פייתון

שיעור 10: תכנות מונחה עצמים OOP

ברק גונן

תכנית הסייבר הלאומית

שלבי הלימוד

- מבוא ל-OOP- בשביל מה זה טוב?
- כתיבת class בסיסי
- בניית class משופר
- ירושה inheritance
- פולימורפיזם

חלק 1: מבוא ל-OOP

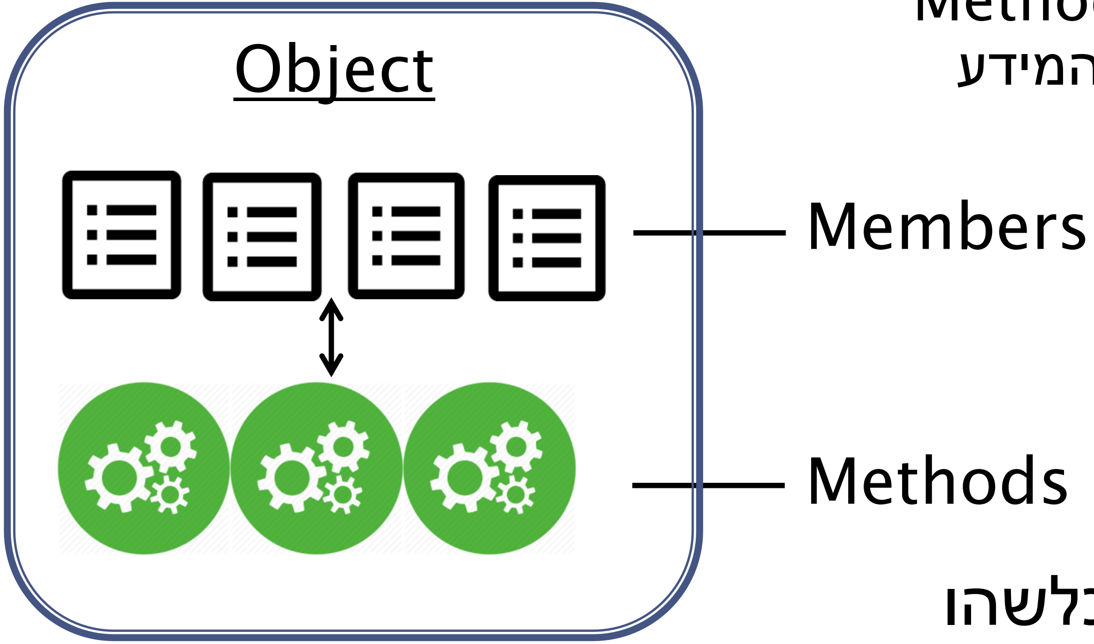
- עד עכשיו כתבנו קוד שקרא לפונקציות
  - Procedural Programming
- מדי פעם עשינו import למודולים
- כעת נלמד לכתוב כאלה בעצמנו ☺
  - Object Oriented Programming
- ככה בדרך כלל עובדים ב"עולם האמיתי"

מדוע Procedural Programming אינו מושלם?

- תוכנה שמחשבת ממוצע ציוני תלמידים:
  - 200 תלמידים בבית הספר
  - 10 מקצועות
  - נדרשים אלפי משתנים
- נשתמש ברשימה?
  - מסובך לעבוד עם אינדקס לכל מקצוע

Object Oriented Programming

- אז מהו אובייקט?
- ישות תוכנה שמכילה מידע ופונקציות
  - המידע נקרא Members
  - הפונקציות נקראות Methods
  - הפונקציות עובדות עם המידע

