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!)

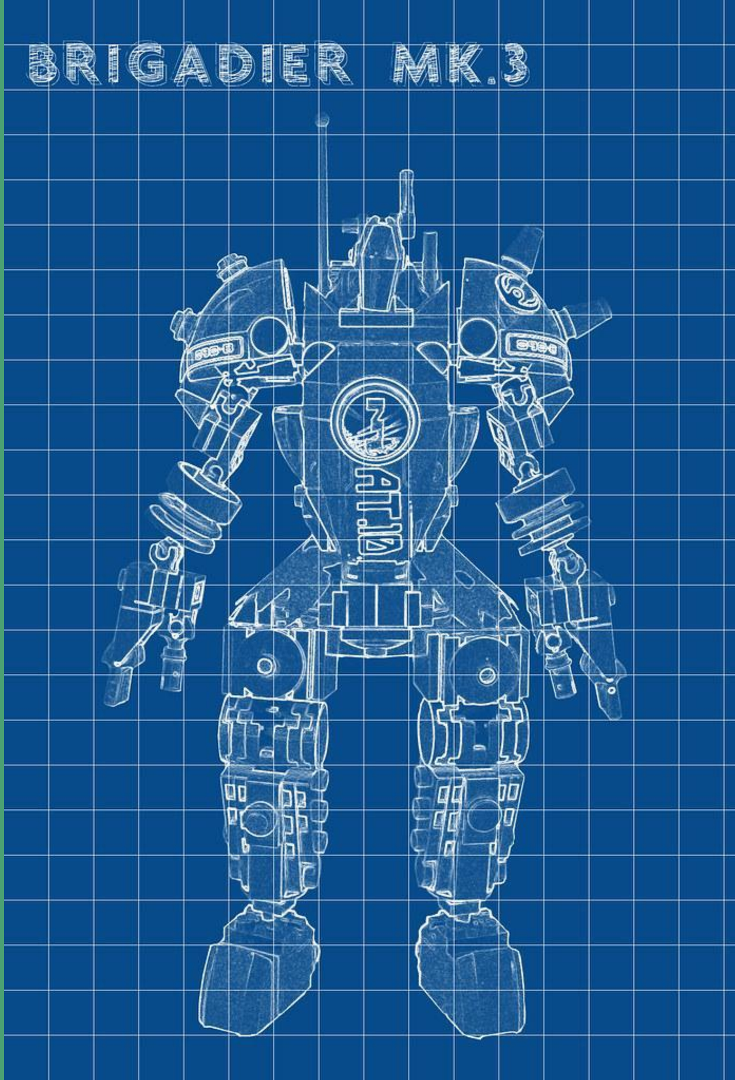
Object-oriented programming (OOP) is a programming paradigm in which code is organized around objects.

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.



