Developing a growth mindset towards your programming ability

Mindsets about intelligence (and programming)

Fixed Mindset

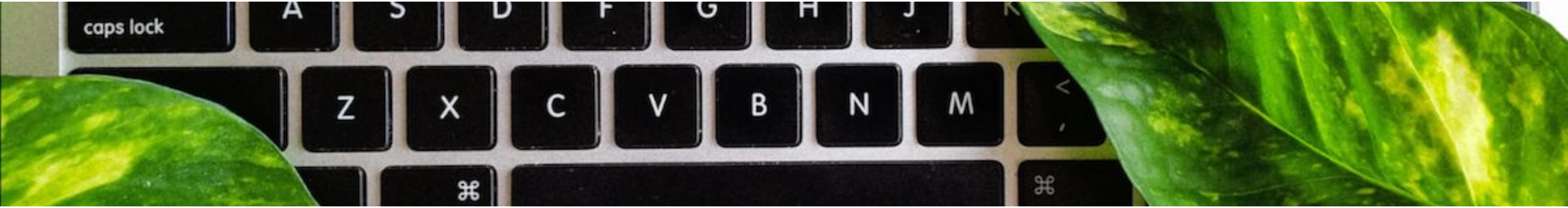
Human traits (including programming skills) are ***fixed/innate***.

Fixed mindset about programming: You have a certain amount of programming ability and *can't do anything to change it.*

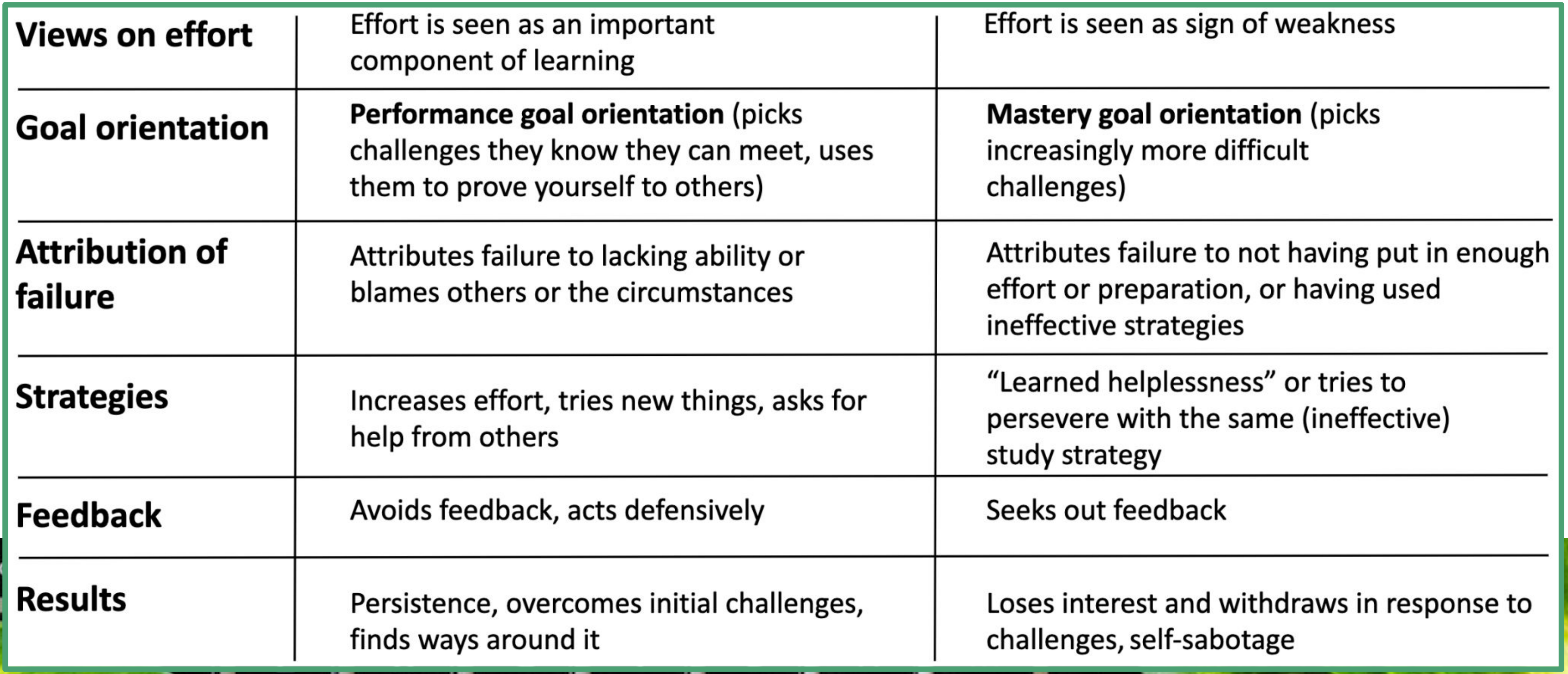
Growth mindset

Human traits (such as programming skills) are ***malleable and can be shaped/developed***.

