AWS Machine Learning Foundations Course

Contents

[Software Practices I 4](#_Toc45885501)

[Writing clean code 4](#_Toc45885502)

[Writing modular code 4](#_Toc45885503)

[Writing efficient code 4](#_Toc45885504)

[optimizing\_code\_common\_books 4](#_Toc45885505)

[optimizing\_code\_holiday\_gifts 5](#_Toc45885506)

[Documentation 5](#_Toc45885507)

[Version Control 5](#_Toc45885508)

[Software Practices II 6](#_Toc45885509)

[Testing 6](#_Toc45885510)

[Test Driven Development 6](#_Toc45885511)

[Logging 6](#_Toc45885512)

[Code Review 6](#_Toc45885513)

[Explain issues and make suggestions 6](#_Toc45885514)

[Keep comments objective 7](#_Toc45885515)

[Provide code examples 7](#_Toc45885516)

[Object Oriented Programming 9](#_Toc45885517)

[Introduction 9](#_Toc45885518)

[Classes and Objects 9](#_Toc45885519)

[shirt\_exercise 9](#_Toc45885520)

[Get and Set methods 9](#_Toc45885521)

[OOP Syntax Exercise - Part 2 10](#_Toc45885522)

[Python Package: Gaussian Distribution 10](#_Toc45885523)

[Theory 11](#_Toc45885524)

[The Gaussian Class 15](#_Toc45885525)

[gaussian\_code\_exercise.ipynb 15](#_Toc45885526)

[Magic Methods in code 16](#_Toc45885527)

[magic\_methods 16](#_Toc45885528)

[Inheritance 17](#_Toc45885529)

[inheritance\_exercise\_clothing 17](#_Toc45885530)

[Inheritance Gaussian Class 17](#_Toc45885531)

[inheritance\_probability\_distribution 17](#_Toc45885532)

[Advanced OOP topics 17](#_Toc45885533)

[Static, Class, and Instance methods 18](#_Toc45885534)

[Class attributes and instance attributes 18](#_Toc45885535)

[Mixins 19](#_Toc45885536)

[Mixin examples 20](#_Toc45885537)

[Python Decorators 21](#_Toc45885538)

[References 25](#_Toc45885539)

[Hello 27](#_Toc45885540)

[Hello 27](#_Toc45885541)

[Hello 27](#_Toc45885542)

# 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.

## Classes and Objects

### 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.

### OOP Syntax Exercise - Part 2

1. Every method in the class must have ‘self’ as the first argument

## Python Package: Gaussian Distribution

Python Package that will be able to

1. Read dataset
2. Calculate mean, standard deviation
3. Plot histogram and probability density function
4. Add 2 Gaussian Distributions

Later, it will be extended to work with binomial distributions

Related links: [Probability density function](https://en.wikipedia.org/wiki/Probability_density_function), [Normal distribution](https://en.wikipedia.org/wiki/Normal_distribution), [Binomial distribution](https://en.wikipedia.org/wiki/Binomial_distribution), [Variance](https://en.wikipedia.org/wiki/Variance), [Standard deviation](https://en.wikipedia.org/wiki/Standard_deviation), [How to do Normal Distributions Calculations](https://statistics.laerd.com/statistical-guides/normal-distribution-calculations.php), [14. Normal Probability Distributions](https://www.intmath.com/counting-probability/14-normal-probability-distribution.php)

(Wikipedia, Probability density function) (Wikipedia, Normal distribution), (Wikipedia, Binomial distribution) (Wikipedia, Variance), (Wikipedia, Standard deviation) (Mathematics), (Laerd)

## Theory

#### Probability Density Function:

n probability theory, a probability density function (PDF), or density of a continuous random variable, is a function whose value at any given sample (or point) in the sample space (the set of possible values taken by the random variable) can be interpreted as providing a relative likelihood that the value of the random variable would equal that sample.[2] In other words, while the absolute likelihood for a continuous random variable to take on any particular value is 0 (since there are an infinite set of possible values to begin with), the value of the PDF at two different samples can be used to infer, in any particular draw of the random variable, how much more likely it is that the random variable would equal one sample compared to the other sample.