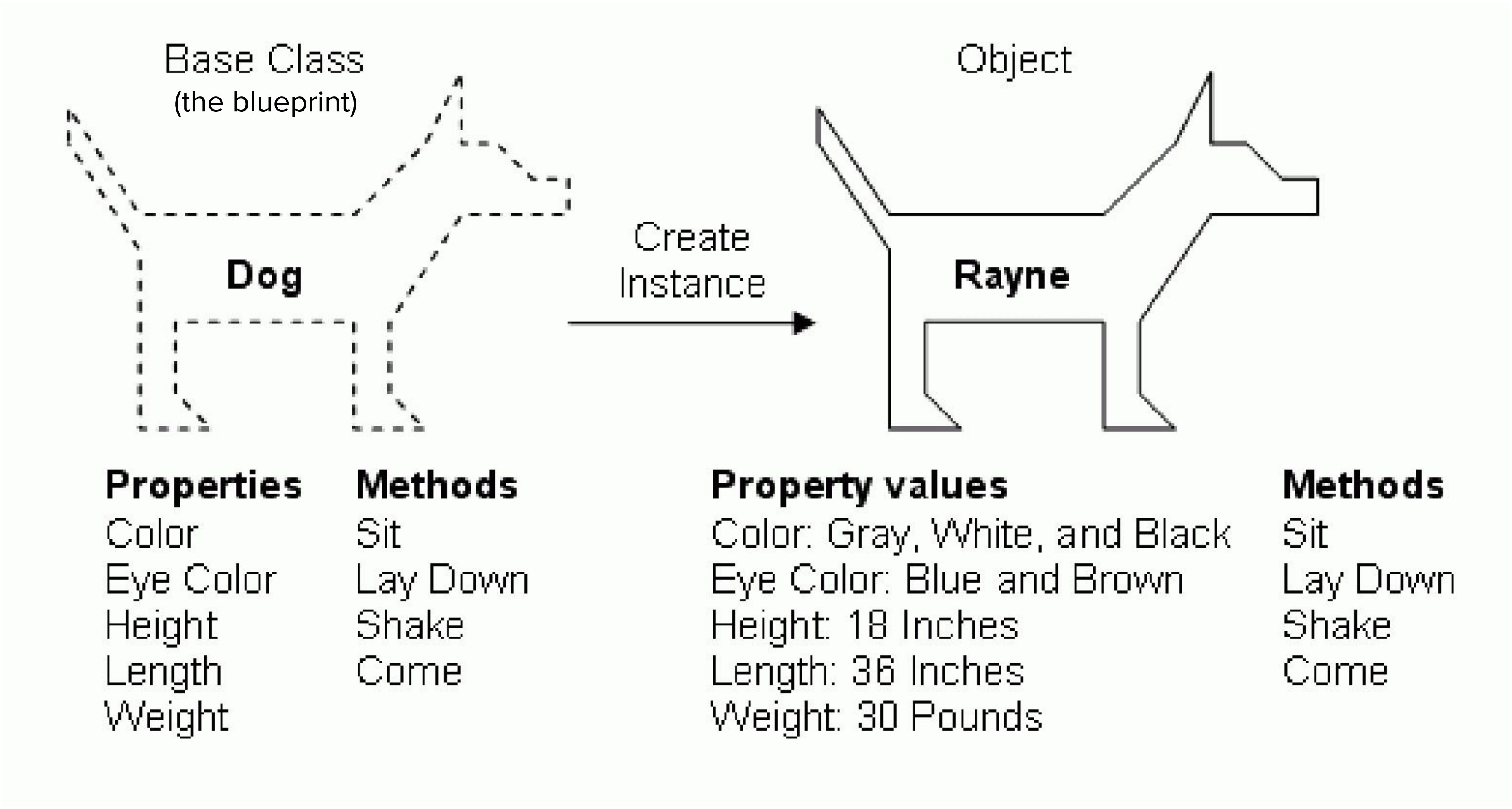


Image: http://justsajid.com/skills/objects-everything-is-an-object-in-csharp/

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`**?

```
1. neuron.spike

2. neuron.spike()

3. spike.neuron

4. spike.neuron()
```

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

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

Functions vs. methods

All methods are functions.

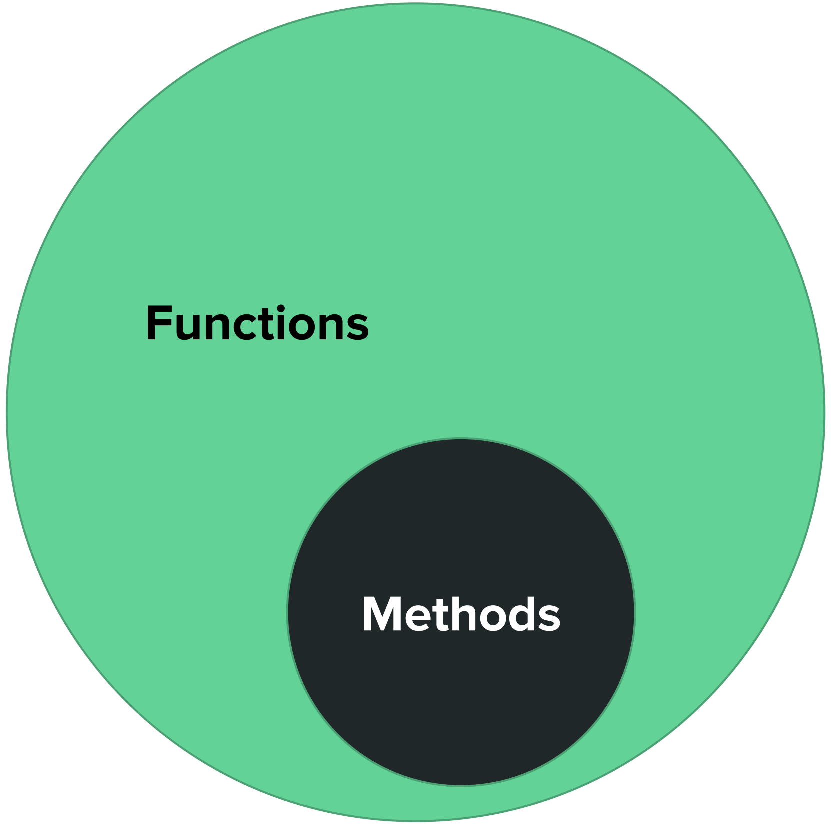
Methods are special functions attached to a variable type.

