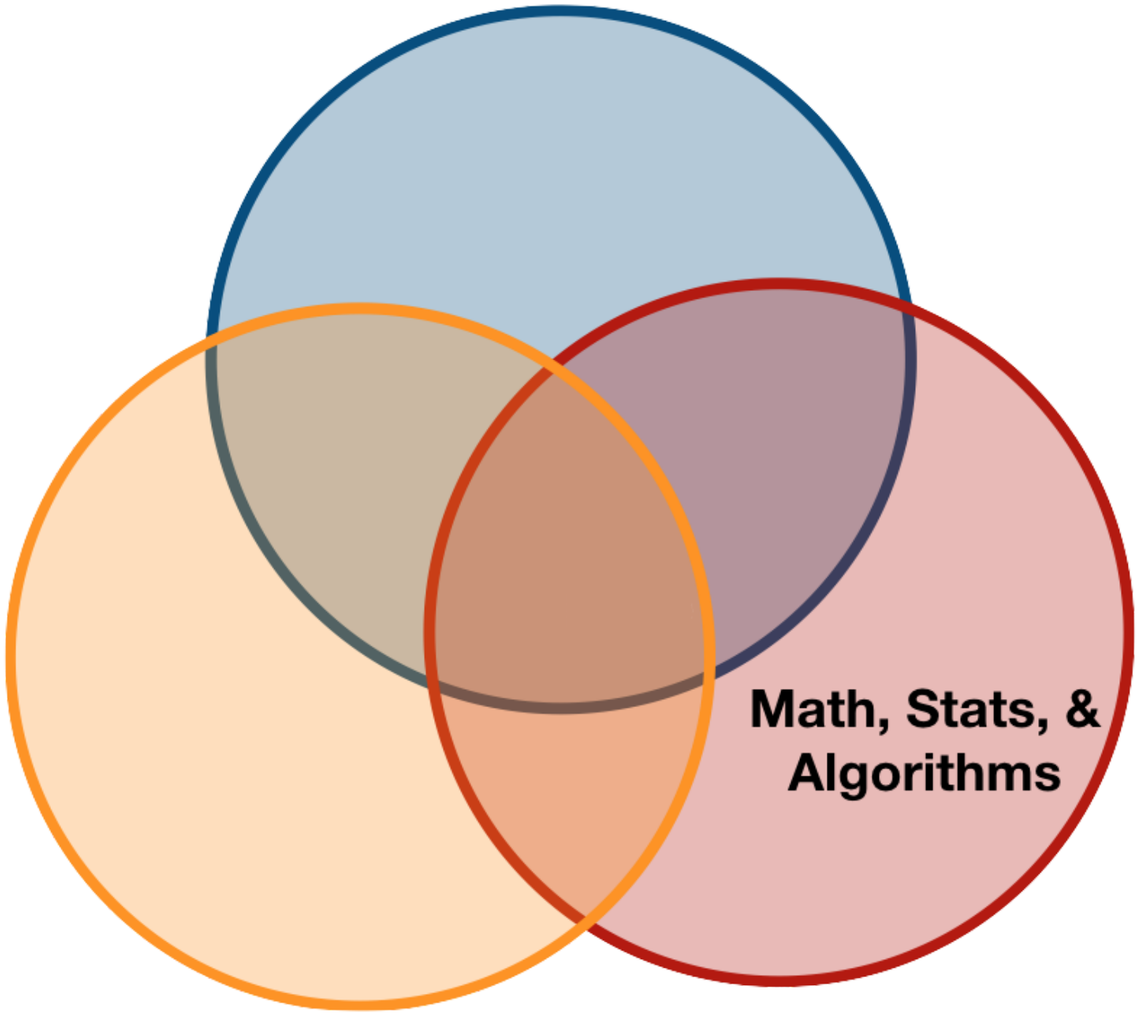
Python, data science, & software engineering

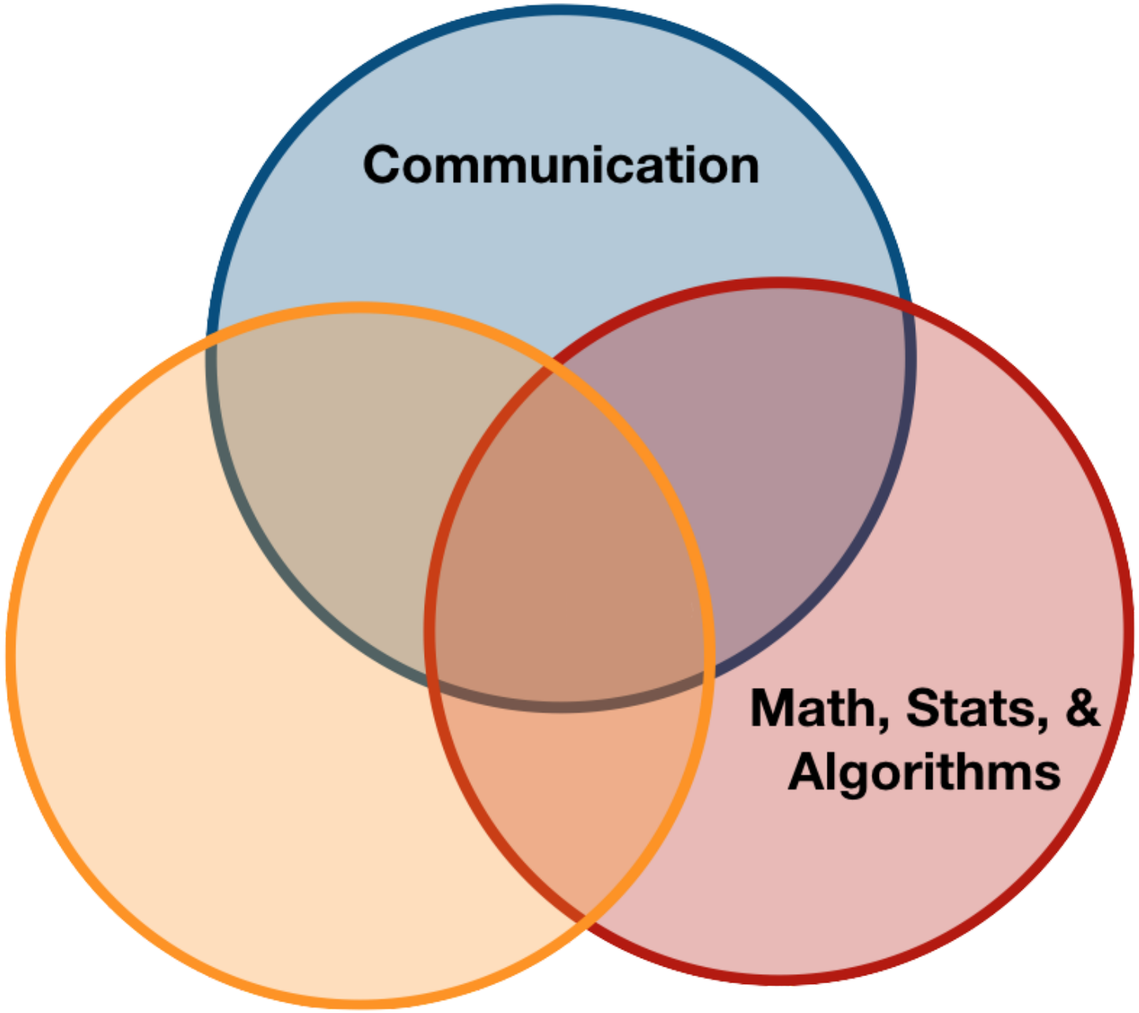
SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

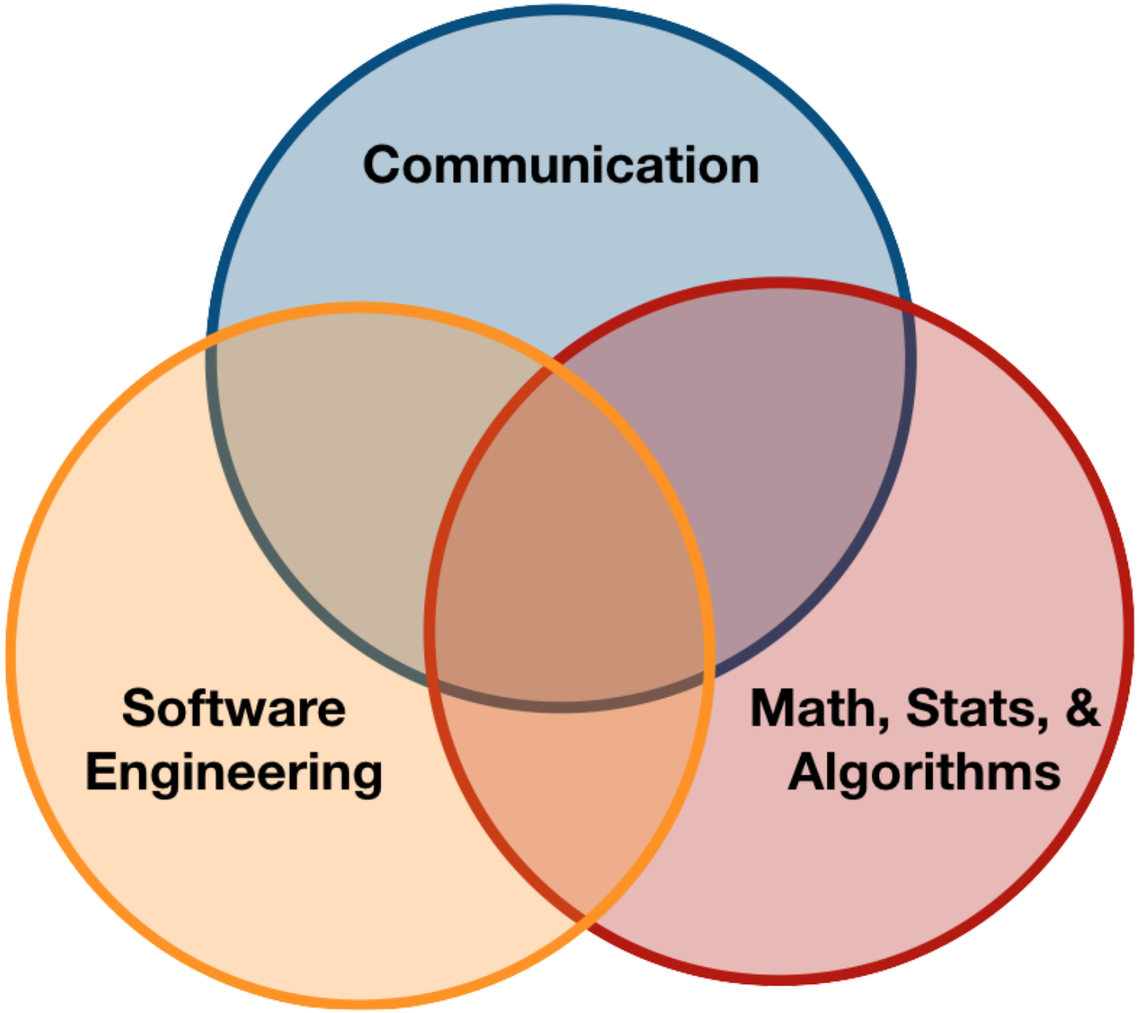
Adam Spannbauer
Machine Learning Engineer at Eastman