In a more precise sense, the PDF is used to specify the probability of the random variable falling within a particular range of values, as opposed to taking on any one value. This probability is given by the integral of this variable's PDF over that range—that is, it is given by the area under the density function but above the horizontal axis and between the lowest and greatest values of the range. The probability density function is nonnegative everywhere, and its integral over the entire space is equal to 1.

#### Example

Suppose bacteria of a certain species typically live 4 to 6 hours. The probability that a bacterium lives exactly 5 hours is equal to zero. A lot of bacteria live for approximately 5 hours, but there is no chance that any given bacterium dies at exactly 5.0000000000... hours. However, the probability that the bacterium dies between 5 hours and 5.01 hours is quantifiable. Suppose the answer is 0.02 (i.e., 2%). Then, the probability that the bacterium dies between 5 hours and 5.001 hours should be about 0.002, since this time interval is one-tenth as long as the previous. The probability that the bacterium dies between 5 hours and 5.0001 hours should be about 0.0002, and so on.

In these three examples, the ratio (probability of dying during an interval) / (duration of the interval) is approximately constant, and equal to 2 per hour (or 2 hour−1). For example, there is 0.02 probability of dying in the 0.01-hour interval between 5 and 5.01 hours, and (0.02 probability / 0.01 hours) = 2 hour−1. This quantity 2 hour−1 is called the probability density for dying at around 5 hours. Therefore, the probability that the bacterium dies at 5 hours can be written as (2 hour−1) dt. This is the probability that the bacterium dies within an infinitesimal window of time around 5 hours, where dt is the duration of this window. For example, the probability that it lives longer than 5 hours, but shorter than (5 hours + 1 nanosecond), is (2 hour−1)×(1 nanosecond) ≈ 6×10−13 (using the unit conversion 3.6×1012 nanoseconds = 1 hour).

There is a probability density function f with f(5 hours) = 2 hour−1. The integral of f over any window of time (not only infinitesimal windows but also large windows) is the probability that the bacterium dies in that window.

#### Variance

In probability theory and statistics, variance is the expectation of the squared deviation of a random variable from its mean. Informally, it measures how far a set of numbers are spread out from their average value.







#### Standard deviation

In statistics, the standard deviation is a measure of the amount of variation or dispersion of a set of values.[1] A low standard deviation indicates that the values tend to be close to the mean (also called the expected value) of the set, while a high standard deviation indicates that the values are spread out over a wider range.

The standard deviation of a random variable, statistical population, data set, or probability distribution is the square root of its variance. It is algebraically simpler, though in practice less robust, than the average absolute deviation.[2][3] A useful property of the standard deviation is that, unlike the variance, it is expressed in the same units as the data.

In addition to expressing the variability of a population, the standard deviation is commonly used to measure confidence in statistical conclusions. For example, the margin of error in polling data is determined by calculating the expected standard deviation in the results if the same poll were to be conducted multiple times. This derivation of a standard deviation is often called the "standard error" of the estimate or "standard error of the mean" when referring to a mean. It is computed as the standard deviation of all the means that would be computed from that population if an infinite number of samples were drawn and a mean for each sample were computed.



In the case where *X* takes random values from a finite data set *x*1, *x*2, ..., *xN*, with each value having the same probability, the standard deviation is



The standard deviation of a continuous real-valued random variable X with probability density function p(x) is



#### Normal distribution

In probability theory, a normal (or Gaussian or Gauss or Laplace–Gauss) distribution is a type of continuous probability distribution for a real-valued random variable. The general form of its probability density function is





The simplest case of a normal distribution is known as the standard normal distribution. This is a special case μ=0 and σ=1, and it is described by this probability density function:



Standard deviation of one means there is a width of 1 between each band in the curve.



This means that the PDF generates, for each x, a probability p, and the plot of those p is the gaussian distribution.

