In [1]:

```
class MyClass():
  name = 'class_name' # class variable
  x = 1.2
  def __init__(self, y):
    self.y = y # instance variable

  def method1(self, a): # instance method
    print('method1: ', a, self.y)

  @staticmethod # decorator
  def max_value(a, b): # static method: can't accessible class attributes
    print ('max: ', max(a,b))

  @classmethod
  def methodC(cls, a): # class method: independent of instance
    print('methodC:', a)
```

In [2]:

```
c = MyClass(-10)
c.y
c.method1('hello')
c.max_value(2, 3)
c.methodC('hey')
```

```
method1:  hello -10
max:  3
methodC: hey
```

In [3]:

```
class MyClass():
  name = 'class_name' # class variable
  __x = 1.2 # private class variable
  def __init__(self, y, w):
    self.y = y # instance variable, public
    self.__w = w # private

  def method1(self, a): # instance method
    print('method1: ', a, self.y)

  def __method1(self, a): # instance method, private
    print('__method1: ', a, self.y)

  @property
  def w(self):
    return self.__w
```

In [4]:

```
cc = MyClass(3, 100)
cc.w
```

Out[4]:

```
100
```

In [5]:

```
# inheritance
class MySubClass(MyClass):
  def __init__(self, y, w, z):
    self.z = z
    super().__init__(y, w)
    # MyClass.__init__(y, w)

  # polymorphism
  def method1(self, a, b):
    print('subclass:method1: ', a, b)
```

In [6]:

```
sc = MySubClass(4, 6, 8)
sc.method1('me', 'two')
```

```
subclass:method1:  me two
```

In [7]:

```
class MySubClass2(MyClass):
  def __init__(self, y, w, z):
    self.z = z
    # not recommended
    self.y = y
    self.__w = w
```

In [8]:

```
sc2 = MySubClass2(2, 4, 9)
sc2.y
```

Out[8]:

```
2
```

In [9]:

```
# multiple inheritance
# MyClass and YourClass have a common class variable __x and name
class YourClass():
  name = 'Your Class'
  __x = 3.4
  def __init__(self, zz) -> None:
    self.zz = zz

class OurClass(MyClass, YourClass):
  def __init__(self, y, w):
    super().__init__(y, w)

  def our_method(self, xx) -> float:
    return 2.3
```

In [10]:

```
ccc = OurClass(3, 4)
ccc.name
ccc.our_method(3)
```

Out[10]:

```
2.3
```

In [11]:

```
# trade class
class Market:
  def __init__(self, date, market_price) -> None:
    self.date = date
    self.market_price = market_price
    # expanded to include market data for option pricing
    # vol, zero_rate, div_rate
    pass

class Trade:
  def __init__(self, name, side, quantity, settle_amount) -> None:
    self.name = name
    self.side = side # 1: buy, -1: sell
    self.quantity = quantity
    self.settle_amount = settle_amount
    pass

  def value(self, mkt):
    if not isinstance(mkt, Market):
      raise TypeError('Input must be a market object')
    return self.quantity * self.side * mkt.market_price

  def totalPnL(self, mkt):
    if not isinstance(mkt, Market):
      raise TypeError('Input must be a market object')
    return self.value(mkt) - self.settle_amount
```

In [12]:

```
class EuropeanOption(Trade):
  def __init__(self, name, side, quantity, settle_amount, strike, option_expiry) -> None:
    self.strike = strike
    self.option_expiry = option_expiry
    super().__init__(name, side, quantity, settle_amount)

  def value(self, mkt):
    if not isinstance(mkt, Market):
      raise TypeError('Input must be a market object')
    value = 0.0
    # do the calculations
    return value
```