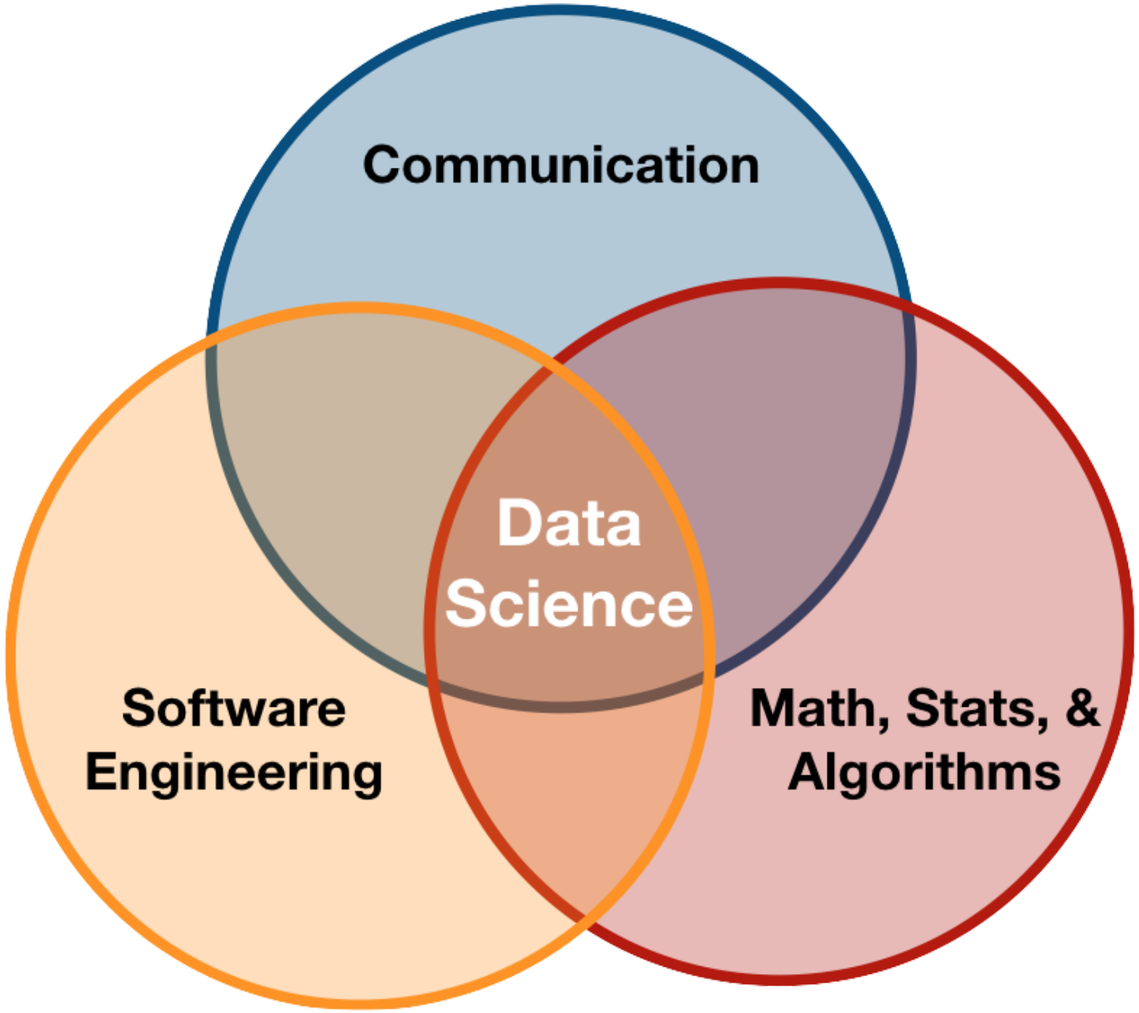
datacamp




Math, Stats, &
Algorithms


Communication
Math, Stats, &
Algorithms


Communication
Software Engineering
Math, Stats, & Algorithms

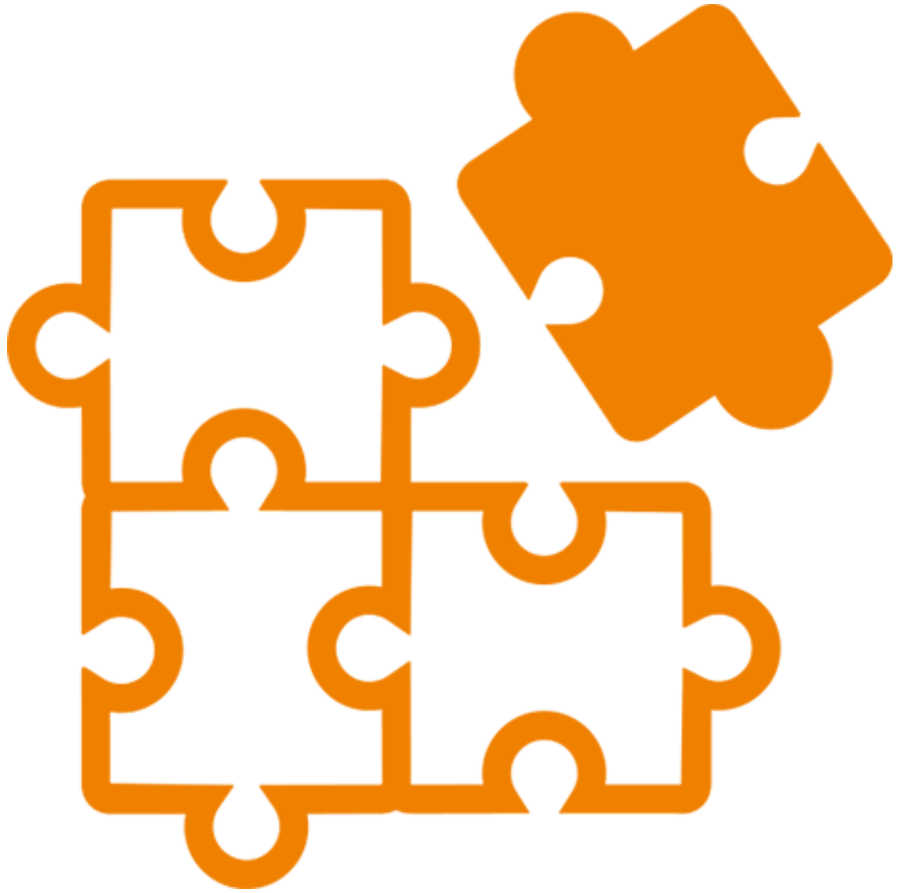

Communication
Data Science
Software Engineering
Math, Stats, & Algorithms

Software engineering concepts

- Modularity
- Documentation
- Testing
- Version Control & Git

Benefits of modularity

- Improve readability
- Improve maintainability
- Solve problems only once

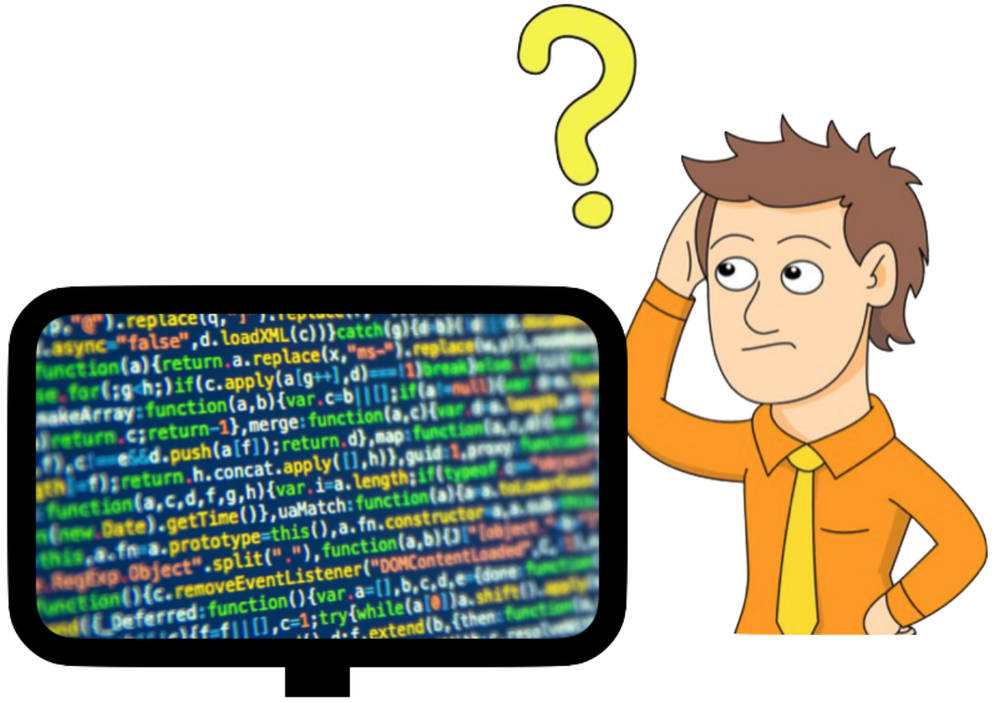


Modularity in python

```
# Import the pandas PACKAGE
import pandas as pd

# Create some example data
data = {'x': [1, 2, 3, 4],
        'y': [20.1, 62.5, 34.8, 42.7]}

# Create a dataframe CLASS object
df = pd.DataFrame(data)

# Use the plot METHOD
df.plot('x', 'y')
```

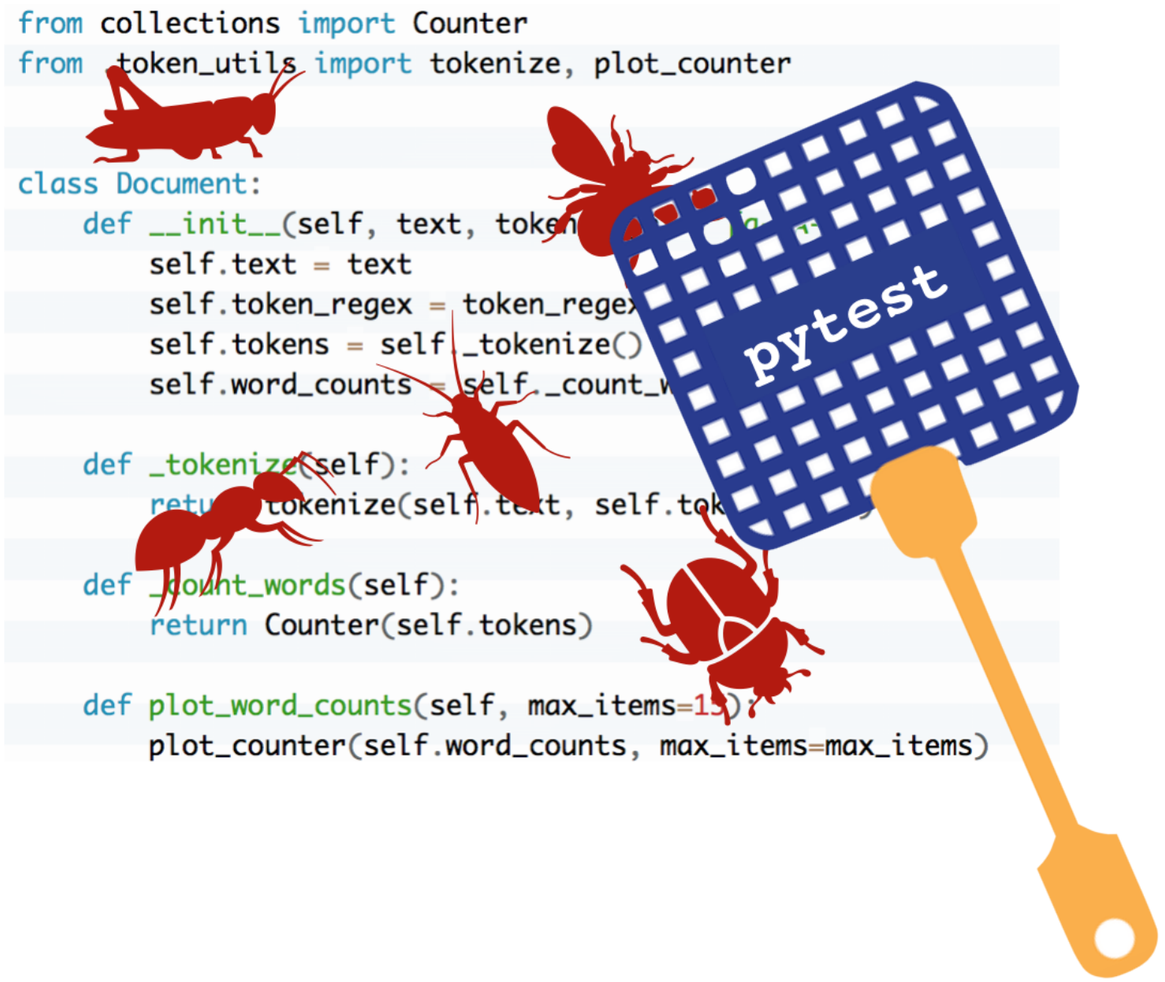
Benefits of documentation

- Show users how to use your project
- Prevent confusion from your collaborators
- Prevent frustration from future you



Benefits of automated testing

- Save time over manual testing
- Find & fix more bugs
- Run tests anytime/anywhere



Let's Review

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

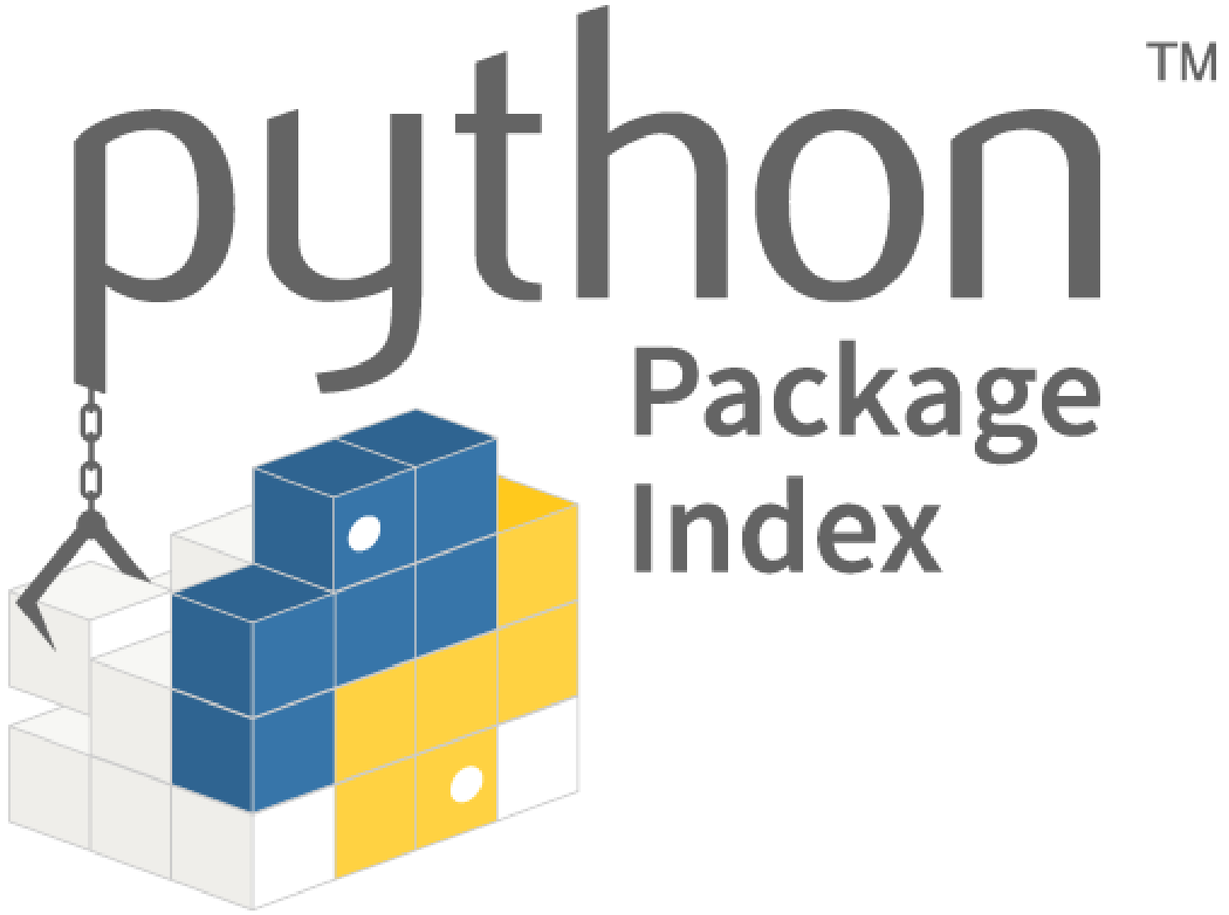
Introduction to Packages & Documentation

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Adam Spannbauer
Machine Learning Engineer at Eastman

datacamp

Packages and PyPi



Intro to pip

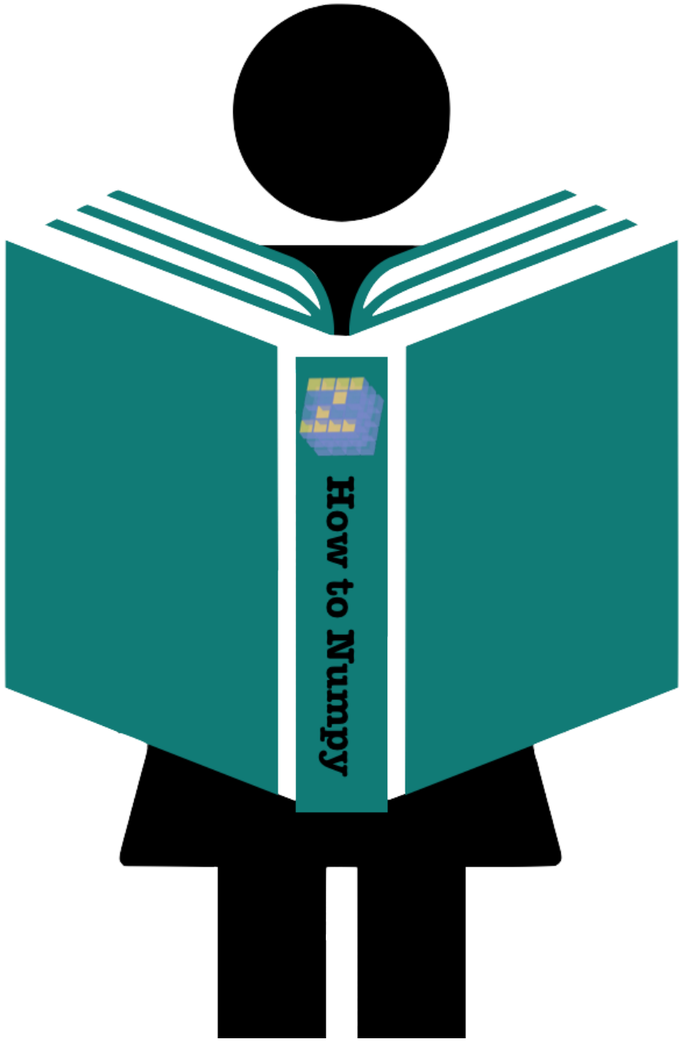
Intro to pip

Using pip to install numpy

```
datacamp@server:~$ pip install numpy
```

```
Collecting numpy
    100% |????????????????????????????????| 24.5MB 44kB/s
Installing collected packages: numpy
Successfully installed numpy-1.15.4
```

How do we use numpy?



