

לתרגול אדפטיבי, בדיקת תשובות אוטומטית, ועוזר בינה מלאכותית חכם, היכנסו ל-www.TIRGUL.ai

הדפסת חישובים

בלי גרשיים המחשב מחשב ומדפיס את התוצאה. עם גרשיים הוא מדפיס את התרגיל כטקסט. הפעולות: + - * . כפל לפני חיבור וחיסור, וסוגריים לפני הכול.

```
Console.WriteLine(2 + 3);  
Console.WriteLine("2 + 3");  
Console.WriteLine(3 + 1 * 2);  
Console.WriteLine((3 + 1) * 2);
```

הפלט:

```
5  
2 + 3  
5  
8
```

חיבור רכיבי הדפסה באמצעות +

כשאחד הצדדים של ה- + הוא מחרוזת, ה- + מצמיד טקסט. כששני הצדדים מספרים, הוא מחבר.

```
Console.WriteLine("Sum: " + (2 + 3));  
Console.WriteLine("Total: " + 4 * 5);
```

הפלט:

```
Sum: 5  
Total: 20
```

שימו לב: "Sum: " + 1 + 2 מדפיס Sum: 12, כי ההצמדה נעשית משמאל לימין. עם סוגריים, "Sum: " + (1 + 2) יודפס Sum: 3.

הערות

מה שאחרי // הוא הערה. המחשב מדלג עליה.

```
// זאת הערה, המחשב מדלג עליה  
Console.WriteLine("Hello"); // גם זאת
```

איפה כותבים את הקוד

הקוד נכתב בין הסוגריים המסולסלים של Main. את השאר התוכנה יוצרת לבד.

```
static void Main(string[] args)  
{  
    // הקוד שלכם נכתב כאן  
}
```

שתי פקודות ההדפסה

Console.Write מדפיסה ונשארת באותה שורה. Console.WriteLine מדפיסה ויורדת שורה. כל פקודה מסתיימת ב- ;.

```
Console.Write("Hi ");  
Console.Write("there");  
Console.WriteLine("!");  
Console.WriteLine("Bye");
```

הפלט:

```
Hi there!  
Bye
```

Console.WriteLine(); בלי כלום בסוגריים רק יורדת שורה.

מחרוזת: מה שבין הגרשיים

מחרוזת מודפסת בדיוק כמו שהיא, בלי הגרשיים. המחשב לא מוסיף רווחים: רווח מודפס רק אם כתבתם אותו בתוך הגרשיים.

```
Console.Write("Hi");  
Console.WriteLine("there");  
Console.Write("Hi ");  
Console.WriteLine("there");
```

הפלט:

```
Hithere  
Hi there
```

לתרגול אדפטיבי, בדיקת תשובות אוטומטית, ועוזר בינה מלאכותית חינם, היכנסו ל-www.TIRGUL.ai

2 * n הוא 4.

```
string s2 = "2";  
int n = int.Parse(s2);  
double d = double.Parse("3.5");  
bool b = bool.Parse("true");  
char c = char.Parse("a");
```

המרה בסוגריים

(int) חותכת את מה שאחרי הנקודה, בלי לעגל. int
נכנס ל- double גם בלי המרה, ופעולה בין int
ל- double נותנת double. המרה חלה רק על מה
שצמוד אליה, ולכן חישוב שלם עוטפים בסוגריים.

```
double d = 9.7;  
int n = (int)d;  
Console.WriteLine(n);  
Console.WriteLine(n + 0.5);  
Console.WriteLine((int)(d * 2));  
Console.WriteLine((int)d * 2);
```

הפלט:

9
9.5
19
18

Parse פועלת רק על מחרוזות. המרה בסוגריים פועלת
רק על מספרים ועל תווים.

