

## סיכום דגשים

### חוקיות כתיבה נכונה:

מעבר על מחרוזות (strings):

מחרוזת היא לא ניתנת לשינוי. אי אפשר לעשות `s[0]='a'`.

כתיבה נכונה (לפי תווים) –

```
for char in my_string: # המיקום את צריך
    if char == "A":
        # עשה משהו
```

כתיבה נכונה (לפי אינדקס) –

```
for i in range(len(my_string)): # לשנות או להשוות
    לאיבר סמוך
    if my_string[i] == my_string[i + 1]: # יכול לעשות שגיאה
```

טיפ למניעת שגיאה: אם משתמשים ב-`i + 1`, הטווח חייב להיות –

`range(len(my_string) - 1)`

• אסור להשתמש ב-`append` על מחרוזת. יש להשתמש ב-`+=` במקום.

מעבר על רשימות (Lists):

רשימה היא ניתנת לשינוי.

שגיאה נפוצה – ניסיון למחוק איברים מהרשימה בזמן שעוברים עליה בלולאת `for`. זה משבש את האינדקסים.

הפתרון – אם צריך לשנות את הרשימה, עדיף ליצור רשימה חדשה ריקה ולמלא אותה

`(new_list.append(item))`

`l.sort()` – מיון הרשימה המקורית (בלי להחזיר כלום).

`sorted(l)` – מחזיר רשימה חדשה ממוינת (המקורית נשארת כפי שהיא).

`l.count(x)` – כמה פעמים איבר מופיע ברשימה.

מעבר על טאפל (Tuple):

כמו רשימות, אבל "נעול". משתמשים בו בעיקר כשרוצים נתונים שלא אמורים להשתנות (כמו ימי שבוע או

נקודות `(x, y)`).

• במה משתמשים? רק בפעולות שלא משנות: `index()`, `count()`, סליסינג `([:])`.

• אם במבחן מבקשים לשנות טאפל: התשובה היא בד"כ "אי אפשר" או שצריך להפוך את הטאפל לרשימה

עם `list(t)`, לשנות, ולהחזיר לטאפל עם `tuple(l)`.

מעבר על מילון (Dictionaries):

כאן אין סדר, יש מפתחות (keys) וערכים (values).

• איך עוברים על מילון?

• `for key in d:`

# עובר רק על המפתחות.

• `for val in d.values():`

# עובר רק על הערכים.

• `for key, val in d.items():`

# הדרך המומלצת. נותן גם את המפתח וגם את הערך.

## חוקים שחובה לזכור:

### אתחול משתנים:

תמיד לאתחול את המשתנה הצובר לפני הלולאה.

- למספרים: `count = 0`
- למחרוזות: `new_s = ""`
- לרשימות: `new_l = []`

### טווחים (range):

לזכור ש-`range(1, 5)` עובר על 1, 2, 3, 4 בלבד (עוצר אחד לפני ה-stop).

### החזרת ערך (return לעומת print):

במבחן בד"כ מבקשים "פונקציה שמחזירה". צריך לוודא שמשתמשים ב-`return` בסוף הפונקציה ולא רק ב-`print`. ברגע שיש `return`, הפונקציה מפסידה לעבוד!

### בדיקת קלט:

אם השאלה מבקשת לבדוק אם מחרוזת היא מספר, להשתמש ב-`char.isdigit()`.

### חיתוך (slicing):

הדרך הכי קלה להפוך מחרוזת/רשימה היא `s[::-1]`.

בסלייסינג (`[start:stop:step]`), האינדקס של ה-stop **לעולם לא נכלל**.

- `my_list = [10, 20, 30, 40]`
- `my_list[0:2]` יחזיר רק את `[10, 20]` (אינדקסים 0 ו-1).

### מלכודת בוליאנים (Boolean Logic):

טעות נפוצה מאוד היא כתיבת תנאי "כמו שמדברים":

- הטעות: `if x == 5 or 6:`

• בפיתון זה תמיד יהיה `True`, כי הוא בודק האם `x` שווה ל-5 או האם המספר 6 קיים (וכל מספר

שאינו 0 נחשב ל-`True`).

- הכתיבה הנכונה: `if x == 0 or x == 6:` או `if x in [5, 6]`.

### סיכום תמציתי:

- צריכים לעבור על הכל? `for item in sequence:`
- צריכים אינדקסים? `for i in range(len(sequence)):` (חשוב לזכור מקרי קיצון ונקודות קצה, כלומר חריגות).
- צריכים לשנות מחרוזת? תבנו אחת חדשה.