Reading documentation with help()

```
help(numpy.busday_count)
```

```
busday_count(begindates, enddates)
    Counts the number of valid days between `begindates` and
    `enddates`, not including the day of `enddates`.

    Parameters
    ----------
    begindates : the first dates for counting.
    enddates : the end dates for counting (excluded from the count)

    Returns
    -------
    out : the number of valid days between the begin and end dates.

    Examples
    --------
    >>> # Number of weekdays in 2011
    ...  np.busday_count('2011', '2012')
    260
```

Reading documentation with help()

```
import numpy as np
help(np)
```

```
    Provides
      1. An array object of arbitrary homogeneous items
      2. Fast mathematical operations over arrays
      3. Linear Algebra, Fourier Transforms, Random Number Generation
```

```
help(42)
```

```
class int(object)
 |  int(x=0) -> integer
 |  int(x, base=10) -> integer
 |
 |  Convert a number or string to an integer, or return 0 if no arguments
 |  are given.  If x is a number, return x.__int__().  For floating point
 |  numbers, this truncates towards zero.
```

Let's Practice

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

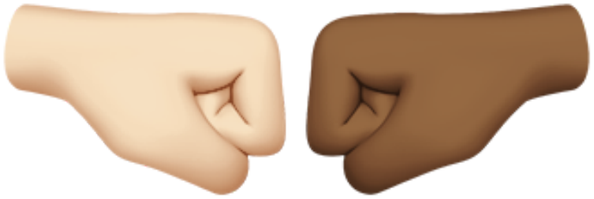
Conventions and PEP 8

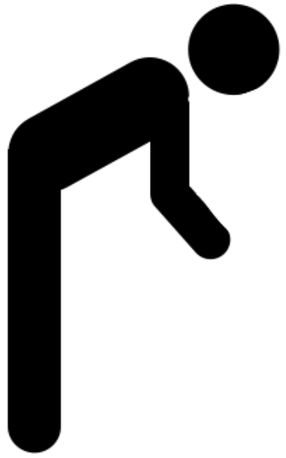
SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Adam Spannbauer
Machine Learning Engineer at Eastman

datacamp

What are conventions?





Introducing PEP 8

> "Code is read much more often than it is written"

Violating PEP 8

```
#define our data
my_dict ={
    'a'  : 10,
'b': 3,
    'c'  :   4,
             'd': 7}
#import needed package
import numpy as np
#helper function
def DictToArray(d):
    """Convert dictionary values to numpy array"""
    #extract values and convert
              x=np.array(d.values())
              return x
print(DictToArray(my_dict))
```

```
array([10,  4,  3,  7])
```

Following PEP 8

```
# Import needed package
import numpy as np

# Define our data
my_dict = {'a': 10, 'b': 3, 'c': 4, 'd': 7}



# Helper function
def dict_to_array(d):
    """Convert dictionary values to numpy array"""
    # Extract values and convert
    x = np.array(d.values())
    return x


print(dict_to_array(my_dict))
```

```
array([10,  4,  3,  7])
```

PEP 8 Tools

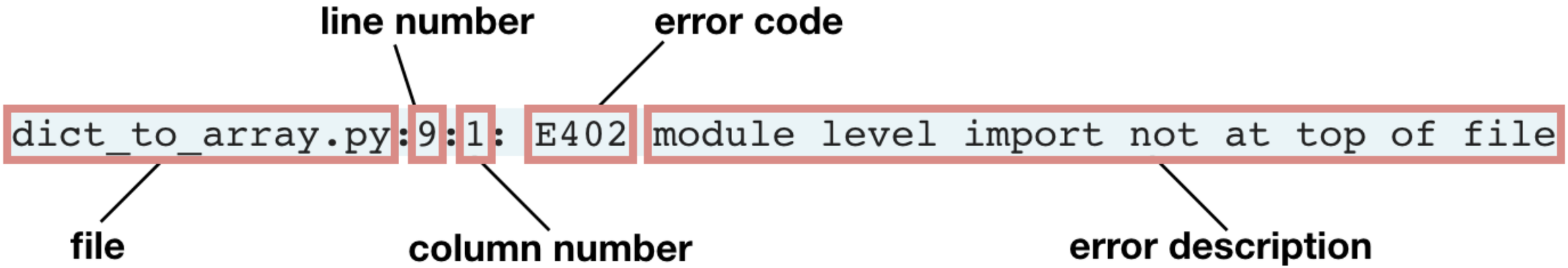
```
1  x=1
   PEP 8: missing whitespace around operator
```

Using pycodestyle

```
datacamp@server:~$ pip install pycodestyle
datacamp@server:~$ pycodestyle dict_to_array.py
```

```
dict_to_array.py:5:9: E203 whitespace before ':'
dict_to_array.py:6:14: E131 continuation line unaligned for hanging indent
dict_to_array.py:8:1: E265 block comment should start with '# '
dict_to_array.py:9:1: E402 module level import not at top of file
dict_to_array.py:11:1: E302 expected 2 blank lines, found 0
dict_to_array.py:13:15: E111 indentation is not a multiple of four
```

Output from pycodestyle



Let's Practice

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

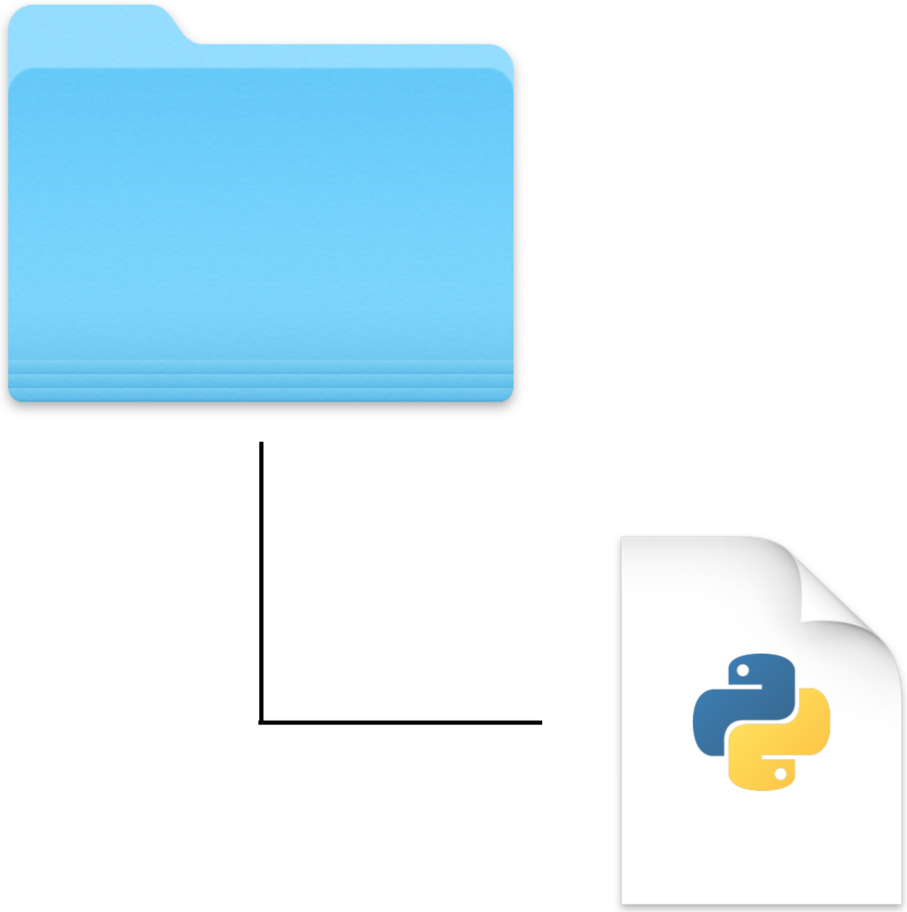
Writing Your First Package

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

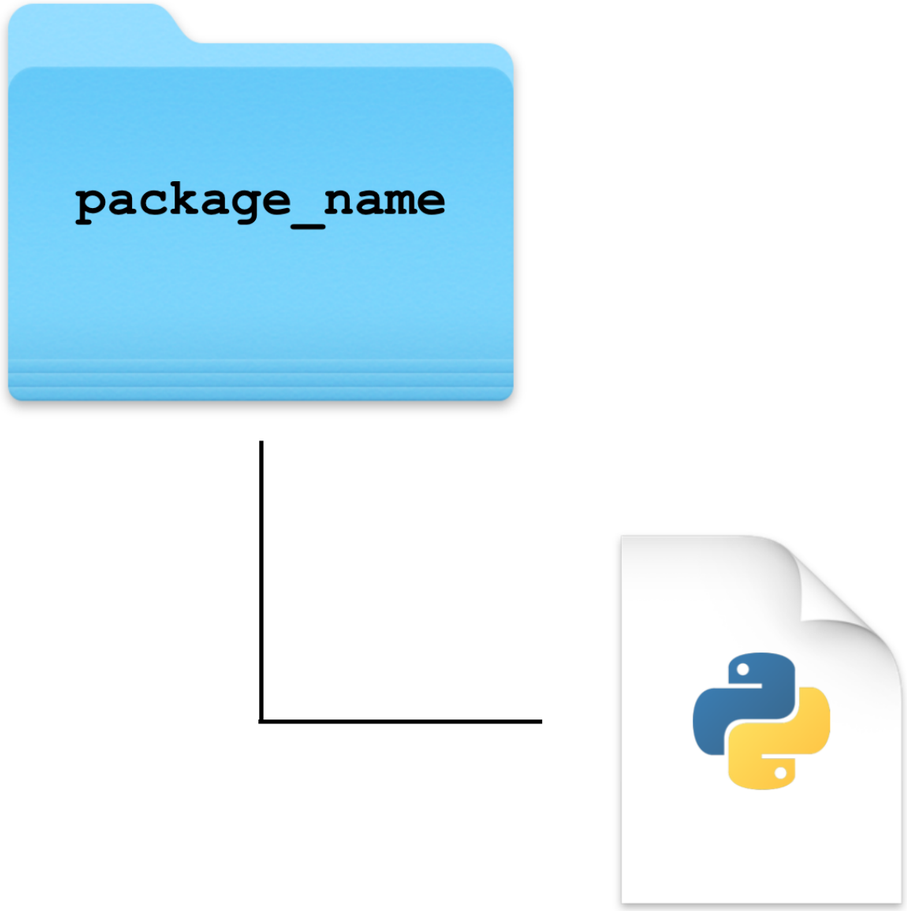
Adam Spannbauer
Machine Learning Engineer at Eastman

