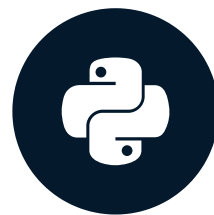


# Documentation

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON



**Adam Spannbauer**

Machine Learning Engineer at Eastman

# 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

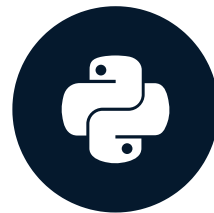
>>> square(2)
4
```

# Let's Practice

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON

# Readability counts

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON



**Adam Spannbauer**  
Machine Learning Engineer

# 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

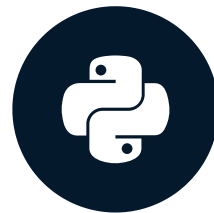
```
def check_if_temperature_is_above_boiling_point(  
    temperature_to_check,  
    celsius_water_boiling_point=100):  
    return temperature_to_check >= celsius_water_boiling_point
```

# Let's Practice

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON

# Unit testing

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON



**Adam Spannbauer**

Machine Learning Engineer at Eastman

# 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`



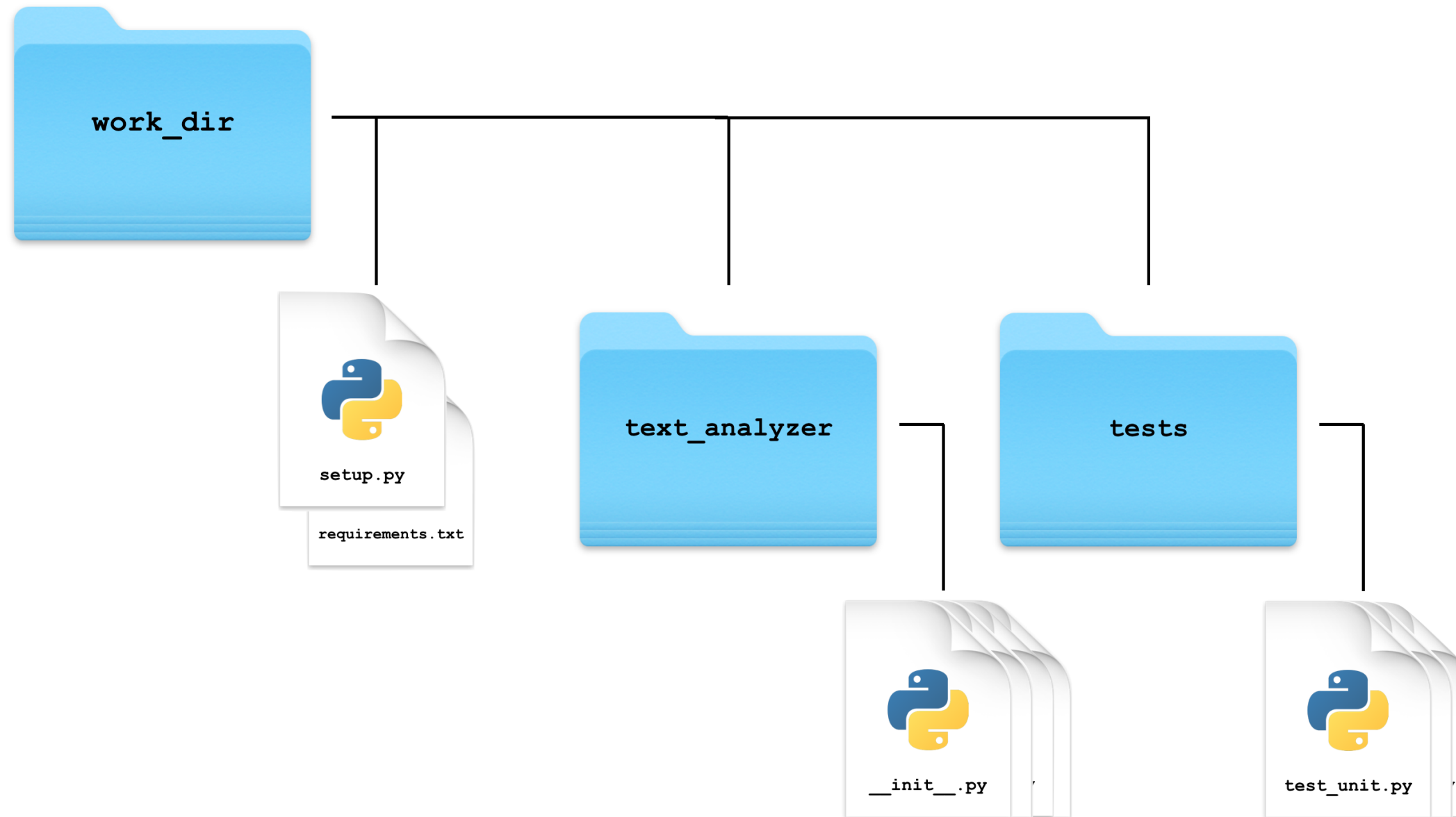
pytest

# Using doctest

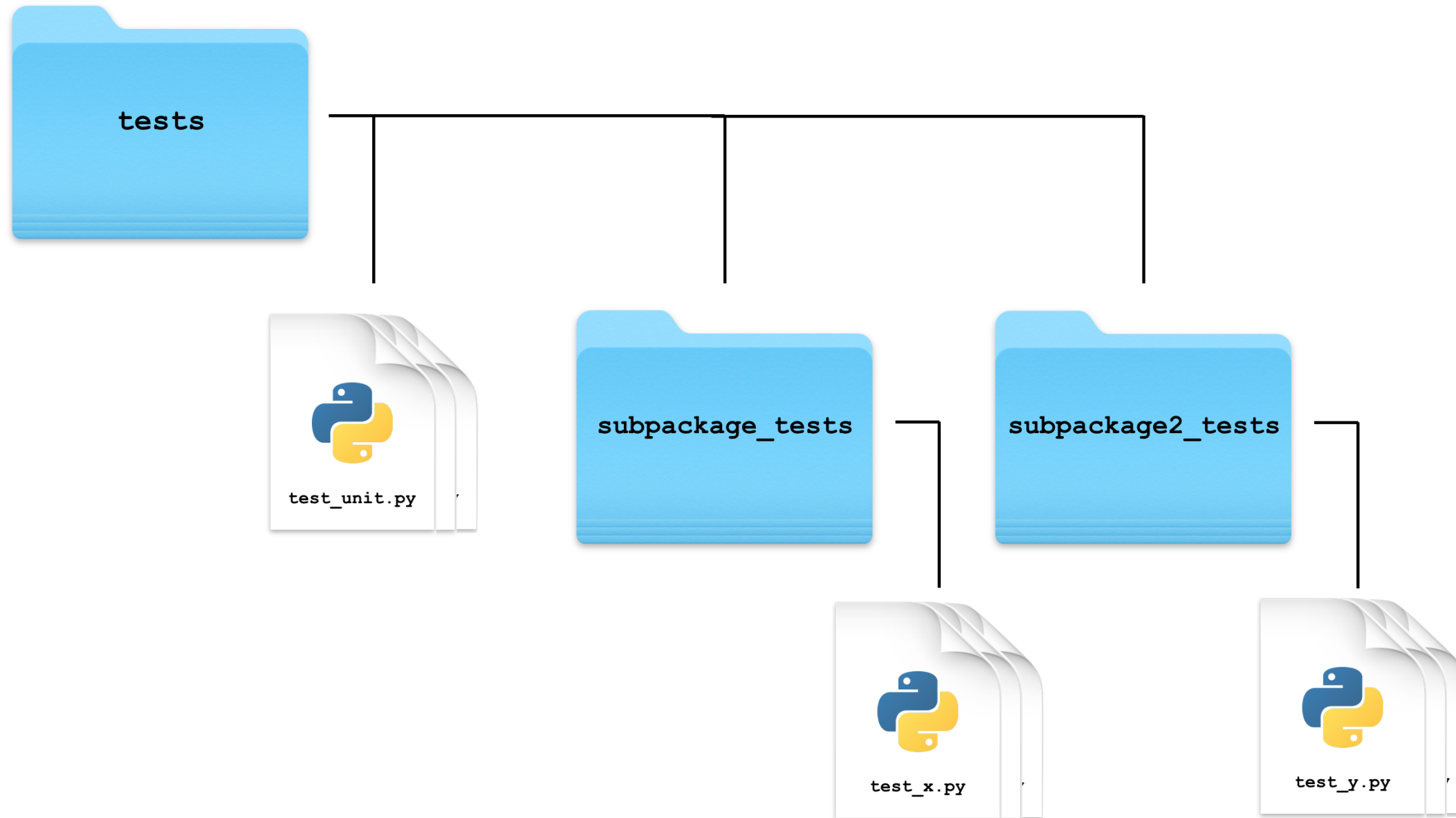
```
def square(x):  
    """Square the number x  
  