Programming skill can be developed through personal effort, good learning strategies, and feedback.



Which of these are indicative of a growth mindset?



Views on effort	Effort is seen as an important component of learning	Effort is seen as sign of weakness
Goal orientation	**Performance goal orientation** (picks challenges they know they can meet, uses them to prove yourself to others)	**Mastery goal orientation** (picks increasingly more difficult challenges)
Attribution of failure	Attributes failure to lacking ability or blames others or the circumstances	Attributes failure to not having put in enough effort or preparation, or having used ineffective strategies
Strategies	Increases effort, tries new things, asks for help from others	"Learned helplessness" or tries to persevere with the same (ineffective) study strategy
Feedback	Avoids feedback, acts defensively	Seeks out feedback
Results	Persistence, overcomes initial challenges, finds ways around it	Loses interest and withdraws in response to challenges, self-sabotage

How do these individuals demonstrate growth mindsets?

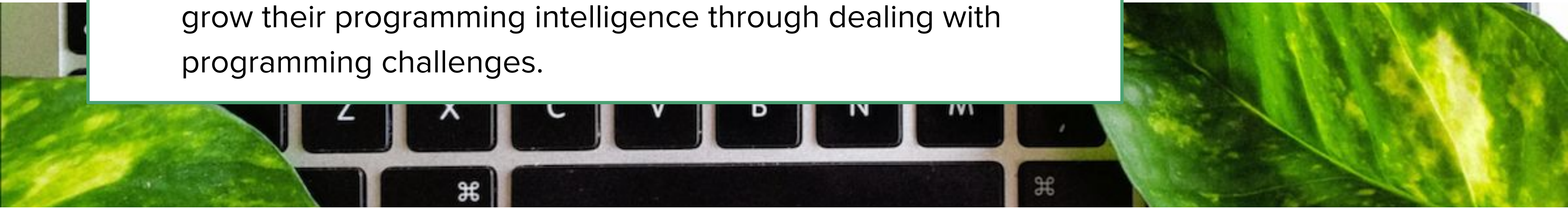
Learn more about their beginnings in programming!

Thinking back on mindset…

- Describe a time when you were learning something new other than programming (e.g., from school, at work or in everyday life) where you had to work really hard on a challenging task. Maybe you made a lot of mistakes, became extremely frustrated and wanted to give up, but with practice and perseverance you were able to succeed. Please be specific about the kinds of mistakes you made and how you overcame them.
- What advice would you give a beginning programmer in BILD 62 to help them cope with the challenge of learning to write and debug Python programs? Be sure to emphasize to them how to grow their programming intelligence through dealing with programming challenges.

Respond on Canvas for credit.

Your (anonymous) input will be shared with future classes!



Topics from this lecture & corresponding notebook

- Syntax of **for** and **while** loops
- How to iterate through strings, lists, and dictionaries
- Using a counter to count loop iterations
- Looping over lists of indices
- Calling functions within functions
- Using **`break`** to interrupt a loop, and **`continue`** to skip a loop
- Functions we learned: **`range(), enumerate()`**

Object-oriented programming

BILD 62

Objectives for today

- Access **attributes** and execute **methods** of objects
- Define **classes** and recognize class definition syntax
- Understand how to manipulate **instances** of a class

Everything in Python is an **object** (even functions!)