תווים וקוד Unicode

לכל תו יש מספר, קוד ה-Unicode שלו. char נכנס
ל- int גם בלי המרה, ובכיוון ההפוך חייבים (char).
כאן code הוא 97, ו- next הוא התו 'b'.

```
char ch = 'a';  
int code = ch;  
char next = (char)(code + 1);
```

חמשת סוגי המשתנים

משתנה הוא קופסה בזיכרון ששומרת ערך מסוג אחד.

```
int num = 90;           // מספר שלם  
double time = 3.8;      // מספר ממשי  
bool isOn = true;       // true או false  
string name = "noa";    // מחרוזת, בגרשיים  
char letter = 'A';      // תו אחד, בגרש בודד
```

יצירה, השמה והדפסה

יוצרים משתנה בכתיבת הסוג והשם, ומכניסים ערך
באמצעות =. אפשר גם באותה שורה. חישוב מתבצע
פעם אחת, ברגע ההשמה: שינוי מאוחר של num7 לא
משנה את theSum.

```
int num7;  
num7 = 90;  
int num8 = 100;  
int theSum = num7 + num8;  
num7 = 1;  
Console.WriteLine(theSum);
```

הפלט:

190

חיבור משתנים ומחרוזות באמצעות +

```
string name = "Noa";  
int age = 16;  
Console.WriteLine(name + " is " + age);  
Console.WriteLine("Next: " + (age + 1));
```

הפלט:

Noa is 16
Next: 17

המרה ממחרוזת: Parse

מחרוזת שכתוב בה מספר היא עדיין טקסט, ואי אפשר
לחשב איתה. Parse ממירה אותה לסוג המבוקש, ואז

לתרגול אדפטיבי, בדיקת תשובות אוטומטית, ועוזר בינה מלאכותית חכם, היכנסו ל-www.TIRGUL.ai

בלי ההמרה, `int age = Console.ReadLine();` היא שגיאה.

סדר הקליטות

כל `Console.ReadLine()` קולטת שורה אחת. הערך הראשון שהוקלד נכנס לפקודת הקלט הראשונה בקוד, השני לשנייה, וכך הלאה.

תוכנית קלט, חישוב, פלט

```
Console.WriteLine("Enter a number");  
int n1 = int.Parse(Console.ReadLine());  
Console.WriteLine("Enter a number");  
int n2 = int.Parse(Console.ReadLine());  
Console.WriteLine("Sum: " + (n1 + n2));
```

הפלט, כשהשתמש הקליד 15 ואחר כך 900:

```
Enter a number  
Enter a number  
Sum: 915
```

קליטת מחרוזת

`Console.ReadLine()` עוצרת את התוכנית עד שהשתמש מקליד ולוחץ Enter, ומחזירה את מה שהוקלד כמחרוזת. לפני הקליטה מדפיסים הודעה שמסבירה למשתמש מה להקליד.

```
Console.WriteLine("Enter your name");  
string userName = Console.ReadLine();  
Console.WriteLine("Hi " + userName + " :");
```

הפלט, כשהשתמש הקליד noa:

```
Enter your name  
Hi noa :)
```

קליטת מספר

הקלט מגיע תמיד כמחרוזת. כדי לקלוט מספר עוטפים את `Console.ReadLine()` בהמרה לסוג המבוקש.

```
int age = int.Parse(Console.ReadLine());  
double h = double.Parse(Console.ReadLine());
```

לתרגול אדפטיבי, בדיקת תשובות אוטומטית, ועוזר בינה מלאכותית חינם, היכנסו ל-www.TIRGUL.ai

הפלט:

```
123
12
4
34
```

שימוש חוזר על אותו מספר

ספרה מהאמצע: קודם מסירים בחילוק את מה שמימין לה, ואז לוקחים אותה במודולו 10. ספרת העשרות של n היא $n / 10 \% 10$. כש- n הוא 234, $234 / 10$ הוא 23, ו- $23 \% 10$ הוא 3.

בדיקת התחלקות

$a \% b$ שווה 0 בדיוק כש- a מתחלק ב- b בלי שארית. $3 \% 9$ הוא 0, ולכן 9 מתחלק ב-3. $3 \% 8$ הוא 3, ולכן 8 לא מתחלק ב-3. מספר זוגי הוא מספר ש- $n \% 2$ שלו שווה 0.

