Goals

- Class Inheritance
- Sub Classes

Class Inheritance

Allows use to inherit attributes and methods from a partent classes.

- Create a sub class that inherits all the functonality of our parent class
- Overwrite or modify the sub class without affecting the parent class

Example: Particle Types

We want to create different types of particles: leptons, hadrons, or bosons.

Sub classes would be good, since the particle types will include all information from our parent Particle class.

In [93]:
```
import numpy as np
```

Working with the class and not the instance

In [94]:
```
class Particle:
    c = 3.0e8 #speed of light

    def __init__(self, name, mass, mass_unit, charge, vel):
        self.name = name
        self.mass = mass
        self.mass_unit = mass_unit
        self.vel = vel #added velocity attribute
        self.mass_list = '{} {}'.format(mass, mass_unit)

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

    def get_beta(self):
        return self.vel/self.c # Also use Particle.c

    @classmethod  # use decarator to distinquish following method as a c
lass method
    def set_c_value(cls, val): #cls is used as convention. class can not
be used (it is python key work)
        cls.c = val #note we are now working with the class and not the
 instance
```

In [95]:
```
class Hadron:
    pass
class Boson:
    pass
class Lepton(Particle): #classes to inherit go in the '()'
#class Lepton(Hadron, Boson, Particle): #classes to inherit go in the
 '()'
    pass
#Lepton class is now a clone of the Particle class, so has all of its fu
nctionality
```

In [96]:
```
par_1 = Particle('Electron', 0.511, 'MeV', -1,1.2e7)
par_2 = Particle('Proton', 0.938, 'GeV', 1,1.2e6)

lep_1 = Lepton('Electron', 0.511, 'MeV', -1,1.2e7)


print(par_1.name)
print(lep_1.name)
```

```
Electron
Electron
```

What happened?

1) When instantiating our Lepton, it first looked at our Lepton class 2) There is no init method in Lepton class, so it then works through the chain of inheritances unitl it finds an init method. This is Particle class in our case.

The inheritance chain the python walks through is known as the **Method Resolution Order**

In [97]:
```
print(help(Lepton)) # use help function with class as argument to get lo
ts of info includeing Meth. res. order
#Method resolution order follows the argument order in which classes are
listed

#All python objects inherit from builtins.object

#can also see what methods and attributes were inherited
```

```
Help on class Lepton in module __main__:

class Lepton(Particle)
 |  Lepton(name, mass, mass_unit, charge, vel)
 |
 |  Method resolution order:
 |      Lepton
 |      Particle
 |      builtins.object
 |
 |  Methods inherited from Particle:
 |
 |  __init__(self, name, mass, mass_unit, charge, vel)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  get_beta(self)
 |
 |  mass_square(self)
 |
 |  ----------------------------------------------------------------------
---
 |  Class methods inherited from Particle:
 |
 |  set_c_value(val) from builtins.type
 |
 |  ----------------------------------------------------------------------
---
 |  Data descriptors inherited from Particle:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
---
 |  Data and other attributes inherited from Particle:
 |
 |  c = 300000000.0

None
```

We can modify our Lepton class variable without affecting the parent one

In [98]:
```
print(par_1.c)
print(lep_1.c)
```

```
300000000.0
300000000.0
```

In [99]:
```
lep_1.set_c_value(2.99e8)

print(par_1.c)
print(lep_1.c)
```

```
300000000.0
299000000.0
```

Initiating Sub Class with More Information than the Parent Class:

We can access the parent class (Particle) init method in two way:

1) Using the class

> Particle.**init**()

2) Using the super method

> super( ).**init**()

Using super has some benifites

- We don't need to worry about the class name
- Better when using multiple inheritances

In [100]:
```
class Lepton(Particle): #classes to inherit go in the '()'
#class Lepton(Hadron, Boson, Particle): #classes to inherit go in the
 '()'

    def __init__(self,name, mass, mass_unit, charge, vel, anti):
        super().__init__(name, mass, mass_unit, charge, vel) #lets Parti
cle class init method handle this stuff
        #Particle.__init__(self,name, mass, mass_unit, charge, vel) #sam
e result as above

        self.anti = anti

#Lepton class is now a clone of the Particle class, so has all of its fu
nctionality
```

In [101]:
```
par_1 = Lepton('Electron', 0.511, 'MeV', -1, 1.2e7,'Positron')

print(par_1.name)
print(par_1.anti)
```

```
Electron
Positron
```

Let's do another Example

In [102]:
```
class Hadron(Particle): #class inherits from Particle class
     def __init__(self,name, mass, mass_unit, charge, vel, particles = N
one): #mod init to use list of particles
        super().__init__(name, mass, mass_unit, charge, vel) #lets Parti
cle class init method handle this stuff
        if particles is None:
            self.particles = [] #if true create an empty list
        else:
            self.particles = particles # if False then add the particles
to the list

    #method to add particles to the list
     def add_part(self,part):
        if part not in self.particles:
            self.particles.append(part)

    #method to remove particles to the list
     def remove_part(self,part):
        if part in self.particles:
            self.particles.remove(part)

    #method to print particle list
     def print_part(self):
        for part in self.particles:
            print('-->', part.name)
```

In [107]:
```
par_1 = Lepton('Electron', 0.511, 'MeV', -1, 1.2e7,'Positron')
par_2 = Particle('Proton', 0.938, 'GeV', 1,1.2e6)

par_3 = Hadron('Neutron', 0.939, 'GeV', 0, 3.5e7, [par_3])
par_3.add_part(par_2)
par_3.print_part()
```

```
--> Neutron
--> Proton
```

In [108]:
```
par_3.add_part(par_1)
par_3.print_part()
```

```
--> Neutron
--> Proton
--> Electron
```

In [109]:
```
par_3.remove_part(par_1)
par_3.print_part()
```

```
--> Neutron
--> Proton
```

Two Built in Methods

1) isinstance: is instance and instance of the class

2) issubclass : is class a subclass of another

In [113]:
```
print(isinstance(par_3,Hadron))
print(isinstance(par_3,Particle))
print(isinstance(par_3,Lepton))
```

```
True
True
False
```

In [117]:
```
print(issubclass(Lepton,Hadron))
print(issubclass(Hadron,Particle))
print(issubclass(Particle,Lepton))
print(issubclass(Lepton,Particle))
```

```
False
True
False
True
```

In [ ]:
```

```