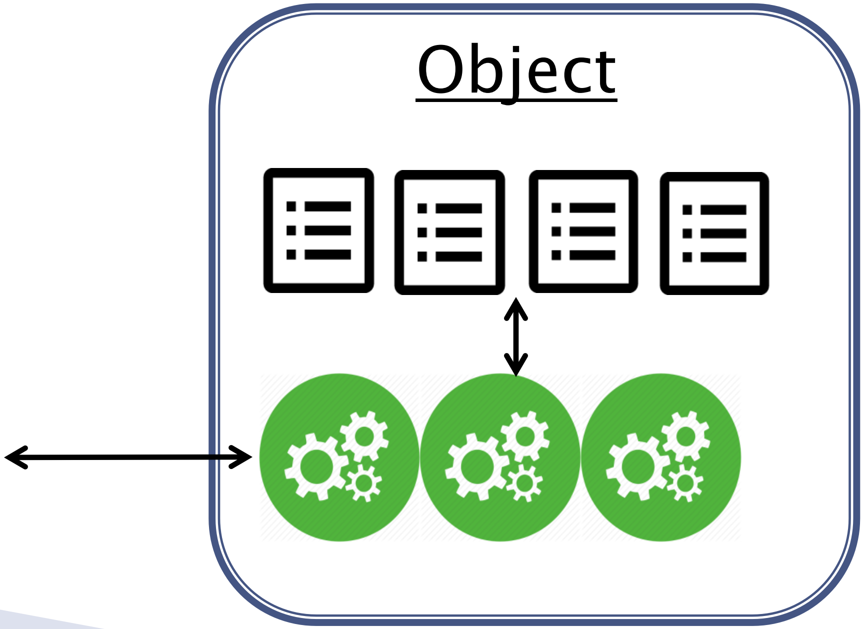


- תנו דוגמה לאובייקט כלשהו
  - המציאו לו methods ו-Members

איך OOP משפר את מצבנו?

- אפשר לקרוא בקלות ל-members ולמתודות של כל אובייקט
- אפשר לשנות מתודות ו-members של אובייקט בלי לשנות את הקוד שמשתמש באובייקט

קוד תוכנית



class

- class, מחלקה, הוא קטע תוכנה שמגדיר את כל ה-members וה-methods של אובייקט
- נמחיש:
  - שבתאי הוא כוכב לכת אחד מתוך רבים
  - למרות שכל כוכב לכת הוא ייחודי, לכולם יש רדיוס, צפיפות, מרחק מהשמש...
  - אפשר להגדיר מחלקה של כוכב לכת, Planet
  - המחלקה Planet תכיל את המאפיינים המשותפים לכל כוכבי הלכת

איך נוצר אובייקט מתוך מחלקה?

- מחלקה מתארת את המאפיינים של אובייקט
- כשהתוכנית רצה, המחלקה משמשת ליצירת אובייקטים בזיכרון המחשב
- המחשה: עוגיה נוצרת מתבנית לעוגיות. התבנית לעוגיות אינה עוגיה, אבל היא מתארת את הצורה של כל העוגיות.

Instance

- כל אובייקט הוא instance של המחלקה ממנה נוצר
- לדוגמה:
  - כל עוגיה היא instance של תבנית העוגיות
  - שבתאי הוא instance של המחלקה Planet

סיכום ביניים

- הסבירו את ההבדל בין Class, Instance, Object
- השלימו את התבניות הבאות תוך שימוש במושגים הנ"ל:
  - __________ הוא __________ של __________
  - __________ הוא תבנית של __________
  - לכל __________ של __________ יש data attributes משלו
- כעת נלמד איך להגדיר מחלקה

חלק 2: כתיבת class

- אנחנו מפעילים מגדל פיקוח וצריכים למנוע התנגשויות מטוסים. אבל כל הטייסים השתגעו וטסים בכיוונים אקראיים... משימתנו לעקוב אחרי המיקום של כל המטוסים

הגדרת class

- ניצור class בשם CrazyPlane:

מקובל ששם ה-class יתחיל באות גדולה

```
class CrazyPlane:

    def __init__(self):
        self.x = 0
        self.y = 0
```

מיד נסביר מה זה __init__

מיד נסביר מה זה self

x, y- data attributes של ה-class

המתודה __init__

- המתודה __init__ רצה אוטומטית בכל פעם שנוצר instance של CrazyPlane בזיכרון המחשב
- בתוכה נהוג לשים אתחול של ה-data attributes
- בשפת פייתון ידועה בשם initializer
  - באופן כללי בעולם ה-OOP: constructor, בנאי

```
def __init__(self):
    self.x = 0
    self.y = 0
```

self

- איך פייתון יודע על איזה אובייקט להפעיל את המתודה?
  - הרי יכולים להיות הרבה instanceים של אותו class
- תשובה: self מצביע על האובייקט שעליו המתודה פועלת
- לכן מוסיפים למתודות את הפרמטר self

הוספת מתודות - CrazyPlane

- המתודה update_position משנה את קואורדינטות המטוס
  - נדרש import random
- המתודה get_position מאפשרת לקרוא את מיקום המטוס

```
def update_position(self):
    self.x += random.randint(-1, 1)
    self.y += random.randint(-1, 1)

def get_position(self):
    return self.x, self.y
```