To calculate the probability:





#### Binomial distribution

In probability theory and statistics, the binomial distribution with parameters n and p is the discrete probability distribution of the number of successes in a sequence of n independent experiments, each asking a yes–no question, and each with its own boolean-valued outcome: success/yes/true/one (with probability p) or failure/no/false/zero (with probability q = 1 − p). A single success/failure experiment is also called a Bernoulli trial or Bernoulli experiment and a sequence of outcomes is called a Bernoulli process; for a single trial, i.e., n = 1, the binomial distribution is a Bernoulli distribution. The binomial distribution is the basis for the popular binomial test of statistical significance.

The binomial distribution is frequently used to model the number of successes in a sample of size n drawn with replacement from a population of size N. If the sampling is carried out without replacement, the draws are not independent and so the resulting distribution is a hypergeometric distribution, not a binomial one. However, for N much larger than n, the binomial distribution remains a good approximation, and is widely used.

In general, if the random variable X follows the binomial distribution with parameters n ∈ ℕ and p ∈ [0,1], we write X ~ B(n, p). The probability of getting exactly k successes in n independent Bernoulli trials is given by the probability mass function:



for *k* = 0, 1, 2, ..., *n*, where



is the binomial coefficient, hence the name of the distribution. The formula can be understood as follows. k successes occur with probability pk and n − k failures occur with probability (1 − p)n − k. However, the k successes can occur anywhere among the n trials, and there nCk different ways of distributing k successes in a sequence of n trials.

This means that for each PDF value of x, it generates k successes with probability p. Plotting those probabilities gives us the Binomial Distribution plot.

## The Gaussian Class

The class stores:

1. Mean
2. Standard Deviation
3. The dataset

It has functions for:

1. Loading data
2. Calculating mean
3. Calculating standard dev
4. Plotting a histogram
5. Calculating the probability density function

### gaussian\_code\_exercise.ipynb

1. SQRT function is in math.sqrt
2. Mean of a data set is the simple average, calculated by summing the row and then dividing it by the number of records
3. Standard deviation differs between a population and a sample of a population
4. For the standard deviation of the population
   1. Calculate the mean
   2. Calculate the squared difference by subtracting the mean from each value and then squaring the difference
   3. Calculate the mean of the squared difference by simple average of the values
   4. Calculate the standard deviation by finding the square root of the mean squared difference
   5. 
5. For the standard deviation of a sample of the population
   1. Calculate the mean
   2. Calculate the squared difference by subtracting the mean from each value and then squaring the difference
   3. Calculate the mean of the squared difference by dividing the N values of the squared differences by N-1
   4. Calculate the standard deviation by finding the square root of the mean squared difference
   5. 
6. Create a histogram of data using plt.hist()

Observations from answers.py

1. Self.stdev() can be refactored to be less repetitive
2. I did better on readability on calculating the PDF function
3. I did better in making sure I was getting only floating-point values

## Magic Methods in code

The aim is to be able to use overriding of default python behavior to implement addition of 2 objects of the Gaussian class.

Magic methods allow us to customize default python behavior. For example, the \_\_init\_\_ method can customize how python instantiates an object. \_\_add\_\_ can be used to add 2 objects, and \_\_repr\_\_ can be used to change how the object is represented when printed.

### magic\_methods

1. Add the means to get mean value, assign that to a new object’s mean and return it
2. Sqrt the sum of squared values of standard deviations, assign that to a new object’s standard deviation and return it
3. Return a string as part of \_\_repr\_\_

## Inheritance

Inheriance is where a choild class inherits methods and attributes from it’s parent class.

Updates to the parent class trickle down to the child class.

### inheritance\_exercise\_clothing

1. Create a new class Blouse
2. Add methods to new class: triple\_price
3. Add methods to parent class—this is called in the child class as well: calculate\_shipping
4. Make sure to call constructor of parent class in child class

## Inheritance Gaussian Class

The distribution super class is used as the base class for the Gaussian and the Binomial classes.

