AWS Machine Learning Foundations Course

Contents

[Software Practices I 3](#_Toc45240542)

[Writing clean code 3](#_Toc45240543)

[Writing modular code 3](#_Toc45240544)

[Writing efficient code 3](#_Toc45240545)

[optimizing\_code\_common\_books 3](#_Toc45240546)

[optimizing\_code\_holiday\_gifts 4](#_Toc45240547)

[Documentation 4](#_Toc45240548)

[Version Control 4](#_Toc45240549)

[Software Practices II 5](#_Toc45240550)

[Testing 5](#_Toc45240551)

[Test Driven Development 5](#_Toc45240552)

[Logging 5](#_Toc45240553)

[Code Review 5](#_Toc45240554)

[Explain issues and make suggestions 5](#_Toc45240555)

[Keep comments objective 6](#_Toc45240556)

[Provide code examples 6](#_Toc45240557)

[References 9](#_Toc45240558)

[Hello 11](#_Toc45240559)

[Hello 11](#_Toc45240560)

[Hello 11](#_Toc45240561)

# Software Practices I

## Writing clean code

Use meaningful names, and proper whitespace.

## Writing modular code

* DRY (Don’t repeat yourself)
* Abstract out logic to improve readability—like into a function
* Minimize number of entities
* Functions should do one thing—Single responsibility principle
* Arbitrary variable names can be more effective in certain functions
* Minimize number of arguments to a functions to 3

## Writing efficient code

* Code that runs infrequently for a short time need not be highly optimized
* Code that needs to run fast, such as a live feed, should be highly optimized
* Code can be refactored to be optimized after an initial solution
* Use vector operations over loops whenever possible
* Refactor using different data structures to make code more efficient
* When searching for solutions, it’s better to experiment with different solutions to find methods that are optimum, rather than stick with the most popular solution

Related links: [What makes sets faster than lists?](https://stackoverflow.com/questions/8929284/what-makes-sets-faster-than-lists/8929445#8929445)

### optimizing\_code\_common\_books

1. Using NumPy and it’s intersect1d instead of lists and loops makes a difference of 1386.01 times speed increase
2. Using sets over lists and their intersection method makes a difference of 4595.53 times speed increase
3. Set and intersection() is 3.315 times faster than using NumPy and it’s intersect1d

Related links: [numpy.intersect1d](https://numpy.org/doc/stable/reference/generated/numpy.intersect1d.html) and [Intersection() function Python - GeeksforGeeks](https://www.geeksforgeeks.org/intersection-function-python/)

### optimizing\_code\_holiday\_gifts

1. Arithmetic operations can be optimized over numpy arrays. Scalar values (entire rows) or vectors can be easily all multiplies, divied, added to, subtracted from, etc. much faster than using loops or other iterations

Related links: [1.4.2. Numerical operations on arrays](https://scipy-lectures.org/intro/numpy/operations.html), [How do I select elements of an array given condition?](https://stackoverflow.com/questions/3030480/how-do-i-select-elements-of-an-array-given-condition), and [numpy.sum](https://numpy.org/doc/stable/reference/generated/numpy.sum.html)

## Documentation

1. Using inline comments to add line level docs
   1. Useful for explaining code when code can’t speak
2. Using doc strings to add docs at the function and module level
   1. They can be one line to explain a single function
   2. Multiline docstrings have more parts such as Arguments: , Returns: , and a longer descriptions

## Version Control

Version control can be used to store, retrieve, and search through changes in a project. It helps protect the project developers from losing work, and takes care of the work of managing versions and change control.

# Software Practices II

## Testing

1. Unit tests are used to test small units of code
2. Pytest is used to process tests
3. We should only have one assert statement per test function
4. Pytest stops if there are syntax errors

## Test Driven Development

1. Writing tests before writing implementation code
2. Tests can check for all the different scenarios and edge cases you can think of, before even starting to write your function
3. You can also write better tests this way as your program evolves, rather than writing one hurried test at the end
4. When refactoring or adding to your code, tests help you rest assured that the rest of your code didn't break while you were making those changes (regression testing).

