



# ניצנים שנה א'

## ספר קורס חלק א'

### חניכים יקרים

ברוכים הבאים לספר הקורס של תכנית ניצנים! הדרך שלכם להפוך לתוכניתנים מעולים מתחילה כאן. ספר זה מכיל את ההגדרות וההסברים למונחים שילוו אתכם במהלך הסמסטר הראשון, מוזמנים להעזר בו לחזור על החומר בכיתה, בבית ותוך כדי תרגול. בהצלחה, מאמינים בכם.

צוות פיתוח ניצנים 2022

מעודכן נכון לתאריך 19.10.22

© כל הזכויות שמורות

## תוכן עניינים

2.....	מבוא לתכנות
2.....	הגדרות בסיסיות
2.....	משתנים
4.....	קבועים
5.....	טיפוסי נתונים בסיסיים
5.....	פקודות קלט פלט
8.....	אופרטורים
9.....	אופרטורים מיוחדים
10.....	casting
12.....	סימוני שגיאות ב-PyCharm
13.....	מבוא לתנאים
13.....	ביטויים לוגיים
14.....	משתנים בוליאניים:
15.....	ביטויים לוגיים מורכבים:
19.....	תנאים בסיסיים
19.....	IF
22.....	ELSE
23.....	elif
26.....	שיטות לתכנון קוד
27.....	שגיאות נפוצות בשימוש בתנאים
29.....	תנאים מתקדמים
29.....	:match
29.....	תנאי מקונן:
30.....	עיצוב קוד:
32.....	שלבים בפתרון בעיה
32.....	דיבאג
33.....	סטנדרטים וקריאות
34.....	עיצוב קוד

35.....	שלבים בפתרון בעיה
36.....	לולאות
36.....	לולאות FOR
37.....	break
38.....	ריצה על מחרוזות
38.....	לולאות for מקוננות
40.....	לולאות WHILE
41.....	while VS for

## מבוא לתכנות

ברוכים הבאים לפרק המבוא! בפרק זה נלמד את המושגים הבסיסיים שיכניסו אותנו לעולם התכנות כמו משתנים, טיפוסים, אופרטורים ועוד. תחת כל נושא תוכלו למצוא את הSYNTAX, הגדרות ודוגמאות בקוד.

## הגדרות בסיסיות

- **פקודת מחשב** – הוראה למחשב לבצע פעולה מסוימת
- **אלגוריתם** – אוסף סופי וחד משמעי של הוראות שביצוען, זו אחרי זו, מוביל לפתרון בעיה מוגדרת מראש.
- **סינטקס (תחביר)** – מערכת הכללים של השפה, שקובעת "איך" לכתוב ביטוי חוקי בשפה.
- **תכנה** – אוסף של הוראות ומידע הניתנות לביצוע על ידי מחשב על פי סדר. התכנה מורכבת מאוסף מאורגן של תוכניות מחשב המשרתות כולן יישום מסוים.
- **ריצה** – ביצוע התכנית מתחילתה ועד סופה על ידי המחשב, שורה אחרי שורה. ככל שהתכנית תהיה מורכבת יותר, זמן הריצה יהיה ארוך יותר.
- **באג BUG** – כשל בתוכנה הגורם להתנהגות שגויה או לקריסה של התכנה.
- **שפת תכנות** – אוסף פקודות ומילים שמורות שיתורגמו לשפת המחשב (אנחנו לומדים את שפת PYTHON)

## משתנים

**הגדרה** – משתנה הוא מכולה של מידע. כלומר, משתנה הוא תא בזיכרון של התוכנה בתוכו אנחנו יכולים לשמור מידע.

פעולות אפשריות של משתנים –

- **הכנסה/השמה** – נוכל להכניס אל המשתנה שלנו מידע כרצוננו.
- **החלפה** – במידה וכבר שמור מידע במשתנה שלנו, נוכל להחליפו במידע אחר, במצב כזה רק המידע החדש ישמר.

- **הוצאה/שליפה** – נוכל להשתמש במידע השמור במשתנה בכל עת.

כמה דוגמות למשתנים בחיינו – גיל, משקל, מספר תושבים וכדומה.

## **SYNTAX** בסיסי - משתנה

הכנסת מידע למשתנה:

לשם כך נשתמש ב- "=" :

```
my_age = 17
```

מצד שמאל של ה- "=" (מוקף בירוק) נגדיר את שם המשתנה, ובכך נוכל לפנות אליו גם בהמשך התוכנית.

שם המשתנה ייכתב **באותיות קטנות** עם \_ מפריד בין המילים וללא רווחים.

מצד ימין (מוקף בכתום) אנחנו כותבים את הערך שיאוחסן בתוך המשתנה:

- אפשר לתת למשתנה ערך מילדי. כלומר לכתוב כבר באותה שורה את המידע שנרצה לשמור בתוך המשתנה.
- אפשר גם לבצע השמה בין משתנים. כלומר, לתת למשתנה ערך של משתנה אחר.

דוגמאות:

**דגש: כל קטע קוד שמגיע אחרי הסימן '#' הוא הערה למתכנת ולא נדרש לכתוב אותו על מנת להגדיר משתנה!**

- בשורה מס' 2 אנחנו מגדירים משתנה מספרי שלם בשם first\_number שהערך שלו יהיה המספר 120.

- בשורה מס' 3 אנחנו מגדירים משתנה מספרי שלם בשם second\_number שהערך שלו יהיה הערך של המשתנה שנקרא first\_number, כלומר גם 120.

- בשורה מס' 6 אנחנו מגדירים משתנה מספרי בעל נקודה עשרונית ששמו הוא decimal\_number וערכו 1.25.

- בשורה מס' 9 אנחנו מגדירים משתנה תוו בשם letter שערכו הוא התו 'a'.

- בשורה מס' 12 אנחנו מגדירים מחרוזת בשם sentence שערכה הוא – Hello World :).

```

var.py x
1  # Int
2  first_number = 120
3  second_number = first_number
4
5  # Float
6  decimal_number = 1.25
7
8  # Char
9  letter = 'a'
10
11 # String
12 sentence = "hello world :)"
13
  
```

```

1. # Int
2. first_number = 120
3. second_number = first_number

4. # Float
5. decimal_number = 1.25
  
```

```

6. # Char
7. letter = 'a'

8. # String
9. sentence = "hello world :)"
  
```

**שימו ⚡ - בהמשך הספר יהיה הסבר על טיפוסים הנתונים השונים (מספרים, תווים ומחרוזות).**

החלפת המידע במשתנה:

דוגמה:

```

number_1 = 7
number_1 = 85
  
```

לאחר כתיבת שורות הקוד האלו, מה שישמר במשתנה שלנו number\_1 יהיה הערך 85 (החליף את 7).

**קבועים**

**הגדרה – קבוע,** כשמו כן הוא, יכיל בתוכו מידע שאינו מתחלף. ברגע שנכנס אליו מידע בפעם הראשונה, לא ניתן להחליפו.

עקרון הגדרת הקבוע והרעיון הלוגי של להחזיק משתנים שערכם אינו משתנה קיים בשפות תכנות רבות, לעקרון משמעותיות רבות אשר ישמשו אתכם בהמשך דרככם בעולם התוכנה. אומנם בשפת פייתון אין לנו אפשרות להגדיר קבוע. כלומר, אם ננסה לשנות את הערך השמור בקבוע לא נקבל שגיאה במהלך ריצת התוכנית, ובכל זאת אנו נשמר את הרעיון הלוגי בשל חשיבותו.

כמה דוגמאות לקבועים בחיינו – תאריך לידה, מספר תעודת זהות, מספר שעות ביממה וכדומה.

פעולות אפשריות של קבועים – הכנסה/השמה, הוצאה/שליפה (**ללא** החלפה!).

