תוכן עניינים פרק 6 – טיפוסי נתונים מורכבים

פרק 6.1 עבודה עם רשימות - Working with Lists

פרק 6.2 Tuples

פרק 6.3 מילונים Dictionaries

בפרק 6 בתוכנית הלימודים להנדסאים של משרד החינוך במקצוע שפת פייתון רשום:

פרק 6 : טיפוסי נתונים מורכבים - Python collections

יעדים

היכרות עם הפשטת מידע מתקדמת בעזרת טיפוסי נתונים ייחודיים לשפה.

תכנים

1. עבודה עם רשימות – List
2. עבודה עם צירופים – Tuples
3. עבודה עם מילונים – Dictionary

סיכום שעות ההוראה : 10 שעות עיוניות ו- 4 שעות התנסותיות. סה"כ 14 שעות.

6.1 עבודה עם רשימות - Working with Lists

6.1.1 כללי

רשימה – list – משמשת לאחסון מספר פריטים במשתנה יחיד.

הערה : בהמשך הפרק נקרא לפעמים לפריט גם בשם אלמנט או איבר או רכיב.

רשימות הן אחד מתוך 4 סוגי נתונים מובנים ב- Python המשמשים לאחסון אוספי נתונים, 3 האחרים הם Tuple, Set ו Dictionary , שהם בעלי תכונות ושימוש שונים.

רשימות נוצרות באמצעות סוגריים מרובעים: לדוגמה :

```
myColors = ["red", "blue", "green"]
```

אם נבקש הדפסה print(myColors) נקבל : ['red', 'blue', 'green']

6.1.2 הפריטים ברשימה – List Items

פריטי רשימה מסודרים, ניתנים לשינוי ומאפשרים ערכים כפולים.

פריטי רשימה כלולים באינדקס, לפריט הראשון יש אינדקס [0],לפריט השני יש אינדקס [1] וכו'.

6.1.3 סדר הרשימה

כאשר אנו אומרים רשימות מסודרות , זה אומר כי לפריטים יש סדר מוגדר, וכי הסדר לא ישתנה.

אם נוסיף פריטים חדשים לרשימה , הפריטים החדשים ימוקמו בסוף הרשימה.

קיימות מספר מתודות שמשנות את הסדר של הרשימה אבל באופן כללי סדר הפריטים ברשימות לא ישתנה.

6.1.4 שינויים ברשימה

הרשימה ניתנת לשינוי, כלומר אנו יכולים לשנות, להוסיף ולהסיר פריטים ברשימה לאחר יצירתה.

6.1.5 כפילויות – Duplicates

מאחר שמיקום הפריטים ברשימות נמצא לפי אינדקס, רשימות יכולות לכלול פריטים בעלי ערך זהה (חברים כפולים).
לדוגמה :

```
myColors = ["red", "blue", "green", "red", "green"]
```

אם נבקש הדפסה : print(myColors) נקבל : ['red', 'blue', 'green', 'red', 'green']

6.1.6 אורך רשימה

כדי לקבוע כמה פריטים יש לרשימה, נשתמש בפונקציה len() . לדוגמה :

```
myColors = ["red", "blue", "green", "red", "green"]

print(len(myColors))
```

נקבל : 5

6.1.7 הפריטים ברשימה – טיפוסי הנתונים

פריטי הרשימה יכולים להיות מכל סוג נתונים. לדוגמה נראה 3 רשימות מטיפוס מחרוזת, שלם ובוליאני.

```
list1 = ["red", "blue", "green"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

אם נבקש הדפסות :

```
print(list1)
print(list2)
print(list3)
```

נקבל :

```
['red', 'blue', 'green']
[1, 5, 7, 9, 3]
[True, False, True]
```

רשימה יכולה להכיל טיפוסי נתונים שונים. לדוגמה רשימה עם טיפוסי מחרוזת, מספרים שלמים וערכים בוליאניים.

```
list4 = ["red", 29, True, -40, "male"]
```

6.1.8 שימוש ב ( )type

מנקודת המבט של פייתון, רשימות מוגדרות כאובייקטים עם סוג הנתונים 'list' . לדוגמה :

```
list4 = ["red", 29, True, -40, "male"]
        print(type(list4))
```

נקבל : < class 'list' >

6.1.9 הבנאי list() - list constructor

ניתן גם להשתמש בבנאי ( )list בעת יצירת רשימה חדשה. לדוגמה :

```
list4 = list (("red", 29, True, -40, "male"))        # פעמיים סוגריים קטנים
```

אם נרצה לקבל הדפסה נרשום : print(list4)

ונקבל : ['red', 29, True, -40, 'male']

6.1.10 אוספים (מערכים) - collections

קיימים ארבעה סוגי נתונים של אוסף בשפת פייתון והם :

*. **list** - הוא אוסף מסודר וניתן לשינוי. יש אפשרות לחברים כפולים.

*. **tuple** הוא אוסף אשר מסודר ובלתי ניתן לשינוי. יש אפשרות לחברים כפולים.

*. **set** הוא אוסף שאינו מסודר וללא אינדקס. אין חברים כפולים.

*. **dictionary** הוא אוסף אשר מסודר וניתן לשינוי. אין חברים כפולים.

במילה מסודר הכוונה שלכל פריט יש את האינדקס שלו.

החל מגרסה 3.7 של פייתון dictionaries הם מסודרים . בפייתון 3.6 וגרסאות קודמות הם לא היו מסודרים.

כאשר בוחרים טיפוס של אוסף כדאי להבין את המאפיינים של טיפוס זה. בחירת הטיפוס הנכון עבור ערכת נתונים מסוימת עשויה להיות שמירה על המשמעות של הפריטים, וזה יכול לגרום לעלייה ביעילות או באבטחה.

6.1.11 גישה לפריטים ברשימה - Access List Items

6.1.12 גישה בעזרת אינדקס

לכל פריט ברשימה יש אינדקס וניגשים אליו על ידי הפנייה למספר האינדקס שלו. הפריט הראשון הוא באינדקס [0] .

דוגמה: הדפסת פריט כלשהו מהרשימה.

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
```

הדפסת הפריט ה 5 ברשימה : `print (colors[5])`

ההדפסה שנקבל היא : white .

6.1.13 אינדקס שלילי

אינדקס שלילי פירושו להתחיל את הספירה מהפריט האחרון . הפריט האחרון הוא באינדקס [-1] זה שלפניו באינדקס [-2] וכך הלאה . בדוגמה של הרשימה colors הצבע navy נמצא באינדקס [-1] , הצבע white באינדקס [-2] וכך הלאה עד הצבע red שנמצא ב [-7] .

דוגמה : נדפיס את האיבר במיקום 2- ברשימה colors :

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
print(colors[-2])
```

ההדפסה שנקבל היא : white .

6.1.14 טווח של אינדקסים

יש אפשרות לציין טווח אינדקסים על-ידי ציון היכן להתחיל והיכן לסיים את הטווח. בעת ציון הטווח, הערך המוחזר יהיה רשימה חדשה עם הפריטים שצוינו. הטווח נרשם כך : [1:5] ואז מדובר על הפריטים 1 עד 5 (אבל לא כולל את הפריט 5 !).

ניתן לרשום את התחום [:5] ללא ציון הפריט הראשון ואז הכוונה מהפריט באינדקס [0] ועד האינדקס [5] לא כולל הפריט באינדקס 5 !!

ניתן לרשום גם : [2:] ללא ציון הפריט הסופי ואז התחום הוא מהפריט במיקום [2] ועד סוף הרשימה .

דוגמאות

דוגמה 1: יש להדפיס את הפריטים מ 1 עד 5 ברשימה colors .

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
print(colors[1:5])
```

ההדפסה שנקבל היא : ['blue', 'green', 'magenta', 'black']

אם נבקש את ההדפסה הבאה :

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
print(colors[-1:-5])
```

נקבל רשימה ריקה [ ] .

לעומת זאת אם נבקש :

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
print(colors[-5:-1])
```

נקבל את ההדפסה מהפריט ב [-5] שהוא green ועד הפריט [-2] שהוא white (לא מקבלים את [-1] ! :

ההדפסה שנקבל היא : ['green', 'magenta', 'black', 'white']

דוגמה 2: הדפסת הרשימה מהפריט 0 ועד 4 לא כולל הפריט הרביעי .

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
print(colors[:4])
```

ההדפסה שנקבל היא :

```
['red', 'blue', 'green', 'magenta']
```

דוגמה 3: הדפסה מפריט במיקום 2 עד סוף הרשימה.

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
print(colors[2:])
```

ההדפסה שנקבל היא :

```
[ 'green', 'magenta', 'black', 'white', 'navy']
```

6.1.15 בדיקה האם פריט נמצא ברשימה

אם רוצים לבדוק האם פריט מסוים נמצא ברשימה משתמשים במילת המפתח **in** .

דוגמה 1: נבדוק האם הפריט blue נמצא ברשימה colors

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
color="blue"
if color in colors :
  print (color + " is in the list colors")
else :
  print (color + " is not in the list colors")
```

ההדפסה שנקבל : blue is in the list colors

דוגמה 2: נבדוק האם הפריט Blue נמצא ברשימה colors (שמנו B במקום b)

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
```

```
color="blue"
if color in colors :
  print (color + " is in the list colors")
else :
  print (color + " is not in the list colors")
```

ההדפסה שנקבל : blue is not in the list colors

6.1.16 שינוי ערך של פריטים

6.1.17 שינוי ערך של פריט

כדי לשנות את הערך של פריט מסוים, מתייחסים למספר האינדקס שלו ברשימה:

דוגמה :

```
myColors = ["red", "blue", "green"]
myColors[2]="brown"
print(myColors)
```

ההדפסה שנקבל היא : ['red', 'blue', 'brown']

6.1.18 שינוי ערכים של מספר פריטים בתחום רצוי.

כדי לשנות את הערך של פריטים בטווח מסוים מגדירים טווח של מספרי אינדקס ורשימה עם הערכים החדשים שייכנסו לטווח מספרי האינדקסים שרוצים להוסיף בו את הערכים החדשים.

דוגמה : נשנה את האיברים 2 עד 5 (לא כולל הפריט 5) ברשימה colors .

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
colors[2:5] = ["orange", "yellow", "purple"]
print(colors)
```

ההדפסה שנקבל היא : ['red', 'blue', 'orange', 'yellow', 'purple', 'white', 'navy'] .

6.1.18.1 אם נוסיף יותר פריטים ממה שביקשנו להחליף, הפריטים החדשים יתווספו למקום שציינו והפריטים שבהמשך הרשימה יעברו בהתאם. כמובן , שבמקרה כזה כמות הפריטים ברשימה תגדל.

דוגמה : נבקש לשנות את הערכים של הפריטים 2 עד 5 (לא כולל 5) – סה"כ להחליף 3 פריטים - אבל נרשום 5 צבעים במקום 3 . כל 5 הצבעים ייכנסו לרשימה, האיברים שביקשנו להחליף – יוחלפו ושאר האיברים יזוזו ויהיו אחרי הצבעים שהוספנו.

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
print(len(colors))                                    # להדפיס את כמות הפריטים ברשימה
colors[2:5] = ["orange", "yellow", "purple", "pink", "pale blue"]
print(colors)                                         # להדפיס את כמות הפריטים ברשימה החדשה
print(len(colors))
```

ההדפסות שנקבל :

```
7
['red', 'blue', 'orange', 'yellow', 'purple', 'pink', 'pale blue', 'white', 'navy']
9
```

6.1.18.2 אם נוסיף פחות פריטים ממה שביקשנו להחליף , הפריטים החדשים יתווספו למקום שציינו והפריטים הנותרים יעברו בהתאם.

דוגמה : נבקש לשנות ערכים של 4 פריטים אבל נרשום רק 2 ערכים חדשים. 2 הערכים החדשים ייכנסו במיקום שביקשנו, 2 הפריטים הבאים ברשימה יימחקו מהרשימה ואורך הרשימה יקטן ב 2 .

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
print(len(colors))
colors[2:6] = ["orange", "yellow"]
print(colors)
print(len(colors))
```

ההדפסה שנקבל היא :

```
7
['red', 'blue', 'orange', 'yellow', 'navy']
5
```

6.1.19 הוספה של פריטים

6.1.20 הוספת פריט – שימוש במתודה ( )insert

כדי להוסיף פריט רשימה חדש מבלי להחליף אף אחד מהערכים הקיימים, באפשרותנו להשתמש במתודה **( )insert** .

המתודה **( )insert** מוסיפה פריט באינדקס שצוין.

דוגמה : הכנסת פריט נוסף לרשימת ה colors במיקום [3] (הפריט הרביעי). כמובן שאורך הרשימה גדל ב 1 .

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
```

```
colors.insert(3, "bourdeaux")                    # הוספת הצבע בורדו באינדקס 3 - פריט רביעי
print(colors)
```

ההדפסה שנקבל : ['red', 'blue', 'green', 'bourdeaux', 'magenta', 'black', 'white', 'navy']

6.1.21 צירוף של פריט בסוף הרשימה עם המתודה append()

כדי להוסיף פריט לסוף הרשימה, משתמשים במתודה **append( )** :

דוגמה: נוסיף את הצבע כתום orange בסוף רשימת הצבעים colors .