חילוק שלמים מול חילוק מדויק

בין שני `int` מתבצע חילוק שלמים, גם אם התוצאה נשמרת ב-`double`. כדי לקבל תוצאה מדויקת המספרים עצמם צריכים להיות ממשיים. במשתנים מסוג `int` ממירים אחד מהם: `(double)a / b`.

```
Console.WriteLine(7 / 2);
Console.WriteLine(7.0 / 2.0);
double d = 7 / 2;
Console.WriteLine(d);
```

הפלט:

```
3
3.5
3
```

חילוק שלמים /

כששני הצדדים `int`, התוצאה היא כמה פעמים שלמות המספר הימני נכנס במספר השמאלי. מה שאחרי הנקודה נזרק, בלי עיגול.

```
Console.WriteLine(5 / 2);
Console.WriteLine(9 / 10);
```

הפלט:

```
2
0
```

שארית החלוקה % (מודולו)

מה שנשאר במספר השמאלי אחרי שהורדנו ממנו כמה שיותר פעמים שלמות את המספר הימני.

```
Console.WriteLine(5 % 2);
Console.WriteLine(14 % 5);
Console.WriteLine(3 % 10);
```

הפלט:

```
1
4
3
```

חזקות של 10: הסרה וחילוף של ספרות

חילוק ב-10 מסיר את הספרה הימנית, וחילוק ב-100 מסיר שתיים. מודולו 10 נותן את הספרה הימנית, ומודולו 100 את שתי הימניות.

```
int n = 1234;
Console.WriteLine(n / 10);
Console.WriteLine(n / 100);
Console.WriteLine(n % 10);
Console.WriteLine(n % 100);
```

לתרגול אדפטיבי, בדיקת תשובות אוטומטית, ועוזר בינה מלאכותית חינם, היכנסו ל-www.TIRGUL.ai

```
if (age >= 13 && age <= 19)
{
    Console.WriteLine("Teenager");
}
if (day == 6 || day == 7)
{
    Console.WriteLine("Weekend");
}
```

שילוב של שניהם:

תנאי מורכב מחליפים כל השוואה ב-true או ב-false, ומצמצמים שלב אחר שלב.

תנאי מקונן

תנאי בתוך תנאי: if בתוך if, או if בתוך else. התנאי הפנימי נבדק רק אם נכנסנו לבלוק שהוא כתוב בתוכו, והוא לא חייב else. else שייך תמיד ל-if שהבלוק שלו נסגר ממש לפניו. שלוש אפשרויות ויותר כותבים בעזרת else שבתוכו if נוסף.

```
if (x > 0)
{
    Console.WriteLine("Positive");
    if (x > 100)
    {
        Console.WriteLine("Big");
    }
}
else
{
    if (x == 0)
    {
        Console.WriteLine("Zero");
    }
    else
    {
        Console.WriteLine("Negative");
    }
}
```

else-ו if

אם התנאי שבסוגריים נכון, מתבצע הבלוק של ה-if. אחרת מתבצע הבלוק של ה-else. ה-else אינו חובה. בכל מקרה הקוד ממשיך אחרי התנאי.

```
if (grade > 90)
{
    Console.WriteLine("Great");
}
else
{
    Console.WriteLine("Keep going");
}
Console.WriteLine("Bye");
```

הפלט, כש-grade שווה 80:

```
Keep going
Bye
```

אופרטורי ההשוואה

סימן	משמעות	סימן	משמעות
>	גדול מ	<	קטן מ
>=	גדול או שווה	<=	קטן או שווה
==	שווה	!=	שונה

= אחד מכניס ערך למשתנה. בתנאי בודקים שוויון עם ==.

תנאי מורכב: || ו- &&

|| פירושו "או": התנאי נכון כשלפחות אחד הצדדים נכון. && פירושו "וגם": התנאי נכון רק כששני הצדדים נכונים. כמו בחשבון, סוגריים מחושבים קודם.

לתרגול אדפטיבי, בדיקת תשובות אוטומטית, ועוזר בינה מלאכותית חינם, היכנסו ל-www.TIRGUL.ai

סוג התוצאה

`Math.Floor` - `Math.Round`, `Math.Sqrt`, `Math.Pow`
מחזירות `double`. לכן שומרים את התוצאה ב- `double`,
או ממירים אותה.

```
double len = Math.Sqrt(h * h + w * w);  
int root = (int)Math.Sqrt(9);
```

השורה `int root = Math.Sqrt(9);` בלי ההמרה היא
שגיאה.

מספר אקראי

את השורה הראשונה כותבים פעם אחת בתוכנית.
`Next(1, 11)` מגרילה מספר שלם מ-1 עד 10: המספר
השני אינו כלול. `Next(5)` עם מספר אחד מגרילה מ-0
עד 4.

```
Random randomNum = new Random();  
int num1 = randomNum.Next(1, 11);  
int num2 = randomNum.Next(5);  
if (num1 == 1)  
{  
    Console.WriteLine("You won");  
}
```

ב- `Next(1, 11)` יש 10 תוצאות אפשריות, ולכן הסיכוי
ש- `num1` יהיה 1 הוא 1 מתוך 10.