datacamp

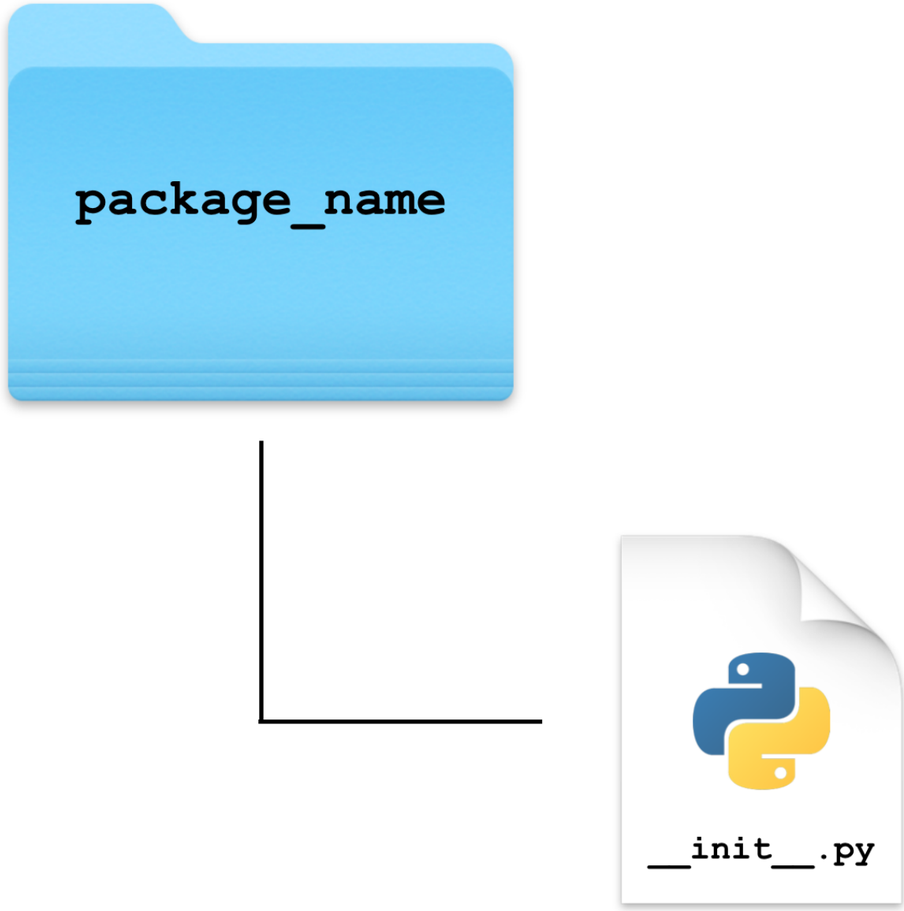
Package structure



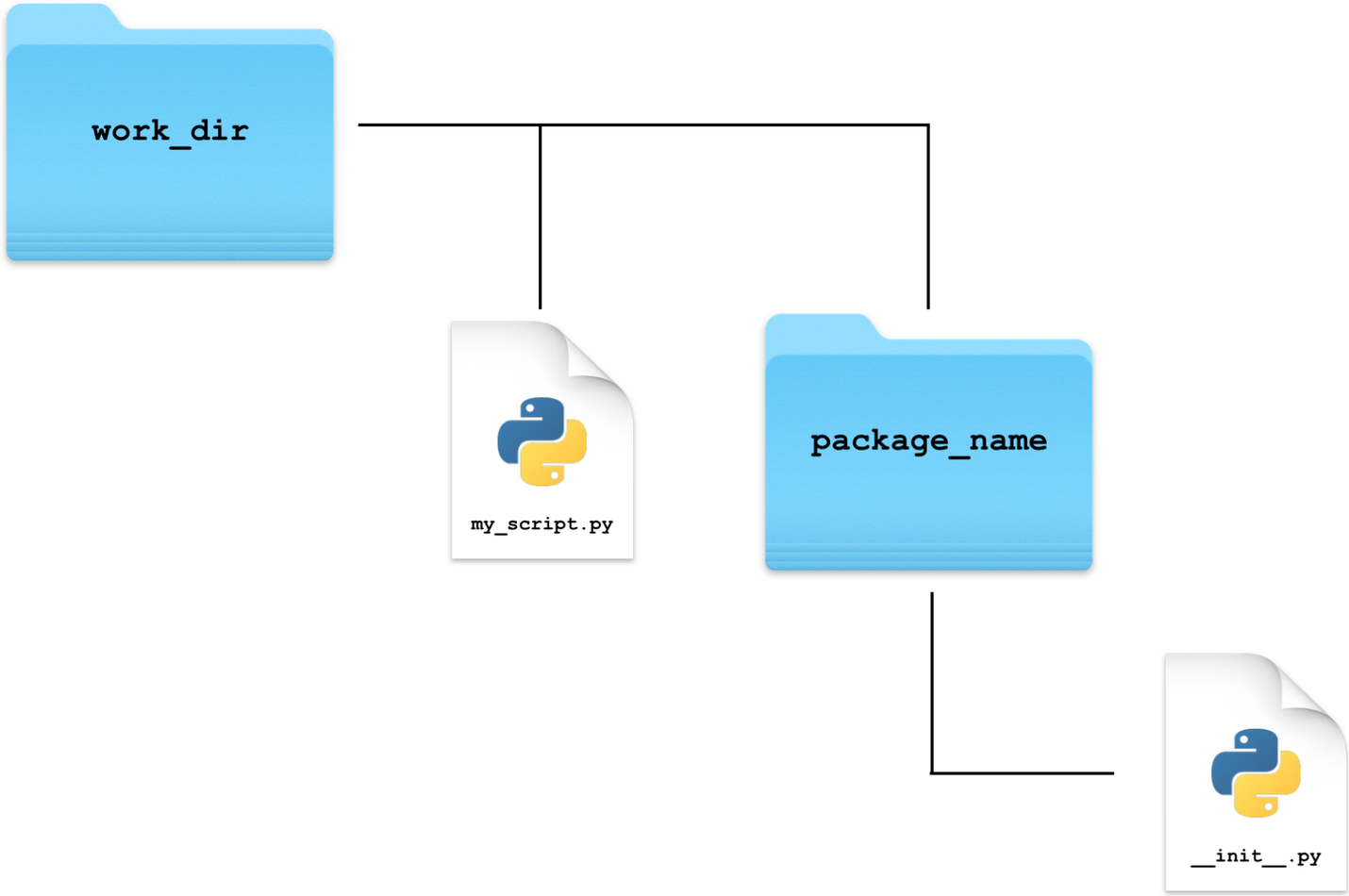
Package structure



Package structure



Importing a local package



Importing a local package

```
import my_package
help(my_package)
```

```
Help on package my_package:
NAME
    my_package

PACKAGE CONTENTS
FILE
    ~/work_dir/my_package/__init__.py
```

Let's Practice

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

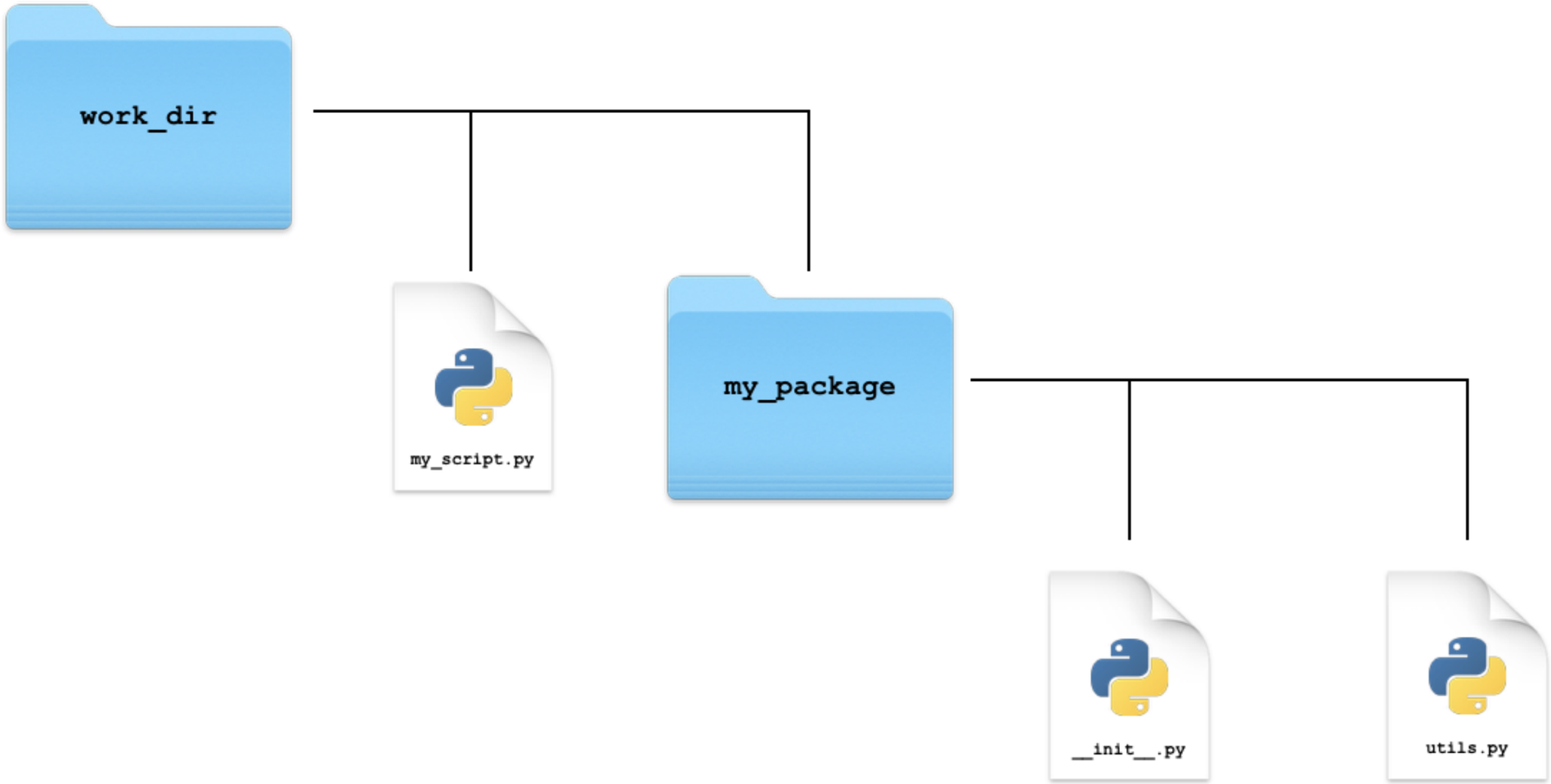
Adding Functionality to Packages

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Adam Spannbauer
Machine Learning Engineer at Eastman

datacamp

Package structure



Adding functionality

working in `work_dir/my_package/utils.py`

```
def we_need_to_talk(break_up=False):
    """Helper for communicating with significant other"""
    if break_up:
        print("It's not you, it's me...")
    else:
        print('I <3 U!')
```

working in `work_dir/my_script.py`

```
# Import utils submodule
import my_package.utils

# Decide to start seeing other people
my_package.utils.we_need_to_talk(break_up=True)
```

```
It's not you, it's me...
```

Importing functionality with __init__.py

working in `work_dir/my_package/__init__.py`

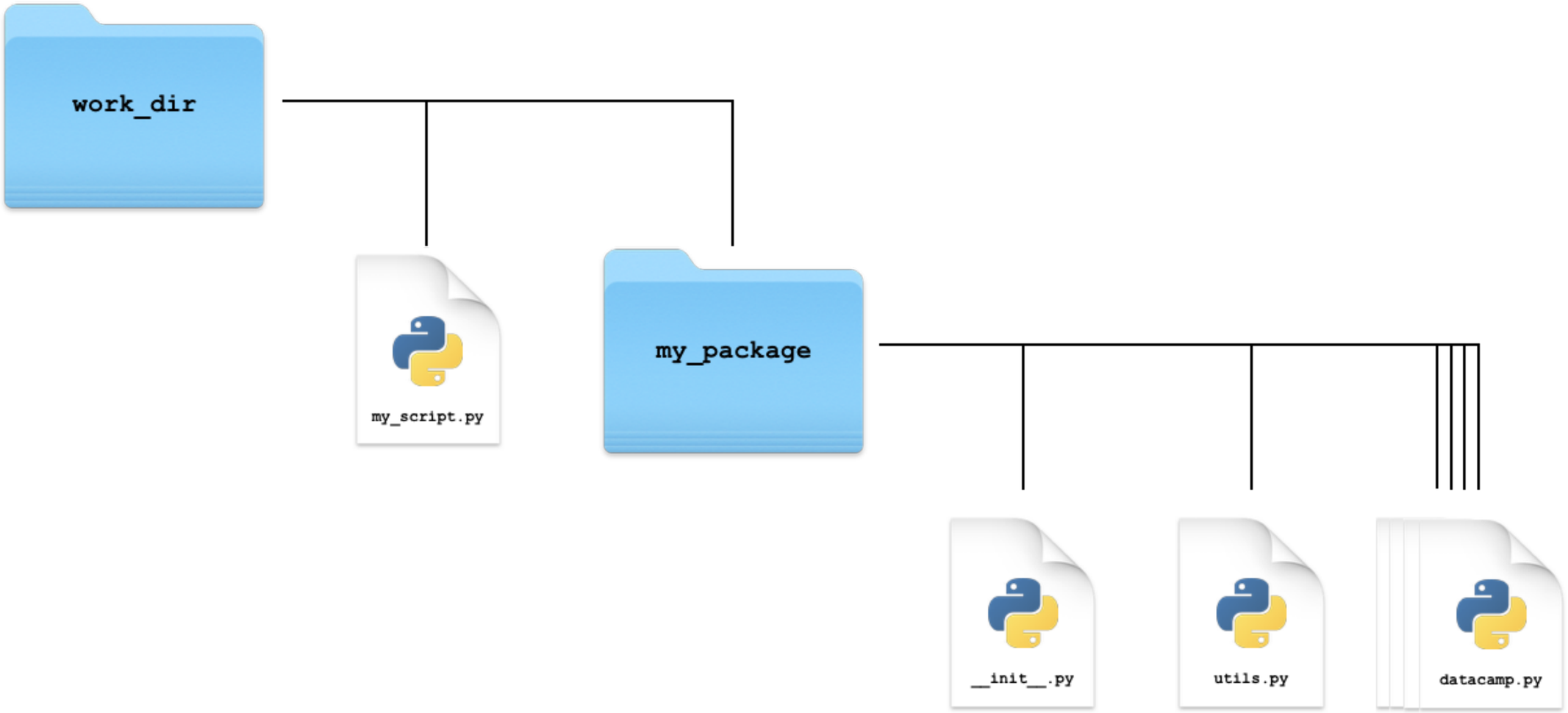
```
from .utils import we_need_to_talk
```

working in `work_dir/my_script.py`

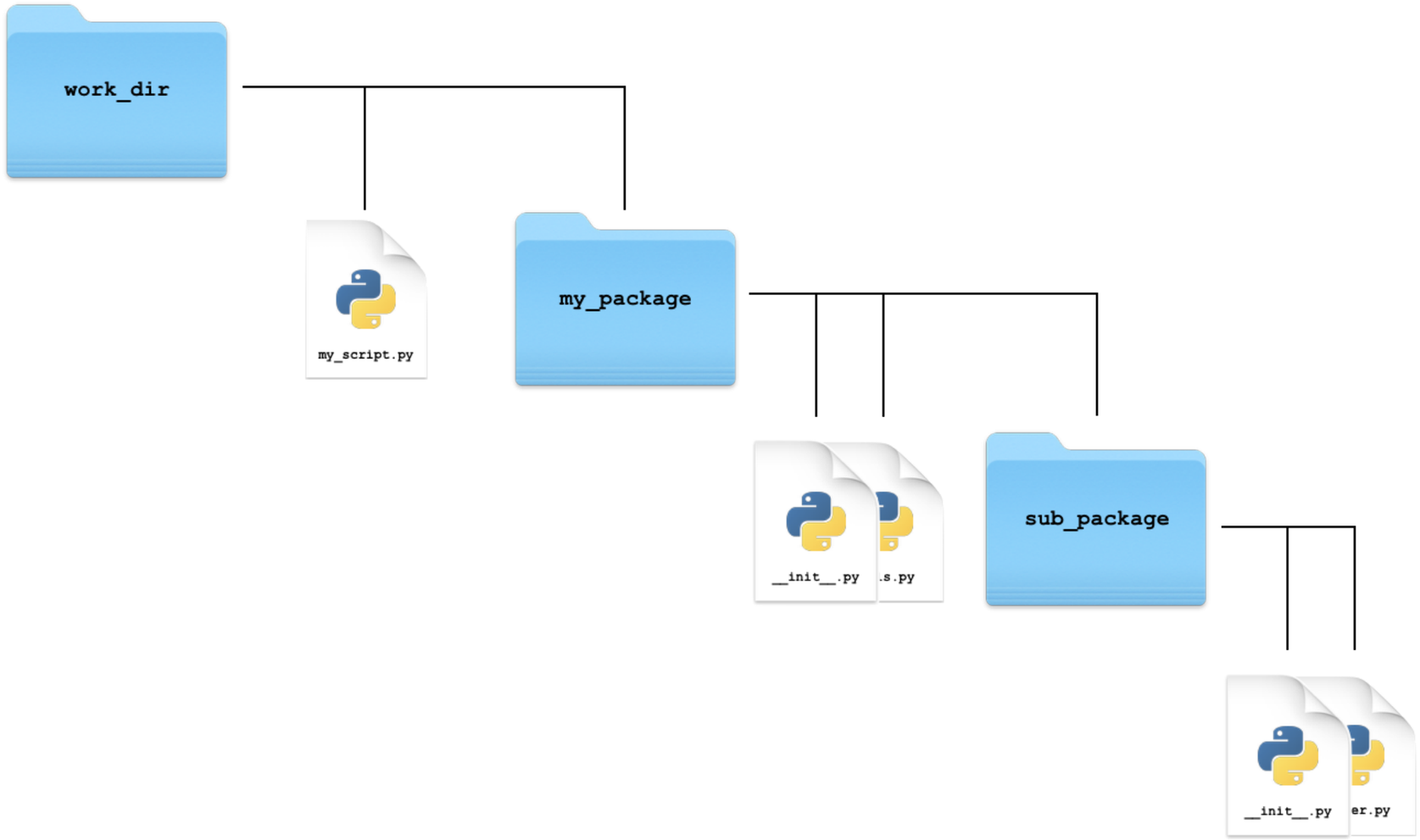
```
# Import custom package
import my_package

# Realize you're with your soulmate
my_package.we_need_to_talk(break_up=False)
```

```
I <3 U!
```

