

סיכום דגשים

חוקיות כתיבה נכון:

מעבר על מחרוזות (strings):

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

כתיבת נכון (לפי תווים) –

```
for char in my_string: # הדרך הכי בטוחה ככלא צריך את המיקום
    if char == "A": # עשה משווה
```

```
חוינו אם צריך לשנות או להשווות #: # לאיבר סמור
    if my_string[i] == my_string[i]: # יכול לעשות שגיאה
```

טיפול למניעת שגיאה: אם משתמשים ב-1 + i, הטווח חייב להיות – .range(len(my_string) - 1)

כתיבת נכון (לפי אינדקס) –

- 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)).

(().sort() – מיון הרשימה המקורי (בלי להחזיר כלום).

(().sorted() – מחזיר רשימה חדשה ממוקנית (המקורית נשארת כפי שהיא).

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

מעבר על טאפל (Tuple):

כמו רשימות, אבל "נעל". משתמשים בו בעיקר כשרוצים נתונים שלא אמרו להשתנות (כמו ימי שבוע או נקודות y, x).

• ומה משתמשים? רק בפעולות שלא משתנות: ((), count, index, סלייסינג [:]).

• אם ב מבחון מבקשים לשנות טאפל: התשובה היא בד"כ "אי אפשר" או צריך להפוך את הטאפל לרשימה עם (t), לשנות, ולהחזיר לטאפל עם (tuple(t)).

מעבר על מילון (Dictionaries):

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

• איך עוברים על מילון?

עובר רק על המפתחות.

עובר רק על הערכים.

הדרך המומלצת. נותן גם את המפתח וגם את הערך.

for key in d:

for val in d.values():

for key, val in d.items():

חווקים חשובות לזכור:

אתחול מעתנים:

תמיד לאות את המשתנה הצבור לפני הלולאה.

- למספרים: `0 = count`

- למחרוזות: `'' = new_s`

- לרשיומות: `[] = new_l`

טוווחים (range):

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

הזרת ערך (return לועמת print):

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

בדיקה קלט:

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

חיתוך (slicing):

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

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

- `my_list = [10, 20, 30, 40]`

- יחזיר רק את `[20, 10]` (אינדקסים 0 ו-1).

מלכודת בוליאנית (Boolean Logic):

טעות נפוצה מאוד היא כתיבת תנאי "כמו שדברים":

- הטעות: `if x == 5 or 6`

• בפייתון זה תמיד יהיה `True`, כי הוא בודק האם `x` שווה ל-5 או האם המספר 6 קיים (וכל מסגר
שאינו 0 נחשב ל-`True`).

- הכתיבה הנכונה: `if x == 0 or x == 6`.

סיכון תמציתין:

• צריכים לעבור על הכל? `for item in sequence:`

• צריכים אינדקסים? `for i in range(len(sequence))` (חסוב לזכור מكري קיצון ונקודות קצרה, כוללן חריגות).

• צריכים לשנות מחרוזת? תבנו אחת חדשה.

מילון (dict)	טאפל (tuple)	רשימה (list)	מחרוזת (str)	תכונה / פקודה
cn (Mutable)	la (Immutable)	cn (Mutable)	la (Immutable)	ניתן לשינוי?
lfy (key['])	cn	cn	cn	גישה לפי אינדקס [0]
cn (כמו מפתחות)	cn	cn	cn	שימוש ב- <code>in</code> (השאלה)
d['new'] = val	ai (אפשר)	append(), insert()	rk בחיבור: s + "!"	הוספה איבר
del d['key']	ai (אפשר)	pop(), remove()	ai אפשר	מחיקת איבר
cn (בודק מפתחות)	cn (בודק איבר)	cn (בודק איבר)	cn (בודק תחזוק)	שימוש ב- <code>in</code>

לולאות:

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

רוב השאלות פשוטות מבקשות לעבור על משה (מחוזת, רשימה, טווח מספרים) ולמצוא משה.

- החוק: תמיד יש משתנה עזר מחוץ ללולאה.

- השגיאה הנפוצה: הגדרת המשתנה בתוך הלולאה (מה שמאפס אותו בכלל סיבוב).

דוגמה (ספרת אותיות מסוימות):

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

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

זה מבלב. הדרך הכי טובה להבין לולאה בתוך לולאה היא לחשב על מהוגי שעון:

- הלולאה החיצונית היא מהוג שעות.

- הלולאה הפנימית היא מהוג דקota.

המהוג הפנימי חייב לסיים סיבובשלום (למשל מ-0 עד ל-59) כדי שהמהוג החיצוני ייזה צעד אחד. מתי משתמשים בזה?

- 1) צריך להשוות כל איבר לכל שאר האיברים (למשל: "אם יש כפילויות בראשימה?").
- 2) שעובדים עם טבלאות (שורות ועמודות).
- 3) צריך להדפיס תבניות (כמו משולש כוכבים).

דוגמה (השוואת כל איבר לכל איבר):

```
nums = [1, 2, 3]
for i in nums:          מהוג שעות
    for j in nums:      מהוג דקota
        print(f"Comparing {i} to {j}")      מה קרה כאן?
                                            עברו 1=i, ה-j יירץ על 1,2,3.
                                            רק אז זה יהיה 2, ושוב j יירץ על 1,2,3.
```

דוגמאות לlolאות מקוננות:

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

```
rows = 5

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

רצף מספרים רצ (Continuous):

כאן לא מתאפשר בכל שורה, אלא ממשיר למספר הלהה בעזרת המשתנה חיצוני (counter).

```
rows = 4
num = 1 # המ המשתנה שמשיר לגודל

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 שמרת חזקות של 2 (או כל בסיס אחר שנבחר).

```
base = 2
limit = 6

print(f"Powers of {base}:")

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

טיפול בטבלה (רשימה של רשימות):

נתונים לר מטריצה וambilשים 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:          # לולאה חיצונית: עוברת על כל "כיתה" (רשימה)
    for grade in class_list:       # לולאה פנימית: עוברת על כל "ציון" בתוך הכיתה
        total_sum += grade

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

בדיקה כפליות או זוגות (השווות "כל אחד עם כולן"):

כדי לפתר שאלות בסגנון "אם יש ברשימה שני מספרים שסכוםם הוא X?", צריך להחזיר איבר אחד (ציון) ולזרץ אליו 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)): # מתחילה 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 שוים - אנתנו 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
```