Motra Usage.

The essential idea of `motra`, our tracking data analysis programming framework is to be able to trivially analyze the large datasets without much hassle.

System Prerequisites.

- Python 3.10
  - Python 3.10 – `pip` package manager (should just be a checkbox, or implicity installed)
  - The following packages (install using `pip`)
    - pandas
    - numpy
    - seaborn
    - matplotlib
- Jupyter Python Notebooks

The installation of all of these programs is trivial and easy enough to google.

Python is:

~~~
For the python packages, you will probably want to open a terminal
and type:
```shell
python3.10 -m pip install [LIBRARY_NAME]
~~~

However, for jupyter, you can install with

```
python3.10 -m pip install jupyterlab
```

For numpy, pandas, seaborn, and matplotlib, the commands are just

```
python3.10 -m pip install pandas
python3.10 -m pip install numpy
python3.10 -m pip install seaborn
python3.10 -m pip install matplotlib
```

Ensure that both `python3.10` and `jupyter` are in your PATH system variable, there are multiple ways of achieving this, generally if you're on a UNIX OS (like Mac and Linux), google how to do add program to path on linux and someone there will help you out.

I want to say on mac and linux you can say,

```
which python3.10
```

and on windows you can say

```
whereis python3.10
```

To acquire a terminal on windows, search for `powershell` in the start menu.

If those commands don't error, and instead print out a directory path, then you're all set, and you don't need to do the next step.

Configuring the Path Environment Variable

To acquire a terminal on mac, find the terminal app in the app drawer.

Generally on Mac and Linux, you have either a `.bashrc` or a `.zshrc`, in your home directory. They both exist as "hidden" files. To use them, open your terminal, and type `ls`.

You should see a listing of all entries in the active directory. If you see the `.bashrc` or `.zshrc`, congratulations, that's exactly what you need.

In the terminal, type,

```
open .bashrc
```

or

```
open .zshrc
```

And then at the bottom of the file add,

```
export PATH="path/to/your/python/install/here/:$PATH"
```

substitute the path with where you installed python, as you're installing python from the website using their installer, it generally tells you where it's going to be storing it.

the process shouldn't need to be repeated for the jupyter notebook, as I believe python will manage your path for that install, but and hopefully, you don't need to do this path work, but sometimes you do.

On Windows

Type "Edit the system environment variables" into the search menu on the taskbar. Hit the button in the dialog that pops up, and then click the path variable and click edit, add the python installation directory if it isn't already there in a new cell.

Digging In

Now that your environment is setup for python coding, you can begin using motra.

To begin, with, you're going to want to open your terminal up in the motra folder, so navigate in your file manager to where you've downloaded motra and then right click on the folder and on mac it should offer you to open terminal, and on windows, you can shift + right click on the folder and the "right click menu" (context menu) will have an option called "open powershell window here".

Now that you're terminal is open, type,

```
jupyter notebook
```

Or, sometimes you can write

```
python3.10 -m notebook
```

You're browser of choice should open at this point, and you should see the files in the current directory, in the web interface, click, `New > Python 3 Notebook`.

Now you will see what we call a `jupyter notebook`.

A jupyter notebook has cells, which can be either `markdown` (you can essentially write plain old english in these cells and the computer will be happy, but for more fancy things, you can google how to use markdown, and it can explain how to make text bold, or headings, or tables, etc).

The other kind of cell, the default, is a python cell, and we'll be using these a lot.

You should see a cell highlighted to begin with, to start out copy and paste the following lines in the first cell, and hit `shift + enter`.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from motra import MotraModel
```

If all goes well, the cell should output nothing, if there is any output, like an `ImportError`, either you have not installed the packages correctly, or you are not in the correct motra directory.

Assuming everything went well, we now have an environment where we can begin programming.

The first thing we're going to briefly oversimplify is a little programming theory.

First of all, for the sake of brevity and not writing a book to explain how to code in python, because such matters are out of the scope of the project.

Disclaimer Please note that the following information is so drastically oversimplified, that it could be nearly considered wrong. This is all extremely handwavy, and if you don't understand all of this, then please just go with it, if you're really curious, you can look up like an 11 hour tutorial on youtube on how to do all things python.