מה ההבדל בין members למשתנים?

```python
class CrazyPlane:

    def __init__(self):
        self.x = 0
        self.y = 0

    def count_down(self):
        for i in range(10,0,-1):
            print i
```

- המשתנה i "חי" רק בתוך לולאת ה-for
- x ו- y חיים כל עוד האובייקט קיים- members

יצירת אובייקט plane

- הקוד הבא יוצר אובייקט בשם plane1 ומשתמש בו:

```
def main():
    plane1 = CrazyPlane()
    xpos , ypos = plane1.get_position()
```

- חישבו- מה יהיה ערכם של xpos, ypos?
  - יצרנו אובייקט plane1 שהוא instance של CrazyPlane
  - מתודת ה-__init__ של CrazyPlane מופעלת אוטומטית ויוצרת משתנים plane1.x, plane1.y
  - מתודת get_position שפועלת על האובייקט plane1 מחזירה את ערכי plane1.x ו-plane1.y

שאלה למחשבה

- ל-class יש משתנים x, y
- האם תקין לכתוב כך בתוכנית הראשית?

```
x, y = plane1.get_position()
```

- כן. המשתנים x, y אינם המשתנים plane1.x ו-plane1.y, ששייכים לאובייקט plane1
  - אין להם את אותו id()
  - אם נשנה את x אז plane1.x לא ישתנה

קריאה למתודה update_position

```
def main():
    plane1 = CrazyPlane()
    x, y = plane1.get_position()
    print 'Coordinates of plane1: ' + \
        '%d , %d' % (x, y)
    for i in xrange(10):
        plane1.update_position()
        x, y = plane1.get_position()
        print 'Coordinates of plane1: ' + \
            '%d , %d' % (x, y)
```

התו \ מסמן המשך
בשורה הבאה
(לא הכרחי)

- תוצאת ההרצה:

```
Coordinates of plane1: 0 , 0
Coordinates of plane1: 1 , 0
Coordinates of plane1: 0 , 1
Coordinates of plane1: -1 , 0
Coordinates of plane1: -2 , 1
Coordinates of plane1: -3 , 0
Coordinates of plane1: -3 , 1
Coordinates of plane1: -2 , 2
Coordinates of plane1: -2 , 1
Coordinates of plane1: -3 , 1
Coordinates of plane1: -2 , 1
```

סיכום ביניים #2

- כיתבו class של החיה האהובה עליכם (Cat, Dog etc)
- הוסיפו ל-class מתודת __init__ שתכלול:
  - שם החיה (לדוגמה Kermit)*
  - גיל החיה (מתחיל מ-0)
- הוסיפו ל-class מתודות:
  - birthday- תעלה את הגיל ב-1
  - get_age- ערך החזרה הוא גיל החיה

* בהמשך נלמד איך לאפשר לתת שם שונה לכל חיה

גבהים
תכנית הסייבר הלאומית

חלק 3: בניית class משופר

- כעת נלמד לבנות class משופר:
  - ניצור משתנים "מוסתרים"
  - נהפוך את ה-class למודול שניתן לייבא עם import
  - נלמד לקבוע ערכים התחלתיים לאובייקט חדש ב-class
  - ניצור פקודת הדפסה מיוחדת ל-class שלנו
  - ניצור מתודות accessor ו-mutator

האם אפשר "לחטוף" מטוס?

- קטע הקוד הבא דורס את נתוני המיקום של המטוס

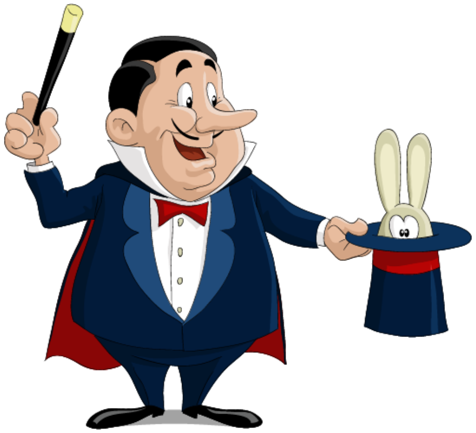
```
plane1.x = 10
plane1.y = 10
x, y = plane1.get position()
```