## Logging

Logging is valuable for understanding the events that occur while running your program. For example, if you run your model over night and see that it's producing ridiculous results the next day, log messages can really help you understand more about the context in which this occurred.

## Code Review

When your coworker finishes up some code that they want to merge to the team's code base, they might send it to you for review. You provide feedback and suggestions, and then they may make changes and send it back to you. When you are happy with the code, you approve and it gets merged to the team's code base.

### Explain issues and make suggestions

BAD: Make model evaluation code its own module - too repetitive.

BETTER: Make the model evaluation code its own module. This will simplify models.py to be less repetitive and focus primarily on building models.

GOOD: How about we consider making the model evaluation code its own module? This would simplify models.py to only include code for building models. Organizing these evaluations methods into separate functions would also allow us to reuse them with different models without repeating code.

### Keep comments objective

BAD: I wouldn't groupby genre twice like you did here... Just compute it once and use that for your aggregations.

BAD: You create this groupby dataframe twice here. Just compute it once, save it as groupby\_genre and then use that to get your average prices and views.

GOOD: Can we group by genre at the beginning of the function and then save that as a groupby object? We could then reference that object to get the average prices and views without computing groupby twice.

It helps if you remember that it’s a group effort, and that it would be better to frame it as something we all would like to get done, since it’s something we would have to work on later as well.

### Provide code examples

Let's say you were reviewing code that included the following lines:

first\_names = []

last\_names = []

**for** name **in** enumerate(df.name):

first, last = name.split(' ')

first\_names.append(first)

last\_names.append(last)

df['first\_name'] = first\_names

df['last\_names'] = last\_names

BAD: You can do this all in one step by using the pandas str.split method.

GOOD: We can actually simplify this step to the line below using the pandas str.split method. Found this on this stack overflow post: <https://stackoverflow.com/questions/14745022/how-to-split-a-column-into-two-columns>

df['first\_name'], df['last\_name'] = df['name'].str.split(' ', 1).str

# Object Oriented Programming

## Introduction

Objects have attributes and methods, or characteristics and actions.

A generic version of an object is it’s class. Classes are blueprints for creating objects.

‘self ‘ is a self-referential pointer, much like ‘this’. It is used to refer to the memory location of the object itself. Using ‘.’ notation, we can access the attributes and methods of a particular object.

‘\_\_init\_\_’ is used to initialize the values of the object upon object creation.

It is easier to modularize code using classes, where classes are stored in separate files, and then called to create objects in other files.

### shirt\_exercise

1. Create 2 objects ‘shirt\_one’ and ‘shirt\_two’ with given values
2. Modify shirt one with ‘change\_price’ and find it’s 12% discount price with ‘discount’
3. Find the total price of the two objects and store in ‘total’
4. Find the 14% and 6% discount prices of these 2 objects and store in ‘total\_discount’
5. Run tests using the cell at the end of the doc

### Get and Set methods

A get method is for obtaining an attribute value. A set method is for changing an attribute value.

Example:

**class** **Shirt**:

**def** **\_\_init\_\_**(self, shirt\_color, shirt\_size, shirt\_style, shirt\_price):

self.\_price = shirt\_price

**def** **get\_price**(self):

**return** self.\_price

**def** **set\_price**(self, new\_price):

self.\_price = new\_price

shirt\_one = Shirt('yellow', 'M', 'long-sleeve', 15)

print(shirt\_one.get\_price())

shirt\_one.set\_price(10)

One of the benefits of set and get methods is that, as previously mentioned in the course, you can hide the implementation from your user. Maybe originally a variable was coded as a list and later became a dictionary. With set and get methods, you could easily change how that variable gets accessed. Without set and get methods, you'd have to go to every place in the code that accessed the variable directly and change the code.

# References

*What makes sets faster than lists?* (n.d.). Retrieved from https://stackoverflow.com/questions/8929284/what-makes-sets-faster-than-lists/8929445#8929445

Hello

# Hello

## Hello

### Hello

#### Hello

Hello