Extending package structure



Extending package structure



Let's Practice

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Making your package portable

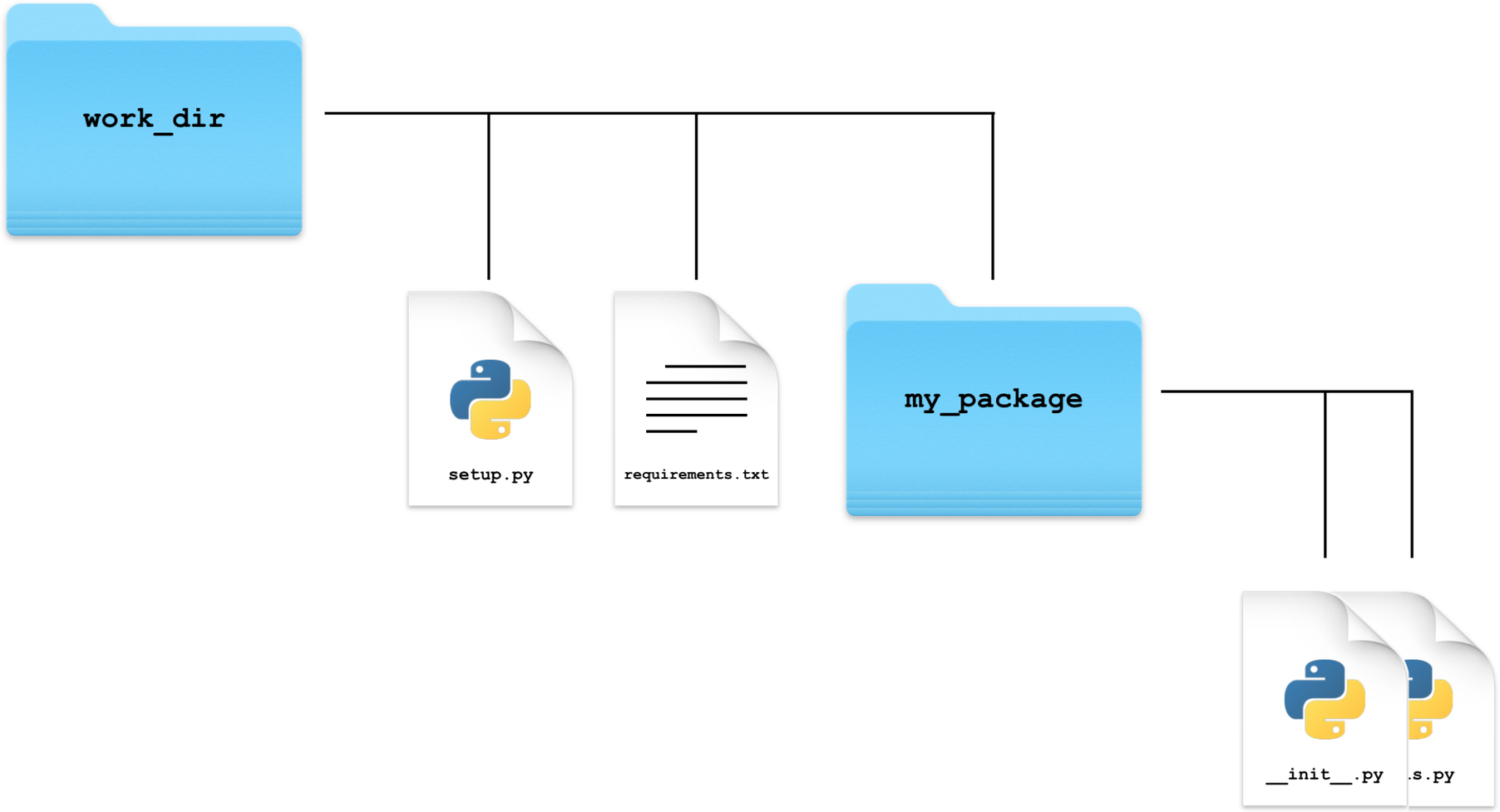
SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Adam Spannbauer
Machine Learning Engineer at Eastman

datacamp

Steps to portability

Portable package structure



Contents of requirements.txt

working in `work_dir/requirements.txt`

```
# Needed packages/versions
matplotlib
numpy==1.15.4
pycodestyle>=2.4.0
```

working with `terminal`

```
datacamp@server:~$ pip install -r requirements.txt
```

Contents of setup.py

```
from setuptools import setup

setup(name='my_package',
      version='0.0.1',
      description='An example package for DataCamp.',
      author='Adam Spannbauer',
      author_email='spannbaueradam@gmail.com',
      packages=['my_package'],
      install_requires=['matplotlib',
                        'numpy==1.15.4',
                        'pycodestyle>=2.4.0'])
```

install_requires vs requirements.txt

working in `work_dir/requirements.txt`

```
# Specify where to install requirements from
--index-url https://pypi.python.org/simple/

# Needed packages/versions
matplotlib
numpy==1.15.4
pycodestyle>=2.4.0
```

Documentation: **install_requires vs requirements files**

pip installing your package

```
datacamp@server:~/work_dir $ pip install .
```

```
Building wheels for collected packages: my-package
  Running setup.py bdist_wheel for my-package ... done
Successfully built my-package
Installing collected packages: my-package
Successfully installed my-package-0.0.1
```

Let's Practice

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

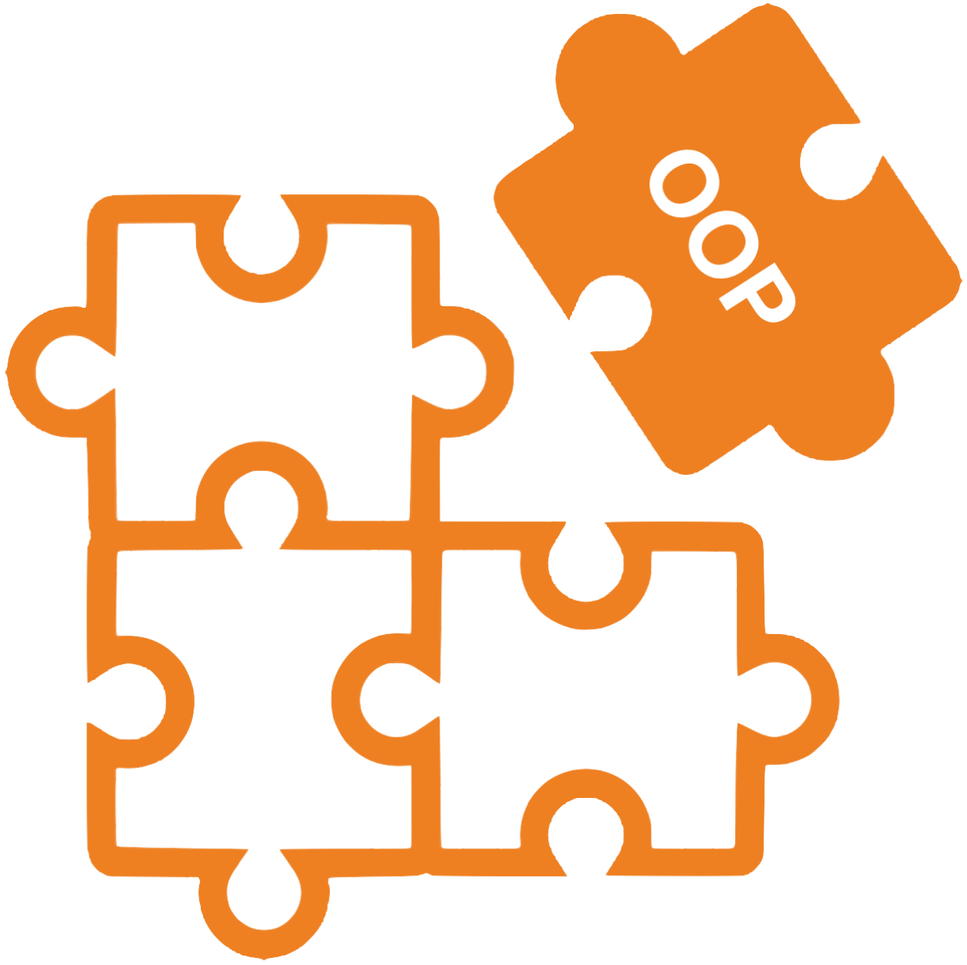
Adding Classes to a Package

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Adam Spannbauer
Machine Learning Engineer at Eastman

datacamp

Object oriented programming



Anatomy of a class

working in `work_dir/my_package/my_class.py`

```
# Define a minimal class with an attribute
class MyClass:
    """A minimal example class

    :param value: value to set as the ``attribute`` attribute
    :ivar attribute: contains the contents of ``value`` passed in init
    """

    # Method to create a new instance of MyClass
    def __init__(self, value):
        # Define attribute with the contents of the value param
        self.attribute = value
```

Using a class in a package

working in `work_dir/my_package/__init__.py`

```
from .my_class import MyClass
```

working in `work_dir/my_script.py`

```
import my_package

# Create instance of MyClass
my_instance = my_package.MyClass(value='class attribute value')

# Print out class attribute value
print(my_instance.attribute)
```

```
'class attribute value'
```

The self convention

working in `work_dir/my_package/my_class.py`

```
# Define a minimal class with an attribute
class MyClass:
    """A minimal example class

    :param value: value to set as the ``attribute`` attribute
    :ivar attribute: contains the contents of ``value`` passed in init
    """

    # Method to create a new instance of MyClass
    def __init__(self, value):
        # Define attribute with the contents of the value param
        self.attribute = value
```

```
my_instance = my_package.MyClass(value='class attribute value')
```

Let's Practice

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Leveraging Classes

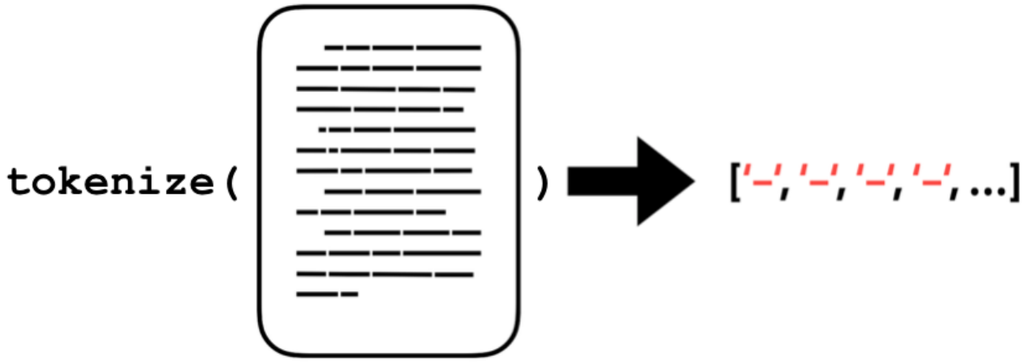
SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Adam Spannbauer
Machine Learning Engineer at Eastman

datacamp

Extending Document class

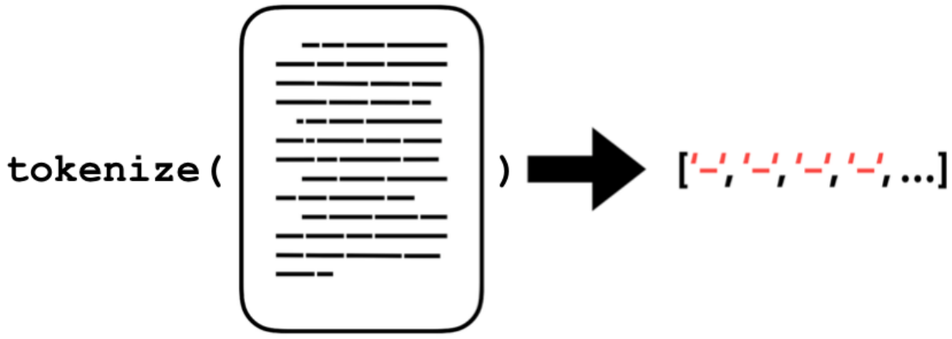
```
class Document:
    def __init__(self, text):
        self.text = text
```



Current document class

```
class Document:
    def __init__(self, text):
        self.text = text
```

```
def __init__():
    ...
```



Revising __init__

```
class Document:
    def __init__(self, text):
        self.text = text
        self.tokens = self._tokenize()

doc = Document('test doc')
print(doc.tokens)
```

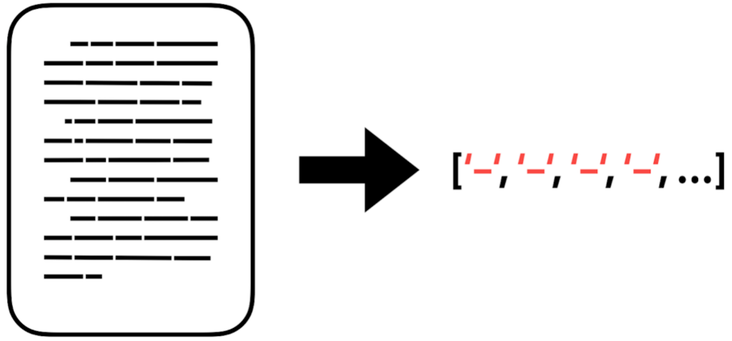
```
['test', 'doc']
```

Adding _tokenize() method

```
# Import function to perform tokenization
from .token_utils import tokenize
class Document:
    def __init__(self, text, token_regex=r'[a-zA-z]+'):
        self.text = text
        self.tokens = self._tokenize()
    def _tokenize(self):
        return tokenize(self.text)
```

Non-public methods

```
def __init__():
    ...
```



The risks of non-public methods

- Lack of documentation
- Unpredictability

Let's Practice

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Classes and the DRY principle