**SYNTAX** בסיסי - קבוע

הכנסת מידע לקבוע:

```

NUMBER_OF_PLAYERS = 2
  
```

כמו במשתנה, נשתמש ב- "=" :

מצד שמאל של ה"=" (מוקף בירוק) נגדיר את שם הקבוע, בעזרתו נוכל לפנות אליו גם בהמשך התוכנית. שם הקבוע יכתב **באותיות גדולות** עם \_ מפריד בין המילים וללא רווחים. מצד ימין (מוקף בכתום) אנחנו כותבים את הערך שיאוחסן בתוך הקבוע. בפיתוח, הדרך שלנו להבדיל בין קבוע למשתנה תהיה לפי השם (אותיות גדולות/קטנות).

#### דוגמאות:

קבועים אלו לא ישתנו לאורך ריצת התוכנית, כי הם קבועים!

#### טיפוסי נתונים בסיסיים

ניתן להכניס למשתנים ולקבועים שלנו סוגי מידע שונים, סוגי מידע אלו נקראים טיפוסי נתונים.

#### טיפוסי נתונים בסיסיים:

- **מספר שלם (Integer)** - מספר טבעי, שלם. לדוגמה: 1, 2, -4, 88.
- **מספר לא שלם (Float)** - מספר ממשי, שאינו שלם, מספר עשרוני. לדוגמה: 0.2, 12.5, -4.52, 8.8.
- **תו (Character)** - סוג משתנה המכיל ייצוג של תו **בודד**. תו יכול להיות מספר, אות קטנה או גדולה, סמל וכ"ו - תווים המופיעים לנו במקלדת. לדוגמה: 'k', '6', '%', 'B'. **המספר השלם 6 והתו 6 שונים! גם רווח הוא תו.**
- **מחרוזת (String)** - טיפוס נתונים המכיל רצף של תווים. לדוגמה: "Nitzanim", "How are you?". **מחרוזות נגדיר באמצעות גרשיים.**

#### פקודות קלט פלט

- פקודות קלט פלט הן פעולות אשר מאפשרות למתכנת לתקשר עם המשתמש, לבקש ממנו נתונים או להודיע הודעות.
- קלט - מה שהמחשב קולט מ-"העולם החיצוני", בעיקר מהמקלדת.
  - פלט - מה שהמחשב פולט אל "העולם החיצוני", בעיקר אל מסך.

ניתן להקביל חיפוש בגוגל לפעולת קלט פלט - האתר מקבל/קולט מהמשתמש "מחרוזת" כלשהי מהמקלדת, ובאמצעותה נפליטים למסך אתרים, קישורים, תמונות וכדומה.

הפעולה **input** משמשת אותנו לקליטת קלט מהמשתמש, הפעולה **print** משמשת להוצאת פלט למסך.

#### הפעולה input:

באמצעות פעולה זו נקבל מידע מהמשתמש. הפעולה עוצרת את ריצת התוכנית ומאפשרת למשתמש להקליד שורת קלט. כשהמשתמש ילחץ Enter לאחר הזנת הקלט, ריצת התוכנית תמשיך. הפעולה מאפשרת להדפיס הערות/הוראות למשתמש לפני ההזנה.

Syntax בסיסי:

```
variable = input("Instructions for the user")
```

במקרה זה, המשתנה variable שומר את המידע שהוזן על ידי המשתמש. הפעולה input קוראת את הקלט כתווים ומחזירה אותו כמחרוזת. כלומר, בvariable תישמר מחרוזת.

**שימו ⚡ - מומלץ להשתמש בשורת ההדפסה לשם מתן הוראות למשתמש, נוכל להכווין אותו לפי מה שנרצה שיוזן ומה ההגבלות שלו.** למשל: הכנס את שמך המלא, הכנס מספר בין 100 ל200 וכדומה.

הפעולה print:

פעולה זו מדפיסה "הודעה מוגדרת" על המסך, הודעה זו יכולה להיות מחרוזת, ערך מסוים או כל טיפוס אחר שנרצה שיודפס - ניתן להדפיס כמה אלמנטים/טיפוסים ללא הגבלה. ההודעה תומר למחרוזת ולאחר מכן תפלט על המסך.

ישנן צורות הדפסה שונות ואותן נדגים בסיכום זה.

Syntax בסיסי:

הסינטקס הבסיסי נראה כך -

```
print(element1, element2, element3)
```

בשורת הקוד הזו האלמנטים יכולים להיות מכל טיפוס נתונים גם נוכל להשתמש בכמה אלמנטים שנרצה. כלומר, ניתן להוסיף עוד אלמנטים ואף להוריד - אין הגבלה ל3 אלמנטים בלבד.

**שימו ⚡ - באופן דיפולטי - מפריד בין האלמנטים תו רווח, וגם התו האחרון תמיד יהיה ירידת שורה.**

דוגמה:

```
element1 = "You have made"
element2 = 600
element3 = "steps today."
print(element1, element2, element3)
```

בשורות קוד אלו ניתן לראות כי האלמנט הראשון והשלישי הם מחרוזות (String) וכי האלמנט השני הוא מספר שלם (Int). לאחר מכן מדפיסים אותם.

יודפס למסך -

```

print.py x
1 print("Hello World")
2
3 number = 55
4 print(number)
5
6 print("There's", number, "countries in africa")
  
```

#### צורות הדפסה שכיחות -

1. הודעה למשתמש (שורה 1).
2. הדפסת ערך משתנה (שורות 3-4)
3. הדפסת ערך + הודעה למשתמש (שורה 6).

#### אפשרויות הדפסה נוספות -

##### היכרות עם **end**:

כפי שצוין, באופן דיפולטי התו האחרון בפעולת `print` יהיה ירידת שורה, `end` יכול לשנות זאת. כלומר, נוכל להגדיר באיזה **תו/מחרוזת** אנחנו רוצים שתגמר ההדפסה.

##### דוגמה:

```

print.py x
1 element1 = "You have made"
2 element2 = 600
3 element3 = "steps today."
4 print(element1, end=" ")
5 print(element2, end=" ")
6 print(element3)
  
```

שורות קוד אלו ידפיסו את אותו הערך שראינו בדוגמה הראשונה, ובכל זאת ניתן לראות כי הגדרנו את `end` להיות תו רווח.

```

print(element1, end="??")
print(element2, end="??")
print(element3)
  
```

נניח והיינו מגדירים את `end` להיות המחרוזת `"??"` -

שורת ההדפסה המתקבלת הייתה:

```
You have made??600??steps today.
```

##### היכרות עם **backslash** (קו מוטה אחורנית - \):

לעיתים נרצה לבצע פעולות בתוך `print` שלא בהכרח נוכל לעשות בדרך ההדפסה "הרגילה". למשל - הדפסת גרש או גרשים, ירידת שורה באמצע המחרוזת וכדומה. באמצעות `backslash` נוכל לעשות פעולות אלו וגם פעולות נוספות.

##### טבלת `backslash` שימושיים -

קוד	תוצאה
<code>\</code>	הוספת גרשים
<code>\n</code>	ירידת שורה



\t	הוספת טאב
\b	מחיקת התו האחרון (כמו לחצן המחיקה במקלדת).

## דוגמאות:

```

1 # \"
2 print("We are the so-called \"Vikings\" from the north.")
3 # \n
4 print("1)\n2)\n3)\n4)")
5 # \t
6 print("tab with tabs: t\t a\t b")
7 # \b
8 print("5 X 2 = 10\b")

```

```

We are the so-called "Vikings" from the north.
1)
2)
3)
4)
tab with tabs: t    a    b
5 X 2 = 10

```

## תוצאות ההדפסה:

" - הוספת גרשיים (ירוק).

\n - ירידת שורה (אדום).

\t - הוספת טאב (צהוב).

\b - מחיקת תו אחרון (ורוד).

## לקריאה נוספת על נושא זה לחצו כאן.

## אופרטורים

### הגדרה

**אופרטור** - סמל המשמש לציון פעולה הפועלת על מספר קבוע או משתנה של איברים.

מתי משתמשים

כאשר יש צורך לבצע פעולות מתמטיות על משתנים וערכים.

