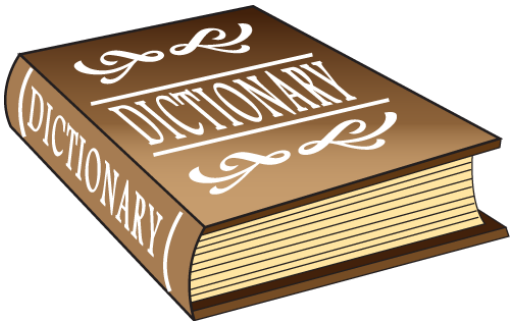
# פייתון

## שיעור 12: מילונים

ברק גונן

# הגדרה של מילון




- מילון, או dictionary, או hash table:
  - זוגות של מפתחות keys וערכים values
  - מילון מוגדר על ידי סוגריים מסולסלים { }
  - לדוגמה: מילון שמכיל ציונים, המפתח הוא מספר ת.ז

הגדרה של מילון ריק

הוספת נתונים למילון

מפתח

ערך

```
>>> students_grades = {}
>>> students_grades['00001234'] = 85
>>> students_grades['00003579'] = 95
>>> students_grades['00002468'] = 65
>>> students_grades['00003579']
95
```

חיפוש ערך במילון לפי מפתח

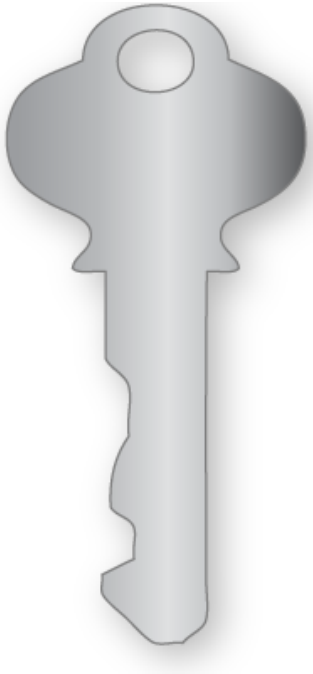
# הגדרה של מילון עם ערכים

- במקום להגדיר מילון ריק ולמלא, אפשר להגדיר מילון עם ערכים

```
>>> students_grades = {'00001234':85, '00003579':95, '00002468':65}
>>> students_grades
{'00003579': 95, '00002468': 65, '00001234': 85}
```

# get

- מה יקרה אם נחפש במילון מפתח שאינו קיים?



```
>>> students_grades['00009999']

Traceback (most recent call last):
  File "<pyshell#51>", line 1, in <module>
    students_grades['00009999']
KeyError: '00009999'
```

- פקודת get מאפשרת לנו לבצע חיפוש "בטוח"
  - אם המפתח אינו קיים יוחזר None
    - None: ערך חוקי, שאומר שמחזירים "כלום"

```
>>> students_grades.get('00001234')
85
>>> students_grades.get('00009999')
>>>
```

# in

- בדיקה אם מפתח נמצא במילון

```
>>> '00003579' in students_grades
True
>>> '00009999' in students_grades
False
```

# pop

- המתודה pop מקבלת מפתח, מוחקת מהמילון את המפתח ואת הערך שלו ומחזירה את הערך

```
>>> students_grades
{'00003579': 95, '00002468': 65, '00001234': 85}
>>> students_grades.pop('00001234')
85
>>> students_grades
{'00003579': 95, '00002468': 65}
```

# המתודות keys, values

- למילון מוגדרות מתודות keys, values
  - keys יוצרת רשימה של כל המפתחות
  - ו-values של כל הערכים

```
>>> students_grades = {'00001234':85, '00003579':95, '00002468':65}
>>> students_grades.keys()
['00003579', '00002468', '00001234']
>>> students_grades.values()
[95, 65, 85]
```

# Loop over a dictionary

- כדי לעבור על איברי מילון, בדומה לרשימה, נשתמש ב:
  - for
  - in
  - משתנה שישמש כ-iterator

```
>>> for key in students_grades:
        print 'Student ID: ', key, '    Student grade: ', students_grades[key]


Student ID:  00003579     Student grade:  95
Student ID:  00002468     Student grade:  65
Student ID:  00001234     Student grade:  85
>>>
```