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
colors.append("orange")
print(colors)
```

ההדפסה שנקבל היא : ['red', 'blue', 'green', 'magenta', 'black', 'white', 'navy', 'orange']

6.1.22 הרחבת רשימה עם המתודה extend()

כדי לצרף רכיבים מרשימה אחרת לרשימה הנוכחית נשתמש במתודה **extend( )** . הרכיבים החדשים מצורפים בסוף הרשימה.

דוגמה: נגדיר 2 רשימות. בשלב ראשון נצרף את הרשימה השנייה בסוף הרשימה הראשונה. בשלב הבא נצרף את הרשימה הראשונה (המורכבת מהרשימה הראשונה ועוד הרשימה השנייה) בסוף הרשימה השנייה. התוכנית נראית כך:

```
colors1 = ["red", "blue", "green"]
colors2 = ["magenta", "black",  "white", "navy"]
colors1.extend(colors2)
print("New colors1 =  " , colors1)
colors2.extend(colors1)
print("New colors2 =  " ,colors2)
```

ההדפסות שנקבל הן :

```
New colors1 =   ['red', 'blue', 'green', 'magenta', 'black', 'white', 'navy']
New colors2 =   ['magenta', 'black', 'white', 'navy', 'red', 'blue', 'green', 'magenta', 'black', 'white', 'navy']
```

6.1.23 הוספה של כל משתנה/נתון/אוסף אל רשימה עם המתודה extend( )

המתודה extend( ) לא חייבת לצרף רשימות . ניתן לצרף גם tuples, sets, dictionaries ועוד.

דוגמה : הוספת tuple לרשימה :

```
colors1 = ["red", "blue", "green"]
colors2 = ("magenta", "black",  "white", "navy")          # colors2 is a tuple and not a list
colors1.extend(colors2)
```

```
print("New colors1 =  " , colors1)
```

ההדפסה שנקבל היא : New colors1 = ['red', 'blue', 'green', 'magenta', 'black', 'white', 'navy']

6.1.24 הסרת פריטים מרשימה

6.1.25 הסרה של פריט מסוים עם המתודה ( )remove

המתודה **( )remove** מסירה פריט מסוים שנבחר.

דוגמה : נסיר פריט מסוים מתוך רשימה.

```
colors1 = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
colors1.remove("red")
print("colors1 =  " , colors1)
```

ההדפסה שנקבל : colors1 = ['blue', 'green', 'magenta', 'black', 'white', 'navy']

אם נרשום עכשיו :

```
colors1.remove("magenta")
print("colors1 =  " , colors1)
```

נקבל : colors1 = ['blue', 'green', 'black', 'white', 'navy']

6.1.26 הסרה של אינדקס מסוים עם המתודה ( )pop

המתודה ( )pop מסירה פריט מסוים על פי האינדקס שנרשום.

דוגמה : הסרה של הפריט השני (שהאינדקס שלו 1) .

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
colors.pop(1)
print(colors)
```

נקבל : ['red', 'green', 'magenta', 'black', 'white', 'navy']

אם לא נציין מספר אינדקס אז המתודה ( )pop תסיר את הפריט האחרון .

דוגמה :

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
```

colors.pop()

print(colors)

נקבל : ['red', 'blue', 'green', 'magenta', 'black', 'white']

6.1.27 מחיקה של פריט ברשימה עם המילה del

גם מילת המפתח del מסירה פריט לפי האינדקס שנרשום.

דוגמה : הסרה של האיבר השלישי (האיבר באינדקס 2) :

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
del colors[2]
print(colors)
```

נקבל : ['red', 'blue', 'magenta', 'black', 'white', 'navy']

6.1.28 מחיקה של רשימה עם המילה del( )

המתודה **del( )** מוחקת את הרשימה עצמה (ולא רק את הפריטים שבה).

דוגמה :

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
del colors
print(colors)
```

נקבל :

```
Traceback (most recent call last):
  File "./prog.py", line 3, in <module>
NameError: name 'colors' is not defined
```

קיבלנו הודעת שגיאה בשורה 3 שהמילה colors לא הוגדרה. ואכן , בשורה השנייה , מחקנו את הרשימה colors ובשורה השלישית מבקשים הדפסה של הרשימה שנמחקה וכבר לא קיימת ולכן קיבלנו הודעת שגיאה.

6.1.29 ניקוי של כל הרשימה עם המתודה clear

המתודה clear( ) מוחקת את כל הפריטים ברשימה (אבל לא את הרשימה עצמה שנשארת ריקה).

דוגמה :

```
colors = ["red", "blue", "green", "magenta", "black",  "white", "navy"]
colors.clear()
print(colors)
```

נקבל : עכשיו לא מקבלים הודעת שגיאה כמו במילה del אלא רשימה ריקה : [ ] .

6.1.30 לולאות בתוך רשימה

6.1.31 לולאת for בתוך רשימה

אפשר לעבור בלולאה בין הפריטים של רשימה באמצעות לולאת **for** .

דוגמה :

```
colors = ["red", "blue", "green", "black",  "white"]
for x in colors :
  print(x)
```

ההדפסה שנקבל:

```
red
blue
green
black
white
```

6.1.32 לולאת for בתוך רשימה עם אינדקס

אפשר גם לעבור בלולאת for בין פריטי הרשימה על-ידי הפניה למספר האינדקס שלהם. לשם כך ניעזר בפונקציות **range( )** ו **len( )**

דוגמה 1: הדפסת כל הפריטים על ידי התייחסות למספר האינדקס שלהם.

```
colors = ["red", "blue", "green", "black",  "white"]
for x in range(len(colors)):
   print(colors[x])
```

ונקבל את כל הפריטם מאינדקס 0 ועד האינדקס שהוא אורך הרשימה (האורך הוא 5 פריטים) אבל ההדפסה לא כוללת 5 אלא רק את הפריטים 0,1,2,3,4 :

```
red
blue
green
black
white
```

דוגמה 2 : הדפסת אורך הרשימה – כמות הפריטים ברשימה - שימוש בפונקציה **len( )** .

```
colors = ["red", "blue", "green", "black",  "white"]
print(len(colors) )
```

נקבל את המספר : 5

על לולאות **for** נרחיב בפרק על לולאות בפייתון.

6.1.33 לולאת while בתוך רשימה

דוגמה 1 : הדפסה של כל הפריטים של רשימה :

אפשר לעבור בלולאה בין הפריטים של רשימה באמצעות לולאת **while** .לשם כך ניעזר בפונקציה **len( )** שבדוגמה הקודמת כדי לקבוע את אורך הרשימה . נתחיל מאינדקס 0 לעבור על כל הפריטים ברשימה בעזרת האינדקסים שלהם. יש לזכור להגדיל ב 1 את האינדקס בסיום כל איטרציה (לולאה של פקודות תוכנה) . בלולאת for ההגדלה מתבצעת אוטומטית !

```
colors = ["red", "blue", "green", "black",  "white"]
x=0
while x <  len(colors) :
  print(colors[x])
  x=x+1                  # x  את 1 לא לשכוח להגדיל ב
```

וההדפסה שנקבל :

```
red
blue
green
black
white
```

שאלה : מה קורה אם שוכחים לרשום את השורה : x=x+1 ?

תשובה : התוכנית תרוץ בלולאה אין סופית ובמסך יודפס כל הזמן red

על לולאות **while** נרחיב בפרק על לולאות בפייתון.

6.1.34 לולאות עם תחביר מקוצר - Looping Using List Comprehension

הבנה מעמיקה של רשימה מאפשרת לרשום פקודות עם תחביר קצר ביותר עבור ביצוע של לולאה בתוך רשימות.

6.1.35 דוגמה : פקודה מקוצרת להדפסת כל הפריטים ברשימה :

```
colors = ["red", "blue", "green", "black",  "white"]
[print (x) for x in colors]
```

הפקודה השנייה כוללת את 2 הפקודות בלולאת ה for שבפסקה קודמת :

```
for x in colors :
```

```
print(x)
```

6.1.36 : העתקה של פריטים מרשימה אחת שיש בהם את התו 'e' לרשימה חדשה והדפסתה

נראה 2 דרכים . האחת בדרך "הרגילה" והשנייה עם משפט מקוצר. כמובן שהרשימה המקורית איננה משתנה.

בדרך הרגילה : ללא הבנה של רשימה נכתוב משפט אם תנאי :

```
colors = ["red", "blue", "green", "black",  "white"]
newColors = [ ]
for x in colors:
  if "e" in x:
    newColors.append(x)
print(newColors)
```

ההדפסה שנקבל : ['red', 'blue', 'green', 'white']

בדרך המקוצרת :

```
colors = ["red", "blue", "green", "black",  "white"]
newColors = [x for x in colors if "e" in x]
print(newColors)
```

ונקבל : ['red', 'blue', 'green', 'white']

ניעזר בדוגמה הקודמת כדי להבין את התחביר במשפט מקוצר :

```
newColors = [x for x in colors if "e" in x]
```

באופן כללי המשפט המקוצר בנוי כך :

[**שם הרשימה החדשה** = [expression **for** item **in** iterable **if** תנאי == True]

הערך המוחזר הוא רשימה חדשה והרשימה הישנה נשארת ללא שינוי.

התנאי דומה למסנן המקבל רק את הפריטים שמעריכים ל- True. אפשר להזניח את התנאי ואז זו ההעתקה של כל הרשימה.

iterable יכול להיות כל אובייקט שניתן לעבור עליו באיטרציה - בלולאה - כמו רשימה , tuple , set וכו' .

במקום **שם הרשימה החדשה** ניתן לרשום print (…) ואז נקבל הדפסה ללא יצירה של רשימה חדשה .

Expression - ביטוי - הוא הפריט הנוכחי באיטרציה אך הוא גם התוצאה שאפשר לטפל בה לפני שהיא מסתיימת כפריט רשימה ברשימה החדשה .

6.1.37 העתקה והדפסה של כל הפריטים שבהם לא מופיע התו 'e'

```
colors = ["red", "blue", "green", "black",  "white"]
newColors = [x for x in colors if "e" not in x]
print(newColors)
```

ההדפסה שנקבל : ['black']

6.1.38 הדפסת הפריטים שבהם מופיעים התווים "re" :

```
colors = ["red", "blue", "green", "black",  "white"]
print ([x for x in colors if "re" in x])
```

ההדפסה שנקבל : ['red', 'green']

6.1.39 העתקת כל הפריטים עם הציון 100 והפריטים שבהם הערך קטן מ 55 (כמות ציונים שליליים)

```
marks = [90,70,54,55,67,89,100,94,50,48,67,100,49,100,65]
marks100 = [x for x in marks if x == 100 ]
negativeMarks = [x for x in marks if x < 55 ]
print ("marks100 =  ", marks100, "and" , len(marks100) , "students has this mark")
print("negativeMarks = ", negativeMarks , "and" , len(negativeMarks) , "students has negative marks" )
```

ההדפסות שנקבל :

```
marks100 =   [100, 100, 100] and 3 students has this mark
negativeMarks =  [54, 50, 48, 49] and 4 students has negative marks
```

6.1.40 שימוש בפונקציה range (טווח)

אפשר להשתמש בפונקציה range( ) כדי ליצור אובייקט שניתן לעבור עליו בלולאה.

דוגמה : נדפיס את כל הפריטים ברשימה colors בטווח מפריט 0 ועד פריט 3 (לא כולל 3).

```
colors = ["red", "blue", "green", "black",  "white"]
print ([colors[x] for x in range(3)])
```

ההדפסה שנקבל : ['red', 'blue', 'green']

6.1.41 הדפסת הפריטים בטווח רצוי עם תנאי

ניתן להדפיס את הפריטים בטווח רצוי עם תנאי.

דוגמה : נדפיס את כל הציונים החיוביים של 10 הסטודנטים הראשונים .