### inheritance\_probability\_distribution

1. The distribution class is the parent class for the gaussian class
2. The code has been refactored to adjust the gaussian class init method—it now initializes the distribution class
3. The read data method has been overridden in the gaussian class
4. The functionality remains the same throughout

## Advanced OOP topics

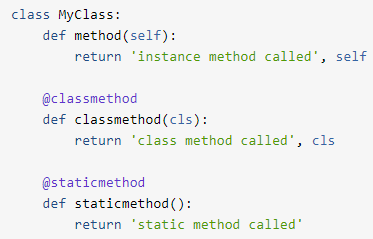
Related links: [Python's Instance, Class, and Static Methods Demystified](https://realpython.com/instance-class-and-static-methods-demystified/), [Class and Instance Attributes](https://www.python-course.eu/python3_class_and_instance_attributes.php), [Mixins for Fun and Profit](https://easyaspython.com/mixins-for-fun-and-profit-cb9962760556), [Primer on Python Decorators](https://realpython.com/primer-on-python-decorators/)

(Python, Python's Instance, Class, and Static Methods Demystified), (Python, Primer on Python Decorators), (Class and Instance Attributes), (Mixins for Fun and Profit)

### Static, Class, and Instance methods

* Classes are objects themselves
* Instance methods are able to modify instance state, as well as class state.
* Class methods can modify class state only, but not the state of any instance
* Class methods can be used to create alternate constructors for a class
* Class methods are used in factory methods
* Static methods can’t modify class or instance state, and act like regular functions
* Static methods are a way for namespacing methods, they belong to the class’s namespace
* Static and class methods communicate and (to a certain degree) enforce developer intent about class design. This can have maintenance benefits.

#### Class, Instance, and Static method example



### Class attributes and instance attributes

Instance attributes are owned by the specific instances of a class. That is, for two different instances, the instance attributes are usually different. Class attributes stay with class objects, while instance attributes are unique to all instances. Class attributes are attributes which are owned by the class itself. They will be shared by all the instances of the class. Therefore they have the same value for every instance. We define class attributes outside all the methods, usually they are placed at the top, right below the class header.

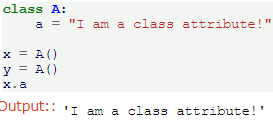
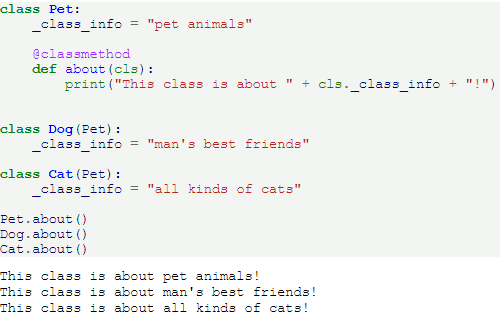




Figure Class attribute example

Class attributes are stored in the  dictionary while the instance attributes are stored in the dictionary.

#### Class attribute and method example

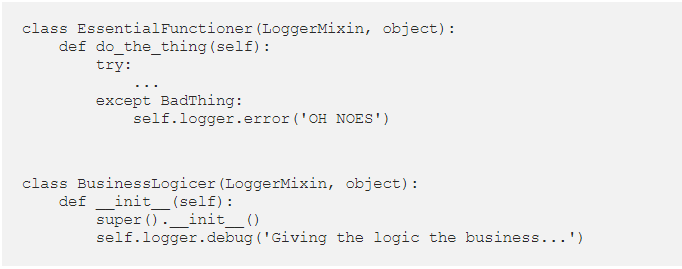
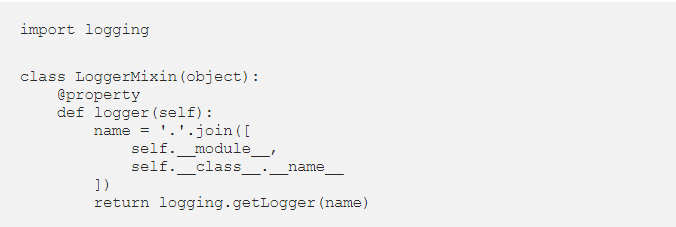


## Mixins

Mixins are classes that encapsulate a specific functionality. They can be inherited to provide specific behavour to an object—not exactly a parent class, more like an enhanced import of functionality—“mixing in functionality”.