- בעולם האמיתי, התוצאה עלולה להיות התנגשות קטלנית

"הסתרה" של משתנים

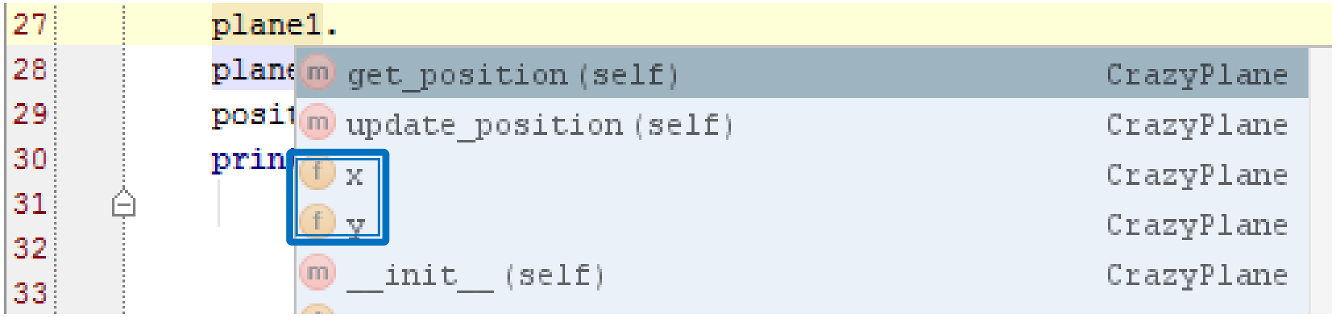
- אז איך אפשר למנוע טעויות שיגרמו להתנגשות מטוסים?
- "נסתיר" את המשתנים בעזרת תחילית __
  - __x
  - __y

```
def __init__(self):
    self.__x = 0
    self.__y = 0
```

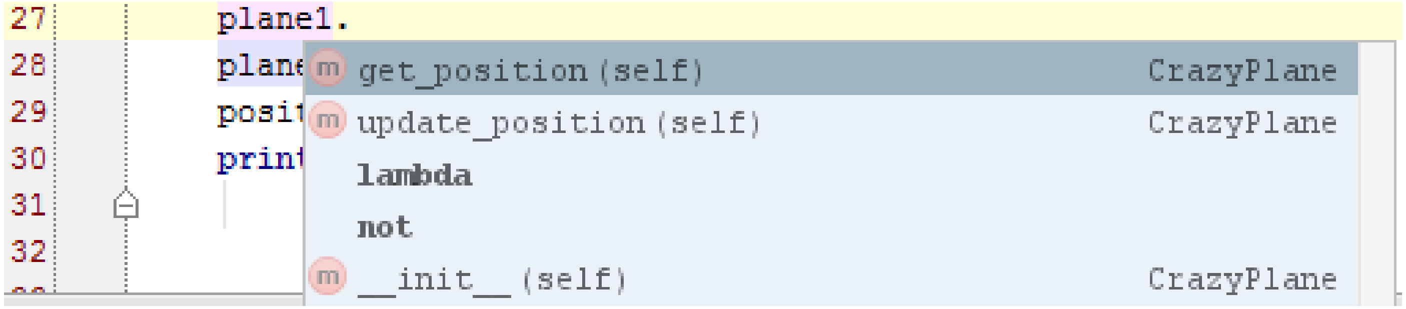


"הסתרה" של משתנים

- כעת x ו-y נסתרים* ממי שכותבים קוד שמשתמש ב-class שלנו
- לפני:



- אחרי:



* עדיין ניתן לגשת אליהם – מיד נראה

גישה למשתנים מוסתרים

- בשפות עיליות ישנו מושג private
  - משתנים או מתודות שאי אפשר לגשת אליהם מחוץ ל-class
- פייתון שפה מיוחדת: יש דרך גישה נסתרת...

גישה למשתנים מוסתרים

- מה יקרה כשנבצע dir על אובייקט crazy_plane?

```
elal = CrazyPlane()
print dir(elal)
```

- "נגלה" משתנים:

```
_CrazyPlane__x
_CrazyPlane__y
```

- בידקו מה מבצעת הפקודה:

```
elal._CrazyPlane__x = 3
```

Accessor, Mutator

- אז איך ניגשים למשתנים של אובייקט?
  - המתכנת צריך לדאוג למתודות מתאימות
- Accessor- מתודה שמאפשרת קריאה
  - נהוג שהשם מתחיל ב-"get"
  - דוגמה: המתודה get_position, מאפשרת קריאת המיקום
- Mutator- מתודה שמאפשרת שינוי
  - נהוג שהשם מתחיל ב-"set"
  - אפשר לממש מתודה בשם set_position שמשנה את המיקום