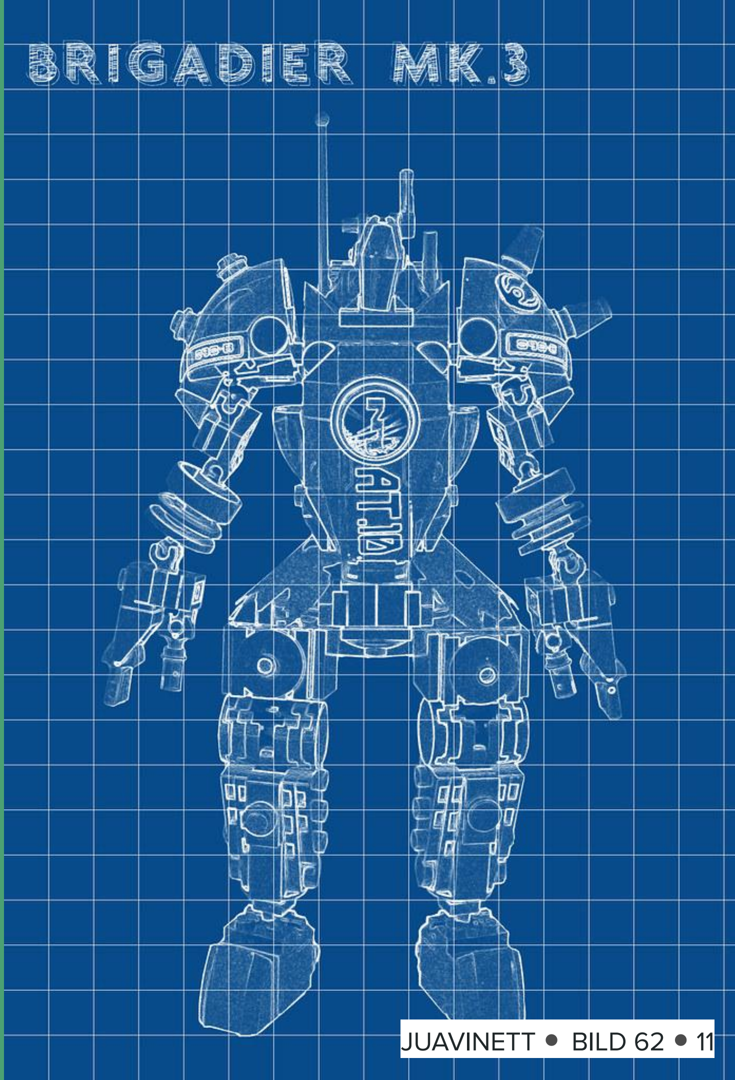
Object-oriented programming (OOP) is a programming paradigm in which code is organized around objects.