# items

- המתודה items מכניסה לרשימה את המילון, כאשר כל צמד של מפתח+ערך הופכים ל-tuple ברשימה

```
>>> for (student_id, grade) in students_grades.items():
        print (student_id, grade)


('00003579', 95)
('00002468', 65)
('00001234', 85)
```

# תרגיל dictionary: קניות במכולת

- צרו מילון בשם prices. המפתח הוא שם המוצר והערך הוא מחיר המוצר. הוסיפו כ-5 מוצרים למילון.
- צרו מילון בשם shopping_cart. המפתח הוא שם המוצר והערך הוא כמות המוצרים בעגלת הקניות. הוסיפו מספר מוצרים.
- הכניסו למשתנה בשם total את סכום הקניות בעגלה.
  - הכניסו ל-shopping_cart מוצר שאינו קיים ב-prices ובידקו שהתוכנית אינה עפה על שגיאה, אלא מדפיסה הודעה מתאימה.

# תרגיל - Word Count

- www.cyber.org.il/python/wordcount.zip
- בתרגיל הבא נקרא את הקובץ alice.txt (אליס בארץ הפלאות) ונדפיס למסך את כמות המופעים של כל מילה בספר
- טיפים:
  - צרו מילון, כאשר המפתחות הם המילים והערכים הם כמות המופעים של כל מילה
  - השתמשו במתודה split כדי להפריד את הטקסט למילים

גבהים
תכנית הסייבר הלאומית

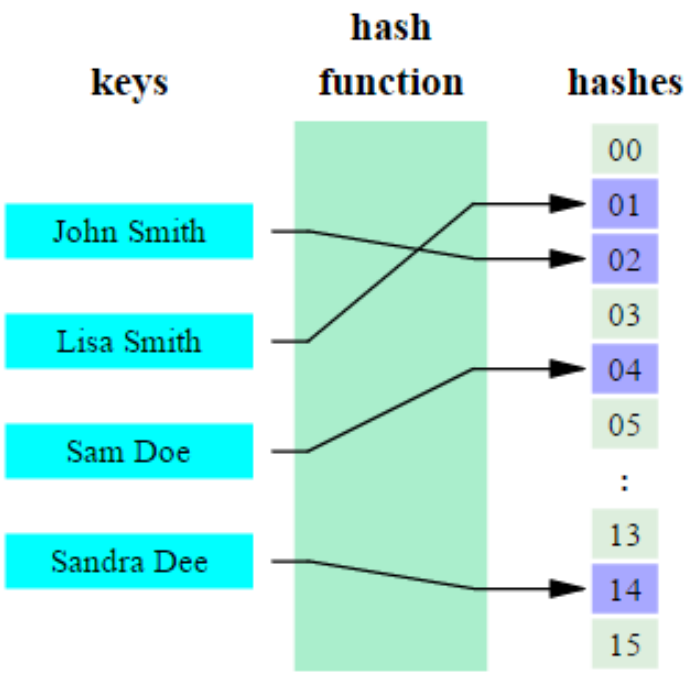
# הרחבה: פייתון, מתחת למכסה המנוע

- Hash
- Mutable
- Immutable

# איך עובד מילון?

- דרך אפשרית אחת למימוש מילון היא בתור רשימה
  - כדי למצוא מפתח ברשימה צריך לעבור על הרשימה מההתחלה עד שמוצאים
  - לכן, כשהמילון גדול החיפוש יקר

# Hash key




- כדי לייעל את החיפוש, כל מפתח עובר המרה על ידי פונקציה מתמטית (hash function), שהופכת אותו למספר, תחת התנאים הבאים:
  - אפשרי ששני מפתחות שונים יומרו לאותו מספר
    - רצוי שכל מפתח יומר למספר שונה
    - מקובל שאם ה-hash כבר קיים, מבוצע re-hash, כדי שלכל מפתח יהיה מספר שונה
  - בלתי אפשרי ששני מפתחות זהים יומרו למספרים שונים