איך משתמשים

אופרטורים בסיסיים:

אופרטור	שם	דוגמה
+	חיבור	$X + Y$
-	חיסור	$X - Y$
*	כפל	$X * Y$
/	חילוק	$X / Y$
**	חזקה	$X ** Y$

אופרטורים מיוחדים

Floor division - כאשר יש צורך שהתוצאה המתקבלת תהיה מטיפוס int (מספר שלם ללא שארית) - ניתן להשתמש ב `//`.

```
# Floor division
result = 7//3
print(result)
```

**שים לב** - אופרטור זה גורם ל"איבוד" החלק העשרוני של המספר.

Modulus - כאשר נרצה לבדוק מה השארית בחלוקת מספר אחד במשנהו נשתמש ב `%`.

```
# Modulus
result = 7 % 3
print(result)
```

דגשים

- כמו במתמטיקה, גם בפייתון אסור לחלק ב-0. פעולה זו תוביל לקריסת התוכנית.
- הסימן # מייצג הערה בפייתון, קוד הנמצא בשורה לאחר כתיבת # - לא ירוץ

דוגמות קוד



```
# Adding
x = 5
y = 5
print(x + y)

# Sub
x = 5
y = 5
print(x - y)

# Mul
x = 5
y = 5
print(x * y)

# Div
x = 5
y = 5
print(x / y)

# Pow
x = 5
y = 5
print(x ** y)

# Div with //
x = 5
y = 5
print(x // y)

# Mod
x = 9
y = 2
print(x % y)
```

## CASTING

הגדרה

פעולת המרת טיפוסו של משתנה לטיפוס אחר.

מתי משתמשים

כאשר יש צורך להתייחס לערך מסוים באופן שונה - כשייך לטיפוס אחר, על מנת לבצע עליו פעולות. לדוגמה, על מנת לחבר בין שני אלמנטים, שני האלמנטים צריכים להיות מאותו הטיפוס, או מטיפוסים שהגיוני לחבר בניהם.

למשל, קוד זה יעלה שגיאה:

```
# Adding number to a String
text = "my age is "
num = 18
result = text + num
print(result)
```

אין משמעות לחיבור המספר 18 עם מחרוזת טקסט.

פתרון הבעיה הוא – המרת המספר לstring:

```
# Adding number to a String
text = "my age is "
num = str(18)
result = text + num
print(result)
```

מה יקרה עכשיו? **יודפס "my age is 18"**.

איך משתמשים

בשביל לבצע Casting יש להשתמש בסוג המשתנה שאליו יש צורך להמיר את הנתון, ובסוגיים אשר יכילו את הערך שאותו ממירים.

```
result = new_variable_type(value)
```

דגשים

- ניתן לקבוע את טיפוסו של משתנה בשני דרכים:
  1. בזמן הגדרתו
  2. בעזרת casting
- ניתן להשתמש ב-casting כדי לבצע הדפסות:

```
print(element1, element2, element3)
```

במפגש הראשון למדתם להדפיס מספר אלמנטים, והשתמשתם בפסיק כדי להפריד בין אלמנט לאלמנט בצורה הבאה:

```
print(str(element1) + str(element2) +
```

דרך נוספת להדפיס מספר אלמנטים בעזרת המרה למחרוזות והצמדת כל המחרוזות אחת לשנייה:

דוגמאות קוד

המרה ל-int:

```
x = "3"
y = 3.6
z = input("Enter a number: ")
int(x)
int(y)
int(z)
```

המרה ל-float:

```
x = "3.9"
y = 3
float(x)
float(y)
```

המרה ל-string:

```
x = 1
y = 1.9
str(x)
str(y)
```

## סימוני שגיאות ב-PYCHARM

**סימון אדום** – מייצג שגיאת הרצה בקוד

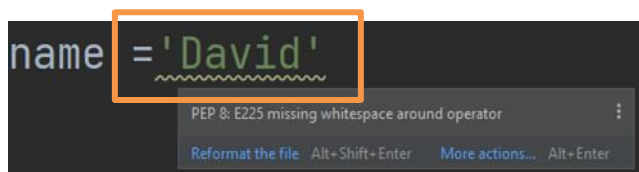
שגיאת הרצה תופיע כאשר כותבים קטע קוד שגוי מבחינה תחבירית.

- מקור השגיאה הוא בסינטקס השפה, ולכן סביבת העבודה לא תוכל לבצע הרצה של הקוד.



**סימון צהוב** – מייצג שגיאת סטנדרטים

- בשפה Python קיים סט של כללים וסטנדרטים לכתיבת קוד בצורה אחידה, נוחה וקריאה.
- סימון צהוב לא ימנע מכם להריץ את הקוד, אך נרצה להימנע משגיאות כאלה.



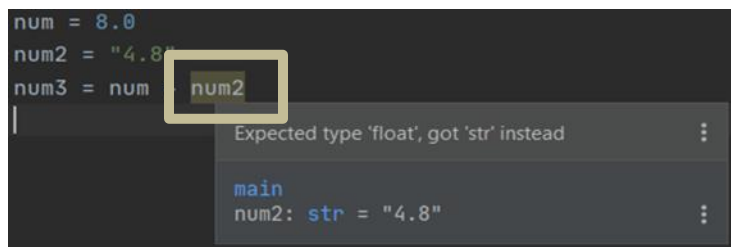
סימון **ירוק** – מייצג שגיאת כתיב.

- במקרה של שגיאת כתיב, סביבת העבודה תסמן את שגיאת הכתיב בצבע ירוק.



סימון **מרקר עדין** – מייצג שגיאה לוגית

- אזהרת שגיאה מסמנת שגיאה לוגית בקוד, שעלולה לגרום לתוכנית לקרוס או לא לקיים קטע מסוים בגלל טעות לוגית.
- כאשר אתם נתקלים באזהרות אלו יש לבדוק אותן ולפתור אותן, על מנת למנוע את התקלות מראש.



## מבוא לתנאים

### ביטויים לוגיים

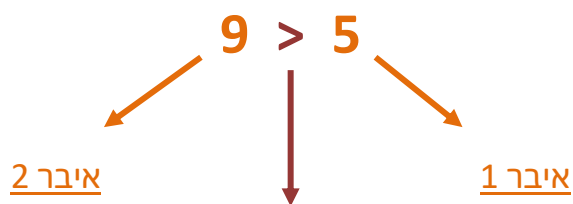
הגדרות

- **ביטוי לוגי** - ביטוי שערכו יכול להיות **אמת** או **שקר**.
- **ביטוי לוגי פשוט** - מורכב מאופרטור אחד ושני איברים שתוצאת היחס בניהם היא **אמת** או **שקר**.
- **אופרטור לוגי** - פעולה לוגית אשר מתבצעת על איבר אחד או יותר.

מבנה ביטוי לוגי פשוט

אופרטור לוגי מציין **יחס** בין שני אלמנטים.

לדוגמה הביטוי הלוגי הפשוט הבא:



הסימן > הוא אופרטור לוגי

הוא מציין שהאיבר משמאל גדול מהאיבר בצד ימין

טענה זו יכולה להיות אמת או שקר

(במהרה הזה הטענה נכונה ולכן ערך הביטוי כולו הוא אמת)

אופרטורים לוגיים בסיסיים:

אופרטור	הסבר	דוגמה	מתי ייתן אמת?
==	שווה	$A == B$	רק אם A שווה ל-B
!=	לא שווה	$A != B$	רק אם A לא שווה ל-B
>	גדול	$A > B$	רק אם A גדול מ-B
>=	גדול שווה	$A >= B$	רק אם A גדול או שווה ל-B
<	קטן	$A < B$	רק אם A קטן מ-B
<=	קטן שווה	$A <= B$	רק אם A קטן או שווה ל-B

דגשים

- על מנת להכניס ערך לתוך משתנה יש להשתמש ב=  
ולקבל תוצאה של אמת ושקר יש להשתמש ב==, שגיאה זו נפוצה מאוד בקרב מתכנתים מתחילים!
- לעיתים מתכנתים עלולים לשכוח את מצב השוויון בין שני ערכים. לכן חשוב להבין טוב מה האופרטור ההופכי לכל אופרטור.