הפעולות של Math

המחלקה `Math` היא אוסף של קיצורי דרך לחישובים.

פעולה	מה היא מחזירה
<code>Math.Abs(-7)</code>	ערך מוחלט: 7
<code>Math.Pow(3, 2)</code>	3 בחזקת 2: 9
<code>Math.Sqrt(25)</code>	שורש: 5
<code>Math.Min(4, 9)</code>	הקטן מבין השניים: 4
<code>Math.Max(4, 9)</code>	הגדול מבין השניים: 9
<code>Math.Round(2.7)</code>	עיגול לשלם הקרוב: 3
<code>Math.Floor(2.7)</code>	עיגול כלפי מטה: 2

שימוש בתוצאה

אפשר להדפיס את התוצאה ישירות או לשמור אותה
במשתנה. בתוך הסוגריים אפשר לכתוב מספר, משתנה,
חישוב, או פעולת `Math` אחרת.

```
int x = -9;  
Console.WriteLine(Math.Abs(x));  
Console.WriteLine(Math.Sqrt(Math.Abs(x)));  
int big = Math.Max(x * 2, 5);  
Console.WriteLine(big);
```

הפלט:

9
3
5

לתרגול אדפטיבי, בדיקת תשובות אוטומטית, ועוזר בינה מלאכותית חינם, היכנסו ל-www.TIRGUL.ai

```
int n = int.Parse(Console.ReadLine());
int count = 0;
for (int i = 0; i < n; i++)
{
    int x = int.Parse(Console.ReadLine());
    if (x > 90)
    {
        count++;
    }
}
Console.WriteLine(count);
```

סוכם וממוצע

משתנה שצובר ערכים. מאתחלים אותו ל-0 לפני הלולאה, ומוסיפים לו בכל סבב. $sum += x$ הוא קיצור של $sum = sum + x$. ממוצע הוא הסכום חלקי הכמות. כדי שהחילוק לא יהיה חילוק שלמים, מגדירים את הסוכם כ- `double`.

```
double sum = 0;
for (int i = 0; i < n; i++)
{
    int x = int.Parse(Console.ReadLine());
    sum += x;
}
Console.WriteLine("Sum: " + sum);
Console.WriteLine("Average: " + sum / n);
```

משתנה שצובר מכפלה מאתחלים ל-1 ולא ל-0.

מקסימום ומינימום

`max` מחזיק את הערך הגדול ביותר שנקלט עד עכשיו, ו-`if` מעדכן אותו כשנקלט ערך גדול ממנו. ערך ההתחלה נקבע לפי תחום הערכים: למקסימום ערך שקטן מכל קלט אפשרי, ולמינימום ערך שגדול מכל קלט אפשרי. בציונים מ-0 עד 100 אלה 0 ו-101.

מבנה הלולאה

```
for (int i = 0; i < 3; i++)
{
    Console.WriteLine("Hi " + i);
}
Console.WriteLine("Done");
```

הפלט:

```
Hi 0
Hi 1
Hi 2
Done
```

בכותרת שלושה חלקים, ונקודה-פסיק ביניהם:

- **אתחול** `int i = 0`: מתבצע פעם אחת, בהתחלה.
- **תנאי** `i < 3`: נבדק לפני כל סבב. כשהוא לא נכון, הלולאה מסתיימת והקוד ממשיך אחריה.
- **קידום** `i++`: מתבצע בסוף כל סבב. זהה ל-`i = i + 1`.

כותרות נפוצות

- `for (int i = 1; i <= 5; i++)` מ-1 עד 5, כולל 5.
- `for (int i = 0; i < 10; i = i + 2)` בקפיצות של 2, מ-0 עד 8.
- `for (int i = 5; i > 0; i--)` ספירה לאחור, מ-5 עד 1.
- `for (int i = 0; i < n; i++)` מתבצע `n` פעמים. את `n` אפשר לקלוט מהמשתמש.

אם התנאי לא נכון כבר בבדיקה הראשונה, גוף הלולאה לא מתבצע אפילו פעם אחת.