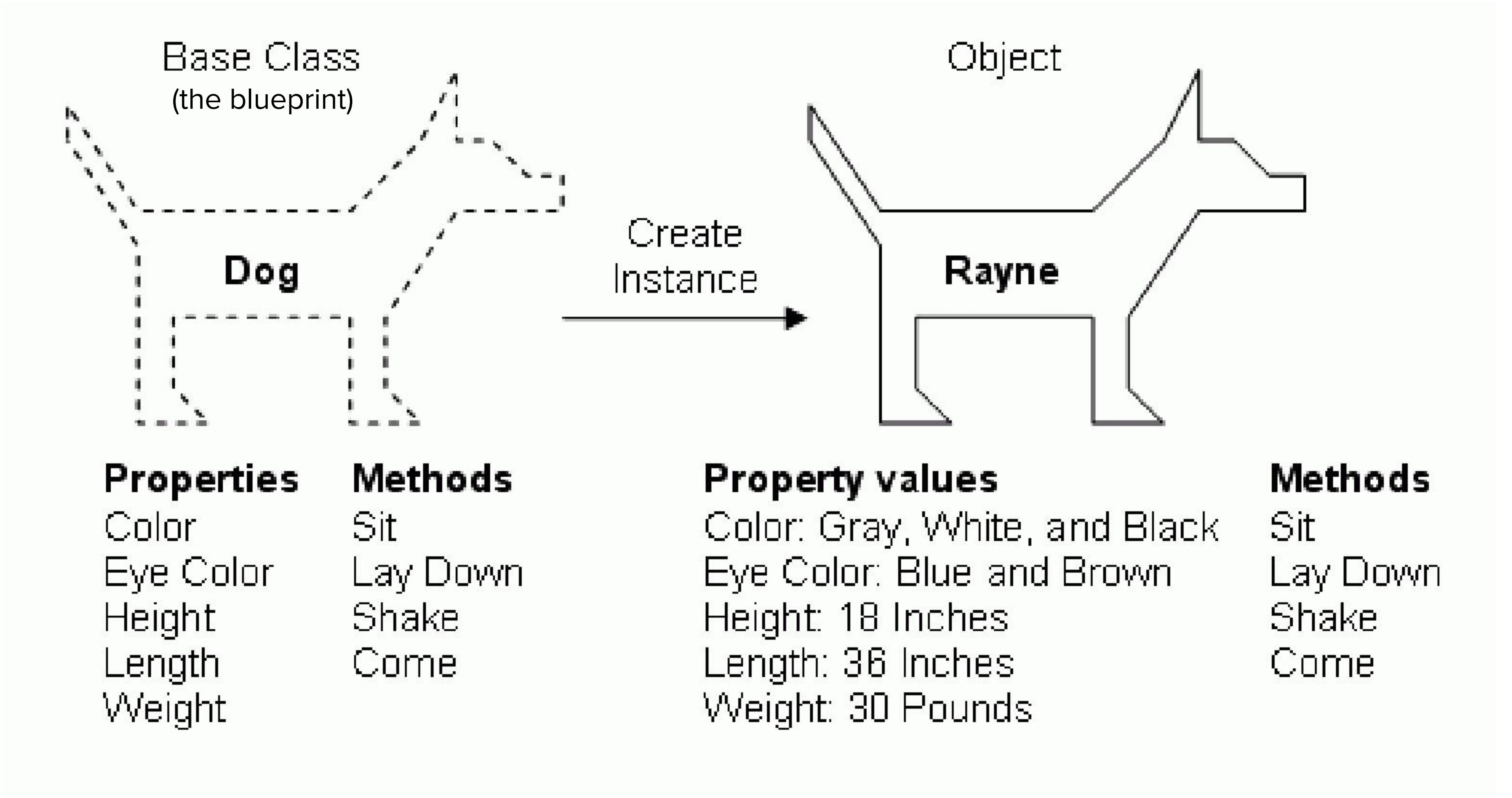
Objects come in different **classes**.*

- An **object** is an entity that stores data.
- An object's **class** defines specific properties objects of that class will have.
- An **instance** is a separate object of a certain **class**

* We've been referring to different "**types**" (e.g., integers, tuples, dictionaries) but even these can be called **classes**.

Think of **classes** as the blueprint for creating and defining objects and their properties (methods, attributes, etc.). They keep related things together and organized.





Base Class
(the blueprint)

Dog

Create Instance

Object

Rayne

Properties	Methods
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

Property values	Methods
Color: Gray, White, and Black	Sit
Eye Color: Blue and Brown	Lay Down
Height: 18 Inches	Shake
Length: 36 Inches	Come
Weight: 30 Pounds	

Image: http://justsajid.com/skills/objects-everything-is-an-object-in-csharp/

Objects are an organization of data (**attributes**), with associated code to operate on that data (**methods**: functions defined and called directly on the objects).

Syntax:

```
obj.method()
```

```
obj.attribute
```

For a hypothetical object called **`neuron`** how would you execute its method, **`spike`**?

https://www.menti.com/bl6714d9u21t

1. `neuron.spike`
2. `neuron.spike()`
3. `spike.neuron`
4. `spike.neuron()`

If neuron has an attribute **`diameter`**, how would you access it?

https://www.menti.com/bl6714d9u21t

1. `neuron.diameter`
2. `neuron.diameter()`
3. `diameter(neuron)`
4. `diameter.neuron`

Functions vs. methods

All methods are functions.

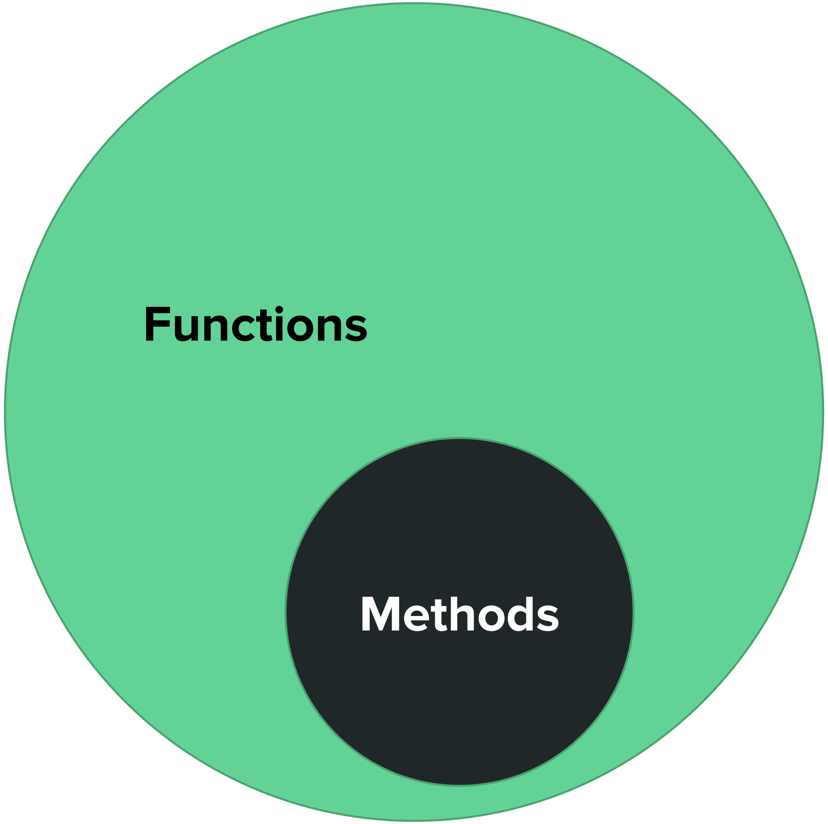
Methods are special functions attached to a variable type.