```
marks = [90,70,54,55,67,89,100,94,50,48,67,100,49,100,65]
print ([marks[x] for x in range(10) if marks[x] >= 55])
```

ההדפסה שנקבל : [90, 70, 55, 67, 89, 100, 94]

6.1.42 הדפסת הפריטים ברשימה עם תווים בדפוס (אותיות "גדולות")

```
colors = ["red", "blue", "green", "black",  "white"]
print ([x.upper() for x in colors])                    # המתודה  upper הופכת כל תו "קטן" לתו "גדול"
```

ההדפסה שנקבל : ['RED', 'BLUE', 'GREEN', 'BLACK', 'WHITE']

6.1.43 שינוי הערך של כל הפריטים ברשימה ל nice

```
 colors = ["red", "blue", "green", "black",  "white"]
newColors = ['nice' for x  in colors]
print(newColors)
```

ההדפסה שנקבל : ['nice', 'nice', 'nice', 'nice', 'nice']

6.1.44 שינוי הערך של פריט רצוי ברשימה לפי תנאי

ניתן לשנות את הערך של פריט מסוים מתוך רשימה.

דוגמה : שינוי הערך "green" ברשימה ל "red"

```
colors = ["red", "blue", "green", "black",  "white"]
newColors = [x if x != "green" else "red" for x in colors1]
print(newColors)
```

בשורה השנייה רשמנו להשאיר את הפריט כמו שהוא אם הוא לא "green" ואם הוא "green" ואז יש לשנות אותו ל "red" .

ההדפסה שנקבל: ['red', 'blue', 'red', 'black', 'white']

אם יש מספר פריטים בשם "green" אז כולם ישתנו ל "red" כמו בדוגמה הבאה :

```
colors1 = ["red", "blue", "green", "black",  "white", "green", "navy"]
newColors = [x if x != "green" else "red" for x in colors1]
print(newColors)
```

נקבל : ['red', 'blue', 'red', 'black', 'white', 'red', 'navy']

6.1.45 מיון רשימות Sort lists

לאובייקטים של רשימה יש מתודה הנקראת ( )sort שתבצע מיון של הרשימה לפי קריטריון שנבקש.

6.1.46 מיון אלפאנומרי של רשימה בסדר עולה - ascending sort

במילה אלפאנומרי הכוונה לתווים מ a ועד z , לשאר תווי הבקרה כמו # או $ וכמו כן הספרות 0 עד 9 . לכל תו יש ערך אסקי (או Unicode משלו) כאשר התו a (ערך בינארי 0x61) והתו A (ערך בינארי 0x41) אינם זהים.
המתודה ( )sort היא case sensitive והמיון בסדר עולה מתבצע התווים הגדולים - Capital letters - לפני הקטנים – lower case.

דוגמה : מיון אלפאנומרי של רשימה בסדר עולה .

```
colors = ["red", "blue", "green", "black",  "white"]
colors.sort( )
print(colors)
```

בשורה השנייה רשמנו colors.sort( ) ובתוך הסוגריים לא רשמנו כלום. ברירת המחדל היא סדר עולה.

נקבל : ['black', 'blue', 'green', 'red', 'white']

דוגמה נוספת עם מספרים:

```
numbers = [100, 55, -20, -40,  1000, 37]
numbers.sort( )
print(numbers)
```

ונקבל : [-40, -20, 37, 55, 100, 1000]

אפשר גם למיין רשימה בעברית רק צריך לזכור שבהדפסה

```
colors = ["שחור", "ירוק", "כחול", "אדום",  "white"]
colors.sort( )
[print (x) for x in colors]
```

ונקבל :

```
white
אדום
ירוק
כחול
שחור
```

אם נבקש בשורה השלישית הדפסה על ידי print(colors) נקבל : ['אדום', 'ירוק', 'כחול', 'שחור', 'white'] .

המיון מתבצע נכון אבל בגלל ההדפסה בעברית נרשם קודם שחור ובסוף אדום.

6.1.47 מיון אלפאנומרי של רשימה בסדר יורד - descending sort

כדי לבצע מיון בסדר יורד נשתמש בארגומנט reverse = True .

דוגמה : מיון יורד של 2 רשימות : רשימה של מחרוזות ורשימה נוספת של מספרים

```
colors = ["red", "blue", "green", "black",  "white"]
numbers = [100, 55, -20, -40,  1000, 37]
colors.sort(reverse = True)
numbers.sort(reverse = True)
print (colors)
print(numbers)
```

ההדפסה שנקבל :

```
['white', 'red', 'green', 'blue', 'black']
[1000, 100, 55, 37, -20, -40]
```

6.1.48 מיון בהתאמה אישית customize sort function

ניתן להתאים אישית את הפונקציה שלך באמצעות הארגומנט key = function . הפונקציה תחזיר מספר שישמש למיון הרשימה (המספר הנמוך ביותר תחילה).

ניתן להשתמש בפונקציות מובנות כפונקציות מפתח בעת מיון רשימה.

דוגמה : מיון רשימה בהתייחס לכמה קרוב המספר ל 100 :

במיון הרשימה נקרא לפונקציה abs שתחזיר את הערך המוחלט של ההפרש בין מספר ששולחים לרשימה והערך 100 .
ההפרש ישמש למיון הרשימה (המספר הנמוך ביותר תחילה):

```
def func1(n):    # הגדרת פונקציה המחזירה את הערך המוחלט של ההפרש בין המספר שהיא מקבלת פחות 100
   return abs(n - 100)

numbers = [100, 55, -20, -40,  1000, 37]
numbers.sort(key = func1)
print(numbers)
```

ההדפסה שנקבל : [100, 55, 37, -20, -40, 1000]

6.1.49 מיון ללא התחשבות ב case sensitive (לא משנה האם התו באותיות קטנות או גדולות)

אם רוצים לבצע מיון ללא התחשבות האם התו באותיות קטנות או גדולות נשתמש ב- str.lower כפונקציית מפתח ואז אין התחשבות באם התו גדול או קטן.

דוגמה : ביצוע מיון ללא התחשבות ב case sensitive

```
newColors = ["Red", "blue", "green", "Black",  "white"]
newColors.sort()
print("newColors after normal sort : " , newColors)
newColors = ["Red", "blue", "green", "Black",  "white"]
newColors.sort(key = str.lower)
print("newColors without case sensitive sort : " , newColors)
```

ההדפסה שנקבל :

```
newColors after normal sort :  ['Black', 'Red', 'blue', 'green', 'white']
newColors without case sensitive sort :  ['Black', 'blue', 'green', 'Red', 'white']
```

דוגמה נוספת : שימשו במילת המפתח key=str.upper

```
newColors = ["Red", "blue", "green", "Black",  "white"]
newColors.sort()
print("newColors after normal sort : " , newColors)
newColors = ["Red", "blue", "green", "Black",  "white"]
newColors.sort(key = str.upper)
print("newColors without case sensitive sort : " , newColors)
```

ההדפסה שנקבל :

```
newColors after normal sort :  ['Black', 'Red', 'blue', 'green', 'white']
newColors without case sensitive sort :  ['Black', 'blue', 'green', 'Red', 'white']
```

6.1.50 מיון הפוך reverse order

ניתן להפוך את הסדר של רשימה כך שהפריט הראשון יהיה האחרון והאחרון יהיה הראשון ובהתאמה שאר הפריטים ללא קשר לאלפבית נשתמש במתודה ( )reverse.

דוגמה: נהפוך את סדר הפריטים ברשימה colors

```
colors = ["red", "blue", "green", "black",  "white"]
```

```
colors.reverse()
print(colors)
```

ההדפסה שנקבל : ['white', 'black', 'green', 'blue', 'red']

6.1.52 העתקה של רשימות

לא ניתן להעתיק רשימה אחת לאחרת על ידי הפקודה : list2=list1 . הסיבה לכך היא שהכתובות בזיכרון של list1 יהיה גם אותן הכתובות של list2 ושינויים שנעשה באחת הרשימות יתבצעו גם ברשימה השנייה כי מדובר על אותן כתובות בזיכרון.

אחת הדרכים להעתיק רשימה אחת לאחרת היא בעזרת המתודה **copy( )** .

6.1.52 : דוגמה להעתקת רשימה עם המתודה copy( )

```
colors1 = ["red", "green", "blue"]
colors2 = colors1.copy()
print(colors2)
```

ההדפסה שנקבל : ['red', 'green', 'blue']

6.1.53 : דוגמה להעתקת רשימה עם המתודה list( )

```
colors1 = ["red", "green", "blue"]
colors2 = list(colors1)
print(colors2)
```

ההדפסה שנקבל גם כאן היא : ['red', 'green', 'blue']

6.1.70 צירוף של רשימות

ישנן מספר דרכים לצרף או לשרשר שתי רשימות או יותר בפייתון. אחת הדרכים הקלות ביותר היא באמצעות האופרטור + .

6.1.71 צירוף רשימה בעזרת האופרטור +

```
colors1 = ["red", "green", "blue"]
colors2 = ["magenta", "white", "black", "orange"]
colors3 = colors1+colors2
print(colors3)
```

ההדפסה שנקבל : ['red', 'green', 'blue', 'magenta', 'white', 'black', 'orange']

המחרוזת colors2 שורשרה בסיום color1 .

אם נשנה את סדר השרשור ונרשום את התוכנית :

```
colors1 = ["red", "green", "blue"]
colors2 = ["magenta", "white", "black", "orange"]
colors3 = colors2+colors1
print(colors3)
```

ההדפסה שנקבל : ['magenta', 'white', 'black', 'orange', 'red', 'green', 'blue']

המחרוזת color1 שורשרה בסיום color2 .

6.1.56 צירוף רשימה בעזרת צירוף הפריטים מהרשימה השנייה אחד אחרי השני לרשימה הראשונה

דרך נוספת לצרף שתי רשימות היא על ידי צירוף כל הפריטים מהרשימה השנייה לרשימה הראשונה בזה אחר זה .

דוגמה :

```
colors1 = ["red", "green", "blue"]
colors2 = ["magenta", "white", "black", "orange"]
for x in colors2 :
    colors1.append(x)
print(colors1)
```

ההדפסה שנקבל : ['red', 'green', 'blue', 'magenta', 'white', 'black', 'orange']

6.1.57 צירוף רשימה בעזרת המתודה ( )extend

דוגמה להוספת רשימה שנייה בסיום הרשימה הראשונה עם המתודה ( )extend :

```
colors1 = ["red", "green", "blue"]
colors2 = ["magenta", "white", "black", "orange"]
colors1.extend(colors2)
print(colors1)
```

ההדפסה שנקבל : ['red', 'green', 'blue', 'magenta', 'white', 'black', 'orange']

6.1.58 מתודות של רשימה

לפייתון יש מתודות שניתן להשתמש בהן בעבודה עם רשימות. הטבלה הבאה מתארת את רשימת המתודות של רשימה list.

מס"ד	המתודה	תפקיד
1	append( )	מוסיפה פריט בסוף רשימה
2	clear( )	מנקה את הרשימה מכל הפריטים
3	copy( )	מחזירה עותק של הרשימה
4	count( )	מחזירה את מספר הפריטים עם הערך שנשלח אליה
5	extend( )	הוספה של פריט (או רשימה) בסוף הרשימה הנוכחית.
67	index( )	מחזירה את האינדקס של הפריט הראשון עם הערך שנשלח אליה
8	insert( )	מוסיפה פריט במיקום שצוין
9	pop( )	מסירה/מוחקת פריט במיקום שצוין
10	remove( )	מסירה פריט עם הערך שנשלח לה
11	reverse( )	הופכת את סדר הרשימה
12	sort( )	מיון הרשימה

טבלה 1 : רשימת המתודות של רשימה

6.1.59 מתודת append( )

המתודה מוסיפה פריט בסיום רשימה.

התחביר :

list.append(elmnt)

כאשר **elmnt** הוא מחרוזת, מספר, רשימה .

דוגמה 1 : הוספת פריט בסיום רשימה

```
colors = ["red", "green", "blue"]
colors.append("white")
print(colors)
```

ההדפסה שנקבל : ['red', 'green', 'blue', 'white']

דוגמה 2 : הוספת רשימה בסיום של רשימה :

```
colors1 = ["red", "green", "blue"]
colors2 = ["white", "black"]
colors1.append(colors2)
print("colors1 =  ", colors1)
```

```
colors2.append(colors1)
print("colors2 =  ", colors2)
```

ההדפסה שנקבל :

```
colors1 =   ['red', 'green', 'blue', ['white', 'black']]
colors2 =   ['white', 'black', ['red', 'green', 'blue', [...]]]
```

6.1.60 מתודת clear( )

המתודה מסירה את כל הפריטים מרשימה.

התחביר :

list.clear( )

לא שולחים פרמטר אל הפונקציה (לא רושמים בסוגריים כלום)

דוגמה :

```
colors = ["red", "green", "blue"]
colors.clear()
print(colors)
```

ההדפסה שנקבל : [ ] .

6.1.61 מתודת copy( )

המתודה מחזירה עותק של הרשימה שמצוינת בפקודה.

התחביר :

list.copy( )

לא שולחים פרמטר אל הפונקציה (לא רושמים בסוגריים כלום)

דוגמה : העתקה של רשימה אחת לרשימה אחרת

```
colors1 = ["red", "green", "blue"]
colors2=colors1.copy()
print(colors2)
```

ההדפסה שנקבל : ['red', 'green', 'blue']

6.1.62 מתודת count( )

המתודה מחזירה את מספר הפריטים עם הערך ששולחים לה.

התחביר:

list.count(value)

value הוא הערך ששולחים ורוצים לדעת כמה פעמים הוא מופיע ברשימה. הוא יכול להיות מסוג של מספר, מחרוזת, רשימה, tuple וכו' .

דוגמה 1: נספור כמה פעמים מופיע הפריט green הרשימה.