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Adam Spannbauer
Machine Learning Engineer at Eastman

datacamp

Creating a SocialMedia Class

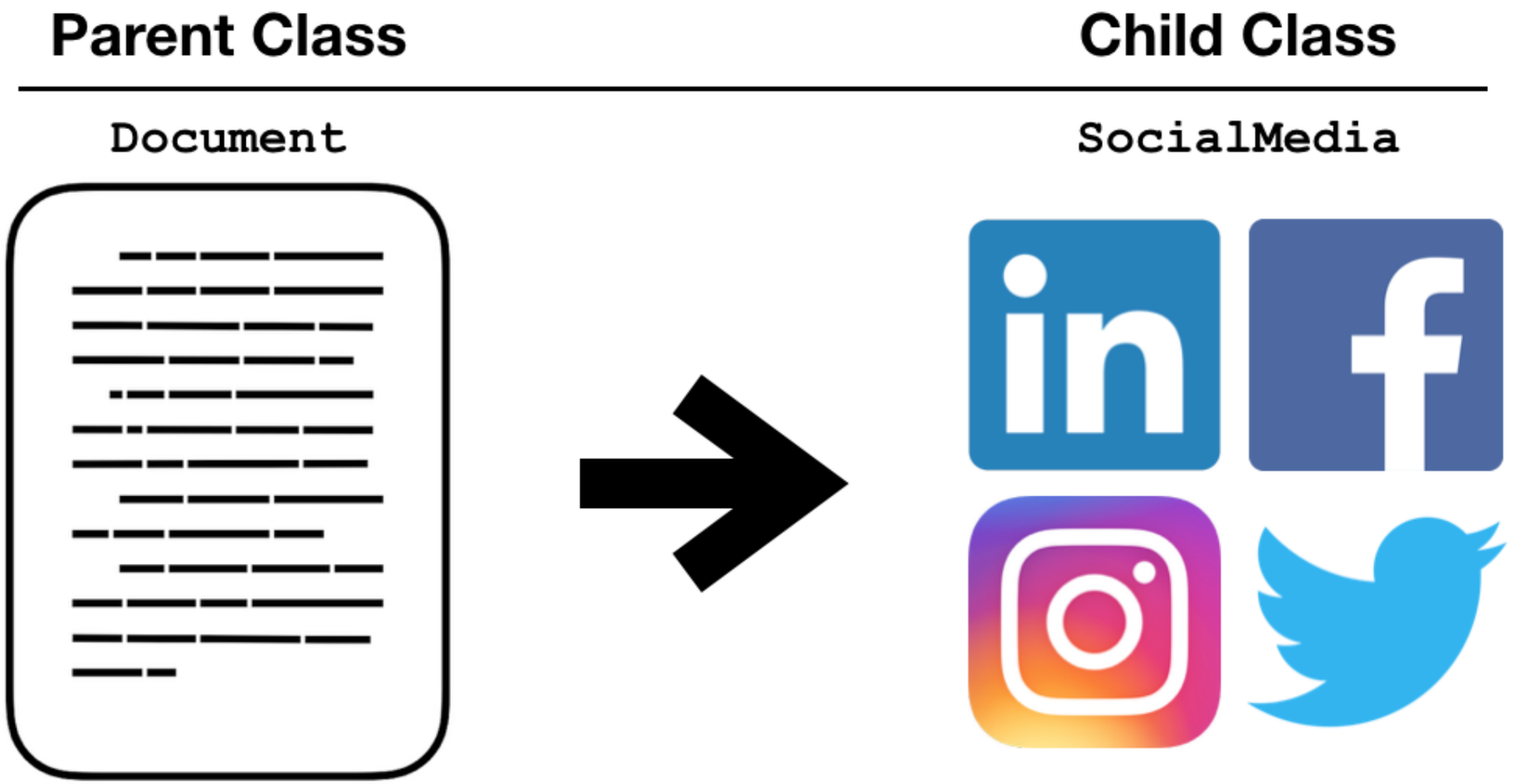
The DRY principle

The DRY principle

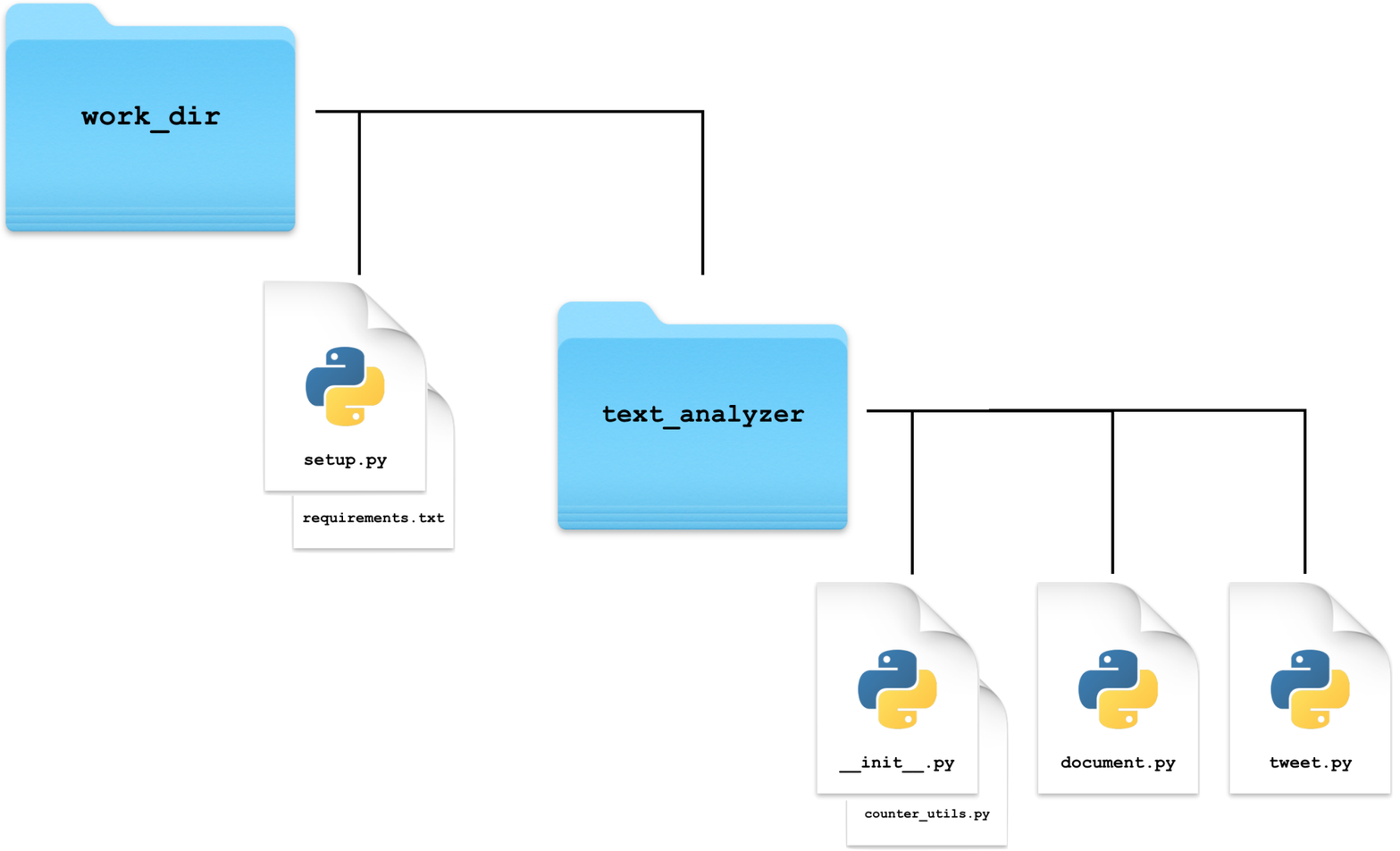
The DRY principle

The DRY principle

Intro to inheritance



Inheritance in Python



Inheritance in Python

```
# Import ParentClass object
from .parent_class import ParentClass

# Create a child class with inheritance
class ChildClass(ParentClass):
    def __init__(self):
        # Call parent's __init__ method
        ParentClass.__init__(self)
        # Add attribute unique to child class
        self.child_attribute = "I'm a child class attribute!"

# Create a ChildClass instance
child_class = ChildClass()
print(child_class.child_attribute)
print(child_class.parent_attribute)
```

Let's Practice

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

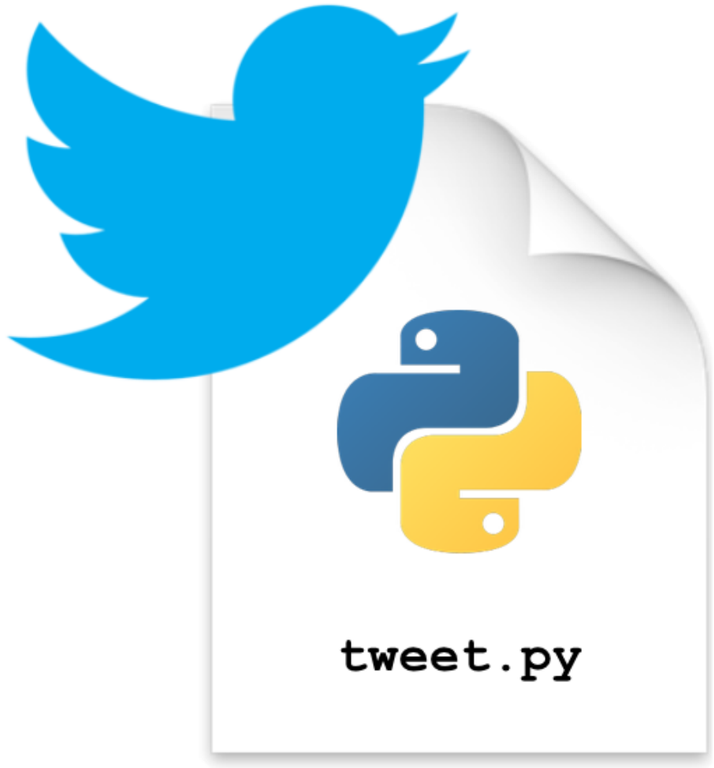
Multilevel inheritance

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

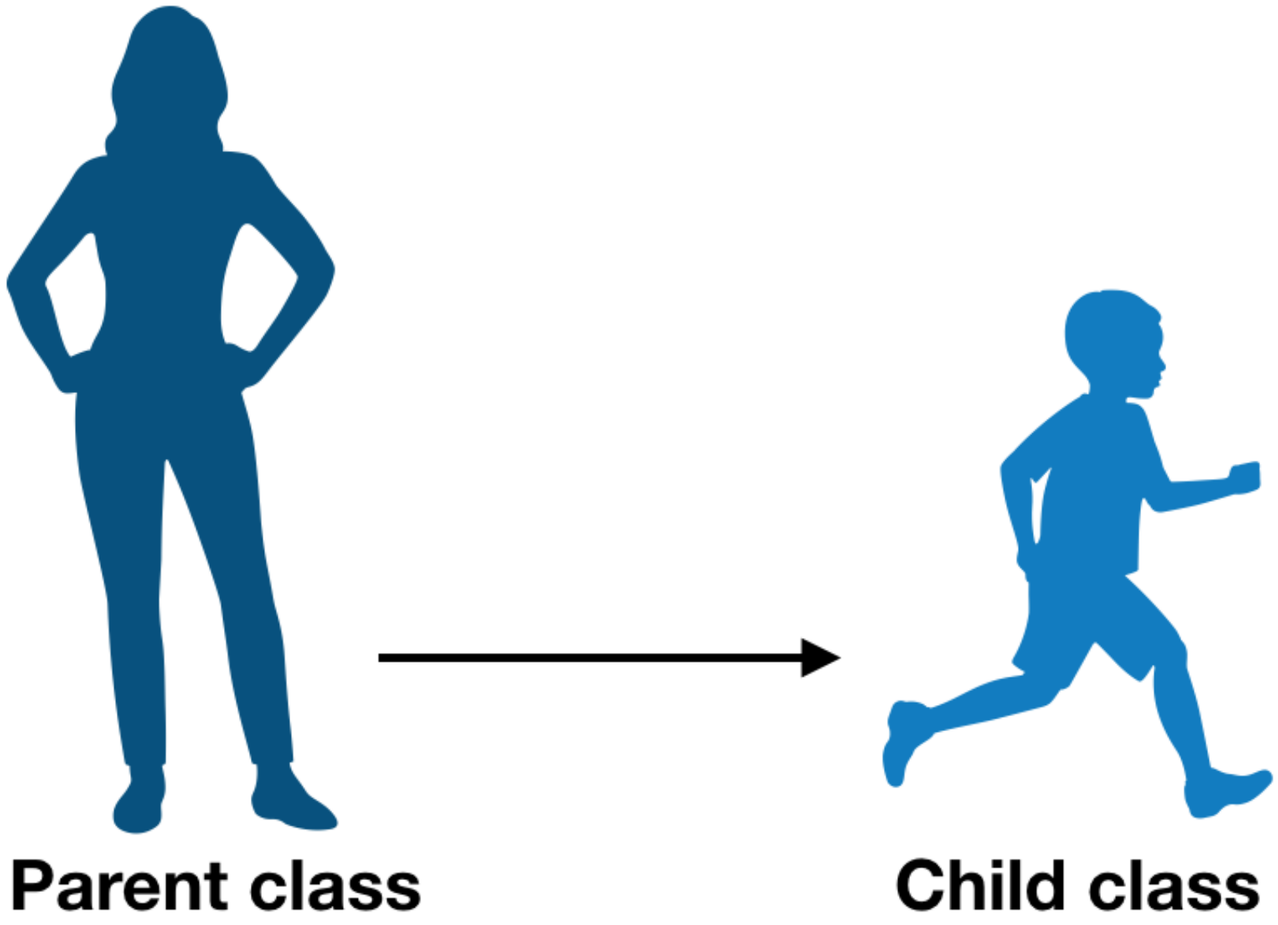
Adam Spannbauer
Machine Learning Engineer at Eastman