    :param x: number to square  
    :return: x squared  
  
    >>> square(3)  
    9  
    """  
    return x ** 3  
  
import doctest  
doctest.testmod()
```

```
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
```

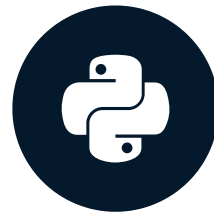
```
===== 1 failed in 0.57 seconds =====
```

# Let's Practice

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON

# Documentation & testing in practice

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON



**Adam Spannbauer**

Machine Learning Engineer at Eastman

# Documenting projects with Sphinx

## text\_analyzer

### Navigation

#### Classes

#### Utility Functions

### Quick search

## 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

 DataCamp / text\_analyzer  build passing

Current

Branches

Build History

Pull Requests

Build #231

More options

✓ new\_feature fix bug in SocialMedia

Commit 09eb5e9

Compare 3080c4a...09eb5e9

Branch new\_feature

DataCamp

Python: 3.6

#231 passed

Ran for 1 min 39 sec

11 days ago

# Links and additional tools

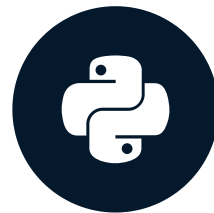
- [Sphinx](#) - Generate beautiful documentation
- [Travis CI](#) - Continuously test your code
- [GitHub](#) & [GitLab](#) - Host your projects with git
- [Codecov](#) - Discover where to improve your projects tests
- [Code Climate](#) - Analyze your code for improvements in readability

# Let's Practice

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON

# Final Thoughts

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON



**Adam Spannbauer**

Machine Learning Engineer at Eastman

# Looking Back

- Modularity

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

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



# Looking Back

- Modularity
- Documentation

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