```
colors = [ "red", "blue", "green", "red", "green", "white","green"]
 colors.count("green")
```

ההדפסה שנקבל : 3

דוגמה 2 : נספוק כמה פעמים מופיע הערך 100 ברשימה הבאה .

```
marks = [90,70,54,55,67,89,100,94,50,48,67,100,49,100,65]
print("100 appears : ", marks.count(100), "times ")
```

ההדפסה שנקבל : 100 appears : 3 times

6.1.63 מתודת ( )extend

המתודה מוסיפה את הפריטים ברשימה שבסוגריים (מחרוזת, רשימה, tuple , set וכו') לסוף הרשימה הנוכחית.

התחביר :

list.extend(iterable)

iterable הוא רשימה , tuple , set וכו'.

דוגמה 1 : הוספה של פריטים מהרשימה marks בסוף הרשימה colors :

```
colors = ["red", "green", "blue"]
numbers = [90,70,54,55,67,89]
colors.extend(numbers)
print(colors)
```

ההדפסה שנקבל : ['red', 'green', 'blue', 90, 70, 54, 55, 67, 89]

דוגמה 2 : הוספה של ה tuple points לרשימה colors

```
colors = ["red", "green", "blue"]
points = [90,70,54,55,67,89]
colors.extend(points)
print(colors)
```

ההדפסה שנקבל : ['red', 'green', 'blue', 90, 70, 54, 55, 67, 89]

6.1.64 מתודת ( )index

המתודה מחזירה את המיקום ברשימה של הפריט ששלחנו אליה. היא מחזירה רק את המיקום של הפריט הראשון של הערך .

התחביר :

list.index(elmnt)

elmnt הוא כל סוג של מחרוזת, מספר, רשימה וכו' .

דוגמה 1 : נמצא את מיקום "green" ברשימה colors .

```
colors = ["red", "green", "blue","white","black", "green", "gray"]
x= colors.index("green")
print(x)
```

ההדפסה שנקבל היא 1 כי ברשימה מופיע "red" פעמיים , גם במיקום 1 וגם במיקום 5 . ההדפסה שנקבל היא 1 כי זהו המיקום הראשון שבו הופיע .

אם נבקש פריט שאיננו ברשימה נקבל הודעה שהפריט איננו ברשימה לדוגמה נבקש "gereen" במקום green ונקבל :

```
ValueError: 'gereen' is not in list
```

6.1.65 מתודת ( )insert

המתודה מוסיפה את הערך שצוין במיקום שצוין.

תחביר :

list.insert(*pos, elmnt*)

pos הוא מספר המציין באיזה מיקום להוסיף את הפריט הרצוי.

elmnt הוא הפריט שרוצים להוסיף. הוא יכול להיות מכל סוג כמו מחרוזת, מספר אובייקט וכו'.

דוגמה : הכנסת צבע gray במיקום 1 ברשימה colors .

```
colors = ["red", "green", "blue","white","black"]
colors.insert(1,"gray")
print(colors)
```

ההדפסה שנקבל : ['red', 'gray', 'green', 'blue', 'white', 'black']

6.1.66 מתודת ( )pop

המתודה מסירה פריט מהרשימה במיקום שציינו. המתודה מחזירה את הערך שהוצא.

התחביר :

list.pop(pos)

pos הוא המיקום ברשימה שממנו רוצים להוציא את הפריט.

דוגמה : הוצאת הפריט השני מהרשימה colors (הפריט השני נמצא במיקום 1 כי הפריט הראשון נמצא במיקום 0) .

```
colors = ["red", "green", "blue","white","black"]
colors.pop(1)
```

```
print(colors)
```

ההדפסה שנקבל : ['red', 'blue', 'white', 'black']

אם רוצים להדפיס גם את הערך שהוצאנו נרשום :

```
colors = ["red", "green", "blue","white","black"]
x=colors.pop(1)
print("colors =  ", colors, "  the value taken out : " ,x)
```

ההדפסה שנקבל : colors = ['red', 'blue', 'white', 'black'] the value taken out : green

6.1.67 מתודת remove( )

המתודה מסירה את הפריט הראשון עם הערך שציינו.

התחביר :

list.remove(elmnt)

elmnt הוא כל סוג של פריט שרוצים להסיר כמו מחרוזת, מספר, רשימה וכו'.

דוגמה : הסרה של הצבע red הראשון שמופיע ברשימה colors1 :

```
colors1 = ["red", "green", "blue","white","black", "green", "gray"]
colors1.remove("green")
print(colors1)
```

ההדפסה שנקבל : ['red', 'blue', 'white', 'black', 'green', 'gray']

6.1.68 מתודת reverse( )

המתודה הופכת את הסדר של הפריטים ברשימה. הפריט הראשון הופך מקומות עם האחרון, הפריט השני הופך מקומות עם הפריט לפני אחרון וכך הלאה.

התחביר : **list.reverse()** 0

דוגמה : נהפוך את סדר הפריטים ברשימה colors

```
colors = ["red", "green", "blue", "white", "black", "gray"]
colors.reverse()
print(colors)
```

ההדפסה שנקבל : ['gray', 'black', 'white', 'blue', 'green', 'red']

6.1.69 מתודת ( )sort

המתודה מבצעת מיון אלפא בית בסדר עולה על רשימה . ניתן גם לבצע מיון יורד.

התחביר :

list.sort(reverse=True|False, key=myFunc)

reverse הוא אופציונלי.

אם לא רושמים כלום בתוך הסוגריים של ה reverse אז המיון הוא בסדר עולה.

אם רושמים בסוגריים reverse = False אז שוב המיון בסדר עולה.

אם רושמים reverse=True אז המיון בסדר יורד וזה דומה למתודה **list.reverse()** .

במילים key=myFunc ניתן לקרוא לפונקציה (בדוגמה נקראת myFunc) שתקבע מהם הקריטריונים של המיון.

דוגמה 1 : מיון לפי סדר אלפ0אנומרי :

```
colors = ["red", "green", "blue", "white", "black", "gray"]
colors.sort()
print(colors)
```

ההדפסה שנקבל : ['black', 'blue', 'gray', 'green', 'red', 'white']

גם עבור התוכנית הבאה שדומה לקודמת חוץ מהשורה השנייה שרשמנו colors.sort(reverse=False) והיא נראית כך :

```
colors = ["red", "green", "blue", "white", "black", "gray"]
colors.sort(reverse=False)
print(colors)
```

נקבל את אותה ההדפסה : ['black', 'blue', 'gray', 'green', 'red', 'white']

דוגמה 2 : הפיכת סדר הפריטים ברשימה :

```
colors = ["red", "green", "blue", "white", "black", "gray"]
colors.sort(reverse=True)0
print(colors)
```

ההדפסה שנקבל : ['white', 'red', 'green', 'gray', 'blue', 'black']

דוגמה 3 : שימוש במילה **key =** למיון לפי קריטריון : מיון לפי אורך הפריט :

```
def func1(e):
  return len(e)

colors = ["red", "green", "blue", "white", "black", "gray"]
```

```
colors.sort(key=func1)
print(colors)
```

הסבר התוכנית : ב 2 השורות הראשונות הגדרנו פונקציה בשם func1 המקבלת פריט ומחזירה את האורך שלו (את כמות התווים שבפריט). בתוכנית עצמה הגדרנו רשימה השולחת לפונקציה func1 פריט אחרי פריט מהרשימה , מקבלת את אורך הפריט וממיינת אותם לפי האורך. פריטים עם אותו אורך היא משאירה לפי הסדר שהיה להם ברשימה. פריט שהופיע קודם – יופיע לפני הפריט ברשימה עם אותו אורך. בהדפסה נקבל את הצבעים עם 3 תווים , אחריהם יופיעו פריטים עם 4 תווים ובסוף אלו עם החמישה התווים. בתווים השווים הסדר הוא לפי הסדר שהיה ברשימה המקורית.

ההדפסה שנקבל : ['red', 'blue', 'gray', 'green', 'white', 'black']

דוגמה 4 : מיון בסדר הפוך לפי אורך הפריט **:**

```
def func1(e):
  return len(e)

colors = ["red", "green", "blue", "white", "black", "gray"]
colors.sort(reverse=True, key=func1)
print(colors)
```

הסבר התוכנית : התוכנית דומה לקודמת להבדיל מזה שביקשנו לבצע מיון עם סידור הפוך.

ההדפסה שנקבל : ['green', 'white', 'black', 'blue', 'gray', 'red']

דוגמה 5 : מיון רשימה של dictionaries (מילונים) בשם workers התבסס על הערך של שנת הקבלה לעבודה .

```
def myFunc(e):
  return e['start']

workers = [
  {'electric': 'Moshe', 'start': 2005},
  {'mechanic': 'Dani', 'start': 2000},
  {'secretary': 'Sara', 'start': 2019},
  {'nurse': 'Kati', 'start': 2011}
]

workers.sort(key=myFunc)
```

```
print(workers)
```

ההדפסה שנקבל :

```
[{'mechanic': 'Dani', 'start': 2000}, {'electric': 'Moshe', 'start': 2005}, {'nurse': 'Kati', 'start': 2011},
{'secretary': 'Sara', 'start': 2019}]
```

אם במקום מיון בסדר עולה נבקש מיון בסדר יורד : `workers.sort(reverse=True, key=myFunc)`
נקבל :

```
[{'secretary': 'Sara', 'start': 2019}, {'nurse': 'Kati', 'start': 2011}, {'electric': 'Moshe', 'start': 2005},
{'mechanic': 'Dani', 'start': 2000}]
```

6.1.70 תרגילים ברשימות

כל התרגילים יתייחסו לאחת או שתי הרשימות הבאות :

1. `colors = ["red", "green", "blue", "white", "black", "gray"]`
2. `numbers = [90,70,54,55,67,89]`

1. הדפס את האיבר השלישי בכל אחת מהרשימות.
2. שנה את הפריט השלישי בכל רשימה . ברשימה הראשונה לצבע האהוב עליך וברשימה השנייה לציון שאת/ה רוצים לקבל במבחן.
3. החלף את האיבר הראשון ברשימה colors מ "red" ל "orange" וברשימה השנייה החלף את המספר הקטן ביותר במספר 100 .
4. יש להשתמש במתודה append כדי להוסיף לרשימה הראשונה את הפריט "orange" ואת המספר 100 לרשימה numbers .
5. יש להשתמש במתודה insert כדי להכניס את הצבע "brown" כפריט השלישי ברשימה colors ואת המספר 100 כפריט הרביעי ברשימה numbers .
6. יש להשתמש במתודה remove כדי להסיר את הצבע "blue" ברשימה colors ואת המספר 70 ברשימה numbers.
7. יש להשתמש באינדקס שלילי כדי להדפיס את הפריט האחרון בכל אחת משתי הרשימות.
8. יש להשתמש בטווח של אינדקסים כדי להדפיס את הפריטים השני עד החמישי (כולל החמישי) בכל אחת מהרשימות.
9. יש להדפיס את מספר הפריטים בכל רשימה (ניתן להיעזר בפונקציה len).
10. יש לצרף את הרשימה numbers בסיום colors ולהדפיס את הרשימה החדשה שנוצרה.

6.2 רשומה / צירוף - Tuple

6.2.1 כללי

הגדרה - tuple משמש לאחסון פריטים מרובים במשתנה יחיד.

ההגדרה של **tuple** דומה ל list רק שההבדל הוא ש **tuple** כותבים בין סוגריים עגולים קטנים ולא בין סוגריים מרובעים.

דוגמה : myTuple = ("red", "blue", "green")

tuple הוא אחד מתוך 4 סוגי נתונים מוכללים ב- Python המשמשים לאחסון אוספי נתונים, 3 האחרים הם **list,**
set ו **dictionary** . כולם בעלי תכונות ושימוש שונים.

tuple הוא אוסף שיש לו סדר **והוא בלתי ניתן לשינוי.**

האינדקס של הפריט הראשון הוא [0] של השני [1] וכו'.

tuples אינם ניתנים לשינוי, כלומר אין באפשרותנו לשנות, להוסיף או להסיר פריטים לאחר יצירת ה tuple.

ל **tuple** יכולים להיות פריטים בעלי ערך זהה.

גם כאן כדי לקבל את טיפוס הנתון רושמים את הפונקציה **type( )** . מנקודת המבט של פייתון, tuples מוגדרים כאובייקטים עם סוג הנתונים 'tuple' :

דוגמה :

```
colors = ("red", "blue", "green", "blue")
print(myTuple)
print (type(myTuple))
```

ההדפסה שנקבל :

```
('red', 'blue', 'green', 'blue')
<class 'tuple'>
```

6.2.2 אורך של tuple

כדי לדעת מהו אורך ה tuple (כמה פריטים יש) נשתמש בפונקציה len( )

דוגמה :

```
myTuple = ("red", "blue", "green")
print(len(myTuple))
```

ההדפסה שנקבל : 3

6.2.3 יצירת tuple עם פריט 1

כדי ליצור tuple עם פריט אחד בלבד יש להוסיף פסיק (,) אחרי הפריט אחרת פייתון לא תזהה שמדובר ב tuple ותזהה אותו כ str .

דוגמה : (יש לשים לב לפסיק אחרי הפריט הראשון).