### Mixin examples

#### Example 1



#### Example 2



## Python Decorators

Placeholder for notes about python decorators.

## Modularized Code

1. The Distribution class has been shifted to the GeneralDistribution.py file.
2. The GaussianDistribution.py file contains the Gaussian class
3. example\_code.py is used to create a Gaussian object and print it’s mean

## Packages

Packages have a fixed structure. They need an \_\_init\_\_.py file to initialize, and a setup.py file to install.

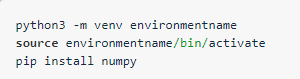
Multiple modules make up a package. The name of the package should be the same as the name of the folder the modules are in, along with the \_\_init\_\_.py file.

setup.py has metadata about the package, such as name, version number, description, etc. To install a local python package, type **pip install .** in the folder where the package is stored.

To see where the package has been installed, type **package-name**.\_\_file\_\_ .

## Virtual Environments

Conda can be used to create environments.



pip and venv will be used for this exercise.

## Exercise: Making a Package and Pip Installing

1. Copy files into directory 3a\_python\_package with **cp -i example\_code.py Gaussiandistribution.py Generaldistribution.py /home/workspace/3a\_python\_package**
2. Create a new folder with **mkdir 'distributions'**
3. Copy files to distributions folder with **cp -i example\_code.py Gaussiandistribution.py Generaldistribution.py distributions**
4. Remove files with **rm example\_code.py Gaussiandistribution.py Generaldistribution.py** from 3a\_python\_package
5. Create \_\_init\_\_.py file and add **from .Gaussiandistribution import Gaussian**  to it in **'distributions'**  folder
6. Create setup.py file in folder
7. Add a . and change the import to **from .Generaldistribution import Distribution** in **Gaussiandistribution.py**
8. Add the following code to setup.py

from setuptools import setup

setup(name = 'distributions',

version = '0.1',

description = 'Gaussian Distribution',

packages = ['distributions'],

zip\_safe = False)

1. Install package with **pip install .**
2. Try using the package in the python interpreter

# References

*14. Normal Probability Distributions.* (n.d.). Retrieved from Interactive Mathematics: https://www.intmath.com/counting-probability/14-normal-probability-distribution.php

*Class and Instance Attributes.* (n.d.). Retrieved from https://easyaspython.com/mixins-for-fun-and-profit-cb9962760556

*Class and Instance Attributes.* (n.d.). Retrieved from https://www.python-course.eu/python3\_class\_and\_instance\_attributes.php

Laerd. (n.d.). *How to do Normal Distributions Calculations.* Retrieved from https://statistics.laerd.com/statistical-guides/normal-distribution-calculations.php

Mathematics, I. (n.d.). *14. Normal Probability Distributions.* Retrieved from https://www.intmath.com/counting-probability/14-normal-probability-distribution.php

*Mixins for Fun and Profit.* (n.d.). Retrieved from https://easyaspython.com/mixins-for-fun-and-profit-cb9962760556

Python, R. (n.d.). *Primer on Python Decorators.* Retrieved from https://realpython.com/primer-on-python-decorators/

Python, R. (n.d.). *Python's Instance, Class, and Static Methods Demystified.* Retrieved from https://realpython.com/instance-class-and-static-methods-demystified/

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

Wikipedia. (n.d.). *Binomial distribution.* Retrieved from https://en.wikipedia.org/wiki/Binomial\_distribution

Wikipedia. (n.d.). *Normal distribution.* Retrieved from https://en.wikipedia.org/wiki/Normal\_distribution

Wikipedia. (n.d.). *Probability density function.* Retrieved from https://en.wikipedia.org/wiki/Probability\_density\_function

Wikipedia. (n.d.). *Standard deviation.* Retrieved from https://en.wikipedia.org/wiki/Standard\_deviation

Wikipedia. (n.d.). *Variance.* Retrieved from https://en.wikipedia.org/wiki/Variance

Hello

# Hello

## Hello

### Hello

#### Hello

Hello