משתנים בוליאניים:

הגדרה

**משתנה בוליאני** – משתנה אשר יכול להחזיק אחד מהערכים הבוליאניים - True או False. למשתנה זה ניתן לבצע השמה ישירה של הערך True או False, או לבצע השמה של ביטוי לוגי שיפורש לערך True או False.

**שימו לב** – זהו טיפוס נתונים חדש – bool, לצידם של str, int, float.

מתי משתמשים

כשנרצה להביע ביטוי לוגי בקוד.

איך משתמשים

נבצע השמה של ביטוי לוגי לתוך משתנה כך:

```
result = 5 < 10
print(result) #True
```

ביטויים לוגיים מורכבים:

הגדרות

**ביטוי לוגי מורכב** – מספר ביטויים לוגיים פשוטים שביניהם אופרטור לוגי מורכב.

**אופרטור לוגי מורכב** – אופרטור שמחבר בין שני ביטויים לוגיים ויוצר בניהם קשר לוגי.

מתי משתמשים

כשנרצה לשלב בין שני ביטויים לוגיים פשוטים.

למשל:

## במרוכב יש ארבע צלעות וגם במשלוש יש ארבע צלעות





איך משתמשים

אופרטורים לוגיים מורכבים:

מתי ייתן אמת?	דוגמה	הסבר	אופרטור
אם A שקר	$\text{not } A$	לא	not
אך ורק אם $A > B$ וגם $C < D$	$(A > B) \text{ and } (C < D)$	וגם	and
אם $A > B$ או $C < D$	$(A > B) \text{ or } (C < D)$	או	or

טבלאות אמת:

מייצגות את תוצאות האופרטורים המורכבים בהתאם לביטויים הלוגיים הפשוטים אשר האופרטור יושב עליהם.

### NOT

מתי ייתן אמת?	דוגמה	הסבר	אופרטור
אם A שקר	$\text{not } A$	לא	not
אך ורק אם $A > B$ וגם $C < D$	$(A > B) \text{ and } (C < D)$	וגם	and
אם $A > B$ או $C < D$	$(A > B) \text{ or } (C < D)$	או	or

משמעות: הופך את ערך הביטוי

למשל: אם הביטוי "היום יום שני" הוא אמת אז הביטוי "היום יום שני NOT" הוא שקר.

### AND

טבלת 0/1

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

טבלת True/False

A	B	A and B
False	False	False
False	True	False
True	False	False
True	True	True

משמעות: מחזיר אמת רק אם שני הביטויים הם אמת

למשל: אם היום יום שני ויורד היום גשם הביטוי "היום יום שני AND יורד גשם" הוא אמת.

## OR

טבלת 0/1

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

טבלת True/False

A	B	A or B
False	False	False
False	True	True
True	False	True
True	True	True

משמעות: מחזיר אמת אם אחד מהביטויים לפחות הוא אמת

למשל: אם היום יום שני ולא יורד גשם הביטוי "היום יום שני OR יורד גשם" הוא אמת.

דגשים

- ניתן לייצג ביטויים לוגיים גם בעזרת מספרים:
  - שקר – 0
  - אמת – כל השאר

דוגמאות קוד

NOT

```
# Using True/False
a = False
result = not a
print(result)

# Using 1/0
a = 1
result = not a
print(result)
```

### AND

```
# Using True/False
a = False
b = False
c = a and b
print(c)
b = True
c = a and b
print(c)

# Using 1/0
a = 1
b = 0
c = a and b
print(bool(c))
b = 1
c = a and b
print(bool(c))
```

### OR

```
# Using True/False
a = False
b = False
c = a or b
print(c)
b = True
c = a or b
print(c)

# Using 1/0
```

```

a = 1
b = 0
c = a or b
print(bool(c))
b = 1
c = a or b
print(bool(c))

```

## תנאים בסיסיים

חניכים יקרים, שיחקתם פעם עץ או פאלי?  
 כנראה שכן, ובטח שיחקתם כך: "אם יצא עץ אז הפסדתי/ניצחתי...אם יצא פאלי אז...".  
 צורת החשיבה "**אם... אז**.." פוגשת אותנו בכל יום בחיים, כאשר קיימות מספר אפשרויות לתרחיש בסיטואציה מסוימת התלויות בדבר שיקרה או לא יקרה. למשל כאן – **אם** יצא עץ **אז** ניצחתי. בשיטה זו משתמשים גם בקוד, קוראים לזה **תנאים**.

מסמך זה מסכם את החומר הנלמד במפגש תנאים בסיסיים.

### הגדרה

**תנאי (condition)** - ביטוי שקובע אם קטע קוד מסוים יתבצע או לא.

בפייתון קיימות שלוש מילים שמורות המשמשות לכתיבת תנאים:

1. if
2. else
3. Elif




### IF

### הגדרה

**if (אם)** - מגדיר קטע קוד שירוצץ אם ערך הביטוי הבוליאני שכתוב בו הוא אמת.

### מתי משתמשים?

כשנרצה לבצע פעולה רק במקרה ספציפי.  
 למשל:

- תיוג בפוסט באינסטגרם - **אם** המשתמש הקליד @, **אז** הצג את רשימת האנשים שהוא יכול לתייג. 
- כיפת ברזל - **אם** טיל חודר לשטחי המדינה, **אז** שלח טיל נגדי למיקום XX. 
- **Spotify** - **אם** משתמש סימן שיר כ"אהוב", **אז** הוסף אותו לרשימת המועדפים 

איך משתמשים?

בעברית מבנית התחביר יראה כך:

**אם ביטוי בולאני:** #אחרי הביטוי יהיו נקודותיים

קטע קוד שירוצק אם הביטוי הבולאני הוא True # קטע הקוד יהיו מוזח פנימה

בפייתון התחביר יראה כך:

**If** number == 8: # אחרי הביטוי יהיו נקודותיים

print("The number is 8") # קטע הקוד יהיה מוזח פנימה

קוד שאינו מוזח פנימה # Code that runs anyway

**שימו לב** – הקוד שנמצא בהזחה אחת קדימה נקרא "בלוק". בכל פעם שאתם מזיחים קוד טאב אחד קדימה, אתם יוצרים "בלוק" חדש של קוד. בדוגמה שמעל, פקודת print היא חלק מהבלוק החדש שנוצר כחלק מהתנאי.

**הזחה** – הרחקה של שורה כתובה מהשוליים.  
 נבצע הזחה באמצעות כפתור ה-tab במקלדת.

תרשים זרימה:

כך יראה התהליך של תנאי if:

ראשית יבדק הביטוי הבולאני, אם ערכו אמת ירוץ קטע הקוד בבלוק התנאי (זה שבתוך קו ההזחה בקוד). לאחר מכן נעבור לקוד הבא (הנמצא מחוץ לקו ההזחה בקוד).

אם הביטוי הוא שקר, נעבור ישיר לקוד הנמצא מחוץ לקו ההזחה.



#### דגשים:

- אחרי כל משפט תנאי יש לשים נקודתיים.
- את קטע הקוד השייך לתנאי מסמנים באמצעות הזחתו טאב אחד פנימה. על מנת לצאת מהבלוק של התנאי, יש לכתוב אותו בקו ההזחה הראשי.
- קוד הנמצא בהזחה פנימה יקרא "בלוק" של קוד.
- בלוק יכול או לרוץ, או שמדלגים עליו. כלומר, או ש"נכנסים אל הבלוק" ומבצעים את הפעולות הכתובות בו או שמדלגים עליו ולא מבצעים אף פעולה שכתובה בו.

#### דוגמאות קוד:

בדיקה אם מספר שווה ל-8:

```
number = 8
if number == 8:
    print("The number is 8")
```

בדיקת יחס בין זוג מספרים:

```
first_number = int(input())
second_number = int(input())
if first_number > second_number:
    print(first_number)
if second_number > first_number:
    print(second_number)
if first_number == second_number:
    print(first_number * 2)
```

כך נראה בלוק של קוד:

```
if number > 7:
    number = 12          #
    number_2 = number - 2 # block of
    number_3 = number_2 * 7.5 # code
```

```
number = number_3 // 2      #
print(number)
```

ELSE

הגדרה

**else (אחרת)** – מגדיר קטע קוד שירוצ אם ערך הביטוי הבוליאני של התנאי הקודם לא מתקיים.

מתי משתמשים?