```
myTuple = ("red",)
print(myTuple)
print(type(myTuple))
```

ההדפסה שנקבל :

```
('red',)
<class 'tuple'>
```

קיבלנו הדפסה ש myTuple הוא tuple .

דוגמה : אם לא נשים פסיק אחרי הפריט הראשון :

```
myTuple = ("red")
print(myTuple)
print(type(myTuple))
```

ההדפסה שנקבל :

```
red
<class 'str'>
```

קיבלנו הדפסה שהמחלקה היא מחרוזת ולא tuple .

6.2.4 הפריטים ב tuple

הפריטים ב tuple יכולים להיות מכל טיפוס נתונים.

דוגמה : ניצור 3 דוגמאות של tuples מהטיפוסים מחרוזת – str , שלם - int ובוליאני :

```
tuple1 = ("red", "blue", "green")
tuple2 = (10, -5, -17, 19, 35)
tuple3 = (True, False, False)
```

ב tuple יכולים להיות משתנים מטיפוסי נתונים שונים.

דוגמה ל tuple עם טיפוסי נתונים שונים :

```
tuple1 = ("red", -17, 35, "blue", True, 100, False)
```

6.2.5 הבנאי constructor

ניתן ליצור tuple בעזרת שימוש בבנאי tuple .

דוגמה : יש לשים לב לפעמיים סוגריים קטנים בשורה הראשונה !

```
myTuple = tuple (("red", "blue", "green"))
print(myTuple)
```

```
print(type(myTuple))
```

ההדפסה שנקבל :

```
('red', 'blue', 'green')
<class 'tuple'>
```

6.2.6 אוספים בפייתון (מערכים)

קיימים ארבעה טיפוסי נתונים של אוסף בשפת פייתון:

- **List** - אוסף מסודר שניתן לשינוי ומאפשר פריטים – חברים – כפולים.
- **Tuple** - אוסף מסודר ובלתי ניתן לשינוי. גם בו יש אפשרות לפריטים כפולים.
- **Set** - אוסף לא מסודר וללא אינדקסים. אין אפשרות לחברים כפולים.
- **Dictionary** - אוסף מסודר (ראה הערה בהמשך) שניתן לשינוי . אין חברים כפולים.

הערה : החל מפייתון 3.7 ה dictionaries מסודרים. בפייתון 3.6 וקודמים הם לא מסודרים.

כאשר בוחרים טיפוס של אוסף כדאי להבין את המאפיינים של הטיפוס . בחירת הטיפוס הנכון עבור מערכת של נתונים נותנת שמירה על המשמעות שלהם וזה מגדיל את היעילות והבטיחות.

6.2.8 גישה לפריטים ב tuple

6.2.8 גישה בעזרת האינדקס שלהם

ניתן לגשת לפריט ב tuple בעזרת האינדקס שלו אותו נכתוב בתוך סוגריים מרובעים - [האינדקס].

לדוגמה : להדפיס את הפריט השני ב tuple הנקרא myTuple .

```
myTuple = ("red", "blue", "green")
print(myTuple[1])
```

ההדפסה שנקבל היא : blue (האיבר הראשון הוא עם אינדקס [0])

6.2.9 אינדקס שלילי

אינדקס שלילי פירושו להתחיל מהסוף. הפריט האחרון יהיה במיקום 1- הפריט לפני אחרון יהיה עם אינדקס 2- וכך הלאה.

דוגמה : נדפיס את האיבר האחרון של ה tuple .

```
myTuple = ("red", "blue", "green")
print(myTuple[-1])
```

ההדפסה שנקבל : green .

6.2.10 טווח של אינדקסים

ניתן לציין טווח של אינדקסים על ידי ציון איפה להתחיל ואיפה לסיים את הטווח.

כאשר מציינים טווח הערך המוחזר יהיה tuple חדש עם הפריטים שציינו.

דוגמה : הדפסה של הפריטים השני, השלישי והרביעי . הטווח שנרשום הוא [2:5] אבל זה לא כולל את החמישי !

```
myTuple = ("red", "blue", "green", "white", "blue")
print(myTuple[2:5])
```

ההדפסה שנקבל : ('green', 'white', 'blue')

אם בטווח לא נציין את האינדקס הראשון אז הכוונה היא החל מאינדקס 0 .

דוגמה :

```
myTuple = ("red", "blue", "green", "white", "blue")
print(myTuple[:5])
```

ההדפסה שנקבל : ('red', 'blue', 'green', 'white', 'blue')

אם בטווח לא נרשום את האינדקס הסופי אז נקבל הדפסה עד פריט האחרון.

דוגמה :

```
myTuple = ("red", "blue", "green", "white", "blue")
print(myTuple[2:])
```

ההדפסה שנקבל : ('green', 'white', 'blue')

6.2.11 טווח של אינדקסים שליליים

יש לציין אינדקסים שליליים אם רוצים להתחיל את החיפוש מסוף ה tuple .

דוגמה : הדפסת הפריטים מ 4- עד 1- (לא כולל 1-) :

```
myTuple = ("red", "blue", "green", "white", "blue")
print(myTuple[-4:-1])
```

ההדפסה שנקבל : ('blue', 'green', 'white')

6.2.12 בדיקה האם פריט נמצא ?

כדי לדעת האם פריט מסוים נמצא ב tuple נשתמש במילת המפתח **in** .

תרגיל : נבדוק האם הצבע green נמצא ב tuple :

```
myTuple = ("red", "blue", "green", "white", "blue")
if "green" in myTuple:
    print("yes, green is in the tuple")
```

ההדפסה שנקבל : yes, green is in the tuple

הערה : גם אם נרשום במקום "green" את המילה 'green' התוכנית תעבוד באופן זהה.

6.2.13 עדכון של tuple

אין אפשרות לשנות tuple , כלומר לא ניתן להוסיף או להסיר פריטים אחרי יצירת ה tuple , אבל ישנן כמה דרכים לעקיפת הבעיה.

6.2.14 שינוי הערכים של tuple

יש אפשרות להמיר את ה tuple ל list , לשנות את ה list ולהמיר בחזרה את ה list ל tuple

דוגמה : נשנה את הצבע green ל yellow ב tuple ששמו myTuple .

```
myTuple = ("red", "blue", "green", "white", "blue")
myList=list(myTuple)          # הפיכה לרשימה
myList[2]="yellow"            # [2] החלפת הפריט במיקום
myTuple=tuple(myList)         # tuple המרה של הרשימה ל
print(myTuple)
```

ההדפסה שנקבל : ('red', 'blue', 'yellow', 'white', 'blue')

6.2.15 הוספת פריט

היות ו tuple איננה ניתנת לשינוי אין לה מתודה כמו append() אבל יש דרכים אחרות להוסיפה של פריטים ל tuple .

6.2.16 המרה לרשימה

בדיוק כמו הדרך לעקיפת *הבעיה לשינוי* tuple , יש אפשרות להמיר אותה לרשימה, להוסיף את הפריטים הרצויים ולהמיר אותם בחזרה ל tuple .

דוגמה: המרה של myTuple לרשימה list הוספה של פריט בסוף הרשימה והחזרת הרשימה ל tuple .

```
myTuple = ("red", "blue", "green")
myList=list(myTuple)          # הפיכה לרשימה
myList.append("yellow")                    # הוספת הפריט בסוף הרשימה]
myTuple=tuple(myList)# המרה של הרשימה ל tuple
print(myTuple)
```

ההדפסה שנקבל : ('red', 'blue', 'green', 'yellow')

6.2.17 הוספה של tuple ל tuple

ניתן להוסיף tuple ל tuple כך שאם ברצוננו להוסיף פריט אחד (או כמה פריטים) יוצרים tuple נוסף עם הפריט או הפריטים שרוצים להוסיף ואז מוסיפים את tuple הנוסף ל tuple הרצוי.

דוגמה : הוספה של tuple חדש של 3 פריטים בסוף tuple רצוי.

```
myTuple = ("red", "blue", "green")
newTuple = ("white", "black", "brown" )        # יצירת tuple חדש
myTuple += newTuple            # הוספת ה tuple החדש לרצוי
print(myTuple)
```

ההדפסה שנקבל : ('red', 'blue', 'green', 'white', 'black', 'brown')

הערה : אם יוצרים tuple עם פריט אחד לא לשכוח לשים פסיק לאחר הפריט אחרת הפריט לא יזוהה כ tuple !!

6.2.18 הסרה של פריט מ tuple

לא ניתן להסיר פריט מ tuple אבל ניתן לעקוף את הבעיה כמו שעשינו בסעיפים הקודמים ששינינו או הוספנו פריט ל tuple .

דוגמה : הסרה של פריט על ידי הפיכת tuple ל list , הסרת הפריט ב list והמרה חזרה ל tuple . בדוגמה כאן נסיר את blue .

```
myTuple = ("red", "blue", "green")
myList=list(myTuple)          # הפיכה לרשימה
myList.remove("blue")         # הסרת הפריט blue
myTuple=tuple(myList)         # המרה של הרשימה ל tuple
print(myTuple)
```

ההדפסה שנקבל : ('red', 'green')

6.2.21 מחיקה של tuple

ניתן למחוק את כל ה tuple בעזרת מילת המפתח **del** .

דוגמה :

```
myTuple = ("red", "blue", "green")
del myTuple
print(myTuple)
```

ההדפסה שנקבל :

```
Traceback (most recent call last):
  File "./prog.py", line 3, in <module>
NameError: name 'myTuple' is not defined
```

קיבלנו שגיאה בשורה 3 כי מחקנו את myTuple ולכן היא איננה מוגדרת ולא ניתן להדפיס אותה.

6.2.22 פרוק/ריווח של tuple - Unpack Tuples

כאשר יוצרים tuple אנחנו מקצים להם ערכים. הדבר נקרא "packing" (אריזה או צפיפות).

בפייתון ניתן לחלץ את הערכים בחזרה למשתנים. הפעולה נקראת "unpacking" (פירוק או ריווח בעברית).

דוגמה: הכנסת הערכים של myTuple למשתנים:

```
myTuple = ("red", "blue", "green")
(x, y, z) = myTuple      # העברה של הערך red למשתנה x הערך blue למשתנה y ושל הערך green למשתנה z
print(x)
print(y)
print(z)
```

ההדפסה שנקבל :

```
red
blue
green
```

כמות המשתנים צריכה להיות שווה לכמות הערכים שב tuple . אם הכמות לא שווה נשתמש בכוכבית * ההסבר בפסקה הבאה.

אם בשורה 2 היינו רושמים כמות גדולה יותר של משתנים כמו (x, y, z) = myTuple היינו מקבלים את ההדפסה :

ValueError: not enough values to unpack (expected 4, got 3)

שאומרת שאין מספיק ערכים לפירוק (ציפינו ל 4 ערכים וקיבלנו 3).

אם בשורה 2 היינו שמים כמות קטנה יותר של משתנים כמו (x, y) = myTuple היינו מקבלים את ההדפסה :

ValueError: too many values to unpack (expected 2)

שאומרת שיש ערכים רבים מידי לפירוק (ציפינו רק ל 2 ערכים ויש 3).

6.2.21 השימוש בכוכבית * Using Asterisk

אם מספר המשתנים קטן ממספר הערכים ניתן להוסיף * לשם המשתנה והערכים יוקצו למשתנה כרשימה - list :

דוגמה : נעביר ערכים ל x ו y ואת שאר הערכים כרשימה ל z

```
myTuple = ("red", "blue", "green", "black", "white")
(x, y, *z) = myTuple     # העברה של הערך red למשתנה x הערך blue למשתנה y ושאר הערכים למשתנה z
print(x)
print(y)
print(z)
```

ההדפסה שנקבל :

```
red
blue
```

['green', 'black', 'white']

אם נוסיף כוכבית לשם של משתנה לפני האחרון אז יוקצו ערכים למשתנה הזה עד שמספר הערכים שנותרו יתאים למספר המשתנים שנותרו.

דוגמה :

```
myTuple = ("red", "blue", "green", "black", "white")
(x, *y, z, w) = myTuple
print(x)
print(y)
print(z)
print (w)
```

ההדפסה שנקבל :

```
red
['blue', 'green']
black
white
```

הערך שהוקצע ל x הוא red . ל y הוקצתה רשימה של 2 ערכים blue ו green . הוקצו 2 ערכים כי אחרי y יש לנו עוד 2 משתנים והם קיבלו את 2 הערכים האחרונים של ה tuple .

6.2.23 לולאות ב tuple - Python - Loop Tuples

6.2.23 לולאת for

ניתן לבצע לולאה ב tuple בעזרת לולאת for .

דוגמה : מעבר בלולאה על הפריטים והדפסה שלהם.