המשך - accessor, mutator

- יש טיעונים בעד ונגד accessor, mutator

JAVA TOOLBOX

By Allen Holub, JavaWorld | SEP 5, 2003 1:00 AM PT

HOW-TO

Why getter and setter methods are evil

Make your code more maintainable by avoiding accessors

Getters and Setters Are Not Evil

by Bozhidar Bozhanov MVB · Oct. 14, 11 · Java Zone

- לא נכריע בדיון, חשוב להכיר את השיטות
  - בעד- שליטה בערכים של ה-members (לדוגמה- מיקום של מטוס לא יהיה שלילי)
  - נגד – קוד מסובך יותר

יצירת מודולים

- נוכל לשמור את ה-class שכתבנו בקובץ נפרד (מודול)
- שם הקובץ (לדוגמה) crazy_plane.py

crazy_plane.py

```
import random


class CrazyPlane:

    def __init__(self):
        self.__x = 0
        self.__y = 0

    def update_position(self):
        self.__x += random.randint(-1, 1)
        self.__y += random.randint(-1, 1)

    def get_position(self):
        return self.__x, self.__y
```

הקוד של ה-class נותר זהה, הסרנו את ה-main

import

- כדי להשתמש במודול, נבצע לו import בקובץ הראשי-

```
import crazy_plane

def main():
    plane1 = crazy_plane.CrazyPlane()
```

שם המודול (הקובץ)

שם ה-class

import - המשך

- ניתן לעשות import למודול בדרכים נוספות:
  - לייבא את ה-class שאנחנו רוצים

```
from crazy_plane import CrazyPlane
```

  - לייבא את כל ה-classes שבקובץ

```
from crazy_plane import *
```

- כעת נוכל לכתוב קוד מקוצר:

```
plane1 = CrazyPlane()
```

שם ה-class בלבד. אין צורך בשם המודול.

אתחול של אובייקט

- פתחנו שדה תעופה נוסף במיקום 5,5 וחלק מהמטוסים ממריאים ממנו
- נאפשר ל-CrazyPlane לקבל מיקום התחלתי

```
def __init__(self, x, y):
    self.__x = x
    self.__y = y
```

אתחול אובייקט - המשך

- כשניצור אובייקט חדש נעביר לו את המיקום כפרמטר

```
from crazy_plane import *

AIRPORT_X = 5
AIRPORT_Y = 5

plane1 = CrazyPlane(AIRPORT_X, AIRPORT_Y)
```

אתחול אובייקט - המשך

- לפעמים נרצה שאובייקט יקבל ערך ברירת מחדל
- הקוד הבא מכניס את ערכי ברירת המחדל אם לא הועבר פרמטר:

```
def __init__(self, x=0, y=0):
    self.__x = x
    self.__y = y
```

__str__

- נלמד איך להדפיס בקלות מידע של אובייקט
- נוסיף מתודת "קסם" __str__ ל-class:

```
def __str__(self):
    return 'Coordinates: ' + '%d , %d' \
           % (self.__x, self.__y)
```

- נבצע בתוכנית print:

```
for i in xrange(10):
    plane1.update_position()
    print plane1
```

- נסו בעצמכם!

יצירת אובייקטים מרובים

- אפשר ליצור מכל class אובייקטים רבים
- כל אובייקט חייב שם ייחודי
- הקוד הבא יוצר 4 אובייקטים ומאתחל אותם:

שימוש בערך ברירת המחדל

```
elal = CrazyPlane()
american = CrazyPlane(NEW_YORK_X, NEW_YORK_Y)
british = CrazyPlane(LONDON_X, LONDON_Y)
lufthansa = CrazyPlane(BERLIN_X, BERLIN_Y)
```

לולאה על אובייקטים

- נכניס את האובייקטים למבנה נתונים, כגון רשימה, וכך נוכל לבצע עליהם פעולות בלולאה:

```
planes = [elal, american, british, lufthansa]
for plane in planes:
    print plane
```