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

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

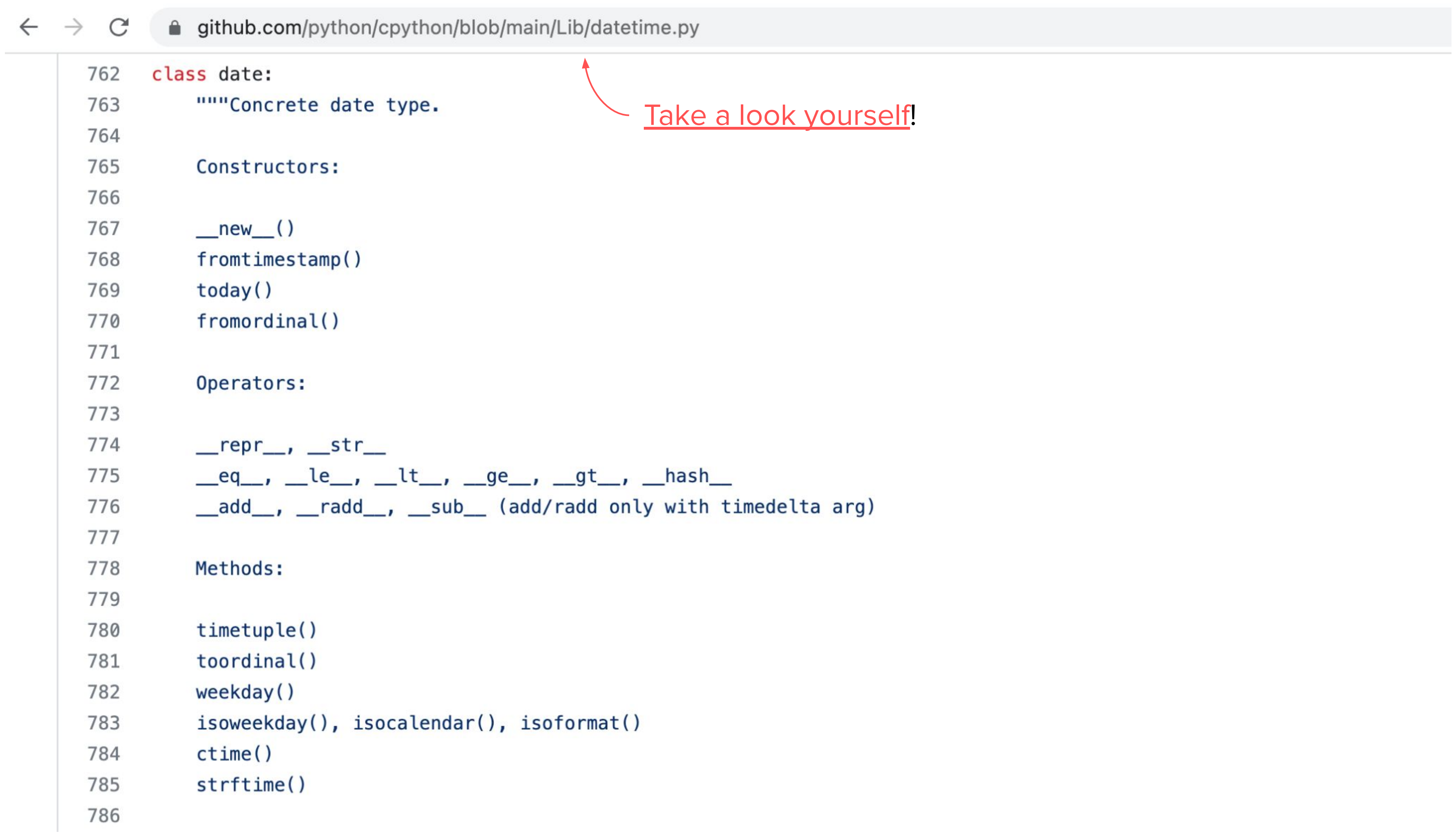
class name

colons

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

body of class

indented by 4 spaces (or tab)

