Class terminology

**member variables** - the data that belongs to a class

Ex. PairCalculator has two member variables: value1 and value2

**accessors** - methods which expose member variables

Ex. the method getVal1 and getVal2 in PairCalculator

**setters** - methods which change the values of member variables

Exercise: try implementing setVal1 and setVal2 in PairCalculator

Best practice is to give access to member variables only through methods! This enforces **encapsulation**
Classes are types!

Ex. pair = PairCalculator(10,2) # pair has type PairCalculator

Ex. win = GraphWin("Hi", 200, 200) # win has type GraphWin

Ex. float, int, str, bool are called **built-in types** because they are included as part of the Python language. Classes allow us to define our own types!
Classes: constructor vs methods

The constructor (ctor) is invoked when you create an instance of a class

Defining the constructor

```python
# Write the constructor
def __init__(self, a, b):
    """
    Constructor, Initializes member variables for values 1 and 2.
    Param a (int): value 1
    Param b (int): value 2
    Implicit returns (PairCalculator): an instance of this class
    """
    self.value1 = a
    self.value2 = b
    print("Creating", self.value1, self.value2)
```

Using the constructor (uses class name!)

```python
myCalculator = PairCalculator(10, -5)
```
Classes: constructor vs methods

Methods determine what you can do with a class (aka the **class interface**)

Defining the method

```python
# TODO: Write an accessor for value 1
def getVal1(self):
    """
    Returns value 1 (int)
    """
    return self.value1
```

Using the method

(All methods use “dot”)

```python
print("value1", myCalculator.getVal1())
```
Classes: method scope

Methods have member variables, parameters, and local variables in scope

def swap(self):
    """
    Swaps both member variables
    """
    tmp = self.value1
    self.value1 = self.value2
    self.value2 = tmp

    # What variables are in scope here?

def getVal1(self):
    """
    Returns value 1 (int)
    """
    self
    self.value1
    self.value2 (still in scope even though we don’t use it!!!)

    # What variables are in scope here?

    return self.value1
Classes: method scope

Methods have member variables, parameters, and local variables in scope

```python
def __init__(self, a, b):
    """
    Constructor, Initializes member variables for values 1 and 2.
    Param a (int): value 1
    Param b (int): value 2
    Implicit returns (PairCalculator): an instance of this class
    """
    self.value1 = a
    # 1. what variables are in scope here?
    self.value2 = b
    print("Creating", self.value1, self.value2)
    # 2. what variables are in scope here?

1. self, a, b, self.value1 (self.value2 has not been created yet!)
2. self, a, b, self.value1, self.value2
Classes: function stack

Method calls get their own function frame, same as non-class functions
Classes: function stack

Method calls get their own function frame, same as non-class functions
Classes: function stack

Method calls get their own function frame, same as non-class functions
Classes are mutable!

Recall: lists are mutable

So are classes! e.g. changes to a class inside a function persist after the function returns

Ex: testCalculator() in PairCalculator.py

```python
def testCalculator(calc):
    """
    Tests the member functions of calc
    """
    print("TEST ADD: ", calc.getVal1(), "+", calc.getVal2(), ":=", calc.add())
    print("TEST SUB: ", calc.getVal1(), "-", calc.getVal2(), ":=", calc.sub())
    calc.neg()
    calc.swap()
```
Classes are mutable!

```python
def testCalculator(calc):
    """
    Tests the member functions of calc
    """
    print("TEST ADD: ", calc.getVal1(), "+", calc
    print("TEST SUB: ", calc.getVal1(), "-", calc
    calc.neg()
    calc.swap()
```

* We will omit the methods
  from our drawings with the
  understanding that they are
  still there!
Classes are mutable!

Recall: lists are mutable

So are classes! e.g. changes to a class inside a function persist after the function returns

Ex: testCalculator() in PairCalculator.py