Goals

- How to create a simple class
- The difference between a class and an instance of a class
- Define class atributes and methods

Why Classes?

- Used in a lot of modern languages (e.g. C++, Java,...)
- Logically group data and functions
- Build upon them
- Do not need to redefine each time

Associated with classes are attributes and methods (data/functions)

Example: A particle class

We'll make a class Particle and for each particle define specific attributes and methods

- name
- mass
- mass unit
- charge
- actions

In [42]:
```python
#for an empty class use pass
class Particle:
    pass
```

Class:

Is a template for creating instances

Instance of a class:

Each particle created from the Particle class is an instance (object) of that class.

We can define unique instances of the Particle class. Note they have their own location in memory

In [23]:
```
par_1 = Particle()
par_2 = Particle()

print(par_1)
print(par_2)
```

```
<__main__.Particle object at 0x106764f50>
<__main__.Particle object at 0x10675d950>
```

Instance variables:

Contain data that is unique to an instance

In [44]:
```
#par_1 abd par_2 are instances of Particle class
par_1.name = 'Electron'
par_1.mass = 0.511
par_1.mass_unit = 'MeV'
par_1.charge = -1

par_2.name = 'Proton'
par_2.mass = 0.938
par_2.mass_unit = 'GeV'
par_2.charge = 1

print(par_1.name)
print(par_2.name)
```

```
Electron
Proton
```

Init Method

Cool, but that is a lot of code we need for each particle. We can have the class do this automatically using the ***init*** method.

init method is an **initialization** method for the class (e.g. a constructor).

Methods created in a class recieve the instance as the first argument. Additional arguments can follow. By convention, the instance argument is called ***self***

In [47]:
```
class Particle:
    def __init__(self, name, mass, mass_unit, charge):
        #variables with self. are instance variables
        self.name = name # same as par_1.name = name
        self.mass = mass
        self.mass_unit = mass_unit
        self.mass_list = '{} {}'.format(mass, mass_unit)
```

In [50]:
```
#init method will run automatically and par_1 is passed in as self and s
ets atributes
par_1 = Particle('Electron', 0.511, 'MeV', -1) # instance is passed auto
matically, don't need to specify
par_2 = Particle('Proton', 0.938, 'GeV', 1)

print(par_1.name)
print(par_1.mass_list)
```

```
Electron
0.511 GeV
```

Adding Actions

Can define methods (functions) within the class

In [51]:
```
class Particle:
    def __init__(self, name, mass, mass_unit, charge):
        self.name = name
        self.mass = mass
        self.mass_unit = mass_unit
        self.mass_list = '{} {}'.format(mass, mass_unit)

    def mass_square(self):
        return self.mass**2
```

In [53]:
```
par_1 = Particle('Electron', 0.511, 'MeV', -1)
par_2 = Particle('Proton', 0.938, 'GeV', 1)

print(par_1.name)
print(par_1.mass_list)
print(par_1.mass_square())

#method needs (), attributes don't
#what if we don't us ()
print(par_1.mass_square) #need ()
```

```
Electron
0.511 GeV
0.261121
<bound method Particle.mass_square of <__main__.Particle object at 0x10
6757e50>>
```

In [63]:
```
#what if you forget 'self'?
class Particle:
    def __init__(self, name, mass, mass_unit, charge):
        self.name = name
        self.mass = mass
        self.mass_unit = mass_unit
        self.mass_list = '{} {}'.format(mass, mass_unit)

    #def mass_square():
    def mass_square(self):
        return '{} {}^2'.format(self.mass**2, self.mass_unit)

par_2 = Particle('Proton', 0.938, 'GeV', 1)
print(par_2.mass_square())
# 1 given argument is the instance of the class emp_2 (self) in this cas
e
```

```
0.8798439999999998 GeV^2
```

Two ways to call a method

1) Call the method on the class instance

In [66]:
```
par_2.mass_square()
```

Out[66]:
```
'0.8798439999999998 GeV^2'
```

2) Call the method on teh class. For this we need to provide the instance (e.g. self)

In [67]:
```
Particle.mass_square(par_2)
```

Out[67]:
```
'0.8798439999999998 GeV^2'
```