```
myTuple = ("red", "blue", "green")
for x in myTuple :
   print(x)
```

ההדפסה שנקבל :

```
red
blue
green
```

נרחיב על לולאות for בפרק על לולאות for בפייתון.

6.2.27 לולאה בעזרת מספר האינדקס

ניתן לעבור בלולאה בין הפריטים של tuple על ידי התייחסות למספר האינדקס שלהם. לשם כך ניעזר בפונקציות range( ) ו len( )
דוגמה : נדפיס את הפריטים לפי האינדקס שלהם בעזרת הפונקציות range ( ) ו len( ) :

```
myTuple = ("red", "blue", "green")
for i in range (len(myTuple)) :
    print(myTuple[i])
```

ההדפסה שנקבל :

```
red
blue
green
```

6.2.28 שימוש בלולאת while

ניתן לעבור על הפריטים ב tuple בעזרת לולאת while . נשתמש בפונקציה len( ) כדי לקבוע את אורך הלולאה . נתחיל מ 0 ונעבור בלולאה בין הפריטים של ה tuple בעזרת התייחסות לאינדקס שלהם. יש לזכור להגדיל את האינדקס ב 1 בסיום כל איטרציה (לולאה , מעבר על סדרת פקודות).

דוגמה : הדפסת הפריטים בעזרת לולאת while

```
myTuple = ("red", "blue", "green")
j=0
while j < (len(myTuple)) :
    print(myTuple[j])
    j=j+1
```

ההדפסה שנקבל :

```
red
blue
green
```

נרחיב על לולאות while בפרק על לולאות while בפייתון.

6.2.29 צירוף tuples

6.2.27 צירוף של 2 tuples

כדי לצרף 2 tuples או יותר נשתמש **באופרטור** + .

דוגמה : צירוף של tuple בסיום tuple **:**

```
myTuple1 = ("red", "blue", "green")
myTuple2 = (1, 2, 3)
myTuple3 = ("white", "black")
```

```
myTuple4 = myTuple3 + myTuple2 + myTuple1
print(myTuple4)
```

ההדפסה שנקבל : ('white', 'black', 1, 2, 3, 'red', 'blue', 'green')

6.2.28 הכפלה של tuples

אם רוצים להכפיל את התוכן של tuple מספר פעמים נשתמש **באופרטור** * .

דוגמה : נכפיל את myTuple 3 פעמים :

```
myTuple1 = ("red", "blue", "green")
myTuple2 = myTuple1*3
print(myTuple2)
```

ההדפסה שנקבל : ('red', 'blue', 'green', 'red', 'blue', 'green', 'red', 'blue', 'green')

6.2.30 מתודות של tuple

לפייתון יש 2 מתודות מוכללות בשימוש עם tuples .

- מתודת ( )count המחזירה את כמות הפעמים שמופיע ערך רצוי ב tuple.
- מתודת ( )index המחזירה את האינדקס ב tuple של הפריט עם הערך אותו מחפשים .

6.2.30 המתודה ( )count

המתודה ( )count מחזירה את מספר הפעמים שערך שצוין מופיע ב tuple .

התחביר :

***tuple*.count(*value*)**

value הוא הערך (הפריט) שאותו מחפשים.

דוגמה : נבדוק כמה פעמים מופיע הערך "red" ב tuple הבא :

```
myTuple = ('red', 'blue', 'green', 'red', 'green', 'red', 'blue')
print(myTuple.count("red"))
```

ההדפסה שנקבל : 3

6.2.33 המתודה ( )index

המתודה ( )index מוצאת את האינדקס של הפריט הראשון ב tuple של הערך שצוין.

המתודה מבצעת תעופה (חריג) אם הערך לא נמצא.

תחביר :

***tuple*.index(*value*)**

value הוא הערך (פריט) אותו מחפשים.

דוגמה : מציאת האינדקס של הערך "blue" :

```
myTuple = ('red', 'blue', 'green', 'red', 'green', 'red', 'blue')
print(myTuple.index("blue"))
```

ההדפסה שנקבל : 1

6.2.32 תרגול tuples

בתרגול נשתמש ב 2 tuples שעליהן נבצע את התרגול:

```
myTuple1 = ("red", "blue", "green")
myTuple2 = (1, -7,10)
```

1. להדפיס את הפריט הראשון ב tuple1 ואת הפריט השני ב tuple2
2. להדפיס את כמות הפריטים בכל tuple .
3. להשתמש באינדקס שלילי כדי להדפיס את הפריט האחרון ב myTuple1 ואת הפריט לפני האחרון ב myTuple2 .
4. להדפיס את הפריטים השני והשלישי ב myTuple1 ואת הראשון והשני ב myTuple2 .
5. לבדוק ולהדפיס האם "blue" נמצא ב myTuple1 והאם 5 נמצא ב myTuple2 .
6. להוסיף "black" ל myTuple1 ואת הערכים 100, 200, 1000 ל myTuple2 .
7. לרשום פקודה שתכפיל את myTuple1 בכמות הפריטים שיש ב myTuple2.

6.3 מילונים – dictionaries

מילונים משמשים לאחסן ערכים של נתונים בצמד של **key : value** או בעברית ערך : מפתח .

מילון הוא אוסף מסודר, שניתן לשנות אותו והוא לא מאפשר כפילויות.

החל מפייתון 3.7 מילונים הם מסודרים. בפייתון 3.6 ומטה המילונים אינם מסודרים.

מילונים נכתבים בין סוגריים מסולסלים ויש להם מפתחות וערכים .

דוגמה:

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
print(my_dictionary)
```

ההדפסה שנקבל : {'brand': 'Chevrolet', 'model': 'Malibu', 'year': 2021}

6.3.1 הפריטים במילון – Dictionary Items

הפריטים של מילון מסודרים, ניתן לשנות אותם ולא מאפשרים כפילויות.

הפריטים מוצגים בזוגות של key:value וניתן להתייחס אליהם על ידי שימוש בשם ה key (מפתח).

דוגמה :

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
print(my_dictionary["brand"])
```

ההדפסה שנקבל : Chevrolet

אם נרצה הדפסה של כל השלושה :

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
print(my_dictionary["brand"], my_dictionary["model"], my_dictionary["year"])
```

ההדפסה שנקבל : Chevrolet Malibu 2021

6.3.2 מסודרים או לא מסודרים ?

כפי שציינו מקודם - החל מפייתון 3.7 מילונים הם מסודרים. בפייתון 3.6 ומטה המילונים אינם מסודרים.

כאשר אומרים שמילונים הם מסודרים הכוונה שלפריטים יש סדר מוגדר וסדר זה איננו ניתן לשינוי.

כשאומרים מילון לא מסודר הכוונה היא שלפריטים אין סדר מוגדר ולא ניתן לגשת אליהם בעזרת אינדקס.

6.3.3 ניתנים לשינוי Changeable

כשאומרים שמילונים ניתנים לשינוי הכוונה שניתן לשנות , להוסיף או להסיר פריטים אחרי יצירת המילון.

6.3.4 כפילויות אינן מאופשרות - Duplicates Not Allowed

במילון לא יכולים להיות 2 פריטים עם אותו מפתח – key .

דוגמה: ערכים כפולים יגרמו "לדריסת" הערך הקודם.

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021,
  "year" : 2022
}
print(my_dictionary)
```

ההדפסה שנקבל : {'brand': 'Chevrolet', 'model': 'Malibu', 'year': 2022}

המפתח key עבור year "דרס" את השנה 2021 ונרשם 2022 .

6.3.5 אורך של מילון Length of dictionary

כדי לדעת כמה פריטים יש במילון ניעזר בפונקציה ( )len .

דוגמה : נבדוק כמה פריטים יש במילון my_dictionary :

```
print(len(my_dictionary))
```

הדפסה שנקבל : 3

6.3.6 טיפוסי הנתונים במילון – Dictionary items – data types

הערכים של הפריטים במילון יכולים להיות מכל טיפוס.

דוגמה:

```
my_dictionary} =
"  brand": "Chevrolet,"
```

```
"  electric" : "True,"
"  model": "Malibu,"
"  year": 2021,
"  colors" :["Red","Black","White","Silver"]
{
print(my_dictionary)
```

במילון כאן הכנסנו טיפוס נתונים בוליאני ב electric (האם הרכב חשמלי ?) ורשימה של צבעים.

ההדפסה שנקבל:

```
{'brand': 'Chevrolet', 'electric': 'True', 'model': 'Malibu', 'year': 2021, 'colors': ['Red', 'Black', 'White', 'Silver']}
```

6.3.7 טיפוס - Type

מבחינת פייתון ההתייחסות למילונים היא כאובייקטים עם טיפוס נתון 'dict' . הוא שייך ל <class 'dict'>

דוגמה :

```
my_dictionary = {
  "brand": "Chevrolet",
  "electric" : "True",
  "model": "Malibu",
  "year": 2021,
  "colors" :["Red","Black","White","Silver"]
}
print(type(my_dictionary))
```

ההדפסה שנקבל : <class 'dict'>

6.3.8 אוספים בפייתון (מערכים) - Python Collections (Arrays)

קיימים ארבעה טיפוסי נתונים של אוסף בשפת פייתון:

- **רשימה list** - אוסף מסודר וניתן לשינוי. יש אפשרות לכפילות חברים ברשימה.
- **רשומה/צירוף Tuple** - אוסף מסודר ולא ניתן לשינוי. יש אפשרות לכפילות חברים.
- **Set** - אוסף לא מסודר וללא אינדקס. אין אפשרות לכפילות חברים.
- **מילון Dictionary** – אוסף מסודר ולא ניתן לשינוי. אין כפילות חברים.

בעת בחירת טיפוס אוסף, כדאי להבין את המאפיינים של טיפוס זה. בחירת הטיפוס הנכון עבור סט נתונים מסוים תאפשר שמירה על משמעות, ומשמעות הדבר עשויה להיות עלייה ביעילות או באבטחה.

6.3.9 גישה לפריטים – Accesssing Items

ניתן לפנות לפריטים של מילון על ידי התייחסות לשם המפתח שלו בתוך סוגריים מרובעים (כמו בדוגמה שהראנו מקודם) :

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
print(my_dictionary["brand"])
```

ההדפסה שנקבל : Chevrolet

6.3.10 גישה לפריט עם המתודה ( )get

דוגמה:

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
x=my_dictionary.get("brand")
print(x)
```

ההדפסה שנקבל : Chevrolet

6.3.11 המתודה ( )keys

המתודה מחזירה רשימה של כל ה keys (מפתחות) במילון .

דוגמה :

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
```

```
print(my_dictionary.keys())
```

ההדפסה שנקבל : dict_keys(['brand', 'model', 'year'])

הרשימה של המפתחות שקיבלנו היא תצוגה של המילון, כלומר כל השינויים שהוצעו במילון ישתקפו ברשימת המפתחות.

דוגמה:

בשלב ראשון ניצור משתנה car מטיפוס מילון שיש לו שלושה מפתחות.

```
car = {
"brand": "Chevrolet",
"model": "Malibu",
"year": 2021
}
x = car.keys()
print(x)  # לפני שנשנה את המילון
```

ההדפסה שנקבל : dict_keys(['brand', 'model', 'year'])

עכשיו נוסיף מפתח נוסף של צבע ונבדוק את המפתחות אחרי השינוי :

```
car["color"] = "white"
print(x)    # אחרי תוספת של מפתח למילון
```

ההדפסה שנקבל : dict_keys(['brand', 'model', 'year', 'color'])

6.3.12 המתודה values( )

המתודה מחזירה רשימה של כל הערכים שבמילון .