All functions are NOT methods.

```
my_variable.method_call()
```

acts like

```
function_call(my_variable)
```



Function reminders

- **`def`** defines a function
- **`function_name()`** - parentheses are required to execute a function
- **`function_name(input1`**) - input parameters are specified within the function parentheses
- **`function_name(input1, input2)`** - functions can take multiple parameters as inputs
- **`input1`** and **`input2`** can then be used within your function when it executes
- To store the output from a function, you'll need a return statement

Methods can…

- Use the object’s data
- Modify that data (e.g. `my_list.reverse()`) or *not* (e.g., `my_string.swapcase()`)
    - Methods of an **immutable** object will never change its value!
- Return a value (e.g. `my_list.pop()`) or *not* (e.g. `my_list.reverse()`)
- Accept additional arguments in parenthese (e.g. `my_list.pop()`) or *not* (e.g. `my_list.reverse()`)

Make sure you read the documentation!

Classes

A class is defined almost like a function, but using the **`class`** keyword.

The class definition usually contains a number of class method definitions (a function in a class).

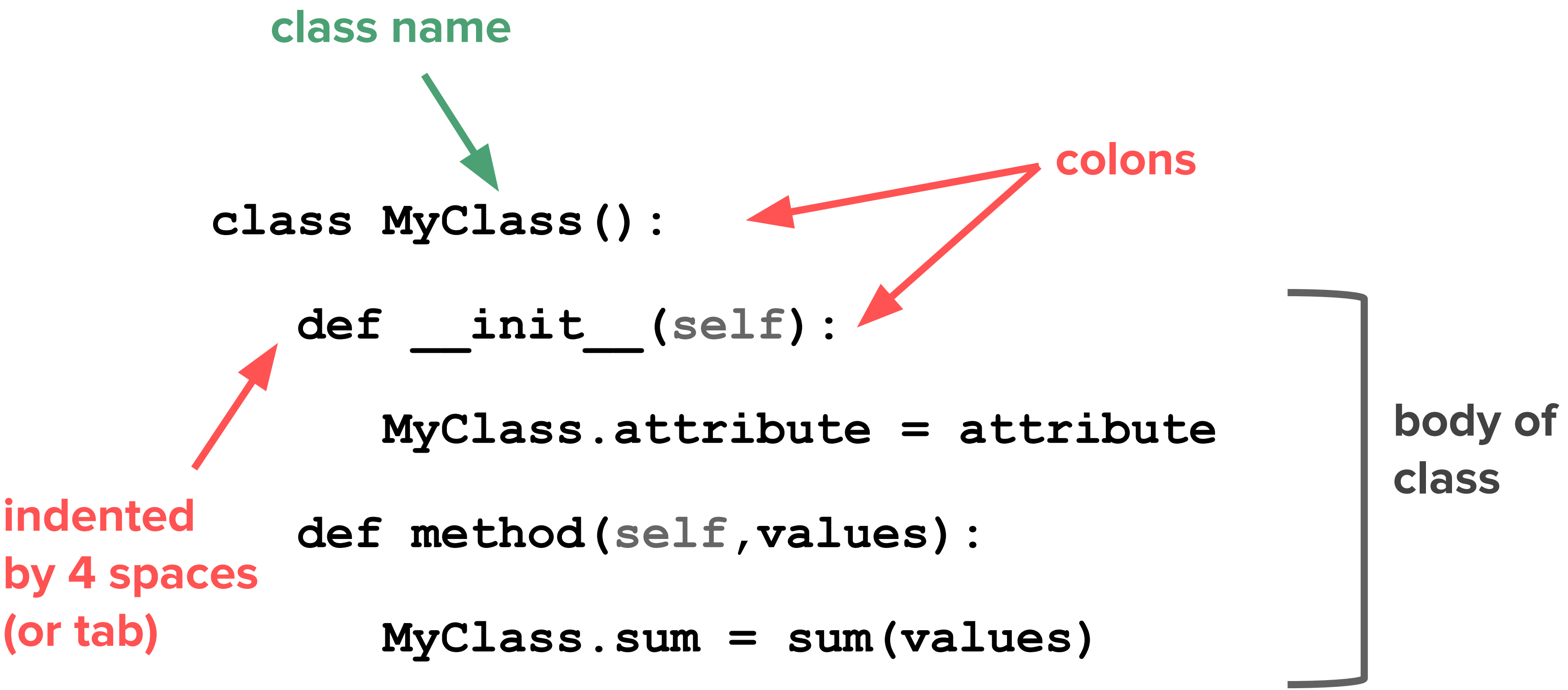
- Each class method should have an argument **`self`** as its first argument. This object is a self-reference.
- Some class method names have special meaning, for example:
  - **`__init__`**: The name of the method that is invoked when the object is first created.
  - (Full list here)

