Object Oriented Programming with examples in Python

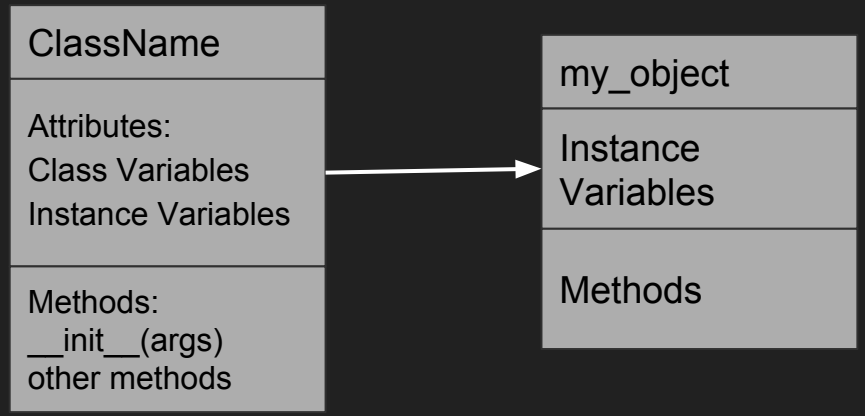
A scientific software point of view

Classes and Objects

An Object has Data and Behavior. The data is often called attributes and can be variable, arrays, etc. The behaviors are often called methods. This is sometimes called encapsulation.

A Class is a blueprint for making Objects. Making an object is called instantiation.

```
my_object = ClassName(args)
```



```
class Vehicle(object):
    def __init__(self, number_of_wheels, type_of_tank, seating_capacity,
                 maximum_velocity):

        self.number_of_wheels = number_of_wheels
        self.type_of_tank = type_of_tank
        self.seating_capacity = seating_capacity
        self.maximum_velocity = maximum_velocity

    def get_number_of_wheels(self):
        return self.number_of_wheels

    def set_number_of_wheels(self, number):
        self.number_of_wheels = number

if __name__ == '__main__':
    tesla_model_s = Vehicle(4, 'electric', 5, 250)
    print(tesla_model_s.get_number_of_wheels())
    tesla_model_s.number_of_wheels = 2          # don't do this
    print(tesla_model_s.get_number_of_wheels())
    print(tesla_model_s.number_of_wheels)       # don't do this
    tesla_model_s.set_number_of_wheels(1)
    print(tesla_model_s.get_number_of_wheels())
```

```
darwin:~ dlytle$ python3 test.py
4
2
2
1
darwin:~ dlytle$
```

Inheritance

A generalized class is created and then more specialized classes are created as extensions of the general class.

An example would be a generalized “Vehicle” class as we saw in the last slide and then we could have more specialized classes like “Automobile” or “Steam Engine” that would inherit characteristics of the Vehicle class but extend it with new attributes like “transmission_type” for Automobile and “whistle_type” for Steam Engine. The child classes inherit all of the attributes and methods of the parent type but can add their specialized attributes and methods. Child classes can also override methods from the parent class, in essence, replacing the inherited methods with their own methods.

Also, introducing the concept of private variables using “__variable” syntax. This is a way to implement “Information Hiding”

```
class Vehicle(object):

    def __init__(self, name, color):
        self.__name = name # __name is private to Vehicle class.
        self.__color = color  # __color is also private.

    def getColor(self):
        # getColor() function is accessible to class Car
        return self.__color

    def setColor(self, color):
        # setColor is accessible outside the class
        self.__color = color

    def getName(self):  # getName() is accessible outside the class
        return self.__name

class Car(Vehicle):

    def __init__(self, name, color, model):
        # call parent constructor to set name and color
        super().__init__(name, color)
        self.__model = model

    def getDescription(self):
        return (self.getName() + self.__model + " in " +
        self.getColor() + " color")
```

```
if __name__ == '__main__':
    c = Car ("Ford Mustang", "red", "GT350")
    print (c.getDescription())
    print (c.getName())
```

```
darwin:~ dlytle$ python3 test.py
Ford Mustang GT350 in red color
Ford Mustang
darwin:~ dlytle$
```

Polymorphism

Polymorphism is an important feature of class definition in Python that is utilized when you have commonly named methods across classes or subclasses. This allows functions to use objects of any of these polymorphic classes without needing to be aware of distinctions across the classes.

```
class Vehicle(object):

    def __init__(self, name, color):
        self.__name = name # __name is private to Vehicle class
        self.__color = color

    def getName(self):  # getName() is accessible outside the
class
        return self.__name

class Car(Vehicle):

    def __init__(self, name, color, model):
        super().__init__(name, color)
        self.__model = model

    def getDescription(self):
        return (self.getName() + " model is " + self.__model)
```

```
class Locomotive(Vehicle):

    def __init__(self, name, color, weight):
        # call parent constructor to set name and color
        super().__init__(name, color)
        self.__weight = weight

    def getDescription(self):
        return (self.getName() + " weighs " + str(self.__weight))

if __name__ == '__main__':
    c = Car("Ford Mustang", "red", "GT350")
    l = Locomotive("DLW WDM-2", "black", 112800)
    for v in (c,l):
        print(v.getDescription())
```

```
darwin:~ dlytle$ python3 test.py
Ford Mustang model is GT350
DLW WDM-2 weighs 112800
darwin:~ dlytle$
```

Class Variables vs Instance Variables

```
class Robot:

    Three_Laws = (
        """A robot may not injure a human being or, through inaction,
           allow a human being to come to harm.""",
        """A robot must obey the orders given to it by human beings,
           except where such orders would conflict with the First Law.,""",
        """A robot must protect its own existence as long as such protection
           does not conflict with the First or Second Law."""
    )

    def __init__(self, name, build_year):
        self.name = name
        self.build_year = build_year

    # other methods as usual
```

Static Methods

Access hidden variables from class name or from an instance.

