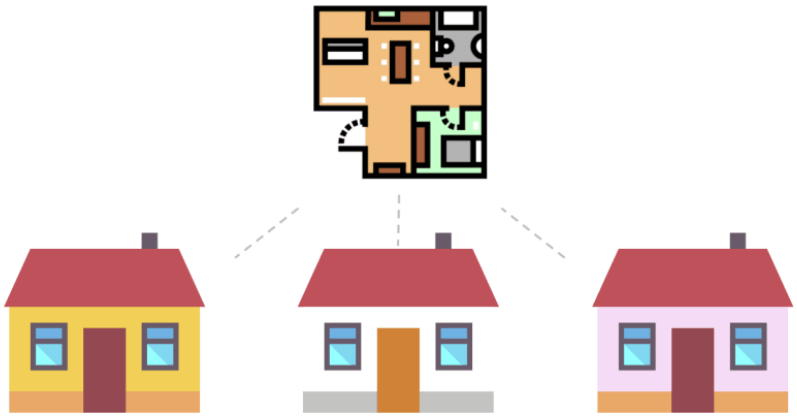
Classes

A **class** is a template for creating objects that contains both **attributes** and **methods**. A common analogy is a blueprint.



It is possible to create multiple **instances** of a class using its definition. For example, a class might be used to describe an employee, with information about his/her name, salary, and so on. This class can be used to store information about multiple employees, even though their information is different. Classes are defined using the `class` keyword.

Classes typically have an **initializer** method, which sets the initial values of attributes. In python, this is `__init__(self, ARGUMENTS)`. Here, `self` represents the specific instance of the class, and allows access to its attributes and methods. To set the value of an attribute, use the format `self.ATTRIBUTE = VALUE`. Note that `self` is never passed as an argument from main program when a new object is **instantiated**. Other common methods are **setters** and **getters**. Setters are used to set or modify values or attributes, while getters are used to access values.

Typical Definition of a Class

```
class CLASSNAME:
      def __init__(self, ARGUMENTS):
            # initializer code goes here
            # includes setting initial values for attributes
      def METHODNAME(self, ARGUMENTS):
            # method code goes here
      ...
```

Instantiating an object of a particular class:

```
my_instance = CLASSNAME(ARGUMENTS)  # note: 'self' is not included
```

Classes

Answer the following questions.

1. Define each term as it relates to classes.
   a. attribute
   b. method
   c. instantiate

2. Another common analogy for a class is a cookie cutter. Explain.

Write programs that accomplish each task, using appropriate programming conventions.

3. Create a class, `Monster`, that has the following attributes. Initialize `type` with a random selection from Goblin, Orc and Imp, `life` with a random value between 10-20, and the other attributes with random values between 7-12. Instantiate a monster, and access the attributes from your main program. Display the values of the attributes in a nicely formatted manner.

Attributes	
`type`	`life`
`strength`	`skill`
`agility`	`luck`

4. Create three different instances of monsters using the `Monster` class above, and display them.

5. Create a class, `Widget`, that has the following attributes and methods. In your main program, create two instances of a `Widget` with default values of 'black' and 0. Use the setter methods to change the values to those input by the user, then use the getter methods to display them in a nicely formatted matter.

Attributes	Methods	
`colour` (e.g. white, purple)	`set_colour(self, colour)`	`get_colour(self, colour)`
`price` (e.g. $12.99, $700.00)	`set_price(self, price)`	`get_price(self, price)`

6. Write a class, `Fraction`, that has attributes for the numerator and denominator. Write methods to reduce the fraction to lowest terms (e.g. 4/6 becomes 2/3), to set the numerator and denominator, and to display the value as a fraction.