Day 2 Notes

Object Oriented Programming

We started this yesterday. Some key ideas:

- class - A class is an outline of an object. It describes the properties and methods of the objects that follow that class
- object - An object is a data structure that follows that has its own version of properties, and can run the programs described in the class.
- properties - pieces of data that are associated with an object (nouns/adjectives)
- methods - functions and procedures that an object can run (verbs)

The Point of OOP The point of OOP is that when you are writing programs for the real world it is often helpful to be able to model the world in the code that you write. It is helpful for us, as programmers of a robot to be able to have code that reflects that robot that we have.

Creating classes You create a class by using the following declaration:

```
class My_Class:

    def __init__(self, args, more_args):
        self.property= args

    def my_cool_method(self, args, more_args):
        #my code
        pass
```

In this code, the class is called My_Class and objects made from it will be instantiations of My_Class. All the objects of this type would have a property named 'property', and they would all be able to use 'my_cool_method' with code like this:

```
my_object=My_Class() # make an object of My_Class type
my_object.my_cool_method() # This makes that object do its cool method
```

How to import Another very important point about OOP is that it allows abstraction. When other people make classes we only need to know what they do (methods) and what information they track (properties). We don't necessarily need to know how the code works. So when REV lets us use some software they developed for their motors. And it has class called RevMotor. And we can look online and see that every REVMotor has a method called `set_speed` that that takes a number from -1 to 1 to give a sense of how fast in what direction the motor runs, then getting a motor to run is as simple as:

```
from ctre.motors import REVMotor
my_motor=REVMotor()
my_motor.set_speed(1)
```

So to program a robot, it often isn't so much about programming or coming up with great algorithms. The first task is to become familiar and proficient with the libraries and classes that others have made to make the robot work.

init method Whenever a class is made it runs a special method called `__init__`. This gives an opportunity for important properties to get set. When you make a class, you will most often need to make this method and set the appropriate properties.

`self` keyword The `self` keyword is a way for python to refer to an object before it actually exists. In a class, we don't know the variable name for an object because it doesn't exist yet. We use `self` to indicate the object that we will make later. Every property needs to be made using the `self` keyword. This makes that property have scope (it will continue to be available throughout the all the methods of the class.)

Practice Let's make sure that all this makes sense.

Choose one of these tasks depending on your level:

- (Beginner) Open a new project and make a class file that represents a motor. Call the class `Motor`. The motor has a property called `speed`. When an object is made set the speed property to be 0.5. Write a method called `set_speed` that takes a number in and sets the speed property to that number. Then write a method called `speed_up` and another called `slow_down`. The first method makes the motor go twice as fast, and the other one makes it go half as fast.
- (Intermediate) Create two classes. One like the one that is described above and another called a `MotorControllerGroup`. The `MotorControllerGroup` should be able to have a number of motors associated to it. The MotorControllerGroup should have
  the ability to be made with a list of motors, and you should be able to use an `add_motor` method that takes in a motor and adds it to the list of motors the controller group manages. The MotorControllerGroup needs to have a method called `set_speed` that takes a number and sets the speed of all the motors it controls to that speed.
- (Advanced) Investigate the difference between classmethods and staticmethods. Write a short description in your own words of differences. Try to come up with a use case in programming the robot.

Git and GitHub Process

Before we start we need some basic terminology for git and GitHub. The first word is a *repository*. A repository is a folder that contains files and subfolders that have files that you wish to track the changes in. Repositories can be stored on your local machine (a **local repository (repo)) or on a computer in the cloud at a site like GitHub (a remote repository).

Once you have a repository you will tell the git program which files you want to track. This is called adding the file to the repository. Once added, git will track the changes you make in the file.

Many people can have the same repository locally on their computer, but their versions may be different. Git has ways of checking to see if there is a **conflict** between different versions of the repository once people look to **merge** their versions.

Let's work with an example.

When making a change Once a change is complete, you should **commit** your changes to repository. On PyCharm you do this by clicking on the green check mark then adding a message about what you did and hitting the button labeled commit.

Make a new branch Sometimes you want to make changes free of other's changes. You want to work on your own area, or you want to keep a version of what you have done intact without. This is most easily done by making a new branch. Branches can be made using the PyCharm Git menu. Branches allow you have several versions of the same code on your machine. When you check out a new branch Git will update the tracked files to be in alignment with the most recent version of that branch.

Make commits in that branch You make commits in a branch the way you always make commits. You click on the checkmark or goto Git/Commit then type a short message about the commit ##### Make a pull-request

Once you have made changes in a branch that you want to tell other people that are working on the project that you have completed an important task. To do this you can create a pull request.

Cloning Cloning is a way of copying the project repository on to your own device.

Git Glossary

- **repository** - The highest level of a GitHub project.
- **branch** - A branch is a version of the project that can be updated, or it may be the main branch which contains all the completed changes.

- **commit** - A step in the completion of the project. This is a moment in the development of your project where you have completed a significant step (sort of like saving).
- **pull-request** - An announcement that an updated version of the code has been completed and needs to be incorporated into the main branch.
- **clone** - making a copy of a remote repository on your own computer.

Practice

1. Make a new project and clone the repo at: https://github.com/CS570-2022/Text_Integrator_Test
2. Create a new branch using your name
3. Create a file using your name and use the markdown ending, ".md" for the name. So make a file called "Jane_Doe.md"
4. In that file write what you like about robotics, and sign your name, and add a few line spaces at the end of the file.
5. Commit that change
6. Make a pull-request to the main branch.