



קריית חינוך "פארק המדע"

בית לערכים, למצוינות וחדשנות

# תכנות מתקדם בשפת פייתון

גלעד מרקמן





קריית חינוך "פארק המדע"

בית לערכים, למצוינות וחדשנות

# מחלקות וירושה

כתיבת מחלקה

הגדרת מאפיינים (אתחול)

פעולות (פעולה סטטית)

פעולות מיוחדות כגון: `__str__`, `__call__`

יצירת אובייקטים

ירושה



קריית חינוך "פארק המדע"

בית לערכים, למצוינות וחדשנות

# מחלקה class, בנאי ומאפיינים

- מחלקה מוגדרת באמצעות הפקודה class.
- הבנאי של המחלקה היא הפונקציה `__init__` (קווים תחתונים כפולים בכל צד). במסגרת פונקציה זו מגדירים ומאתחלים את המאפיינים של המחלקה.

```
class Point:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
  
p = Point (5, 7)  
print (p.x, p.y)
```



# מתודות (פונקציות במחלקה)

קריית חינוך "פארק המדע"

בית לערכים, למצוינות וחדשנות

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def print (self):
        print ('x = ', p.x, ' y = ', p.y)

    def setXY (self, x, y):
        self.x = x
        self.y = y

p = Point (5, 7)
p.print()
p.setXY(10, 10)
p.print()
```

```
x = 5  y = 7
x = 10 y = 10
```

- מתודה של המחלקה מוגדרת על ידי פונקציה אליה מוסיפים את הפרמטר self כארגומנט ראשון.
- Self הוא דומה ל this ב CSharp.
- בעת הקריאה למתודה לא מעבירים את הארגומנט self.



קריית חינוך "פארק המדע"

בית לערכים, למצוינות וחדשנות

# Overloading

- אין בפייתון overloading ולא ניתן ליצור שתי מתודות עם אותו שם.
- הסיבה לכך שניתן ליצור מתודות עם ערכי ברירה מחדל, כך שלא חייבים להעביר להם ארגומנטים:

```
def setXorY (self, x = None, y = None):  
    if x is not None:  
        self.x = x  
    if y is not None:  
        self.y = y  
  
p = Point (0, 0)  
p.setXorY ()  
p.setXorY(y = 5)  
p.setXorY(x = 7)  
p.setXorY(x = 3, y = 9)
```



קריית חינוך "פארק המדע"

בית לערכים, למצוינות וחדשנות

# מאפיין מחלקה

- מאפיין מחלקה מוגדר כמשתנה מחוץ לבנאי וללא המילה `.self`.
- בדרך זו ניתן לפנות למשתנה גם ללא יצירת אובייקט.

```
class myClass:
    class_attr = 5

    def __init__(self, attr):
        self.instance_attr = attr

o = myClass(10)
print(myClass.class_attr)
print(o.instance_attr)
print(o.class_attr)
print(myClass.instance_attr) # error - type object 'myClass' has no attribute 'instance_attr'
```



קריית חינוך "פארק המדע"

בית לערכים, למצוינות וחדשנות

# פעולות מיוחדות

- ישנן פעולות מיוחדות המוגדרות לכל הפונקציות שניתן להתאימם למחלקות שלנו:
- הפעולה `__str__` הזוהה לפעולה `ToString` ב `C#`.
- הפעולה `__eq__` המשווה בין שני אובייקטים.
- ניתן לבצע דריסה של פעולות אילו כדי להגדיר התנהגות מתאימה למחלקות שיצרנו.
- <https://docs.python.org/3/reference/datamodel.html#special-method-names>



קריית חינוך "פארק המדע"

בית לערכים, למצוינות וחדשנות

# ירושה

- פייטון מאפשר להגדיר ירושה, באמצעותה מחלקה יכולה לרשת את כל המאפיינים והפונקציות של המחלקה המורישה.
- הגדרת ירושה נעשית באמצעות הוספת שם המחלקה המורישה בהגדרת המחלה:

```
class Student:
    def __init__(self, name, phone, id):
        self.name = name
        self.phone = phone
        self.Id = id

class MathStudent (Student):
    def __init__(self, name, phone, id, algebra, calulos):
        super().__init__(name, phone, id)
        self.Algebra = algebra
        self.calculos = calulos
```





קריית חינוך "פארק המדע"

בית לערכים, למצוינות וחדשנות

# super

- על מנת לקרוא לפונקציות של המחלקה המורשה (לרבות הבנאי `__init__`), אנו יכולים להשתמש בשתי דרכים, שימוש בשם המחלקה המורשה או במילה השמורה `:super`:

```
class Student:
    def __init__(self, name, phone, id):
        self.name = name
        self.phone = phone
        self.Id = id

class MathStudent (Student):
    def __init__(self, name, phone, id, algebra, calulos):
        super().__init__(name, phone, id)
        self.Algebra = algebra
        self.calculos = calulos
```

```
class Student:
    def __init__(self, name, phone, id):
        self.name = name
        self.phone = phone
        self.Id = id

class MathStudent (Student):
    def __init__(self, name, phone, id, algebra, calulos):
        Student.__init__(self, name, phone, id)
        self.Algebra = algebra
        self.calculos = calulos
```



קריית חינוך "פארק המדע"

בית לערכים, למצינות וחדשנות

# Overriding Function

- במחלקה היורשת ניתן לכתוב פונקציה המחליפה את הפונקציה של המחלקה המורישה על ידי כתיבת הפונקציה מחדש (אין צורך לציין כי מדובר ב overriding).
- בפונקציה ה"דורסת" ניתן לפנות אל הפונקציה המקורית באמצעות המילה השמורה super או באמצעות שם הפונקציה:

```
class Student:
    def __init__(self, name, phone, id):
        self.name = name
        self.phone = phone
        self.Id = id

    def __str__(self):
        return f'{self.name} {self.phone} {self.Id}'
```

```
class MathStudent (Student):
    def __init__(self, name, phone, id, algebra, Calulos):
        Student.__init__(self, name, phone, id)
        self.Algebra = algebra
        self.Calculos = Calulos

    def __str__(self):
        return Student.__str__(self) + f'{self.Algebra} {self.Calculos}'
```



# ירושה מרובה

- פייטון מאפשר להגדיר ירושה (מרובה). כלומר מחלקה יכולה לרשת ממספר מחלקות.
- הגדרת ירושה נעשית באמצעות הוספת שם המחלקה המורישה בהגדרת המחלה:

```
class Student:  
    def __init__(self, name, phone, id):  
        self.name = name  
        self.phone = phone  
        self.Id = id
```

```
class MathStudent (Student):  
    def __init__(self, name, phone, id, algebra, Calulos):  
        Student.__init__(self, name, phone, id)  
        self.Algebra = algebra  
        self.Calculos = Calulos
```

```
class ComputerStudent (Student):  
    def __init__(self, name, phone, id, python, CSharp):  
        Student.__init__(self, name, phone, id)  
        self.Python = python  
        self.CSharp = CSharp
```

```
class Math_Com_Student (MathStudent, ComputerStudent):  
    def __init__(self, name, phone, id, algebra, calulos, python, CSharp):  
        MathStudent.__init__(self, name, phone, id, algebra, calulos)  
        ComputerStudent.__init__(self, name, phone, id, python, CSharp)
```