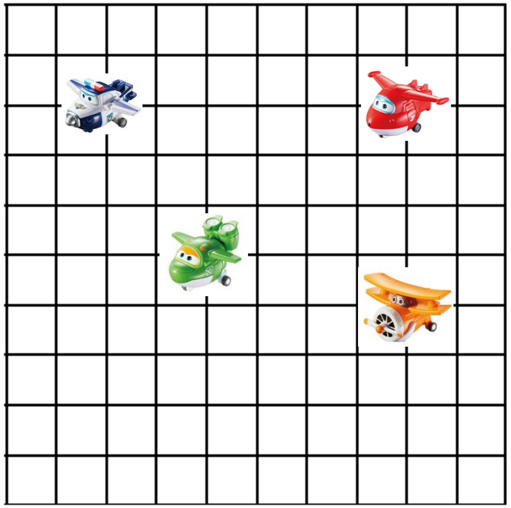
גבהים
תכנית הסייבר הלאומית

סיכום ביניים #3

- שפרו את ה-class של החיה שיצרתם:
  - היפכו את ה-class למודול ועשו לו import
  - הסתירו את שם החיה ואת הגיל שלה (תזכורת: __)
  - אפשרו לקבוע את שם החיה בזמן יצירת האובייקט
  - אפשרו לשנות את שם החיה (מתודת set)
  - אפשרו לקרוא את שם החיה ואת הגיל החיה (מתודת get)
  - אפשרו הדפסת פרטי החיה ע"י print (מתודת __str__)
  - לימוד עצמי: ממשו את מתודת __repr__

תרגיל מתקדם – פיקוח אווירי

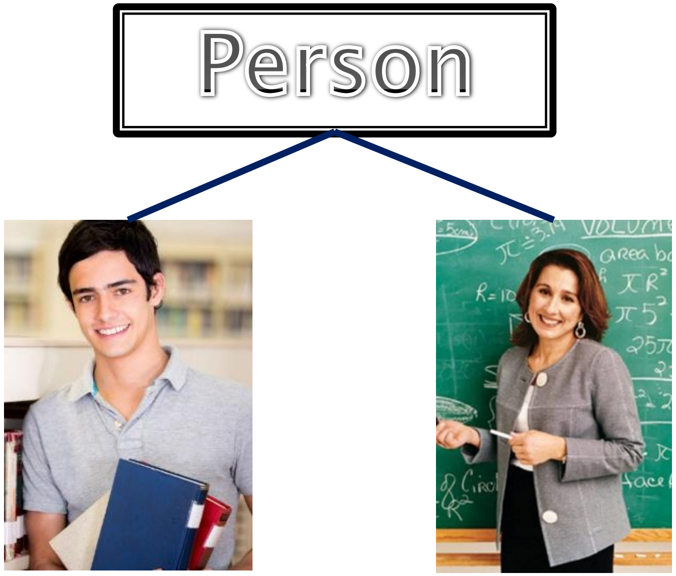
- עליכם למנוע התנגשות מטוסים
- המרחב האווירי שלכם הוא ריבוע בגודל 10X10
- 4 מטוסים מסוג CrazyPlane משייטים במרחב
  - לולאה אינסופית המעדכנת את מיקום המטוסים
- אם שני מטוסים מתקרבים (פחות מ-2 משבצות) שילחו הוראת turn עם קואורדינטה סמוכה למיקום המטוס
  - לדוגמה מטוס שנמצא ב-[1,2] יכול לקבל פקודה לעבור ל-[1,3]
- מטוס שקיבל הוראת turn ישנה את מיקומו בהתאם



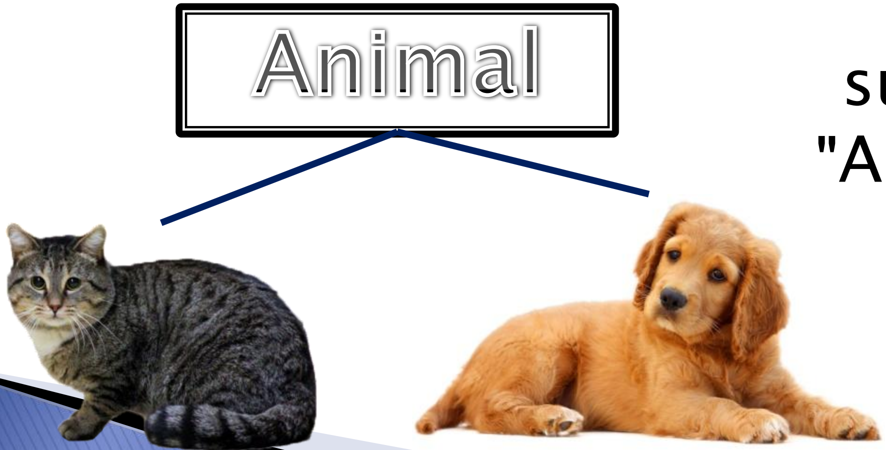
חלק 4: ירושה inheritance

- לעיתים class הוא סוג ספציפי של class אחר:
  - חתול הוא חיה
  - מכונית היא כלי תחבורה
  - חולצה היא פריט לבוש
- נוכל ליצור class שיש לו (מקבל בירושה) את כל התכונות של class אחר ועוד תכונות מיוחדות לו