למשל:



- התחברות לאינסטגרם – **אם** המשתמש הקליד סיסמה לא נכונה אז התרע על סיסמה לא נכונה. **אחרת**, חבר את המשתמש.



- "אהבתי" לשיר ב-Spotify – **אם** השיר כבר ברשימת האהובים אז הורד את השיר מהרשימה. **אחרת** הוסף את השיר לרשימה

איך משתמשים?

בעברית מבנית התחביר יראה כך:

**אם ביטוי בוליאני:** #אחרי הביטוי יהיו נקודותיים

קטע קוד שירוצ רק אם ערך הביטוי הוא True #הזחה פנימה

**אחרת:** #נקודותיים

קטע קוד שירוצ רק אם ערך הביטוי הוא False #הזחה פנימה

קוד שירוצ בכל מקרה #נבדל בכך שאינו מוזח פנימה

בפייתון התחביר יראה כך:

**if** number == 8: # אחרי הביטוי יהיו נקודותיים

print("The number is 8") # הזחה

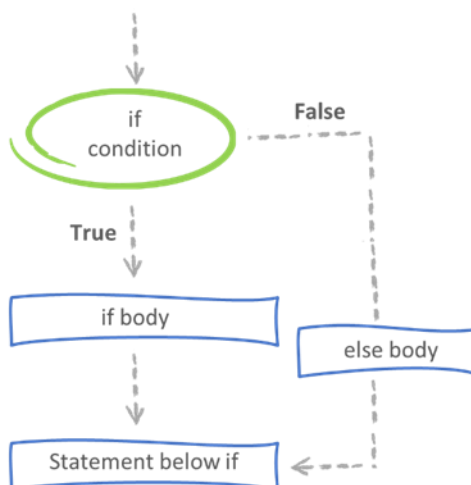
**else:** # נקודותיים

print("The number is not 8") # הזחה

Code that runs anyway # ללא הזחה

## תרשים זרימה:

כך יראה התהליך של תנאי else -



דגשים:

- else נמצא תמיד בזוגיות עם if מסוים הבא לפניו. כלומר, הוא מתייחס לתנאי if ספציפי כלשהו וירוך במידה ותנאי זה **אינו** מתקיים.

דוגמאות קוד:

בדיקה אם מספר שווה ל-8 או שלא:

```

number = 8
if number == 8:
    print("The number is 8")
else:
    print("The number is not 8")
  
```

בדיקה אם המשתמש הוא ג'יימס בונג:

```

name = input("Insert your name: ")
if name == "Bond":
    print("Welcome on board 007")
else:
    print("Good morning", name)
  
```

ELIF

הגדרה

**elif (אחרת אם)** - קיצור של if - else, תנאי נוסף שיבדק רק אם התנאי שלפניו הוא שקר.



מתי משתמשים?

למשל:

- **אם** כמות העוקבים שלך קטנה מ-1000, הצג את כמות העוקבים.  
**אחרת אם** כמות העוקבים שלך קטנה ממיליון: הצג את כמות העוקבים חלקי 1,000 + 'K'.  
**אחרת** הצג את כמות העוקבים חלקי 1,000,000 + 'M'.

איך משתמשים?

בעברית מבנית התחביר יראה כך:

**אם** ביטוי בוליאני: #נקודותיים

קטע קוד שירוצ' רק אם ערך הביטוי הוא True #הזחה

**אחרת אם** ביטוי בוליאני חדש: #נקודותיים

קטע קוד שירוצ' רק אם ערך הביטוי החדש הוא True #הזחה

**אחרת:** #נקודותיים

קטע קוד שירוצ' רק אם ערך הביטוי הוא False #הזחה

נקודותיים: # `if number == 8:`

הזחה # `print("The number is 8")`

נקודותיים: # `elif number == 9:`

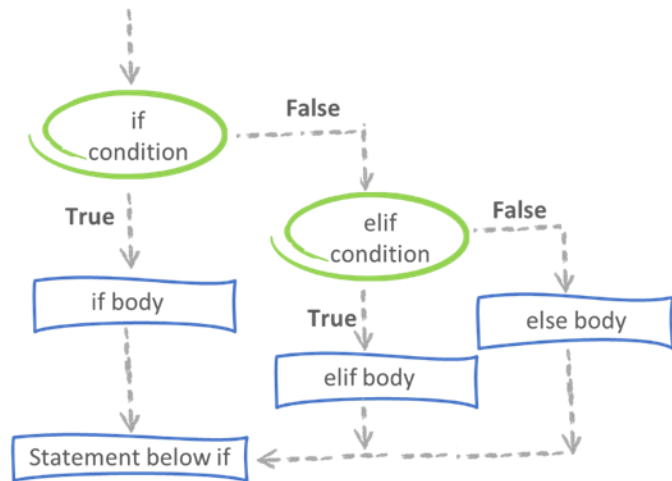
הזחה # `print("The number is 9")`

נקודותיים: # `else:`

הזחה # `print("The number is not 8 nor 9")`

ללא הזחה # `Code that runs anyway`

כך יראה התהליך של תנאי elif-else -



### דגשים:

- בדומה ל-else, גם elif תמיד נמצא בזוגיות עם if ספציפי.
- בכל תנאי חייב להיות if! לעומת זאת, אין חובת שימוש ב- elif, else.
- יכול להיות תנאי שיש בו elif אך אין בו else.
- בכל מערכת יחסים של תנאים יהיה if אחד, else אחד (או אפס), ומספר בלתי מוגבל של elif (או אפס).
- בסדר הופעתם if יהיה תמיד ראשון, else אחרון וביניהם elif:

if

elif

elif....

else

- בכל מערכת if-elif-else, בסופו של דבר ירוץ קטע קוד אחד בלבד הנמצא באחד מהם.
- טיפ של אלופים: במידה ואתם בודקים הרבה מקרים עם if ו-elif, מקמו את המקרה הנפוץ יותר ראשון וכך תמנעו בדיקות מיותרות.

דוגמאות קוד:

בדיקה אם מספר שווה ל-8, או ל-9, או לאף אחד מהם:

```

if number == 8:
    print("The number is 8")
elif number == 9:
    print("The number is 9")
else:

```

```
print("The number is not 8 nor 9")
# Code that runs anyway
```

בדיקת יחס בין שני מספרים:

```
first_number = int(input())
second_number = int(input())
if first_number > second_number:
    print(first_number)
elif second_number > first_number:
    print(second_number)
else:
    print(first_number * 2)
```

#### שיטות לתכנון קוד

לפני כתיבת הקוד, חשוב להבין את המשימה לעומקה ולתכנן את הפתרון בצורה המיטבית. ניתן לתכנן בכל שיטה שתצאו, כאן נמליץ לכם על שתי שיטות טובות ונפוצות.

#### דוגמה:

תרגיל: כתוב תכנית המממשת משחק בו המטרה היא לשלוף מכובע את המספר 8. קבל מהמשתמש את המספר שנשלף והדפס הודעה האם נשלף 8 או שלא נשלף 8.

#### עברית מבנית

ראיתם זאת לאורך כל הסיכום. ניתן לתכנן את הפתרון קודם בעברית, מעין "קוד בעברית", ורק לאחר מכן לעבור לכתוב את הקוד.

