

PRIME LESSONS

By the Makers of EV3Lessons



PROGRAMAREA ORIENTATĂ PE OBIECTE

BY SANJAY AND ARVIND SESHAN

OBIECTIVELE LECȚIEI

- Învățăm programarea orientată pe obiecte

CLASELE

- Clasele îți permit să grupezi împreună o colecție de variabile și funcții cu un scop comun.
- E.g. O clasă cu animale în zoo (ZooAnimal) poate conține:
 - Tip → tiger, monkey, snake
 - Greutate → greutatea în kg
 - Vârstă → vârsta în ani
 - Zi de naștere() → comadați hrana favorită și încrementează vârsta cu 1.

CLASELE VS INSTAȚELE

- Definiți clasele ca funcții și începe cu `class myClass (object) :`
 - În interiorul definiției ar trebui să listați
 - Variabilele asociate cu o clasă → greutate, vârstă
 - methods (functions related to class) → birthday()
- Un program poate crea mai multe instanțe a clasei definite Class -- i.e. Variabile de ce tip
 - E.g. ZooAnimal poate fi o clasă și ambele LeoLion și GeoffGiraffe pot fi instanțe a clasei.

METODE

- Metodele sunt funcții asociate unei clase.
- Definește în interiorul funcțiilor `def myMethod(self, parameters) :`
 - Observați că parametrul “self” parameter este important din moment ce legătura cu clasa
- Există o metodă specială numită `__init__(self)`, care apelează orice creezi în instanța unei clase
- Pentru a rula o metodă, ai nevoie de o instanță.
 - E.g., `LeoLion.Birthday()`

EXEMPLU DE CLASĂ

```
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
```

APELAREA CLASELOR(OBIECTELOR)

- Bazat pe exemplul anterior...

```
myObject = MyClass(7) # sets that object's n-->7  
print(myObject.varPlus(3)) # prints 7+3=10  
print(myObject.myVar) # prints 7
```

- Obiectul are metode care sunt definite în `myClass`, similar cu liste, șiruri de caractere și alte tipuri de date
- Poți personaliza acestea oricum dorești
- Nu plasa “`self`” în apelarea metodei
 - `Self` este automat înlocuit cu o instanță pe care o folosești pentru apelarea unei metode.

METODE STATICE

- Metodele statice țin de o clasă, nu de obiecte individuale
- Începe cu `@staticmethod`
- Aceste metode sunt individuale și nu au nevoie de o instanță pentru a fi apelată
 - Nu au un `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
```


VARIABLELE STATICE VS VARIABLELE OBIECT

- Variabilele statice sunt definite sub definiția unei clase, nu a unei metode
- Variabilele statice pot fi accesate de oriunde (metode statice și nonstatice)
- Variabilele Obiect au ca referință la ele `self.someVariable`
- Variabilele statice au ca referință `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: MOȘTENIREA CLASELOR

- Clasele pot fi „moștenite” metodele/proprietățile sau altă “superclass”
 - Înlocuiești “object” cu numele altei clase
- Metodele pot fi transformate într-o clasă copil prin simpla redefinire
- Metodele copil pot primi ca referință metoda părinte prin utilizarea `super()`....

```
# 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
```

PROVOCAREA

- Crează o clasă care să stocheze informația despre țări și printează-o apelând o metodă
 - Ar trebui stocat numele, populația și zona
 - Metodele ar trebui să fie 1) print info și 2) get population density (population/area)
- Afișează densitatea populației țării tale pe display-ul Hub-ului

SOLUȚIA PROVOCĂRII

```
from spike import PrimeHub, LightMatrix

from spike.control import wait_for_seconds, wait_until, Timer

from math import *

import time

hub = PrimeHub()

class Country(object):

    def __init__(self, name, population, area):

        self.name = name

        self.population = population

        self.area = area

    def info(self):

        print(self.name, self.population, self.area)

    def getDensity(self):

        return self.population/self.area

myCountry = Country("New Country",500000,1000000)

print(myCountry.getDensity())

hub.light_matrix.write(myCountry.getDensity())
```

CREDITS

- Această lecție de SPIKE Prime a fost realizată de Sanjay Seshan și Arvind Seshan.
- Mai multe lecții sunt disponibile pe www.primelessons.org
- Această lecție a fost tradusă în limba română de echipa de robotică FTC – ROSOPHIA #21455 RO20



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).