Python Pointers

CS 6370 / ME EN 6225 Motion Planning
University of Utah **Fall 2020**

I've decided to collect a few odds and ends about the python programming language here to help those wanting to get started.

- python is an object-oriented, interpreted language. You don't need to compile! It's also weakly typed, so some variable x could change from being an int to a string to some other object type without letting you know explicitly.
- The general python language reference is available here: `https://docs.python.org/3/reference/`
- Another great resource about python and basic programming is the textbook from Harvey Mudd's intro to cs course available at `https://www.cs.hmc.edu/csforall/`
- An option for a python interpreter is the IPython interpreter which adds a lot of nice functionality on top of the standard python terminal. You can find out more information at the IPython website `http://ipython.org/`. IPython is available on the Linux CADE machines by running the command *ipython*.
- Independent of the interpreter you are using python has the super awesome `help()` function which allows you to see the documentation associated with a package, class, function, or object instance. In my code I try and put appropriate doc-strings in all functions and classes so that you can easily get these descriptions.
- numpy - Numpy is the numerical python library. This is your go to library for anything linear algebra related.
  - If you want a comparison to Matlab commands see this link `http://mathesaurus.sourceforge.net/matlab-numpy.html`
  - Often, including in my code, numpy as imported with the package alias "np"

    ```
    import numpy as np
    ```

  - To initialize an empty matrix A of r rows and c columns the code would look like

    ```
    A = np.zeros((r,c))
    ```

  - To make the matrix have boolean types instead of floating point use the following:

    ```
    A = np.zeros((r,c), dtype=np.bool)
    ```

  - Somewhat confusingly, the natural datatype in numpy is the array. There is also a matrix type, which can be more convenient for doing linear algebra, but you must explicitly create it:

    ```
    A = np.matrix(np.zeros((r,c)))
    ```

- matplotlib allows you to make plots using similar commands to matlab. See `https://matplotlib.org/` for more details.

- Remember, if you make a class, you need to create an instance of it to use it. Here we make an instance, `f`, of the class `Foo`:

```
class Foo:
    def bar(self, x, y):
        return x + y

f = Foo()
f.bar(3,4) # 7
f.bar('string', 'cheese') # 'stringcheese'
```

- In the above example, see how we used the fact that python is weakly typed to concatenate two strings, using the same function we used to add two integers!

- Note also the `self` variable, this is a python keyword that gives you access to class members and functions, so a more complicated example:

```
class Foo:
    def bar(self, x, y):
        return x + y

    def bar_bar(self, x):
        return self.bar(x, x)

f = Foo()
f.bar_bar(3) # 6
```

- We can also use self for member fields instead of functions, this is particularly useful when we build constructors using the keyword `__init__`:

```
class Foo:
    def __init__(self, z):
        self.q = z

    def bar(self, x, y):
        return x + y

    def bar_bar(self, x):
        return self.bar(x, x)

    def foo_foo(self, x):
        return self.bar(x, self.z)

f = Foo(4)
f.foo_foo(3) # 7
```

- Functions are objects! You can pass functions as arguments to other functions or make them members of classes. This can be very useful, for example if you want to have an abstract planning algorithm that takes as input a transition function you could do something like this:

```
def space_ship_thrusters(s, a):
    s_prime = s + a*thruster_velocity
    return s_prime

def space_ship_warp_drive(s, a):
    # Do crazy warp drive math...
    return s_prime

plan_a = planning_alg(map, init_state, f=space_ship_thrusters)
plan_b = planning_alg(map, init_state, f=space_ship_warp_drive)
```

  I used this in my Project 1 solutions for not only setting the transition function but also sending functions for `is_goal` and heuristic functions.

- Note in passing functions as arguments do not put the parenthesis at the end: Send f not $f()$

- In the previous example I used a named parameter call by specifying $f = ...$ This allows you to call functions with rearranged parameter order. More importantly it allows you to specify default parameter values:

```
def f(a, b, c=None, d=abs):
    if c is None:
        return d(a+b)
    else:
        return d(c(a,b))

def multiply(a,b=1):
    return a*b

f(1,3) # returns 4
f(1,-3) # returns 2
f(1,3,multiply) # returns 3
f(1,3,c=multiply) # returns 3
f(1,3,d=multiply) # returns 4
f(1,-3,d=multiply) # returns -2
f(1,-3,c=multiply,d=multiply) # returns -3
```

  You can see that named parameters do not have to functions, but can also be values, as shown in the function `multiply()`.