מונה

משתנה שסופר כמה פעמים קרה משהו. מאתחלים אותו ל-0 לפני הלולאה, מגדילים אותו ב-1 בתוך `if`, ומדפיסים אותו אחרי הלולאה. `count++` הוא קיצור של `count = count + 1`.

אתחול לפי הערך הראשון

כשתחום הערכים לא ידוע, למשל כשיש גם מספרים שליליים, קולטים את הערך הראשון לפני הלולאה ומאתחלים לפיו. הלולאה מתחילה מ-1, כי ערך אחד כבר נקלט.

```
int first = int.Parse(Console.ReadLine());
int max = first;
int min = first;
for (int i = 1; i < n; i++)
{
    int x = int.Parse(Console.ReadLine());
    if (x > max)
    {
        max = x;
    }
}
```

```
int max = 0;
int min = 101;
for (int i = 0; i < n; i++)
{
    int x = int.Parse(Console.ReadLine());
    if (x > max)
    {
        max = x;
    }
    if (x < min)
    {
        min = x;
    }
}
Console.WriteLine(max + " " + min);
```

כדי לזכור באיזה סבב נמצא המקסימום, שומרים בתוך ה-`if` גם את `i`, למשל `pos = i`. עם `>` נשמר המופע הראשון של המקסימום, ועם `>=` האחרון.

לתרגול אדפטיבי, בדיקת תשובות אוטומטית, ועוזר בינה מלאכותית חינם, היכנסו ל-www.TIRGUL.ai

```
int money = int.Parse(Console.ReadLine());
int sum = 0;
int price = int.Parse(Console.ReadLine());
while (money >= sum + price)
{
    sum += price;
    price = int.Parse(Console.ReadLine());
}
Console.WriteLine(sum);
```

בלי בדיקה מוקדמת, אחרי הלולאה המשתנה כבר חצה את הגבול:

```
int x = 1;
while (x < 100)
{
    x = x * 2;
}
Console.WriteLine(x);
```

הפלט:

128

מבנה הלולאה

`while` פירושו "כל עוד". התנאי נבדק לפני כל סיבוב: אם הוא נכון, הגוף מתבצע וחוזרים לבדוק שוב. אם לא, הקוד ממשיך אחרי הלולאה. `while` הוא כמו `for` שנשאר ממנו רק התנאי: את האתחול כותבים לפני הלולאה, ואת הקידום בסוף הגוף.

```
int i = 0;
while (i < 3)
{
    Console.WriteLine(i);
    i++;
}
```

הפלט:

0
1
2

אם שום דבר בגוף לא משנה את מה שנבדק בתנאי, הלולאה לא תסתיים לעולם.

while עם דגל

דגל הוא משתנה `bool`. מאתחלים אותו לפני הלולאה, ו"מרימים" אותו בתוך הלולאה כשקורה משהו. הלולאה לא נעצרת מיד, אלא רק בבדיקה הבאה של התנאי.

```
bool found = false;
int count = 0;
while (found == false)
{
    int x = int.Parse(Console.ReadLine());
    count++;
    if (x == 99)
    {
        found = true;
    }
}
Console.WriteLine("Tries: " + count);
```

אפשר לשלב את הדגל עם תנאי נוסף, למשל עם זקיף: `while (price != -1 && found == false)`. אחרי לולאה כזאת בודקים את הדגל ב-`if`, כדי לדעת למה היא הסתיימה.

while עם זקיף

זקיף הוא ערך מיוחד שמסמן את סוף הקלט, כאן -1. קולטים פעם אחת לפני הלולאה, ושוב בסוף גוף הלולאה. הזקיף עצמו לא מעובד. כשהזקיף נקלט ראשון, הגוף לא מתבצע כלל.

```
int sum = 0;
int price = int.Parse(Console.ReadLine());
while (price != -1)
{
    sum += price;
    price = int.Parse(Console.ReadLine());
}
Console.WriteLine(sum);
```

while עם מעבר חוסם

בתנאי יש גבול, ובכל סיבוב ערך מתקרב אליו. הלולאה מסתיימת כשהגבול נחצה. כאן התנאי בודק לפני ההוספה, ולכן הסכום לעולם לא עובר את הגבול.