Python crash course.

Variables Variables bind a "thing" to a name.

```
# how great dogs are on a scale of one to ten
how_great_dogs_are = 10
```

The line above assigns ten to the variable name `how_great_dogs_are`.

You are free to customize the variable name however you want, but there are a few rules, which are either just good practice, or necessary for program execution.

Variable Naming Rules

- must not begin with a number, so `4_how_much_i_love_you` is not a valid python variable name. you're going to need to spell out that number.
- must not contain spaces, spaces have meaning in python.
- must have only numbers and letter keys ##### Variable Naming Conventions
- all letters in variables should be lowercased.
- if you want to type seperate words like I did above, use the `_` character to seperate the words.
- Make your variable names descriptive enough that you know what's going on, but don't write a full sentence.

Comments Comments are places where you can make notes about what's happening in the code, or where you can describe how your code does what it does.

```
# This is a comment
```

Comments are any line that begins with a `#` symbol.

They have no discernible cost on computation, and python does not do anything with them, they exist only for you and your other programmers.

imports

```
import this
```

the import keyword brings in the code that someone else wrote. you basically can just take this syntax on faith for now, becuase it's simple, and there's enough footguns when you dive into it that it's not worth explaining.

the alternate that is sometimes available is

```
from math import cos
```

this is, of course, just an example, you can do this for name you want to import, and as you can see, this is exactly what we do when we pull in motra.

The *only* reason we do this is to reduce typing later. We don't need to tell the computer that cos is in the math package every single time we want to use it, we can just use it.

Python Function Call Syntax

Functions are pivotal in python, they are essentially the bread and butter of what you'll be doing.

unless you have been taught how to write your own functions, you will be using other people's functions to achieve your data needs.

the most basic example, I can imagine right now is

```
from math import cos
cos(3.14)
```

calculates the cos of an approximation of pi.

Functions are exectuted ("called"), by adding parenthesis after the function name, and then inside those parenthesis, giving what's called function arguments(arguments are just the "stuff" that is passed to functions).

Function Argument Types Python has two kinds of function arguments and any function can take any combination.

- Positional Arguments
  - Positional arguments are arguments whose meaning is defined by the order in which they are passed, so for instance, if I say, `divide(3, 5)` (which isn't a real function in python), you can assume that this is going to mean, `3/5`, but not `5/3`. This is the exact application of positional arguments, contexts where order makes what's happening clear.
- Keyword Arguments
  - These arguments are passed by name, so if `cos`'s argument was positional, which wouldn't make much sense, but is possible, you would pass, `cos(angle=3.14)`, if `angle` was cos's keyword argument. These *only* appear after positional arguments.

nearly every python library will make extensive use of both, including motra.

When a function calculates something, we say that thing is returned (because the function is "returning" a "thing" back to the context where it was called).

So for instance, `cos`, returns a number back to where it was called, so you can say,

```
from math import cos
c = cos(3.14)
```

this will assign the result of the cosine evaluated using an approximation of pi, to the variable called `c`.

most python functions return "something".

On Objects

You may be wondering at this point, why I use terms like “something” or “thing” in quotes when referring to what a function returns.

Essentially, as nice as it would be for functions to return values, all the time, that’s not really what happens.

`cos` for instance, does return a value, but most functions don’t.

There is a larger set of things, which programmers call “objects”, which are actually what is *always* returned from a python function.

Values are a subset of these “objects”, but certainly do not represent most use cases.

“objects” are kind of tricky to define, but essentially, they represent real world or abstract concepts. So an object can represent the number `42`, but another object could represent the list `[3, 4, 5]`.

Throughout your usage of motra, you will be using what’s called `MotraModel` objects. These objects represent the data and the behaviors which we can apply to the fly tracking data sets.

You don’t need to understand how `MotraModel`’s acomplish what they, do, you just need to know how to talk to them. Just like you don’t know exactly how `excel` generates things, just trust the process.

This above all else,

Everything you interact with in Python *is* an object.

Variables are not so much objects, but they serve the purpose of assigning names to objects, so the programmer and the python interpreter (the computer) can understand what you’re trying to tell it.

```
data + behaviors is an object
```

behaviors essentially is what data can do, and what it means for certain things to be done to that data.