Side note: Case conventions in Python

- Style conventions (often called **style guides**) are useful ways to recognize different types of objects in Python, and can help you understand other people's codes
- Variables and functions are typically in **snake_case** (e.g., **`my_variable`**)
- Classes are in **PascalCase** (e.g. **`MyClass`**)
  - Sometimes called camel case, but more accurately, camel case is: **camelCase**

Full Python style guide here: https://www.python.org/dev/peps/pep-0008/

class syntax



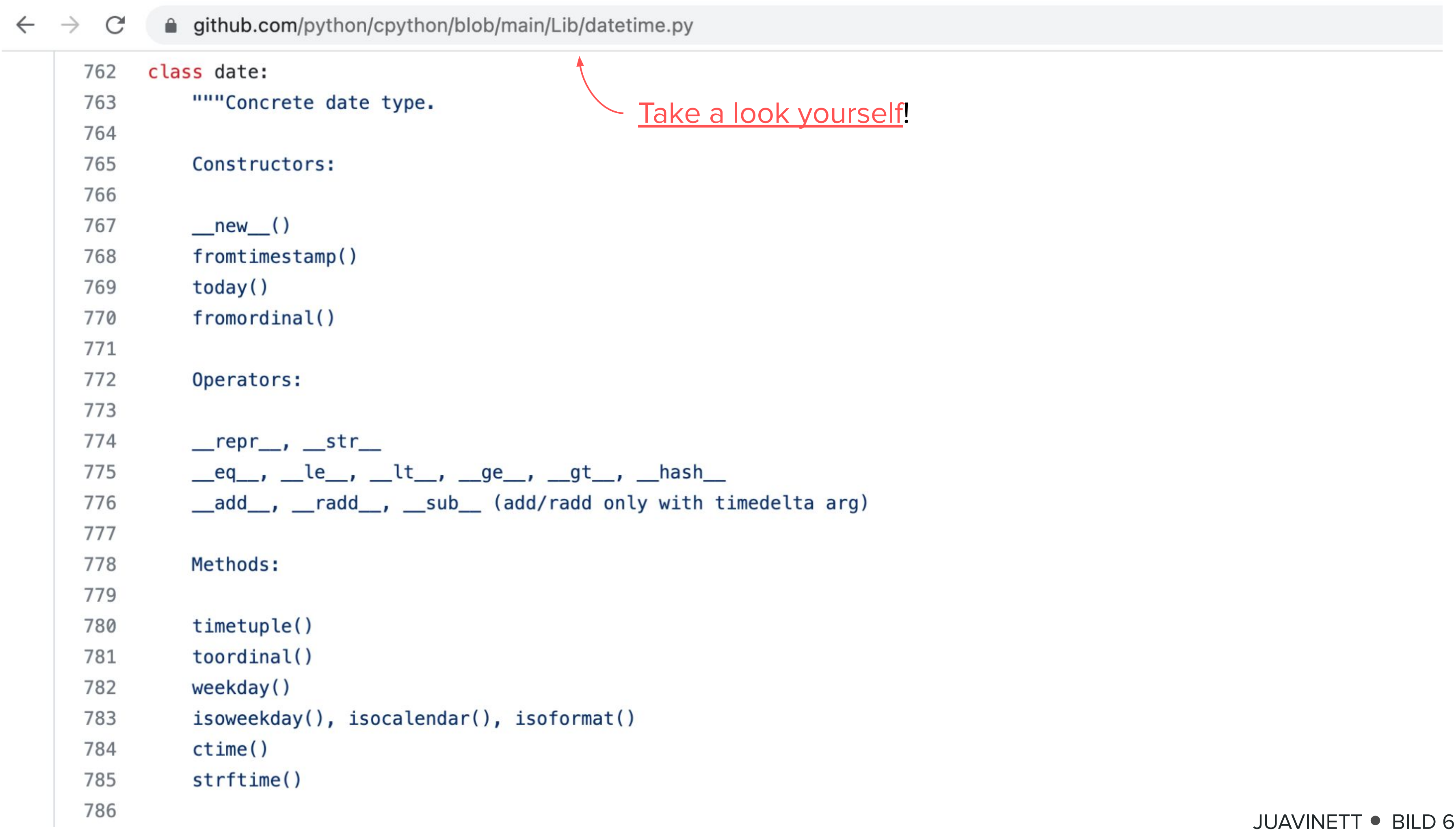
class name

colons

```
class MyClass():
   def __init__(self):
      MyClass.attribute = attribute
   def method(self,values):
      MyClass.sum = sum(values)
```