```
# Comments
```

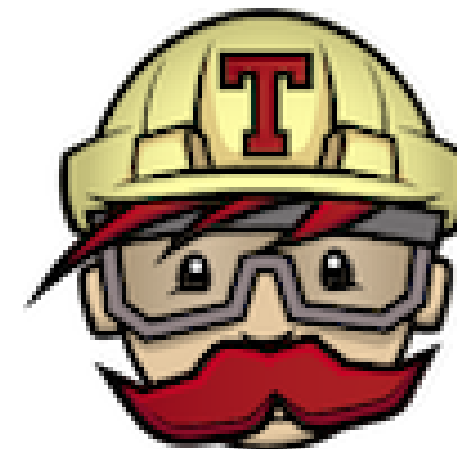




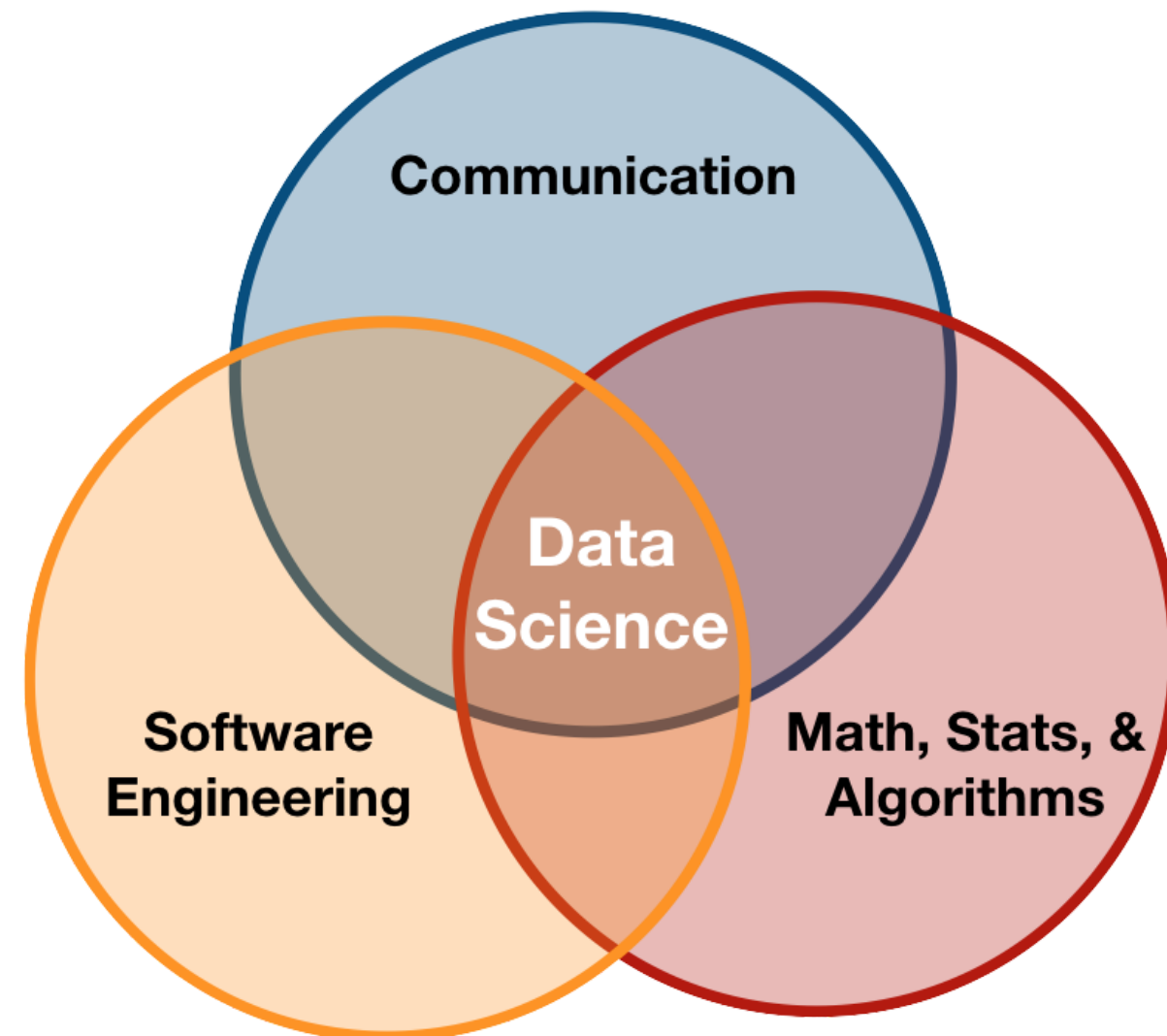
# Looking Back

- Modularity
- Documentation
- Automated testing

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



# Data Science & Software Engineering



# Good Luck!

SOFTWARE ENGINEERING PRINCIPLES IN PYTHON