Printing

Printing, although it means putting ink on paper colloquially, computer scientists named outputting text from the program to the programmer “printing”.

So to print, “i love you”, you write,

```python
print("i love you")
```

note the usage of quotes, text surrounded by quotes are called `string literals` in most programming languages. any `string literal` is interpreted by the computer to become a `string`. `string`s are objects which represent human readable text to the computer. Python abbreviates the word `string` down to just `str`, probably to save keystrokes.

Getting Help

Most python functions will give you some help on how to use them, if you ask them for it.

to do so, you can use the builtin, `help` function.

Oh man, I don't know how to print, but I know it's done with the print function, let's ask for help.

```
help(print)
```

My python edition outputs:

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

While perhaps slightly technical, describes what it menas to print to something. The motra help is much more generous in description.

We have another thing to cover before we dive into programming using motra in python.

The Dot Seperator.

The dot separator is an essential feature in python.

the dot separator essentially is the syntax python chose so that programmers could tell python where to look for the thing that they want to use.

Better explained with an example Let's say that I just brought math right in.

```
import math
import numpy as np
```

I can no longer just use the `cos` function, because python doesn't know where it is, and python will not guess or go searching. Especially in this case, both math and numpy define how a `cos` function, so python couldn't tell which one you want.

so if you want the cos function from math, you need to use,

```
math.cos(3.14)
```

That dot there, between `math` and `cos` tells python that you want to use the cosign function of the math library.

In english this is read as, "call the cos function defined in the math library with the value 3.14"

The other usage of dot. The only time that dot is used, is well, in decimal numbers, so like 3.14, uses a dot, but this usage of the dot is unrelated.

Oh, and the `as` in the import lines, just imports and redefines the name of the library to something the word after the `as`.

When we import numpy, we often say,

```
import numpy as np
```

simply because everytime we want to use a function in numpy, we don't want to have to say, numpy.function, we would rather say, np.function.

Now let's load our datasets in!

To start with, we're going to write,

```
help(MotraModel.
```

Now, press the `[TAB]` key, you will see a barebones listing of the things that you can do. scroll until you find, `read_from_single_csv`.

Now, with that option selected, press enter and it will type the name in for you. add a `)` and then hit shift enter and you'll see a help message something like,

```
Help on function read_from_single_csv in module motra.motra_model:

read_from_single_csv(path: 'os.PathLike', setup_config: 'Optional[SetupConfig]' = None, data
    Reads in a fly tracking file from a csv file and constructs
    a new MotraModel from the data.

    Parameters
    ----------
    path: the path to the fly tracking data.
    setup_config: the setup config to use, if left as none, will use the default.
    dataframe_key_config: the dataframe keys to use for the generated dataframe, if none, wi

    Returns
    -------
    a motra model containing the information at the given path
```

Note when you request help, you just provide the name of the function, not any arguments.

So for the first argument it is a positional argument, you can tell this because keyword arguments have an `=` as part of the argument definition. After the colon you can see the words `os.PathLike`, essentially it means it expects you to give it something that is "like a path to a file", and you can see that below in the parameters description.

The next argument, seperated by a comma, commas separate arguments, you can see a keyword argument called `setup_config`. You can pass a setup config, which is not none, if you have for instance, a situation where your framerate is not 30, but for the most part the default works just fine so we won't override this keyword argument's default.

Now if we try to use this function, we can say that we could define a motra model like the following:

```
one_day_data = MotraModel.read_csv("assets/tracks.csv")
```

because I have a csv track at that file location, your location may be different, so substitute out the string for something else.

Now if I shift+enter, nothing should start textually screaming at me, and I should be silently placed in the next cell, ready for analysis.

To graph this data, in a new cell, below the others, type,

```
one_day_data.plot()
```

Note that if you request the help for the plot method

```
help(one_day_data.plot)
```

you can see that you can actually do a little more with the plot method, like changing how big the plot will be.

Note that you might get a little snippet of ugly text along with this plot call, the most simple way to resolve this is to add a semicolon at the end of the line.

Python notebooks print the returned value of the last function if it is the last line in a given cell, adding a semicolon to that last line suppresses this behavior.