Classes

What is a Class?

Defines properties and behaviors of objects. The properties and functions of a class can have diffrent levelrs of access. - Public: Open to use. You do not need to create an object of said class to have access functions or properties that contain this object modifier. - Protected: A familiy event. Only the class and its desendants have access to the properties and methods. - Private: Only the class has access to properties and methods.

Class Definition

When creating a class you need a **constructor** which define the properties of a class.

Objects

Contain the properties and behaviors defined in their class. Objects are stored in memory and their properties can contain their own copy if attributes defined in the class.

Varibles

Class Variables: Variables *shared* by all objects that share a class. **Instance/Object Variable**: Variables unique to each object.

Into the Constructor

- `__init__()` is a constructor that run automaticaly when you create an object.
- `self` refeences the current instance of the class.
- You can create optional parameters by giving your parametes default values.
  - All prededing parameters are required ie. if class has paramters A, B, and C. If you want B to be optional you have to give both B and C default values.

Dog Class

Imagine you have a class named **Dog**, what feature of a dog will you need to add to your class.

Dog Class Setup Dog

- Name: `str`
- Breed: `str`
- Size: `int`
- Fur Type: `str`
- Fur Color: `str`
- Bark Sound: `str`
- Mood: `str`
- Has Owner: `bool`

Python Implementation In Python, a constructed is defined with the `def` keyword like all python functions and with the method name of `__init__`. Inside the arguments section is the keyword `self` which is needed to access class properties.

```
class Dog:
    def __init__(self):
        self.name = None
        self.breed = None
        self.size = -1
        self.fur_type = None
        self.fur_color = None
        self.bark_sound = None # file path
        self.mood = None
        self.has_owner = None
```

```
class Dog:
    def __init__(self, n, b, s, ft, fc, bs, m:, ho):
        self.name = n
        self.breed = b
        self.size = s
        self.fur_type = ft
        self.fur_color = fc
        self.bark_sound = bs # file path
        self.mood = m
        self.has_owner = ho
```

```
class Dog:
    def __init__(self, n:str = 'NA', b:str = 'NA', s:int = 0, ft:str = 'NA',
                fc:str = 'NA', bs:str = 'NA', m:str = 'NA', ho:bool = False):
```

```
        self.name = n
        self.breed = b
        self.size = s
        self.fur_type = ft
        self.fur_color = fc
        self.bark_sound = bs # file path
        self.mood = m
        self.has_owner = ho
```

Table

			Fur		Bark		Has
Name	Breed	Size	Type	Fur Color	Sound	Mood	Owner
Fido	Whippet	3	Soft/Short	Tan	Fido__bark.mp3	Hyper	True
Husker	Beagle	2	Short/Soft	White/Brown	Husker__bark.m4a	Cranky	True
Philip	Dalmation	4	Soft/Short	White/Balck	Philip__barl.flac	Sleepy	False

---

Encapsulation

Encapsulation is the bundling of data (variables) and the methods (functions) that operate on that data into a single unit — a class. It also means restricting direct access to some of the object's components, which is a way to enforce *data hiding and protection*.

Getters and Setters

Used to access or update private variables in a controlled way.

Code Example

```
class Person:
    def __init__(self, name:str):
        self._name = name

    def get_name(self):
        return self._name

    def set_name(self, name:str):
        self._name = name
```

Code Example Using the `@property` Decorator

```
class Person:
    def __init__(self, name:str):
        self._name = name

    @propery
    def name(self):
        return self._name

    @name.setter
    def name(self, name:str):
        self._name = name
```