אם number שווה ל:8:

למשל:

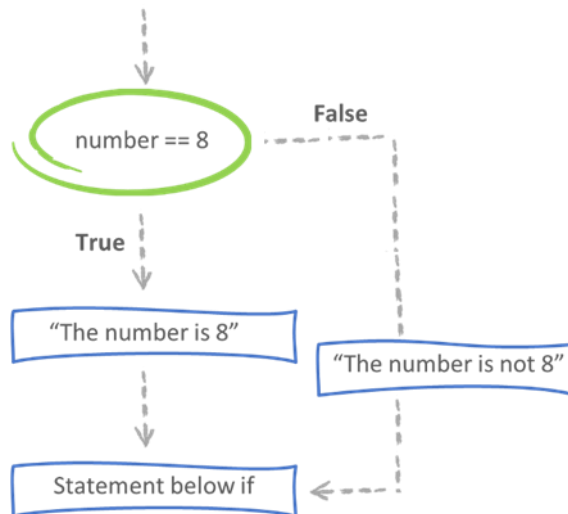
הדפס "המספר הוא 8"

אחרת:

הדפס "המספר אינו 8"

#### תרשים זרימה

תכנון מהלך הריצה של התכנית ומה יקרה בכל שלב שלה.



## שגיאות נפוצות בשימוש בתנאים

### חזרה על קוד

כתיבת שורת קוד גם ב `if` וגם ב `else`, כך שהיא תתבצע בכל מקרה. כתוצאה מכך, נוצרה חזרה על קוד שניתן להימנע ממנה באמצעות הוצאת הפקודה החוצה.

### שגיאה:

```

if number > 0:
    print("Positive!")
    count += 1
else:
    print("Negative!")
    count += 1
  
```

### תיקון:

```

if number > 0:
    print("Positive!")
else:
    print("Negative!")
count += 1
  
```

השוואת ביטוי בוליאני לערך מיידי

כאן אפילו PyCharm כועס. אין צורך להשוות את תוצאתו של ביטוי בוליאני לtrue או לFalse. התנאי מתייחס ישירות לתוצאת הביטוי ופועל על פיו.

שגיאה:

```
can_enter = True
if can_enter == True:
    print("You are in!")
else:
    print("You can't enter!")
```

תיקון:

```
can_enter = True
if can_enter:
    print("You are in!")
else:
    print("You can't enter!")
```

חוסר התייחסות לכל תנאי בפני עצמו

טעות קלאסית.

שגיאה:

```
number = 1
if number == 1 or 2 or 3:
    print("Good!")
else:
    print("Bad!")
```

תיקון:

```
number = 1
if number == 1 or number == 2 or number == 3:
    print("Good!")
else:
    print("Bad!")
```

הקוד שבדוגמה מנסה לבדוק אם הוזן 1,2 או 3. בכל צד של or/and צריך להיות תנאי מלא.

## תנאים מתקדמים

:MATCH

הגדרה

פקודה המאפשרת הצלבה של ערך עם רשימה של ערכים אפשריים. כל אחד מהערכים נקרא case.

מתי משתמשים

כשנרצה לבדוק האם ערך מסוים **שווה במדויק** למספר אפשרויות של ערכים, **ולא לטווח ערכים**.

איך משתמשים

לפניהם הסינטקס:

התאם בין משתנה לבין:

מקרה ראשון:

קטע קוד שירוצ' רק אם המשתנה שווה למקרה הראשון

מקרה שני:

קטע קוד שירוצ' רק אם המשתנה שווה למקרה השני

מקרה אחר:

דוגמות קוד

```
match favorite_animal:
    case "dog":
        print("Dog says Woof!")
    case "cat":
        print("cat says Meow!")
    case _:
        print("This was not one of the options.")
```

תנאי מקונן:

הגדרה

**תנאי מקונן** – פקודת תנאי שנמצאת בתוך פקודת תנאי אחרת.

מתי משתמשים

כשנרצה לבדוק תנאי אך ורק במידה ותנאי אחר מתקיים.

איך משתמשים

לפניכם הסינטקס:

אם ביטוי בוליאני:

אם ביטוי בוליאני נוסף:

קטע קוד שירוף רק אם ערכי הביטוי הבולאני  
וגם הביטוי הבוליאני הנוסף הם True

אחרת:

דוגמות קוד

```
if holiday_summer:
    if even_day:
        print("Beach")
    else:
        print("Football")
else:
    print("Sleep")
```

דגשים

- בתכנות יש מספר דרכים להגיע לאותה התוצאה. לעיתים, ניתן לפתור תרגילים גם באמצעות תנאי מקונן וגם באמצעות תנאים של elif.

עיצוב קוד:

הגדרות


**עיצוב קוד** - מדד לכמה הקוד מסודר, בעל חלוקה לוגית נכונה, מכיל שימוש נכון בכלים שנלמדו, קצר וללא חזרה על קוד.

כללים לעיצוב קוד איכותי

1. נשתמש נכון בכלים שלמדנו.
2. נמנע כמה שאפשר מחזרות על קוד.
3. נקפיד על סדר לוגי הגיוני.
4. נמנע ממשתנים ופעולות מיותרות.

דוגמות קוד


1. נשתמש נכון בכלים שלמדנו.



```


if number <= 20:
    if number % 5 == 0:
        print("dividable by 5")
    elif number % 2 == 0:
        print("number is even")
    print("not even and not dividable by 5")
  
```

2. נמנע כמה שאפשר מחזרות על קוד.



```


if number > 0:
    print("Positive!")
    count += 1
else:
    print("Negative!")
    count += 1
  
```



```

if number > 0:
    print("Positive!")
else:
    print("Negative!")
count += 1
  
```

3. נקפיד על סדר לוגי הגיוני.



```

age = int(input("Enter your age: "))
age = age - 18
if age >= 0:
    print("Age is 18+")
else:
  
```



```
print("Age is 18-")
age = age + 18
```

```
age = int(input("Enter your age: "))
if age >= 18:
    print("Age is 18+")
else:
    print("Age is 18-")
```

4. נמנע ממשתנים ופעולות מיותרות.

```
private_name = input("Enter your private name: ")
family_name = input("Enter your family name: ")
full_name = private_name + family_name
print("Hi " + private_name + ", welcome to code design session")
```

```
private_name = input("Enter your private name: ")
print("Hi " + private_name + ", welcome to code design session")
```

## שליבים בפתרון בעיה

### דיבאג

#### הגדרות

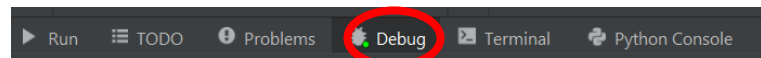
**באג** - בעברית "תקל", הוא תקלה במערכת מבוססת תכנה, שמתנהגת בצורה **שגויה** או **לא צפויה**.

**דיבאג** - בעברית "ניפוי תקלים", הוא תהליך שיטתי של איתור והפחתת באגים בתכנת מחשב.

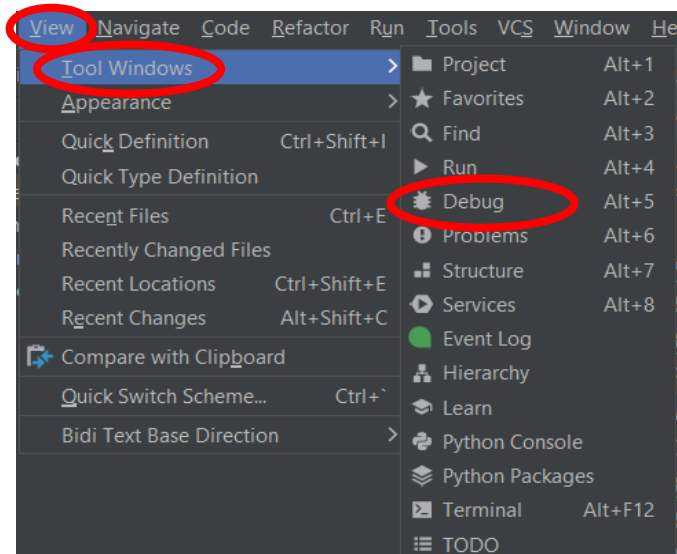
**נקודת עצירה (breakpoint)** - נקודת עצירה מכוונת או נקודת השהייה בתוכנית מחשב, לצורכי דיבאג.