indented by 4 spaces (or tab)

body of class

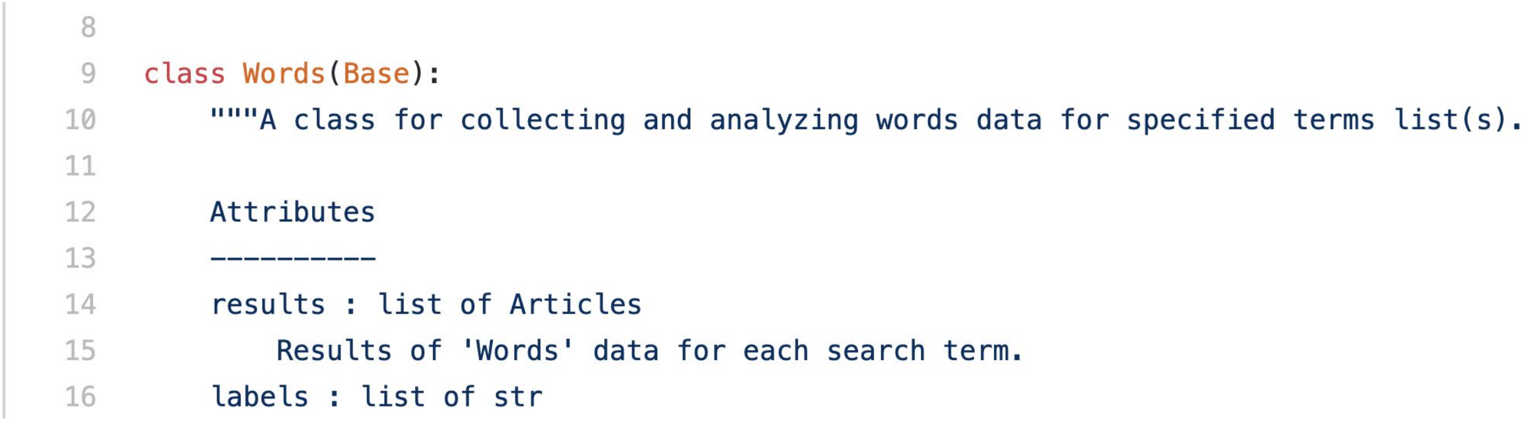


github.com/python/cpython/blob/main/Lib/datetime.py

```
class date:
    """Concrete date type.

    Constructors:

    __new__()
    fromtimestamp()
    today()
    fromordinal()

    Operators:

    __repr__, __str__
    __eq__, __le__, __lt__, __ge__, __gt__, __hash__
    __add__, __radd__, __sub__ (add/radd only with timedelta arg)

    Methods:

    timetuple()
    toordinal()
    weekday()
    isoweekday(), isocalendar(), isoformat()
    ctime()
    strftime()

```

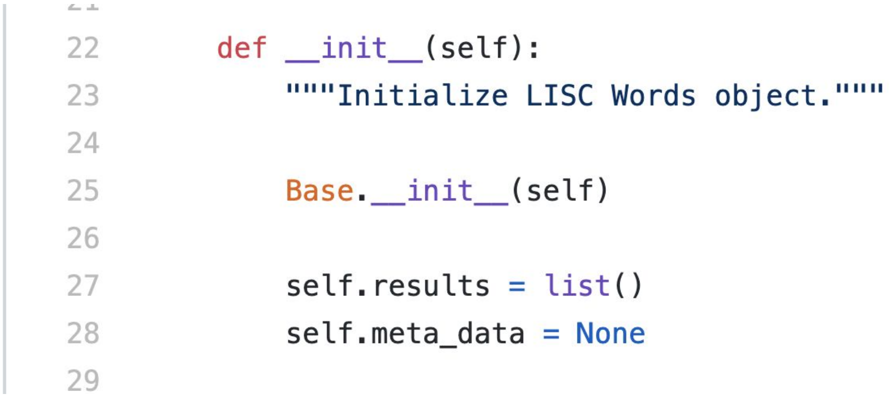
Take a look yourself!

