PRIME LESSONS

By the Makers of EV3Lessons

OBJECT ORIENTED PROGRAMMING

BY SANJAY AND ARVIND SESHAN

This lesson uses SPIKE 3 software

LESSON OBJECTIVES

- Learn Object Oriented Programming

CLASSES

- Classes allow you to group together a collection of variables and functions with a common purpose
- E.g. Class for animals in a zoo (ZooAnimal) could contain:
  - Type → tiger, monkey, snake
  - Weight → current weight in kg
  - Age → age in years
  - Birthday() → orders favorite food and increments age by 1

CLASSES VS INSTANCES

- You define Classes like functions and start with `class myClass(object):`
  - Inside the definition you list both
    - variables associated with a class → weight, age
    - methods (functions related to class) → birthday()

- A program can create many Instances of the defined Class -- i.e. variables of that type
  - E.g. ZooAnimal may be a Class and both LeoLion and GeoffGiraffe may be Instances of that class

METHODS

- Methods are functions associated with a class
- Defined inside start like functions with `def myMethod(self, parameters):`
  - Note that the "self" parameter is important as it defines that it relates to that class
- There is a special method called `__init__(self)`, which is called whenever you create an Instance of a Class
- To run a method, you need an Instance
  - E.g., `LeoLion.Birthday()`

EXAMPLE CLASS

```
class MyClass(object):
    # init method
    def __init__(self, n):
        # define class variables
        self.myVar = n

    # define a method that returns myVar+x
    def varPlus(self, x):
        # note that self. variables belong to the class
        # and can be accessed with calls to that class
        return self.myVar+x
```

CALLING CLASSES (OBJECTS)

- Based on the previous example…

```
myObject = MyClass(7) # sets that object's n-->7

print(myObject.varPlus(3)) # prints 7+3=10

print(myObject.myVar) # prints 7
```

- The object has methods that are defined in `myClass`, similar to lists, strings, and other data types
- You can customize these however you want
- You do not place "`self`" in method calls
  - The self is automatically replaced with the Instance you use to call the method

STATIC METHODS

- Static methods belong to the class, not to an individual object
- Begins with `@staticmethod`
- These methods are universal and do not need an Instance to be called
  - You do not have a `self` "parameter"

```
class MyClass(object):
   ....

   @staticmethod
   def myStaticMethod(x):
       print(x+20)

# You call static methods by
referring to a class, not an object
MyClass.myStaticMethod(10)  # 30
```

STATIC VARIABLES VS OBJECT VARIABLES

- Static variables are defined under the class definition, not a method
- Static variables can be accessed anywhere (static and non-static methods)
- Object variables are referred to by using self.someVariable
- Static variables are referred to by using myClass.someVariable

```
class MyClass(object):
   myStaticVar = 10 # a static variable

   def __init__(self, n):
       # this var cannot be accessed
       # from a static method
       self.myVar = n # variable pertaining
                      # to an object
   def printVar(self):
       # you can call a static and
       # non-static variable here
       return self.myVar

   @staticmethod
   def myStaticMethod():
       # print a static variable
       print(MyClass.myStaticMethod)
```

EXTRA: CLASS INHERITANCE

- Classes can "inherit" the methods/properties of another "superclass"
  - You replace "object" with the name of the other class
- Methods can be overridden in the child class by simply redefining it
- Overridden child methods can still refer back to the parent method by using super()....

```python
# Parent superclass
class MyClass(object):
    def __init__(self, n):
        self.myVar = n

    def printVar(self):
        return self.myVar

# Child class
class ChildClass(MyClass):
    # override a method
    def __init__(self, n, a):
        self.a = a
        # call init of the super class
        super().__init__(n)

c = ChildClass(4, 4)
# printVar() in inherited
print(c.printVar()) # 4
```

CHALLENGE

- Create a class that will store information about countries and print it on a method call
  - It should store name, population, and area
  - Your methods should be 1) print info and 2) get population density (population/area)
- Display your country's population density on the hub screen

CHALLENGE SOLUTION

```
from hub import light_matrix
import runloop, sys

class Country(object):
    def __init__(self, name, population, area):
        self.name = name
        self.population = population
        self.area = area

    def printInfo(self):
        print("Name:", self.name, " Population:", self.population, "Area:", self.area)

    def getDensity(self):
        return self.population/self.area

# Function to stop the program using a system exception
def stopAndExitProgram():
    sys.exit("Stopping")

async def main():
    myCountry = Country("New Country", 500000, 1000000)
    myCountry.printInfo()
    await light_matrix.write(str(myCountry.getDensity())) # convert float to str before writing
    stopAndExitProgram()

runloop.run(main())
```

CREDITS

- This lesson was created by Sanjay and Arvind Seshan for Prime Lessons
- Additional contributions by FLL Share & Learn community members.
- More lessons are available at www.primelessons.org