תהליך הדיבאג - איך זה נראה

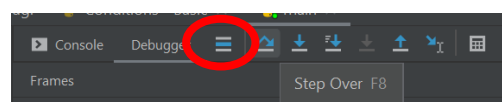
התחלת תהליך הדיבאג:



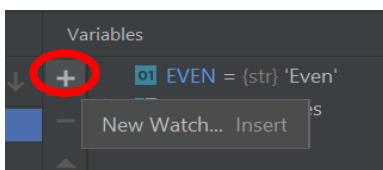
במידה וכפתור Debug לא קיים אצלך, כך מוסיפים אותו:



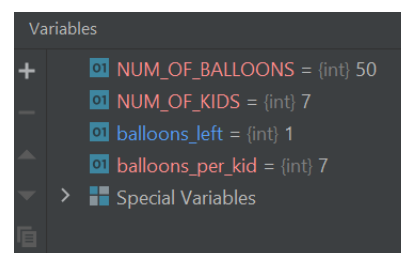
לחיצה על כפתור ה- Step Over (F8), כך נתקדם בתוך הקוד:



כדי להוסיף משתנה חדש לחלון, יש ללחוץ על ה+ ולרשום את שם המשתנה:



חלון ה-watches:



## דגשים

- הדיבאג הוא כלי חשוב שיעזור לכם לפתור תקלות בקוד שלכם!
- זוהי מיומנות ולכן בפעמים הראשונות שתתנסו בתהליך הוא יקח לכם יותר זמן.

## סטנדרטים וקריאות

### הגדרות

**סטנדרטים** - סט של כללים לכתיבת הקוד, שנועדו לשמור על הקוד אחיד ונוח כך שהמתכנת ושאר חברי הצוות יבינו אותו.

**קריאות** - מדד לכמה הקוד קריא ומובן.

## יתרונות קוד קריא

1. תחזוקת הקוד – יכולת שלילפת הקוד בעתיד ושליטה בו.
2. הגברת השליטה בקוד – על ידי הימנעות מסרבול.
3. עבודת צוות – קוד קריא יקל על שילוב נכון ומהיר יותר על החלקים במערכת של צוות.
4. הרחבה – הרחבת הקוד בפשטות בהמשך.

## כללים לכתיבת קוד קריא

1. שמות משמעותיים של משתנים.
2. לא נשתמש במספרים מיידיים בקוד.
3. אם המספר המייד הוא מספר עם משמעות מתמטית שלא משתנה פשוט נשתמש בו.

## דוגמות קוד

כלל 1:

```
a = 18
```



```
if age == 18:
```

```
age = 18
```



```
if age == MAX_AGE:
```

כלל 3:

```
radius = 3.14 * radius
```



## כללים לכתיבת הערות

1. אין צורך בהערות ברורות מאליהן.
2. אם יש צורך בהרבה הערות בקוד – זה סימן שהקוד לא קריא.
3. נוסף הערה לפני קטע קוד שנדרש להסביר ורק אם שמות המשתנים לא מספיקים.

## עיצוב קוד

### הגדרות

**עיצוב קוד** - מדד לכמה הקוד מסודר ובעל חלוקה לוגית נכונה, חסכוני במשאבים בעת הרצתו, יעיל וקצר ללא חזרה על קוד.

### כללים לעיצוב הקוד

1. שימוש נכון בכלים שנלמדו – למשל, מתי יש להשתמש בתנאי בסיסי, תנאי מקונן, elif וכו'.
2. הימנעות מחזרה על קוד – על מנת לחסוך בשורות קוד, יצירת סדר בקוד והקלה על המתכנת במידה ויהיה צורך לשנות את הקוד (המתכנת יצטרך לשנות את הקוד רק במקום אחד – מניעת טעויות וסיבוכים).
3. הקפדה על סדר לוגי הגיוני – אתחול משתנים, יצירת בלוקים של תנאים, קבלת קלט מהמשתמש ועוד. יש לבצע את הפעולות בסדר ההגיוני ביותר כך שהקוד יהיה ברור למתכנתים אחרים.

4. הימנעות ממשתנים ופקודות מיותרות – שני הדברים הללו דורשים מהמחשב שלכם משאבים. נשאף לצמצם את מספר המשתנים והפקודות שלנו על מנת להקל עליו, ולכתוב קוד ברור יותר. לעיתים ניצור מספר משתנים רב על מנת להגדיל את קריאות הקוד – זהו קו דק.

#### שליבים בפתרון בעיה

##### שלב 1 – הבנת הבעיה

1. קריאת הבעיה.
2. הדגשת חלקים חשובים.
3. קריאה נוספת של הבעיה תוך התייחסות לחלקים המודגשים.

##### שלב 2 – מקרי קצה

1. מקרי קצה הם מקרים מיוחדים בריצה שנצטרך לבדוק שהם נתמכים ע"י המערכת שלנו. לדוגמה: קלט לא תקין של המשתמש.
2. נכתוב לנו מקרים כדי שבזמן כתיבת הקוד נזכור להתייחס אליהם, ונסדר אותם בטבלה.

##### שלב 3 – תכנון האלגוריתם

1. יצירת טבלאות עבור המשתנים והקבועים שנשתמש בהם בתוכנית.
2. בניית תבנית מפורטת ככל הניתן של האלגוריתם.

<u>שם המשתנה</u>	<u>טיפוס המשתנה</u>	<u>תפקיד המשתנה</u>

<u>שם הקבוע</u>	<u>ערך הקבוע</u>	<u>תפקיד הקבוע</u>

##### שלב 4 – קידוד

1. כתיבת הקוד בpython.
2. בדיקה לאורך הכתיבה שהקוד עובד (ולא רק בסוף).

##### שלב 5 – בדיקות

1. בשלב זה נבדוק את כל מקרי הקצה!

2. במידה ולא התייחסנו למקרה קצה, נוסיף את הטיפול במקרה זה.
3. יכול להיות שהתיקון שלנו יצר באגים נוספים, לכן נריץ שוב את כל מקרי הריצה שקשורים למקרה הקודם שנכשל

## לולאות

מפגשי לולאות

## FOR לולאות

הגדרות

**לולאת for** - קטע קוד החוזר על עצמו מספר פעמים מוגדר מראש.

מתי משתמשים

כשנרצה לחזור על קטע קוד מסוים מספר פעמים מוגדר מראש.

איך משתמשים

לפניכם הסינטקס לכתיבת לולאת for.

```
for scoop in range(number_of_scoops):
```

עבור **מספר פעמים** מוגדר מראש:

**repeated code**

**קטע קוד** החוזר על עצמו

דגשים

1. בפונקציית range() רצים מ-a עד המספר אחד לפני b.  
למשל: range(1, 5) --> 1,2,3,4
2. בפונקציית range() אם לא נציין את a, פייתון יתייחס אליו כמו 0.  
למשל: range(5) --> range(0, 5) --> 0,1,2,3,4
3. בעולם התכנות מתחילים לספור מ-0.
4. בפונקציית range() ניתן להכניס פרמטר נוסף המציין את גודל הקפיצה בין המספרים בטווח.  
אם לא נכניס אותו, פייתון יתייחס אליו כ-1.  
למשל: range(3, 10, 3) --> 3, 6, 9  
range(3, 10) --> 3, 4, 5, 6, 7, 8, 9

דוגמות קוד

לולאה שמדפיסה את כל המספרים בין 2 ל-9.



```
for current_number in range(2, 10):
    print(current_number)
```

לולאה שמחשבת את הסכום של כל המספרים עד 8.

```
sum = 0
for index in range(8):
    sum += index
print(sum)
```



לולאה שמדפיסה את הערך של 5!

```
factorial = 1
for num in range(1, 6):
    factorial *= num
print(factorial)
```



## BREAK

### הגדרות

**break** - פקודה אשר עוצרת את ריצת הלולאה ומעבירה לקוד שאחריה.

מתי משתמשים

כשנרצה להפסיק את ריצת הלולאה.

איך משתמשים

בגוף הלולאה, נרשום תנאי שיבדוק האם יש לעצור את ריצת הלולאה.

```
for scoop in range(number_of_scoops):
```



בגוף התנאי נרשום את המילה break.

```
    some code
```

```
    if condition:
```

### דוגמות קוד

לולאה שקולטת 5 מספרים ומחשבת את סכומם.