מילון (dict)	טאפל (tuple)	רשימה (list)	מחרוזת (str)	תכונה / פקודה
כן (Mutable)	לא (Immutable)	כן (Mutable)	לא (Immutable)	ניתן לשינוי?
לפי מפתח ['key']	כן	כן	כן	גישה לפי אינדקס [0]
כן (כמות מפתחות)	כן	כן	כן	שימוש ב- <code>len()</code>
<code>d['new'] = val</code>	אי אפשר	<code>append(), insert()</code>	רק בחיבור: <code>s + "!"</code>	הוספת איבר
<code>del d['key']</code>	אי אפשר	<code>pop(), remove()</code>	אי אפשר	מחיקת איבר
כן (בודק מפתחות)	כן (בודק איבר)	כן (בודק איבר)	כן (בודק תת-מחרוזת)	שימוש ב- <code>in</code>

## לולאות:

### לולאה בודדת: תבנית ה"צבירה" (The Accumulator)

- רוב השאלות הפשוטות מבקשות לעבור על משהו (מחרוזת, רשימה, טווח מספרים) ולמצוא משהו.
  - החוק: תמיד יש משתנה עזר מחוץ ללולאה.
  - השגיאה הנפוצה: הגדרת המשתנה בתוך הלולאה (מה שמאפס אותו בכל סיבוב).
- דוגמה (ספירת אותיות מסוימות):

```
count = 0 # 1. הגדרה מחוץ ללולאה
for char in "python is fun":
    if char == "n":
        count += 1 # 2. עדכון בתוך הלולאה
# 3. סיום - התוצאה מוכנה כאן (מחוץ ללולאה)
```

### לולאות מקוננות: מודל ה"שעון" (The Clock Model)

- זה מבלבל. הדרך הכי טובה להבין לולאה בתוך לולאה היא לחשוב על מחוגי שעון:
- הלולאה החיצונית היא מחוג שעות.
  - הלולאה הפנימית היא מחוג דקות.
- המחוג הפנימי חייב לסיים סיבוב שלום (למשל מ-0 עד ל-59) כדי שהמחוג החיצוני יזוז צעד אחד. מתי משתמשים בזה?
- כשצריך להשוות כל איבר לכל שאר האיברים (למשל: "האם יש כפילויות ברשימה?").
  - כשעובדים עם טבלאות (שורות ועמודות).
  - כשצריך להדפיס תבניות (כמו משולש כוכביות).

```
nums = [1, 2, 3]
```

```
for i in nums: # מחוג שעות
    for j in nums: # מחוג דקות
        print(f"Comparing {i} to {j}")
```

דוגמה (השוואת כל איבר לכל איבר):  
מה יקרה כאן?  
עבור  $i=1$ , ה-j ירוץ על 1,2,3.  
רק אז i יהיה 2, ושוב j ירוץ על 1,2,3.

### דוגמאות ללולאות מקוננות:

רצף מספרים מצטבר בכל שורה (בכל שורה מתחילים מ-1 ומדפיסים על למספר השורה הנוכחית).

```
rows = 5

for i in range(1, rows + 1):
    # מספר השורה הנוכחית i הלולאה הפנימית מדפיסה מ-1 ועד i
    for j in range(1, i + 1):
        print(j, end=" ") # end=" " מונע ירידת שורה אוטומטית
    print() # ירידת שורה בסוף כל "שורה" של הלולאה החיצונית
```

רצף מספרים רץ (Continuous):

כאן לא מתאפס בכל שורה, אלא ממשיך לספור הלאה בעזרת משתנה חיצוני (counter).

```
rows = 4
num = 1 # המשתנה שימשיך לגדול

for i in range(1, rows + 1):
    for j in range(1, i + 1):
        print(num, end=" ")
        num += 1 # קידום המספר הרץ
    print()
```

חזקות של מספר נתון:

טבלה קטנה שמראה חזקות של 2 (או כל בסיס אחר שנבחר).

```
base = 2
limit = 6

print(f"Powers of {base}:")
# לולאה אחת פשוטה שעוברת על המעריכים
for i in range(limit):
    result = base ** i # חישוב חזקה: base בחזקת i
    print(f"{base}^{i} = {result}")
```

טיפול בטבלה (רשימה של רשימות):

נותנים לך מטריצה ומבקשים לבצע עליה חישוב. המשימה: לחשב את הסכום של כל איבר במטריצה.

```
# מטריצה של ציונים: כל רשימה פנימית היא "כיתה"
grades = [
    [80, 90, 70],
    [100, 95, 88],
    [60, 75, 80]
]

total_sum = 0

for class_list in grades: # עוברת על כל "כיתה" (רשימה)
    for grade in class_list: # עוברת על כל "ציון" בתוך הכיתה
        total_sum += grade

print(f"The total sum is: {total_sum}")
```

בדיקת כפילויות או זוגות (השוואת "כל אחד עם כולם"):

כדי לפתור שאלות בסגנון "האם יש ברשימה שני מספרים שסכומם הוא X?", צריך להחזיר איבר אחד (חיצוני)

ולרוץ איתו על שאר האיברים הפנימיים.

```
nums = [1, 5, 8, 3, 2]
target = 10

# אנחנו צריכים אינדקסים כדי לא להשוות מספר עם עצמו
for i in range(len(nums)):
    for j in range(i + 1, len(nums)): # מתחיל מ-i+1
        if nums[i] + nums[j] == target:
            print(f"Found pair: {nums[i]} and {nums[j]}")
```

הדפסת צורות עם לוגיקה משתנה:  
משמש למשל להדפסת ריבוע שבו רק האלכסון או המסגרת שונים.

```
size = 4

for i in range(size):          # שורות
    for j in range(size):      # עמודות
        if i == j:             # אם האינדקסים שווים - אנחנו על האלכסון
            print("X", end=" ")
        else:
            print("0", end=" ")
    print() # ירידת שורה בסוף כל שורה של המטריצה
```

התוצאה תהיה:

```
X 0 0 0
0 X 0 0
0 0 X 0
0 0 0 X
```