דוגמה :

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
print(my_dictionary.values())
```

ההדפסה שנקבל : dict_values(['Chevrolet', 'Malibu', 2021])

גם כאן קיבלנו רשימה של הערכים שבמילון. אם נוסיף פריט נקבל רשימה חדשה עם הפריט שהוספנו.

דוגמה:

```
car = {
"brand": "Chevrolet",
```

```
"model": "Malibu",
"year": 2021
}
x = car.values()
print(x)    # לפני השינוי
car["color"] = "white"
print(x)    # אחרי השינוי
```

ההדפסה שקיבלנו :

```
dict_values(['Chevrolet', 'Malibu', 2021])
dict_values(['Chevrolet', 'Malibu', 2021, 'white'])
```

ההדפסה העליונה היא לפני השינוי וההדפסה השנייה אחרי השינוי של התוספת car["color"] = "white" .

6.3.13 המתודה items( )

המתודה מחזירה כל פריט במילון כ tuples .

דוגמה : קבלת רשימה של זוגות key : value של המשתנה car מהדוגמה הקודמת:

```
print(car.items())
```

ההדפסה שנקבל : dict_items([('brand', 'Chevrolet'), ('model', 'Malibu'), ('year', 2021), ('color', 'white')]

הרשימה המוחזרת היא תצוגה של פריטי המילון כזוגות tuple, כלומר כל השינויים שבוצעו במילון ישתקפו ברשימת הפריטים.

6.3.14 בדיקה האם קיים מפתח

כדי לדעת האם מפתח – key נמצא במילון משתמשים במילה in .

דוגמה : בדיקה האם "model" נמצא במילון my_dictionary .

```
my_dictionary} =
"  brand": "Chevrolet,"
"  model": "Malibu,"
"  year": 2021
{
if "model" in my_dictionary:
    print ('"model" is a key in my_dictionary')
else:
    print ('"model" is NOT a key in my_dictionary')
```

ההדפסה שנקבל : "model" is a key in my_dictionary

6.3.15 שינוי ערכים

ניתן לשנות ערך של פריט רצוי על ידי התייחסות לשם ה key .

דוגמה : שינוי ה model של הרכב

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
my_dictionary["model"] = "Camaro"
print ( my_dictionary)
```

ההדפסה שנקבל : {'brand': 'Chevrolet', 'model': 'Camaro', 'year': 2021}

6.3.16 עדכון מילון בעזרת המתודה ( )update

המתודה מעדכנת את המילון עם הפריטים מהארגומנט הנתון .

הארגומנט חייב להיות מילון או אובייקט איטרלי עם צמד key:value .

דוגמה: עדכון ה model בעזרת המתודה ( )update

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
my_dictionary.update({"model" : "Impala"})
print(my_dictionary)
```

ההדפסה שנקבל : {'brand': 'Chevrolet', 'model': 'Impala', 'year': 2021}

6.3.17 הוספת פריטים למילון

הוספת פריט למילון מתבצעת על ידי הוספת אינדקס מפתח key חדש והשמה של ערך אליו כמו שראינו בסעיף 6.3.12.

דוגמה : הוספת צבע - "color" - למילון .

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
```

```
my_dictionary["color"]="white"
print(my_dictionary)
```

ההדפסה שנקבל : {'brand': 'Chevrolet', 'model': 'Malibu', 'year': 2021, 'color': 'white'}

6.3.18 הסרה (סילוק) של פריטים במילון Remove Dictionary Items

ישנן מספר מתודות להסרה (סילוק) של פריטים במילון .

6.3.18.1 המתודה ( )pop

המתודה מסירה את הפריטים עם שם המפתח - key - שאנחנו מציינים.

דוגמה : הסרת הפריט עם המפתח model :

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
my_dictionary.pop("model")
print(my_dictionary)
```

ההדפסה שנקבל : {'brand': 'Chevrolet', 'year': 2021}

6.3.18.2 המתודה ( )popitem

המתודה מסירה את הפריט האחרון שהוכנס (בגרסאות לפני 3.7 היה מוסר פריט רנדומאלי).

דוגמה : הוספת פריט למילון ולאחריה הסרת הפריט האחרון שהוכנס:

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
my_dictionary["color"]="white"
print(my_dictionary)       # הדפסה ראשונה
my_dictionary.popitem()
print(my_dictionary)       # הדפסה שנייה
```

ההדפסות שנקבל :

```
{'brand': 'Chevrolet', 'model': 'Malibu', 'year': 2021, 'color': 'white'}
```

{'brand': 'Chevrolet', 'model': 'Malibu', 'year': 2021}

רואים שהפריט האחרון שהכנסנו הוא זה שהוסר.

אם נבצע שינוי במפתח model מ Malibu ל Impala ונקרא למתודה ( )popitem אז הפריט שיימחק הוא עם המפתח year ולא האחרון שבוצע עליו השינוי !!

דוגמה:

```
my_dictionary["model"]="Impala"    # model  שינוי שם המפתח
my_dictionary.popitem()
print(my_dictionary)
```

ההדפסה שנקבל : {'brand': 'Chevrolet', 'model': 'Impala'}

הפריט שנמחק הוא הפריט האחרון במילון כי זהו הפריט האחרון שהוכנס למילון .

6.3.18.3 - המילה del

המילה מוחקת את הפריט עם שם המפתח שמציינים.

דוגמה 1 : הסרת הפריט עם המפתח year :

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
del my_dictionary["year"]
print(my_dictionary)
```

ההדפסה שנקבל : {'brand': 'Chevrolet', 'model': 'Malibu'}

דוגמה 2 : מחיקת כל הפריטים במילון (כולל המילון עצמו)

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
del my_dictionary
print(my_dictionary)
```

ההדפסה שנקבל :

```
Traceback (most recent call last):
  File "./prog.py", line 7, in <module<
```

NameError: name 'my_dictionary' is not defined

קיבלנו הודעת שגיאה שהמילון my_dictionary לא הוגדר (כי מחקנו את כל הפריטים שלו).

6.3.18.4 - המתודה ( )clear

המתודה מרוקנת את הפריטים שבמילון אבל לא מוחקת את המילון.

דוגמה:

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
my_dictionary.clear()
print(my_dictionary)
```

ההדפסה שנקבל : {}

הפריטים נמחקו אבל המילון קיים.

6.3.19 לולאת מילונים Loop Dictionaries

6.3.19.1 לולאה דרך מילון – loop Through A Dictionary

ניתן לבצע לולאה במילון בעזרת לולאת **for** .

כאשר נמצאים בלולאה במילון הערך המוחזר הם המפתחות keys של המילון, אבל ישנן מתודות שניתן להחזיר את הערכים .

דוגמה 1 : החזרת כל המפתחות של המילון אחד אחרי השני :

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
for x in my_dictionary:
  print(x)
```

ההדפסה שנקבל :

```
brand
model
year
```

דוגמה 2 : החזרת כל הערכים של המילון אחד אחרי השני :

```
my_dictionary = {
"  brand": "Chevrolet",
"  model": "Malibu",
"  year": 2021
}
for x in my_dictionary:
  print(my_dictionary[x])
```

ההדפסה שנקבל :

```
Chevrolet
Malibu
2021
```

דוגמה 3 : השימוש במתודה values( ) להחזרת הערכים של מילון :

```
for x in my_dictionary.values():
 print(x)
```

ונקבל את אותה ההדפסה כמו בדוגמה הקודמת :

```
Chevrolet
Malibu
2021
```

דוגמה 4 : החזרת המפתחות של המילון בעזרת המתודה keys( )

```
for x in my_dictionary.keys():
  print(x)
```

ההדפסה שנקבל:

```
brand
model
year
```

דוגמה 5 : המתודה items( ) להחזרת המפתחות והערכים

```
for x, y in my_dictionary.items():
 print(x, y)
```

ההדפסה שנקבל :

```
brand Chevrolet
model Malibu
year 2021
```

6.3.20 העתקת מילונים - Copy Dictionaries

לא ניתן להעתיק מילון על ידי הקלדה כמו dict2 = dict1 כי במקרה כזה dict2 יהיה רק ייחוס ל dict2 (הם יישבו על אותן כתובות זיכרון) ולכן שינויים באחד מהם ישפיעו אוטומטית על השני !

ישנן דרכים להעתקה של מילון אחד אל השני. אחת הדרכים היא בעזרת המתודה ( )copy .

6.3.20.1 העתקה של מילון בעזרת המתודה ( )copy .

דוגמה : העתקת המילון my_dictionary אל המילון my_car

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
my_car=my_dictionary.copy()
print (my_car)
```

ההדפסה שנקבל :

```
{'brand': 'Chevrolet', 'model': 'Malibu', 'year': 2021}
```

6.3.20.2 העתקה של מילון בעזרת הפונקציה ( )dict

דרך נוספת ליצירת עותק של מילון היא בעזרת הפונקציה המובנית ( )dict .

דוגמה:

```
my_car=dict(my_dictionary)
print (my_car)
```

ההדפסה שנקבל : {'brand': 'Chevrolet', 'model': 'Malibu', 'year': 2021}

6.3.21 קינון של מילונים Nested Dictionaries

מילון יכול להכיל מילונים אחרים . מצב כזה נקרא קינון מילונים.

6.3.21.1 יצירה של מילון מקונן

דוגמה 1 : יצירה של מילון נקרא my_family שבו רשומים שם האישה ושמות חלק מהילדים של אחד מאבותינו :

```
my_family = {
    "wife1" : {
      "wife" : "Lea",
      "year"  : 60
```

```
    },
    "wife2" : {
       "wife" : "Rachel",
       "year"  : 57
    },
    "child1" : {
      "name" : "Reuven",
      "year" : 35
   },
   "child2" : {
      "name" : "Shimon",
      "year" : 34
   },
   "child3" : {
      "name" : "Levi",
      "year" : 33
   }
}
print (my_family)
```

ההדפסה שנקבל :

```
{'wife1': {'wife': 'Lea', 'year': 60}, 'wife2': {'wife': 'Rachel', 'year': 57}, 'child1': {'name': 'Reuven',
'year': 35}, 'child2': {'name': 'Shimon', 'year': 34}, 'child3': {'name': 'Levi', 'year': 33}}
```

6.3.21.2 הוספת מילונים אל מילון

דוגמה 2 : הוספת המילונים child1 ו child2 אל המילון avraham

```
child1 : {
  "name" : "Itzhak",
  "year" : 30
  }
child2 : {
  "name" : "Ishmael",
  "year" : 32
}
```

```
avraham = {
  "child1" : child1,
  "child2" : child2
}
print (avraham)
```

ההדפסה שנקבל : {'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}}

6.3.22 מתודות של פייתון

לפייתון יש קבוצה של מתודות מובנות שניתן להשתמש בהן במילונים. על חלק מהמתודות עברנו בדוגמאות קודמות. נציין את כל המתודות בטבלה הבאה ונצרף דוגמאות :

מס"ד	המתודה	תיאור
1	**clear( )**	מוחקת את כל הרכיבים שבמילון
2	**copy( )**	מחזירה עותק של המילון
3	**fromkeys( )**	המתודה ( ) fromkeys מחזירה מילון עם המפתחות שצוינו והערך שצוין
4	**get( )**	מחזירה את הערך של מפתח key רצוי
5	**items( )**	מחזירה רשימה הכוללת tuple לכל זוג של מפתח : ערך
6	**keys( )**	מחזירה רשימה הכוללת את מפתחות המילון
7	**pop( )**	מסלקת את הרכיב עם המפתח שציינו בקריאה למתודה
8	**popitem( )**	מסלקת את זוג המפתח : ערך האחרון שנוסף
9	**setdefault( )**	מחזירה את הערך של המפתח שצוין. אם המפתח לא קיים : הכנס את המפתח עם הערך שצוין .
10	**update( )**	מעדכנת את המילון עם הצמד מפתח : ערך שצוין
11	**values( )**	מחזירה רשימה של כל הערכים במילון

בסעיפים הבאים נסביר את המתודות בליווי דוגמאות

6.3.22.1 המתודה ( )clear

המתודה מוחקת את כל הרכיבים שבמילון . לא שולחים פרמטרים אל המתודה.

התחביר : *dictionary*.clear()

Dictionary הוא שם המילון שרוצים למחוק. לא שולחים פרמטרים למתודה.

דוגמה:

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
my_dictionary.clear()
print(my_dictionary)
```

ההדפסה שנקבל : { } . קיבלנו מילון ללא רכיבים . המילון קיים כי אם נבקש הדפסה של מילון שלא קיים נקבל הודעת שגיאה.

6.3.22.2 המתודה ( )copy

המתודה מחזירה העתק של המילון שצוין בקריאה למתודה.

התחביר : *dictionary*.copy()

dictionary הוא שם המילון שאת ההעתק שלו יש להעביר למילון אחר. לא שולחים פרמטרים אל המתודה.