For our purposes, we’re familiarizing ourselves with class syntax ***mostly*** so that we can recognize these in other tools and datasets.



```python
class Words(Base):
    """A class for collecting and analyzing words data for specified terms list(s).

    Attributes
    ----------
    results : list of Articles
        Results of 'Words' data for each search term.
    labels : list of str
```

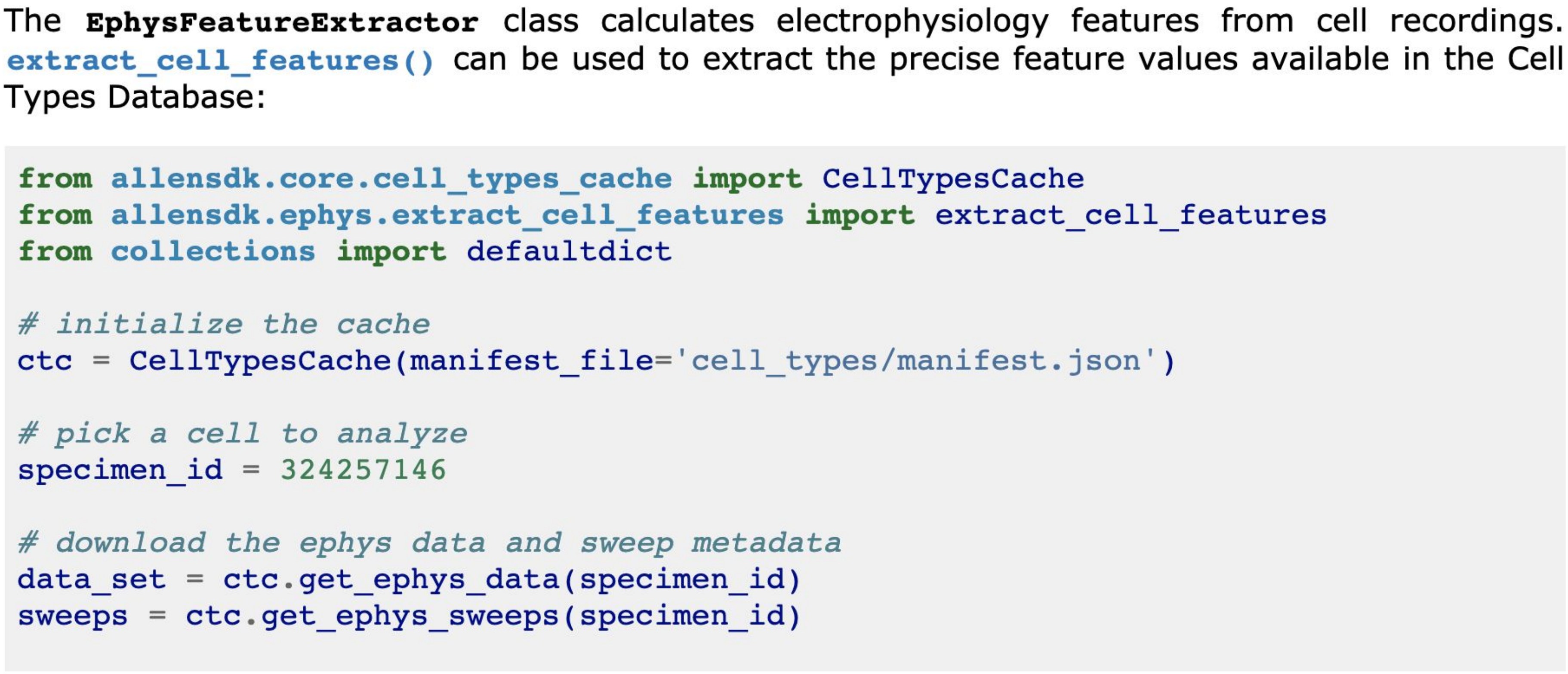
• • •



```python
    def __init__(self):
        """Initialize LISC Words object."""

        Base.__init__(self)

        self.results = list()
        self.meta_data = None

```

From https://github.com/lisc-tools/lisc/blob/c44af07492165f9a35b653b6aa1da1f397044593/lisc/objects/words.py

Feature Extraction



The **EphysFeatureExtractor** class calculates electrophysiology features from cell recordings. `extract_cell_features()` can be used to extract the precise feature values available in the Cell Types Database:

```
from allensdk.core.cell_types_cache import CellTypesCache
from allensdk.ephys.extract_cell_features import extract_cell_features
from collections import defaultdict

# initialize the cache
ctc = CellTypesCache(manifest_file='cell_types/manifest.json')

# pick a cell to analyze
specimen_id = 324257146

# download the ephys data and sweep metadata
data_set = ctc.get_ephys_data(specimen_id)
sweeps = ctc.get_ephys_sweeps(specimen_id)
```

From https://alleninstitute.github.io/AllenSDK/cell_types.html

Resources

Introduction to Python Programming (see section on Classes)

Real Python Tutorial on Object-Oriented Programming