Subclass, Superclass



- ה-class היורש נקרא subclass
- ה-class שיורשים ממנו נקרא superclass
- Teacher ו-Student הם subclasses של ה-superclass "Person"
- Cat ו-Dog הם subclasses של ה-superclass "Animal"



Class “Person”

- ב-Person נשמר השם והגיל של האדם

```
class Person:

    def __init__(self, name='Tal', age=20):
        self.__name = name
        self.__age = age

    def say(self):
        print 'Hi :)'

    # print
    def __str__(self):
        return 'Person %s is %d years old' % \
            (self.__name, self.__age)
```

המשך - Class "Person"

- נוסיף accessors ו- mutators למשתנים name ו-age:

```
# acessors
def get_name(self):
    return self.__name

def get_age(self):
    return self.__age

# mutators
def set_name(self, name):
    self.__name = name

def set_age(self, age):
    self.__age = age
```

יצירת School

- בבית הספר יש מורים ותלמידים
  - למורים יש שם, גיל ושכר
  - לתלמידים יש שם, גיל, וממוצע ציונים
- יש לנו כבר class שמכיל שם וגיל
  - האם אפשר להשתמש בו?
- כמובן ☺ נגדיר יחס ירושה:

```
class Teacher(Person):

class Student(Person):
```

מציין ירושה מ- Person

Teacher

- ניצור class שאין בו כלום

```
class Teacher(Person):
    pass
```

- מה יודפס אם נריץ את שורות הקוד הבאות?

```
teacher1 = Teacher()
print teacher1
```

Person Tal is 20 years old

- חישבו מדוע?

Teacher - המשך

- אנחנו רוצים ש-Teacher יהיה Person עם שכר:

מציין ירושה מ- Person

ערכי ברירת מחדל של Teacher

```
class Teacher(Person):

    def __init__(self, name='Daniel', age=30, salary=50):
        Person.__init__(self, name, age)
        self.__salary = salary
```

הוספת מידע ספציפי של Teacher

אתחול של Person

Teacher - המשך

- מה כעת יודפס אם נריץ את שורות הקוד הבאות?

```
teacher1 = Teacher()
print teacher1
```

Person Daniel is 30 years old

- ומה יודפס אם לא נשתמש בערכי ברירת המחדל?

```
barak = Teacher('Barak', 40, 60)
print barak
```

Person Barak is 40 years old

super

- אפשר לקרוא ל-superclass בלי לכתוב את השם שלו:

```
class Teacher(Person):

    def __init__(self, name, age, salary):
        super(self.__class__, self).__init__(name, age)
        self.__salary = salary
```

- צריך להוסיף ל-superclass ירושה מ-object

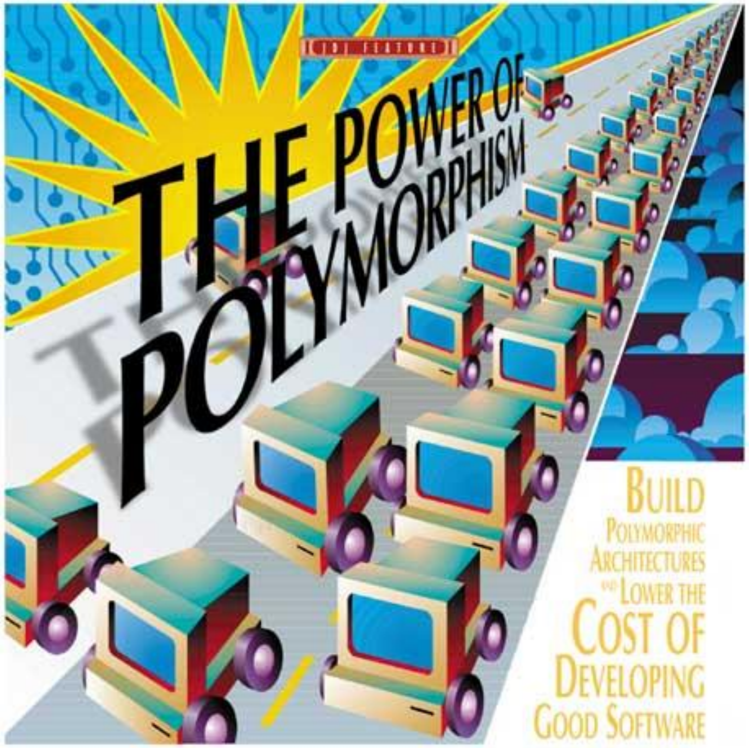
```
class Person(object):
```

- מתי זה שימושי?
  - כשלא ידוע לנו מי ה-superclass
  - נרצה לקרוא לפונקציה שהיורש דרס

