**Association**

Class B has a week association relationship with Class A, as it uses specific attributes from A in the addAllNums method. However, that is the extent of the relationship.

class A(object):

def \_\_init\_\_(self, a, b, c):

self.a = a

self.b = b

self.c = c

def addNums():

self.b + self.c

class B(object):

def \_\_init\_\_(self, d, e):

self.d = d

self.e = e

def addAllNums(self, Ab, Ac):

x = self.d + self.e + Ab + Ac

return x

ting = A("yo", 2, 6)

ling = B(5, 9)

print ling.addAllNums(ting.b, ting.c)

**Aggregation**

Class B forms an aggregation relationship with Class A, as it references an independent A object when initialized, as one of its attributes. Whilst a B object is dependent on A, in the event of B's destruction, A will continue to exist as it is independent of B.

class A(object):

def \_\_init\_\_(self, a, b, c):

self.a = a

self.b = b

self.c = c

def addNums():

self.b + self.c

class B(object):

def \_\_init\_\_(self, d, e, A):

self.d = d

self.e = e

self.A = A

def addAllNums(self):

x = self.d + self.e + self.A.b + self.A.c

return x

ting = A("yo", 2, 6)

ling = B(5, 9, ting)

print ling.addAllNums()

**Composition**

Much like aggregation, however rather than referencing an independent object, B actually initializes an instance of A in it's own constructor as an attribute. If the B object is destroyed then so too is the A object. This is why composition is such a strong relationship.

class A(object):

def \_\_init\_\_(self, a, b, c):

self.a = a

self.b = b

self.c = c

def addNums():

self.b + self.c

class B(object):

def \_\_init\_\_(self, d, e):

self.d = d

self.e = e

self.A = A("yo", 2, 6)

def addAllNums(self):

x = self.d + self.e + self.A.b + self.A.c

return x

ling = B(5, 9)

print ling.addAllNums()