אם נקלט מספר שלילי יש להפסיק את הקלט.

```

sum = 0
for i in range(5):
    number = int(input("Enter a number: "))
    if number < 0:
        break
    sum += number
print(sum)

```

## ריצה על מחרוזות

מתי משתמשים

כשנרצה לעבור על כל התווים של מחרוזת מסוימת.

איך משתמשים

לפניכם דוגמה ללולאה שמדפיסה את כל האותיות במחרוזת "nitzanim".

```

for letter in
    "nitzanim":
    print(letter)

```

דוגמות קוד

לולאה שמחשבת ומדפיסה כמה פעמים מופיעה האות "i" במחרוזת "Nitzanim rules!".

```

count = 0
string = "Nitzanim rules!"
for letter in string:
    if letter == "i":
        count += 1
print(count)

```



## לולאות FOR מקוננות

הגדרות

**לולאה מקוננת** – לולאה בתוך לולאה. במקרה זה, קטע הקוד שחוזר על עצמו הינו לולאה.

עברית מבנית:

עבור מספר פעמים מוגדר מראש:  
 עבור מספר פעמים מוגדר מראש:  
 - קטע קוד -  
 קטע קוד  
 שחוזר על  
 עצמו

Python:

for outer in range (outer\_numbers):  
 for inner in range (inner\_numbers):  
 -code-  
 קטע קוד  
 שחוזר על  
 עצמו

דוגמאות קוד

הדפסת ריבוע סולמיות:

```

SIZE = 10
for outer in range(SIZE):
    for inner in range(SIZE):
        print("#", end = " ")
    print()
  
```

הדפסת לוח הכפל:

```

SIZE = 10
for outer in range(1, SIZE + 1):
    for inner in range(1, SIZE + 1):
        print(outer * inner, end = " ")
    print()
  
```

בדיקת נוכחות בבית הספר:

```

GROUPS = 5
STUDENT_IN_GROUP = 10
for group in range(1, GROUPS + 1):
    print("Group " + str(group) + ":")
  
```



```
for student in range(1, STUDENT_IN_GROUP + 1):
    input("Is student number " + str(student) + " is here? ")
```

## לולאות WHILE

### הגדרות

**לולאת while** - קטע קוד החוזר על עצמו מספר פעמים כל עוד תנאי מסוים מתקיים.

### מתי משתמשים

כשנרצה לחזור על קטע קוד מסוים כל עוד תנאי מסוים מתקיים. במילים אחרות, אנחנו צריכים את לולאת while כדי לחזור על קוד מספר פעמים שאינו מוגדר מראש.

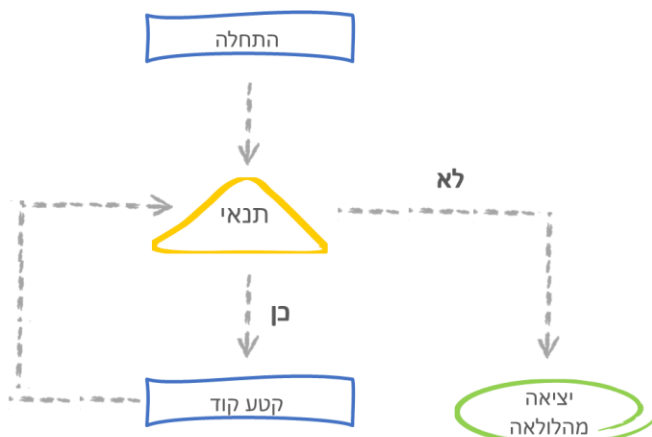
### איך משתמשים

לפניכם הסינטקס לכתיבת לולאת while.

```
number = 0
while number < 10:
    print(number)
    number += 1
```

1. **While** - פקודת ציון לשימוש בלולאה זו
2. **Number < 10** - ביטוי בוליאני המהווה את תנאי הלולאה, כל עוד הביטוי הוא True הלולאה תבצע איטרציה. בפעם הראשונה שהביטוי יהיה False, האיטרציה לא תתבצע והלולאה תסתיים.
3. **קוד שבתוך הלולאה** (הזחה פנימה) - כמו ב-for. קוד זה יבוצע בכל פעם שהתנאי יהיה אמת וניכנס ללולאה.
4. **אתחול המונה והגדלתו** - בניגוד ללולאות for אשר שם המונה מאותחל ומעודכן אוטומטית בתוך הטווח המוגדר לו, כאן אין הדבר קורה. עקרונית, אם לא נשנה דבר, התנאי ישאר זהה תמיד והלולאה לא תעצור. כדי שכמות הריצות של הלולאה תהיה מוגבלת עלינו לבצע שינוי בתנאי הלולאה. במקרה זה - המשתנה number גדל ב-1 בכל ריצה, לבסוף יגיע לערך 10, תנאי הלולאה יהיה false והלולאה תעצר.

### תרשים זרימה:



בתחילת כל איטרציה, נבדק התנאי. אם הוא אמת - יורץ קטע הקוד, נחזור לבדיקת הלולאה וכן הלאה.

בפעם הראשונה שהתנאי יהיה שקר, לא יורץ קטע הקוד והלולאה תעצר.

אם בתנאי העצירה קיים שימוש במשתנה מסוים, על המתכנת להגדיר אותו לפני תחילת הלולאה ולעדכן אותו בגוף הלולאה. אם משתנה זה לא יעודכן, הלולאה תהיה אינסופית – ריצת הלולאה לא תסתיים מכיוון שתנאי העצירה לעולם לא יתקיים.

דוגמות קוד

קליטת מספרים עד קבלת 0:

```
number = int(input("Insert a number: "))
while number != 0:
    number = int(input("Insert a number: "))
```

זריקת קוביות עד קבלת דאבל 6:

```
import random
dice_1 = random.randint(1, 6)
dice_2 = random.randint(1, 6)
while not (dice_1 == 6 and dice_2 == 6):
    print("We got", dice_1, dice_2)
    dice_1 = random.randint(1, 6)
    dice_2 = random.randint(1, 6)
print("We got it!", dice_1, dice_2)
```

לולאה קולטת שם משתמש וסיסמא עד שנקלט שם משתמש ששווה ל"favorite\_food" וסיסמא ששווה ל"sushi".

```
username = "favorite_food"
password = "sushi"
username_guess = input("Enter your username:")
password_guess = input("Enter your password:")

while password_guess != password or username_guess != username:
    print("Wrong data.")
    username_guess = input("Enter your username:")
    password_guess = input("Enter your password:")
```



WHILE VS FOR

הגדרות

**לולאת for** - קטע קוד החוזר על עצמו מספר פעמים מוגדר מראש.

**לולאת while** - קטע קוד החוזר על עצמו מספר פעמים כל עוד תנאי מסוים מתקיים.

מתי משתמשים ב-**FOR** ומתי ב-**WHILE**?

כשנרצה לכתוב קטע קוד, נשאל את עצמנו כמה פעמים הוא ירוץ.

אם אנחנו יכולים להגדיר כמה פעמים הוא ירוץ, נשתמש בלולאת **for** ותנאי העצירה יהיה כמות הפעמים שהלולאה תרוץ (גם אם היא משתנה).

אחרת, ננסה להשלים את המשפט:

הלולאה רצה כל עוד ...

ניקח את המילים שהשלימו את המשפט ונהפוך אותן לביטוי בוליאני – זה תנאי העצירה של לולאת ה-**while** שלנו.

דוגמות

סוג הלולאה	תרגיל
While	1. כתבו תוכנית הקולטת כל פעם זוג מספרים ומדפיסה את המכפלה שלהם. עד לקליטת המספר 1- עבור אחד מהם.
For	2. קבל קלט מהמשתמש - מספר שלם וחיובי A. כתבו תוכנית המדפיסה את לוח הכפל במימדי AXA.
While	3. כתבו תכנית המקבלת מספר זוגי מהמשתמש ומדפיסה את החצי שלו. יש לוודא כי הקלט תקין.
For	4. כתבו תכנית המדפיסה מספר רנדומלי של פעמים את המשפט "I love israel"