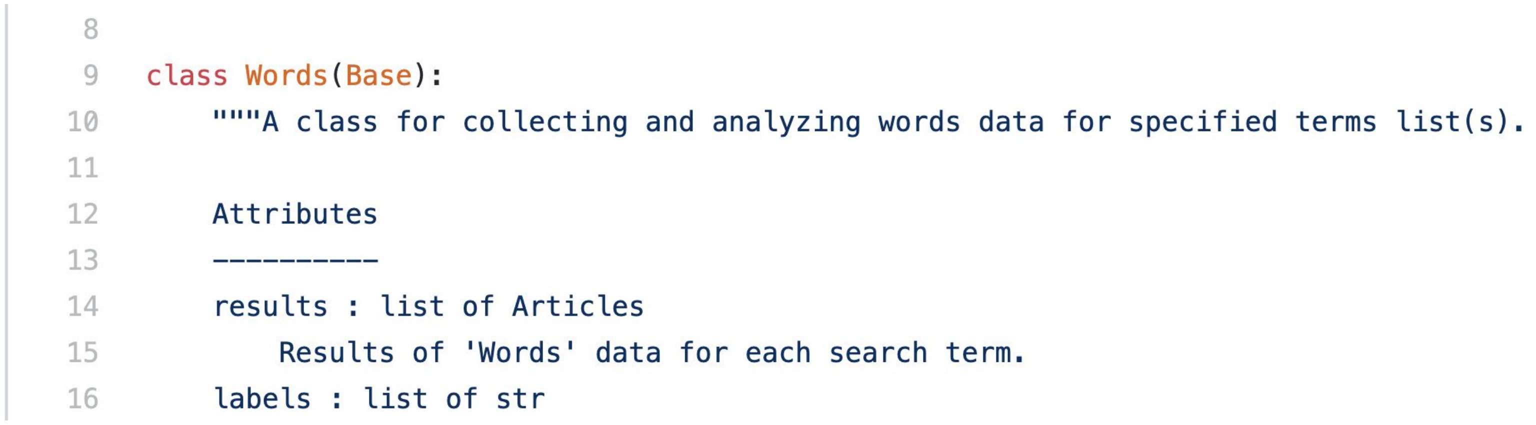


github.com/python/cpython/blob/main/Lib/datetime.py

```
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()

```

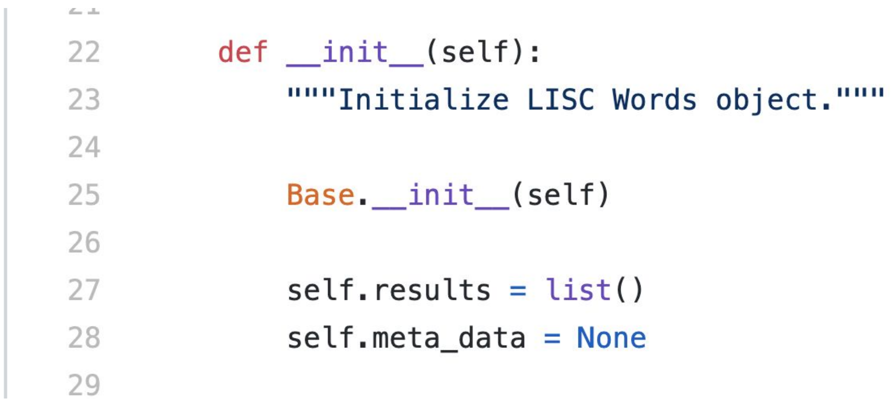
Take a look yourself!

For our purposes, we’re familiarizing ourselves with class syntax 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:

```python
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