דוגמה :

```
car1= {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
print("car1 = ", car1)
car2 =car1.copy()
print("car2 = ", car2 )
```

ההדפסות שנקבל:

```
car1 =  {'brand': 'Chevrolet', 'model': 'Malibu', 'year': 2021}
car2 =  {'brand': 'Chevrolet', 'model': 'Malibu', 'year': 2021}
```

הערה: כדאי לשים לב שאם היינו רושמים car2=car1 אז car2 היה נמצא בדיוק באותן כתובות של car1 ושינוי של אחד המילונים היה גורם לשינוי זהה במילון השני ! במקרה של שימוש במתודה copy שינוי באחד המילונים לא ישפיע על השני.

6.3.22.3 המתודה ( )fromkeys

המתודה ( ) fromkeys מחזירה מילון עם המפתחות שצוינו והערך שצוין.

התחביר : dict.fromkeys(*keys, value*)

ה keys הכרחי והוא מציין את המפתחות של המילון החדש.

ה value הוא אופציונלי. זהו ערך אחד של כל המפתחות ! של המילון החדש . לא חייבים לציין ערך ואז בערך ליד המפתח נרשם None.

דוגמה : יצירה של מילון בשם newdict עם 3 מפתחות mykey1 mykey2 mykey3 שבכולם הערך 0 .

```
x = ('mykey1', 'mykey2', 'mykey3')
y = 0
newdict = dict.fromkeys(x, y)
print(newdict)
```

ההדפסה שנקבל : {'mykey1': 0, 'mykey2': 0, 'mykey3': 0}

הסבר : בשורה הראשונה והשנייה הגדרנו משתנים בשם x עם 3 מחרוזות ו עם ערך y של 0 . בשורה השלישית השתמשנו במתודה fromkeys שבמחלקה dict ליצירת מילון שבו המפתחות הם המחרוזות שב x והערך הוא זה שב y . בשורה הרביעית הדפסנו את המילון החדש שנוצר.

דוגמה : יצירה של מילון עם מפתחות וללא ערכים:

```
x = ('mykey1', 'mykey2', 'mykey3')
#y = (0, 1, 2 )
newdict = dict.fromkeys(x)
print(newdict)
```

ההדפסה שנקבל : {'mykey1': None, 'mykey2': None, 'mykey3': None}

6.3.22.4 המתודה ( )get

המתודה מחזירה את הערך של הפריט שהמפתח שלו צוין.

התחביר : *dictionary*.get(*keyname, value*)

keyname הוא הכרחי והוא מציין את הפריט עם שם המפתח ממנו נקבל את הערך.

value הוא אופציונלי. הערך שיוחזר אם המפתח keyname לא נמצא .

דוגמה: קבלת הערך של המפתח - key - של ה year במילון mydictionary :

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
x=my_dictionary.get("year",2022)
print(x)
```

ההדפסה שנקבל : 2021 .

אם לא היה נמצא המפתח year היינו מקבלים 2022 . לדוגמה נרשום year בטעות כ yer :

```
x=my_dictionary.get("yer",2022)
print(x)
```

ההדפסה במקרה זה הייתה 2022 כי אין את המפתח yer .

6.3.22.5 המתודה ( )items

המתודה מחזירה אובייקט תצוגה המכיל זוגות של מפתח : ערך של המילון כ tuple . אובייקט התצוגה יראה את כל השינויים שבוצעו במילון (נראה בדוגמאות שבהמשך).

התחביר : *dictionary*.items()

dictionary שם המילון שממנו רוצים להחזיר את הזוגות מפתח : ערך . לא שולחים פרמטרים למתודה.

דוגמה 1: העבר לאובייקט התצוגה x את כל זוגות **מפתח: ערך** של המילון my_dictionary והדפס אותם למסך.

```
x=my_dictionary.items()
print(x)
```

ההדפסה שנקבל : dict_items([('brand', 'Chevrolet'), ('model', 'Malibu'), ('year', 2021)])

דוגמה 2 : כאשר משנים ערך של פריט במילון אובייקט התצוגה מתעדכן גם הוא :

```
my_dictionary["model"]="Impala"
my_dictionary["year"]= 2020
print(x)
```

ההדפסה שנקבל : dict_items([('brand', 'Chevrolet'), ('model', 'Impala'), ('year', 2020)])

6.3.22.6 המתודה ( )keys

המתודה מחזירה אובייקט תצוגה המכיל את המפתחות של המילון כרשימה. אובייקט התצוגה ישקף כל שינוי שבוצע במילון.

תחביר : *dictionary*.keys()

Dictionary הוא שם המילון . לא שולחים פרמטרים לפונקציה.

דוגמה 1 : העבר לאובייקט התצוגה y את כל המפתחות של המילון my_dictionary והצג אותם.

```
my_dictionary = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
y=my_dictionary.keys()
print(y)
```

ההדפסה שמקבלים : dict_keys(['brand', 'model', 'year'])

דוגמה 2 : הוסף מפתח color למילון my_dictionary והראה שגם אובייקט התצוגה קיבל את העדכון:

```
my_dictionary["color"]="Silver"
print(y)
```

ההדפסה שמקבלים : dict_keys(['brand', 'model', 'year', 'color']) .

6.3.22.7 המתודה ( )pop

המתודה מסירה את הפריט שצוין מהמילון.

הערך של הפריט שהוסר הוא הערך החוזר מהמתודה.

התחביר : *dictionary*.pop(*keyname, defaultvalue*)

dictionary – שם המילון . keyname – שם המפתח שאותו רוצים להסיר. *dfaultvalue* – אופציונלי. הערך שיוחזר אם ה keyname (המפתח) לא קיים במילון.

אם לא צוין *dfaultvalue* והמפתח לא נמצא נקבל הודעת שגיאה.

דוגמה 1 : הסרה של המפתח brand והערך שלו

```
mycar = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
mycar.pop("brand")
print(mycar)
```

ההדפסה שנקבל : {'model': 'Malibu', 'year': 2021}

דוגמה 2 : הדפסת הערך החוזר שהוא הערך של המפתח brand

```
mycar = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
x=mycar.pop("brand")
print(x)
```

ההדפסה שנקבל: Chevrolet

דוגמה 3: קבלת dfaultvalue במידה ולא קיים המפתח

```
 mycar = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
x=mycar.pop("brnd",100)
print(x)
```

ההדפסה שנקבל : 100

דוגמה 4 : קבלת שגיאה במידה ולא קיים מפתח וגם לא רשמנו defaultvalue .

```
mycar = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
x=mycar.pop("brnd")
print(x)
```

ההדפסה שנקבל :

```
Traceback (most recent call last):
  File "./prog.py", line 6, in <module>
KeyError: 'brnd'
```

6.3.22.8 המתודה popitem( )

המתודה מסירה את הפריט האחרון שהוכנס/התוסף למילון. (בגרסאות פייתון לפני 3.7 היה מוסר פריט אקראי) .

הפריט שהוסר הוא הערך החוזר של המתודה והוא מוחזר כ tuple .

התחביר : *dictionary*.popitem()

ה dictionary הוא המילון ממנו מסירים את הפריט. לא שולחים למתודה פרמטרים.

דוגמה 1: הסרת הפריט האחרון מהמילון :

```
mycar = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
print (mycar.popitem())
```

```
print(mycar)
```

ההדפסה שנקבל :

```
('year', 2021)      # זהו הערך החוזר
{'brand': 'Chevrolet', 'model': 'Malibu'}      # year  המילון אחרי שהוסר המפתח האחרון שהוכנס
```

דוגמה 3 : נוסיף למילון פריט , נדפיס את הפריטים במילון ואז נסיר את הפריט האחרון שהכנסנו.

```
mycar = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
mycar["color"]="white"    # color  הוספת פריט של צבע לבן במפתח החדש
print(mycar)              #  הדפסת המילון החדש עם הצבע לבן
mycar.popitem()           #  הסרה של הפריט האחרון שהוספנו
print(mycar)              #  הדפסה של המילון לאחר הסרת הפריט האחרון שהוכנס
```

ההדפסות שנקבל :

```
{'brand': 'Chevrolet', 'model': 'Malibu', 'year': 2021, 'color': 'white'}
{'brand': 'Chevrolet', 'model': 'Malibu', 'year': 2021}
```

דוגמה 4 : הדפסת הפריט האחרון שהסרנו :

```
mycar = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
mycar["color"]="white"   #color  הוספת פריט עם המפתח צבע
print(mycar.popitem())    # הדפסת הערך של הפריט האחרון שהוסר
```

ההדפסה שנקבל:

```
('color', 'white')
```

6.3.22.9 המתודה ( )setdefault

המתודה מחזירה את הערך של הפריט עם המפתח שצוין.

אם המפתח לא קיים מתווסף המפתח עם הערך שצוין.

התחביר : *dictionary*.setdefault(*keyname, value*)

Dictonary – שם המילון הרצוי. keyname – שם המפתח הרצוי שרוצים להחזיר את הערך שלו . value – אופציונלי. אם המפתח קיים אז ל value אין תפקיד. אם המפתח איננו קיים אז מתווסף הפריט עם המפתח והערך ששלחנו. אם המפתח לא קיים ולא שלחנו ערך אז הערך של המפתח החדש שהוספנו יהיה הוא None .

דוגמה 1: החזרת הערך של הפריט עם המפתח "model" . שולחים את הערך "Savana" אבל במקרה הזה אין לערך תפקיד.

```
mycar = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
x=mycar.setdefault("model","Savana")
print(x)
```

ההדפסה שנקבל : Malibu

דוגמה 2 : קריאה למתודה ושליחה של מפתח שלא קיים וערך כלשהו. המפתח והערך יתווספו למילון .

```
mycar = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
x=mycar.setdefault("modl","Savana")
print(x)       # modl הערך המוחזר יהיה Savana כי אין את המפתח
print(mycar)   # modl יתווסף פריט חדש עם המפתח
```

ההדפסה שנקבל :

```
Savana
{'brand': 'Chevrolet', 'model': 'Malibu', 'year': 2021, 'modl': 'Savana'}
```

דוגמה 3 : קריאה למתודה ושליחה של מפתח שלא קיים ולא נשלח ערך. יתווסף המפתח החדש עם הערך None .

```
mycar = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
```

```
x=mycar.setdefault("modl")
print(x)       # modl הערך המוחזר יהיה None כי אין את המפתח
print(mycar)   # modl יתווסף פריט חדש עם המפתח
```

ההדפסות שנקבל :

```
None
{'brand': 'Chevrolet', 'model': 'Malibu', 'year': 2021, 'modl': None}
```

6.3.22.10 המתודה ( )update

המתודה מוסיפה את הפריטים שנציין למילון.

הפריטים יכולים להיות מילון או אובייקט iterable - שניתן לעבור עליו בלולאה - עם זוג של מפתח : ערך .

התחביר : *dictionary*.update(*iterable*)

Dictionary – המילון הרצוי. Iterable – מילון או אובייקט iterable - שניתן לעבור עליו בלולאה - עם זוג של מפתח :ערך .

דוגמה 1 : הוספה של פריטים למילון :

```
mycar = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
mycar.update({"color": "White"})
mycar.update({"Electric": "True"})
print(mycar)
```

ההדפסה שנקבל : {'brand': 'Chevrolet', 'model': 'Malibu', 'year': 2021, 'color': 'White', 'Electric': 'True'}

6.3.22.11 המתודה ( )values

המתודה מחזירה אובייקט תצוגה המכיל את ערכי המילון כרשימה .

אובייקט התצוגה ישקף את כל השינויים שבוצעו במילון.

התחביר : *dictionary*.values() . לא שולחים פרמטרים למתודה.

דוגמה 1 : החזרת הערכים של המילון mycar

```
mycar = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
x=mycar.values()
print(x)
```

הדפסה שנקבל : dict_values(['Chevrolet', 'Malibu', 2021])

דוגמה 2 : שינוי הערכים במילון והדפסת אובייקט התצוגה שהשתנה גם הוא.

```
mycar = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
x=mycar.values() # אוביקט התצוגה מקבל את הערכים של המפתחות
mycar["year"]=2022        # שינוי הערך של המפתח year
print(x)                  # הדפסת אובייקט התצוגה שקיבל אוטומטית את השינוי
```

ההדפסות שנקבל : dict_values(['Chevrolet', 'Malibu', 2022])

6.3.23 תרגילים במילונים

בתרגילים הבאים יש להשתמש במילון :

```
mycar = {
  "brand": "Chevrolet",
  "model": "Malibu",
  "year": 2021
}
```

1. יש להשתמש במתודת ה get להדפסת הערך של המפתח "model" של המילון mycar .
2. יש לשנות את השנה (המפתח "year") ל 2022 .
3. יש להוסיף את המפתח "electric" עם הערך True או False (האם המכונית חשמלית) .
4. יש להשתמש במתודה ( )pop כדי להסיר את המפתח "year" .
5. מחק את המילון car .