תרגיל מסכם ירושה: Student

- צרו “Student” שיורש מ-“Person”
  - קבעו ערכי ברירת מחדל כרצונכם
  - הוסיפו ל-__init__ שלו את __grade
  - הוסיפו ל-__grade :
    - accessor
    - mutator

חלק 5: פולימורפיזם



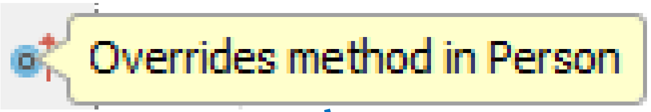
- נפעיל את say() על אובייקט מסוג Teacher:

```
Hi :)
```

- נרצה שהמורה יאמר משהו ייחודי
  - ... ועדיין יירש את כל יתר התכונות של Person

Method overrding

- בתוך Teacher "נדרוס" את המתודה say של Person:




```
def say(self):
    print 'Good morning students!'
```

המשך -Method overriding

- כעת מה יקרה אם נפעיל את say על אובייקט מסוג Teacher?

```
Good morning students!
```

- נעדכן גם את מתודת ה-__str__ כך שתהיה ייחודית ל-Teacher:

```
def __str__(self):
    return 'Teacher %s is %d years old, hourly income %d' % \
        (self.get_name(), self.get_age(), self.__salary)
```

שימו לב איך subclass פונה למשתנים מוסתרים של superclass

isinstance

- בבית הספר המעולה של נווה חמציצים נפתחה מגמת סייבר
- אובייקט מסוג CyberStudent הוא כמו Student אך כולל גם:
  - ציון בסייבר
  - accessor ו-mutator לציון הסייבר

```
class CyberStudent(Student):

    def __init__(self, name, age, grade, cyber_grade):
        Student.__init__(self, name, age, grade)
        self.__cyber_grade = cyber_grade
```

isinstance- המשך

- מנהל בית הספר מבקש שכל תלמיד שהציון שלו בסייבר גבוה מ-80 יקבל הודעת "Wow!"
- **<u>כל</u>** התלמידים נמצאים ב-list שנקרא students
- האם הקוד הבא יעבוד היטב?

```
for student in students:
    if student.get_cyber_grade() >= 80:
        print 'Wow!'
```

- ... ל-Student אין מתודה get_cyber_grade()

```
AttributeError: Student instance has no attribute 'get_cyber_grade'
```

isinstance- המשך

- הפונקציה isinstance מאפשרת לבדוק אם אובייקט שייך ל-class כלשהו
  - מחזירה True או False
  - שימו לב- אובייקט תמיד שייך ל-superclass של ה-class שלו

- isinstance(object_name, class_name)

  - כעת ניתן לבדוק שאובייקט הוא מסוג CyberStudent לפני שמנסים לקרוא ל-cyber_grade שלו

isinstance- דוגמאות

- נתון:

```
noam = Student('Noam', 17, 93)
daniel = CyberStudent('Daniel', 17, 95, 90)
```

- מה תהיה התוצאה בכל אחת מהשורות הבאות:

```
print isinstance(noam, Student)
print isinstance(daniel, CyberStudent)
print isinstance(noam, Person)
print isinstance(noam, CyberStudent)
print isinstance(daniel, Student)
```

תרגיל מסכם פולימורפיזם- BigCat

- הגדירו class בשם BigThing
  - מקבלת אובייקט כלשהו
  - מתודה size:
    - אם האובייקט הוא מספר- מחזירה אותו
    - אם האובייקט הוא רשימה / מילון / מחרוזת- מחזירה את len שלו

- הגדירו class בשם BigCat
  - יורש מ-BigThing
  - מקבל כפרמטר גם משקל
  - אם המשקל > 15, size תחזיר "Fat", אם המשקל > 20 תחזיר "Very fat", אחרת "OK"

```
latif = BigCat('latif', 22)
```

סיימתם ללמוד OOP!