# מילון- תהליך החיפוש

- כאשר מחפשים מפתח במילון, צריך לחפש אותו רק בין המפתחות שיש להם אותו hash
  - אם ה-hash טוב, אז יש רק מפתח יחיד כזה

# סוגי מפתחות

- ראינו שכל מילון בנוי מסט של מפתחות וערכים
- שאלה למחשבה: האם כל אובייקט יכול לשמש כמפתח?
  - רמז: נבחן כיצד משפיע שינוי של משתנים מסוגים שונים על המיקום שלהם בזיכרון

# אובייקט מסוג integer

- כאשר משנים את ערכו של אובייקט מסוג integer-

```
>>> my_int=111
>>> id(my_int)
4959424
>>> my_int=my_int+222
>>> id(my_int)
36964676
```

- נוצר אובייקט חדש מסוג integer, במיקום חדש בזיכרון
- אובייקט כזה נקרא **immutable**- בלתי ניתן לשינוי
  - אי אפשר לשנות, שינוי גורם ליצירת אובייקט חדש

# אובייקט מסוג string

- כאשר משנים את ערכו של אובייקט מסוג string-

```
>>> my_string='aaaa'
>>> id(my_string)
44600096
>>> my_string=my_string+'bbbb'
>>> id(my_string)
44597664
```

- נוצר אובייקט חדש מסוג string, במיקום חדש בזיכרון
- לכן גם string הוא immutable- בלתי ניתן לשינוי

# אובייקט מסוג tuple

- כאשר משנים את ערכו של אובייקט מסוג tuple-

```
>>> my_tuple=('x',1)
>>> id(my_tuple)
44021400
>>> my_tuple[0]='y'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

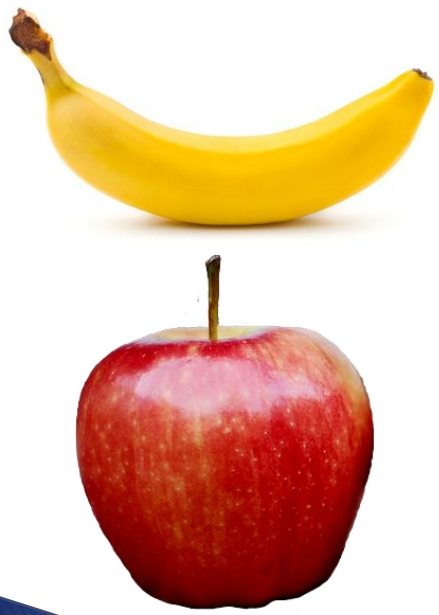
- ובכן, זה פשוט בלתי אפשרי... ☺
- לכן גם tuple הוא immutable

# אובייקט מסוג list

- כאשר משנים את ערכו של אובייקט מסוג list-

```
>>> my_list=[1,1,1]
>>> id(my_list)
44620776
>>> my_list[0]=2
>>> id(my_list)
44620776
```

- לא נוצר אובייקט חדש. ערכו של ה-list משתנה.
- לכן list הוא **mutable**- ניתן לשינוי

# מדוע mutable אינו יכול לשמש בתור key?

- משום שאין לו hash
  - אפשר ליצור hash, אבל הוא לא טוב בתור מפתח.
- מדוע? נניח לשלילה, שהיינו יכולים ליצור מילון שהמפתח בו הוא מסוג list:
  - יצרנו רשימה a=['apple'] ורשימה b=['banana']
  - כעת היינו מגדירים מילון fruits={a:1, b:2}
  - המקום במילון בו נשמרת כל רשימה תלוי בערך שלה
  - בשלב הבא היינו מגדירים b[0]='apple'
  - כעת אם היינו מחפשים את b במילון, הוא לא היה נמצא במקום בו חיפשנו אותו
  - כיוון שבחרנו עבור b ערך זהה לשל a, היינו מקבלים את הערך הצמוד למפתח של a