datacamp

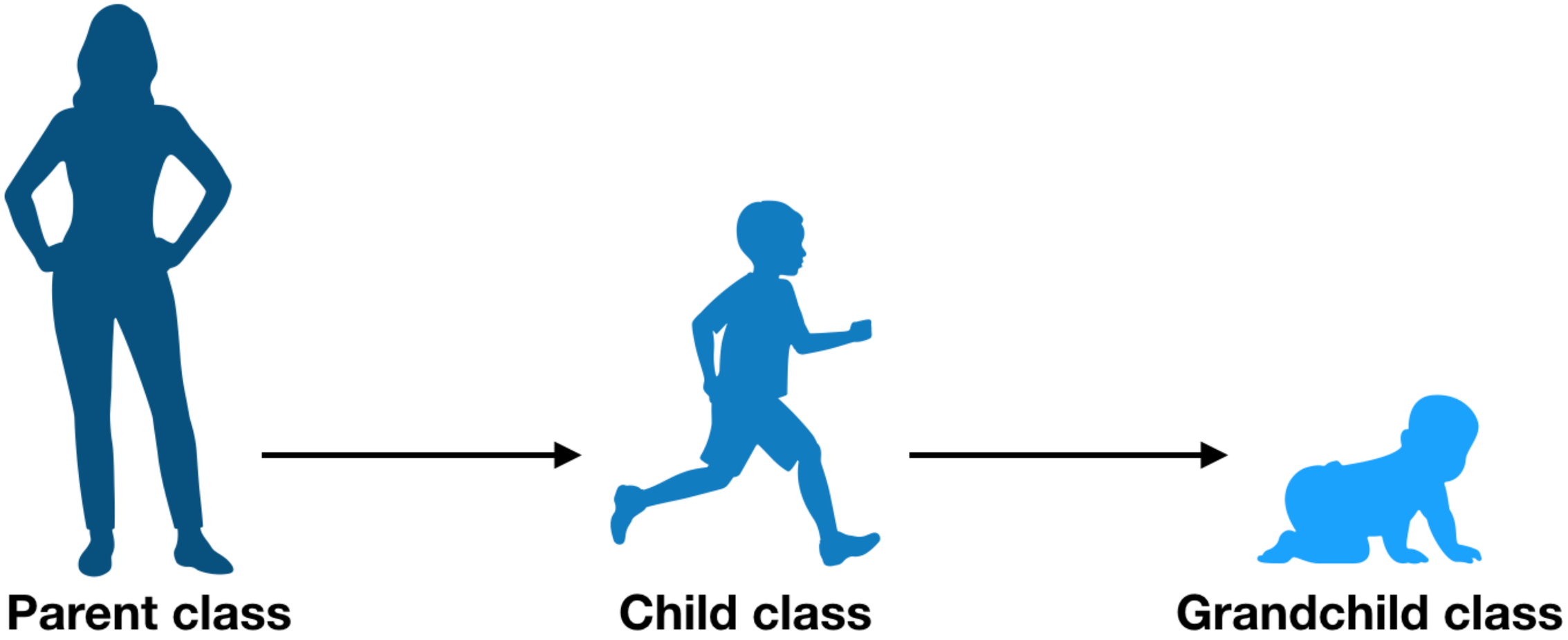
Creating a Tweet class



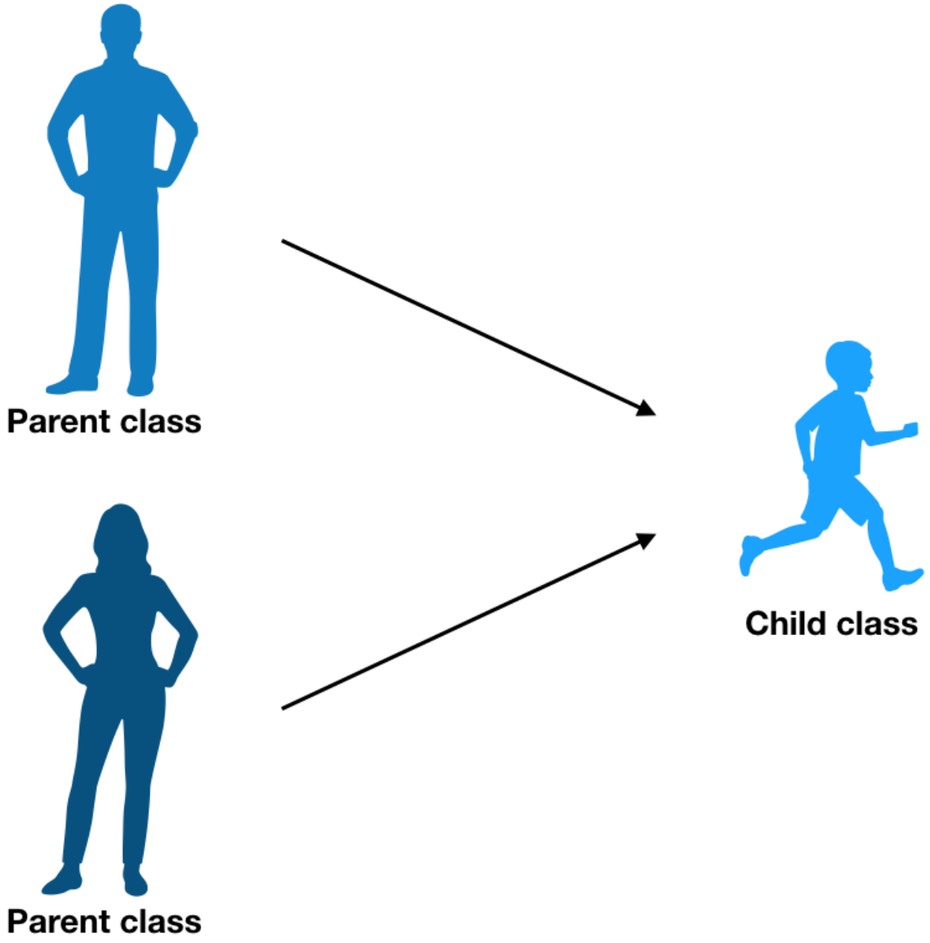
Multilevel inheritance



Multilevel inheritance



Multiple inheritance



Multilevel inheritance and super

```
class Parent:
    def __init__(self):
        print("I'm a parent!")


class Child(Parent):
    def __init__(self):
        Parent.__init__()
        print("I'm a child!")

class SuperChild(Parent):
    def __init__(self):
        super().__init__()
        print("I'm a super child!")
```

Learn more about multiple inheritance & `super()` .

Multilevel inheritance and super

```
class Parent:
    def __init__(self):
        print("I'm a parent!")

class SuperChild(Parent):
    def __init__(self):
        super().__init__()
        print("I'm a super child!")

class Grandchild(SuperChild):
    def __init__(self):
        super().__init__()
        print("I'm a grandchild!")

grandchild = Grandchild()
```

```
I'm a parent!
I'm a super child!
I'm a grandchild!
```

Keeping track of inherited attributes

```
# Create an instance of SocialMedia
sm = SocialMedia('@DataCamp #DataScience #Python #sklearn')
# What methods does sm have?  ¯\_(?)_/¯
dir(sm)
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', '__weakref__', '_count_hashtags',
'_count_mentions', '_count_words', '_tokenize', 'hashtag_counts',
'mention_counts', 'text', 'tokens', 'word_counts']
```

Let's Practice

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Documentation

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Adam Spannbauer
Machine Learning Engineer at Eastman

datacamp

Documentation in Python

- Comments

```
# Square the number x
```

- Docstrings

```
    """Square the number x

    :param x: number to square
    :return: x squared

    >>> square(2)
    4
    """
```

Comments

```
# This is a valid comment
x = 2
```

```
y = 3  # This is also a valid comment
```

```
# You can't see me unless you look at the source code
# Hi future collaborators!!
```

Effective comments

Commenting 'what'

```
# Define people as 5
people = 5

# Multiply people by 3
people * 3
```

Commenting 'why'

```
# There will be 5 people attending the party
people = 5

# We need 3 pieces of pizza per person
people * 3
```

Docstrings

```
def function(x):
    """High level description of function

    Additional details on function
```

Docstrings

```
def function(x):
    """High level description of function

    Additional details on function

    :param x: description of parameter x
    :return: description of return value
```

Example webpage generated from a docstring in the Flask package.

Docstrings

```
def function(x):
    """High level description of function

    Additional details on function

    :param x: description of parameter x
    :return: description of return value

    >>> # Example function usage
    Expected output of example function usage
    """
    # function code
```

Example docstring

```
def square(x):
    """Square the number x

    :param x: number to square
    :return: x squared

    >>> square(2)
    4
    """
    # `x * x` is faster than `x ** 2`
    # reference: https://stackoverflow.com/a/29055266/5731525
    return x * x
```

Example docstring output

```
help(square)
```

```
square(x)
    Square the number x

    :param x: number to square
    :return: x squared

    >>> square(2)
    4
```

Let's Practice

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Readability counts

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Adam Spannbauer
Machine Learning Engineer

datacamp

The Zen of Python

```
import this
```

```
The Zen of Python, by Tim Peters (abridged)


Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
The complex is better than complicated.
Readability counts.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
```

Descriptive naming

Poor naming

```
def check(x, y=100):
    return x >= y
```

Descriptive naming

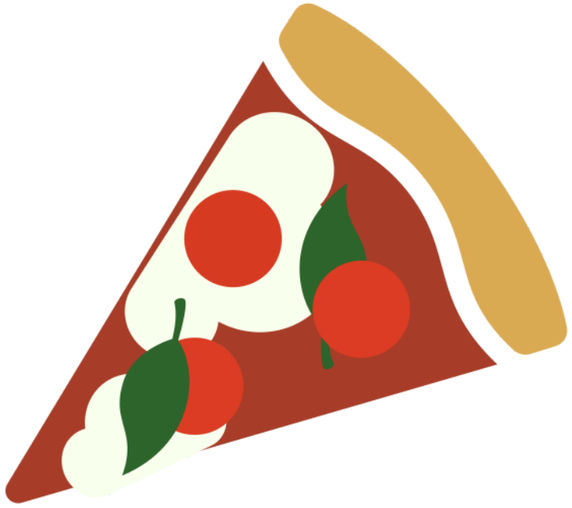
```
def is_boiling(temp, boiling_point=100):
    return temp >= boiling_point
```

Going overboard

```
def check_if_temperature_is_above_boiling_point(
        temperature_to_check,
        celsius_water_boiling_point=100):
    return temperature_to_check >= celsius_water_boiling_point
```

Keep it simple

```
The Zen of Python, by Tim Peters (abridged)

Simple is better than complex.
Complex is better than complicated.
```



Making a pizza - complex

```
def make_pizza(ingredients):
    # Make dough
    dough = mix(ingredients['yeast'],
                ingredients['flour'],
                ingredients['water'],
                ingredients['salt'],
                ingredients['shortening'])

    kneaded_dough = knead(dough)
    risen_dough = prove(kneaded_dough)

    # Make sauce
    sauce_base = sautee(ingredients['onion'],
                                ingredients['garlic'],
                                ingredients['olive oil'])

    sauce_mixture = combine(sauce_base,
                            ingredients['tomato_paste'],
                            ingredients['water'],
                            ingredients['spices'])

    sauce = simmer(sauce_mixture)
    ...
```

Making a pizza - simple

```
def make_pizza(ingredients):
    dough = make_dough(ingredients)
    sauce = make_sauce(ingredients)
    assembled_pizza = assemble_pizza(dough, sauce, ingredients)

    return bake(assembled_pizza)
```

When to refactor

Poor naming

```
def check(x, y=100):
    return x >= y
```

Descriptive naming

```
def is_boiling(temp, boiling_point=100):
    return temp >= boiling_point
```

Going overboard

```
def check_if_temperature_is_above_boiling_point(
        temperature_to_check,
        celsius_water_boiling_point=100):
    return temperature_to_check >= celsius_water_boiling_point
```

Let's Practice

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Unit testing

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Adam Spannbauer
Machine Learning Engineer at Eastman

datacamp

Why testing?

- Confirm code is working as intended
- Ensure changes in one function don't break another
- Protect against changes in a dependency