We use a “Decorator” here, which we won’t talk about, Decorators could be a whole other Seminar.

```
class Robot:
    __counter = 0

    def __init__(self):
        type(self).__counter += 1   # ‘type(self)’ evaluates to ‘Robot’

    @staticmethod
    def RobotInstances():
        return Robot.__counter

if __name__ == “__main__”:
    print(Robot.RobotInstances())
    x = Robot()
    print(x.RobotInstances())
    y = Robot()
    print(x.RobotInstances())
    print(Robot.RobotInstances())
```

```
darwin:~ dlytle$ python3 test2.py
0
1
2
2
darwin:~ dlytle$
```

FITSBrowse

Requirements

UML

Prototype

Browser for FITS files on disk.
These will be 2-D images
X-Y cross section plots
Header information display
ROI statistics
Image stretch - min/max - histogram equalization
Use MatPlotLib for plots

Configuration file
- number of frames to show
- plots to show
- show header information

Show directory list, allow user to click on image to see it, perhaps
also allow decent into subdirectories or switch to directory somewhere else.

Simple Photometry?

Data Structures and Objects:
- the GUI is an object
- the current directory is an object
- each image file will be opened into an object
- the region of interest will be an object
- the settings/configuation will be an object
- the timeline will be an object
- the map of the heavens will be an object

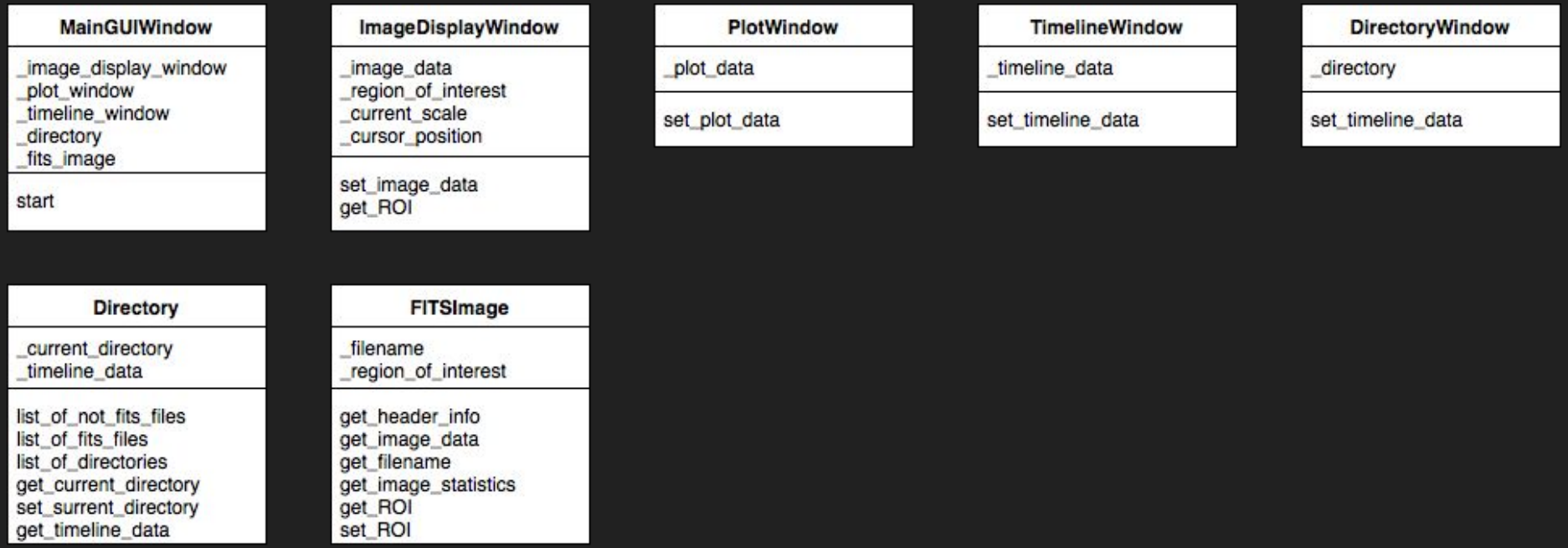
3D display showing target PyQtGraph?
- probably not, can't texture a sphere in PyQtGraph!

However, we can show a map of the heavens and plot the position of the target. There can be a timeline slider that will show the target on the map and show the images as the slider is moved back and forth. We can look through the headers of all the images in the directory and then populate the timeline slider with indicators of the image times.

So the user will see a list of images, they can click on any image and the target position will be displayed, the timeline slider will move to the proper time, and the image will be displayed in the main image window.

The user can also slide the timeline slider and the target position will move around as they slide but I'm not sure we want to flash all the images as they slide by, that might be confusing. More thought needed there.

A little UML


MainGUIWindow
_image_display_window
_plot_window
_timeline_window
_directory
_fits_image
start
ImageDisplayWindow
_image_data
_region_of_interest
_current_scale
_cursor_position
set_image_data
get_ROI
PlotWindow
_plot_data
set_plot_data
TimelineWindow
_timeline_data
set_timeline_data
DirectoryWindow
_directory
set_timeline_data
Directory
_current_directory
_timeline_data
list_of_not_fits_files
list_of_fits_files
list_of_directories
get_current_directory
set_surrent_directory
get_timeline_data
FITSImage
_filename
_region_of_interest
get_header_info
get_image_data
get_filename
get_image_statistics
get_ROI
set_ROI

Demo and Code discussion of FITSBrowse