Testing in Python

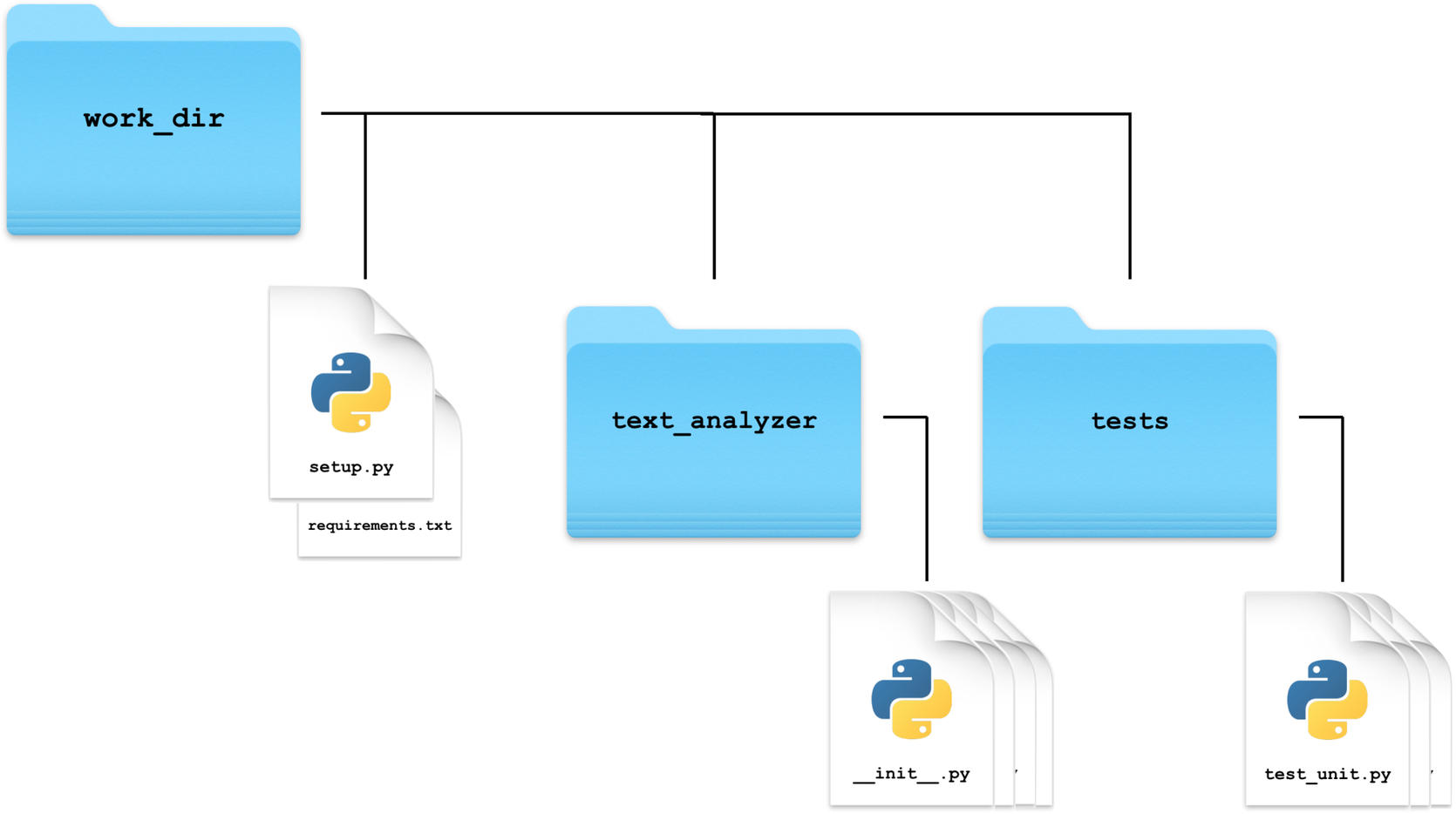
- `doctest`
- `pytest`

Using doctest

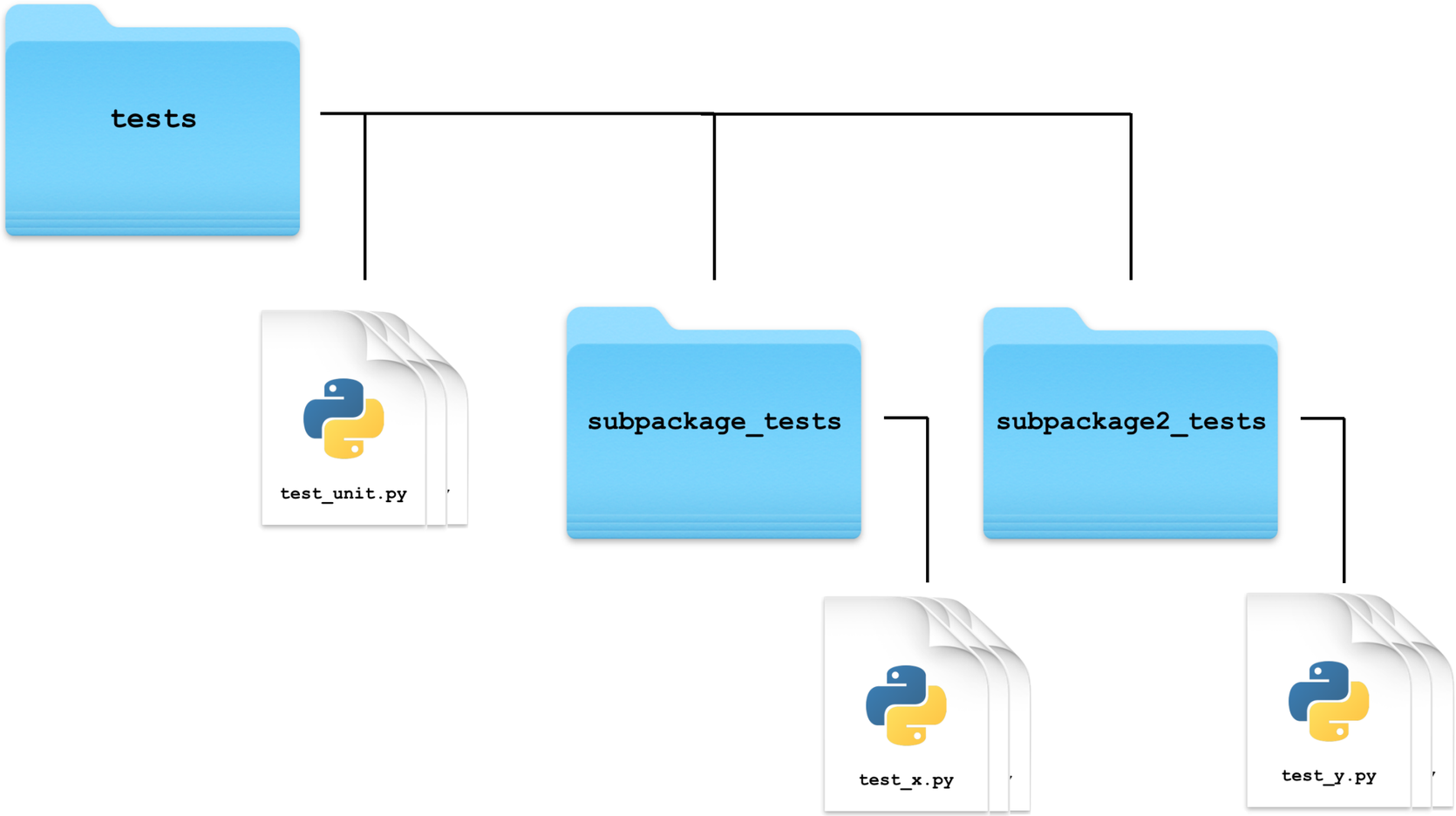
```
def square(x):
    """Square the number x

    :param x: number to square
    :return: x squared

    >>> square(3)
    9
    """
    return x ** x


import doctest
doctest.testmod()
```

```
Failed example:
    square(3)
Expected:
    9
Got:
    27
```

pytest structure



pytest structure



Writing unit tests

working in `workdir/tests/test_document.py`

```
from text_analyzer import Document



# Test tokens attribute on Document object
def test_document_tokens():
    doc = Document('a e i o u')

    assert doc.tokens == ['a', 'e', 'i', 'o', 'u']

# Test edge case of blank document
def test_document_empty():
    doc = Document('')

    assert doc.tokens == []
    assert doc.word_counts == Counter()
```

Writing unit tests

```
# Create 2 identical Document objects
doc_a = Document('a e i o u')
doc_b = Document('a e i o u')

# Check if objects are ==
print(doc_a == doc_b)
# Check if attributes are ==
print(doc_a.tokens == doc_b.tokens)
print(doc_a.word_counts == doc_b.word_counts)
```

```
False
True
True
```

Running pytest

working with `terminal`

```
datacamp@server:~/work_dir $ pytest
```

```
collected 2 items

tests/test_document.py ..                          [100%]

========== 2 passed in 0.61 seconds ==========
```

Running pytest

working with `terminal`

```
datacamp@server:~/work_dir $ pytest tests/test_document.py
```

```
collected 2 items

tests/test_document.py ..                          [100%]

========== 2 passed in 0.61 seconds ==========
```

Failing tests

working with `terminal`

```
datacamp@server:~/work_dir $ pytest
```

```
collected 2 items

tests/test_document.py F.

============== FAILURES ==============
________ test_document_tokens ________

def test_document_tokens(): doc = Document('a e i o u')

assert doc.tokens == ['a', 'e', 'i', 'o']
E AssertionError: assert ['a', 'e', 'i', 'o', 'u'] == ['a', 'e', 'i', 'o']
E Left contains more items, first extra item: 'u'
E Use -v to get the full diff

tests/test_document.py:7: AssertionError
====== 1 failed in 0.57 seconds ======
```

Let's Practice

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Documentation & testing in practice

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Adam Spannbauer
Machine Learning Engineer at Eastman

datacamp

Documenting projects with Sphinx

text_analyzer

Navigation

Classes

Utility Functions

Quick search

Go

Classes

class `text_analyzer.`**`Document`**(*text*)

Analyze text data

Parameters: **text** – text to analyze

Variables:
- **text** – Contains the text originally passed to the instance on creation
- **tokens** – Parsed list of words from `text`
- **word_counts** – `Counter` object containing counts of hashtags used in text

`plot_counts`(*attribute='word_counts', n_most_common=5*)

Plot most common elements of a `collections.Counter` instance attribute

Parameters:
- **attribute** – name of `Counter` attribute to use as object to plot
- **n_most_common** – number of elements to plot (using `Counter.most_common()`)

Returns: `None`; a plot is shown using `matplotlib`

```
>>> doc = Document("duck duck goose is fun")
>>> doc.plot_counts('word_counts', n_most_common=5)
```

Documenting classes

```
class Document:
    """Analyze text data

    :param text: text to analyze

    :ivar text: text originally passed to the instance on creation
    :ivar tokens: Parsed list of words from text
    :ivar word_counts: Counter containing counts of hashtags used in text
    """
    def __init__(self, text):
        ...
```

Continuous integration testing

DataCamp / text_analyzer build failing

Current Branches Build History Pull Requests Build #230 More options

new_feature update SocialMedia class

#230 failed

Commit 3080c4a

Ran for 1 min 13 sec

Compare 43dc3ba..3080c4a

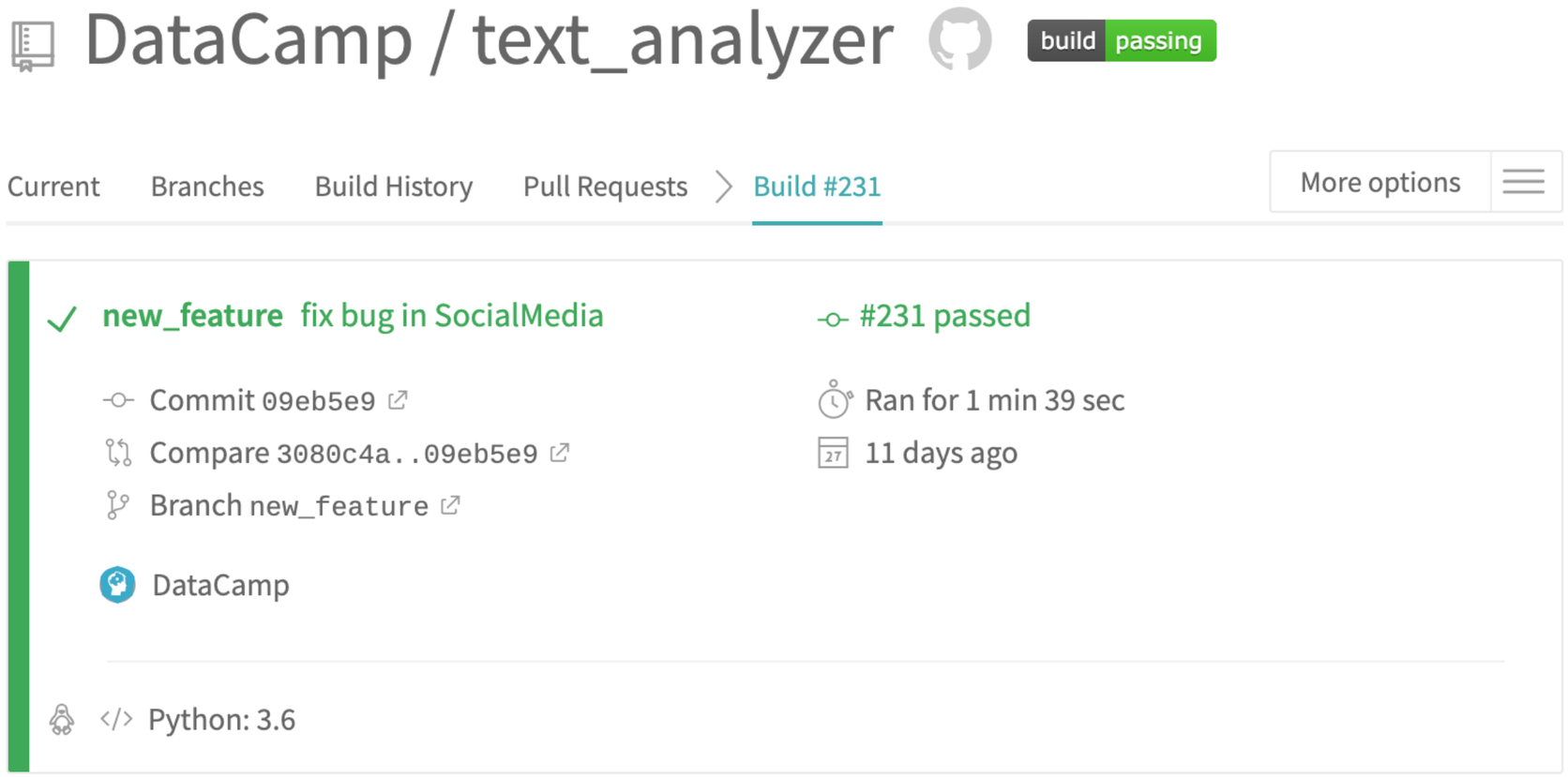
11 days ago

Branch new_feature

DataCamp

Python: 3.6

Continuous integration testing



Links and additional tools

- **Sphinx** - Generate beautiful documentation
- **Travis CI** - Continuously test your code
- **GitHub** & **GitLab** - Host your projects with git
- **Codecov** - Discover where to improve your projects tests
- **Code Climate** - Analyze your code for improvements in readability

Let's Practice

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Final Thoughts

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON

Adam Spannbauer
Machine Learning Engineer at Eastman

datacamp

Looking Back

- Modularity

```
def function()
    ...
```

```
class Class:
    ...
```

Looking Back

- Modularity
- Documentation

```
"""docstrings"""
```

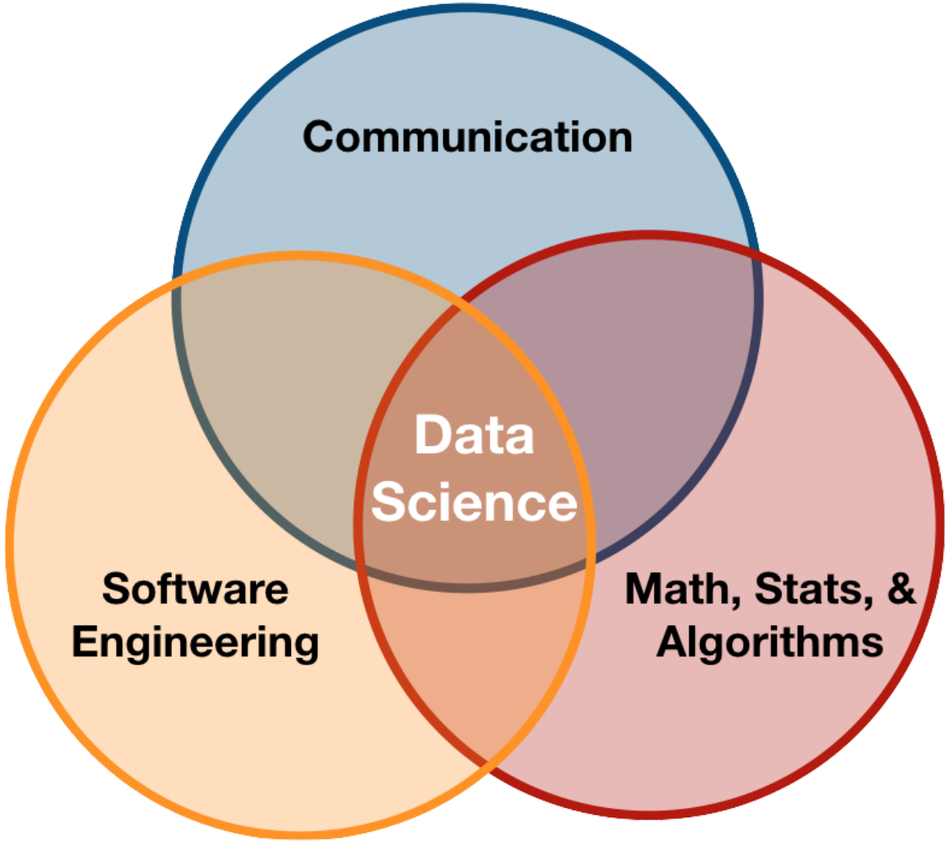
```
# Comments
```

Looking Back

- Modularity
- Documentation
- Automated testing


```
def f(x):
    """
    >>> f(x)
    expected output
    """
    ...
```

Data Science & Software Engineering



Good Luck!

SOFTWARE ENGINEERING FOR DATA SCIENTISTS IN PYTHON