מתי for ומתי while

כשיוודעים מראש כמה פעמים הלולאה תרוץ, נוח יותר `for`. כשלא יודעים מראש, `while`.

while מסוננת קלט

חוזרים על הקלט עד שהוא תקין. התנאי מתאר קלט **לא תקין**. קלט שתקין כבר בניסיון הראשון לא נכנס ללולאה כלל, ואחרי הלולאה הערך תקין בוודאות.

```
int n = int.Parse(Console.ReadLine());
while (n < 1 || n > 10)
{
    Console.WriteLine("Try again");
    n = int.Parse(Console.ReadLine());
}
Console.WriteLine("OK: " + n);
```

לתרגול אדפטיבי, בדיקת תשובות אוטומטית, ועוזר בינה מלאכותית חינם, היכנסו ל-www.TIRGUL.ai

שימוש בשני המשתנים יחד

```
for (int i = 1; i <= 2; i++)  
{  
    for (int j = 1; j <= 3; j++)  
    {  
        Console.WriteLine(i * j + " ");  
    }  
    Console.WriteLine();  
}
```

הפלט:

```
1 2 3  
2 4 6
```

איפוס משתנה בכל סיבוב חיצוני

משתנה שצריך להתחיל מחדש בכל שורה, כמו סכום של שורה, מאתחלים בתוך הלולאה החיצונית ולפני הפנימית.

```
for (int i = 0; i < 2; i++)  
{  
    int sum = 0;  
    for (int j = 0; j < 3; j++)  
    {  
        sum += int.Parse(Console.ReadLine());  
    }  
    Console.WriteLine(sum);  
}
```

קיבול של כל סוגי הלולאות

אין הגבלה על עומק הקיבול: אפשר for בתוך for, while, ואפשר if בכל רמה. כשמבנה לא ברור, עוקבים אחריו שורה אחרי שורה.

```
int again = int.Parse(Console.ReadLine());  
while (again == 1)  
{  
    // כאן שתי הלולאות שמדפיסות מלבן  
    again = int.Parse(Console.ReadLine());  
}
```

לולאה בתוך לולאה

הלולאה החיצונית קובעת את מספר השורות. הלולאה הפנימית קובעת מה מודפס בכל שורה. על כל סיבוב אחד של החיצונית, הפנימית רצה מההתחלה ועד הסוף. לכל לולאה משתנה בשם אחר. בתוך הפנימית מדפיסים

2- Console.WriteLine(), ואחריה Console.WriteLine(). מורידה שורה.

```
for (int i = 0; i < 2; i++)  
{  
    for (int j = 0; j < 3; j++)  
    {  
        Console.WriteLine(" ");  
    }  
    Console.WriteLine();  
}
```

הפלט:

```
***  
***
```

גבול פנימי שתלוי במשתנה החיצוני

כשהגבול של הפנימית הוא i, אורך השורה משתנה משורה לשורה, ומתקבל משולש.

```
for (int i = 1; i <= 3; i++)  
{  
    for (int j = 0; j < i; j++)  
    {  
        Console.WriteLine(" ");  
    }  
    Console.WriteLine();  
}
```

הפלט:

```
*  
**  
***
```

לתרגול אדפטיבי, בדיקת תשובות אוטומטית, ועוזר בינה מלאכותית חינם, היכנסו ל-www.TIRGUL.ai

פעולות קיצור דרך

הדוגמאות על `string s = "Hello";`

פעולה	מה היא מחזירה
<code>s.Contains("ell")</code>	האם הטקסט מופיע בה: <code>true</code>
<code>s.Substring(1, 3)</code>	3 תווים החל ממיקום 1: <code>ell</code>
<code>s.IndexOf('l')</code>	מיקום המופע הראשון: <code>2</code>
<code>s.IndexOf('l', 3)</code>	החיפוש מתחיל ממיקום 3: <code>3</code>
<code>s.LastIndexOf('l')</code>	מיקום המופע האחרון: <code>3</code>

`IndexOf` ו-`LastIndexOf` מקבלות תו או טקסט: `s.IndexOf("lo")` מחזירה 3. הפעולות לא משנות את המחרוזת עצמה, אלא מחזירות ערך חדש. אות גדולה ואות קטנה הן תווים שונים: `s.Contains("h")` היא `false`.

השוואת מחרוזות

משווים שתי מחרוזות באמצעות `==`.

```
string word = Console.ReadLine();
if (word == "yes")
{
    Console.WriteLine("Great");
}
```

שילוב של כמה פעולות

```
string t = "Dan Cohen";
int space = t.IndexOf(' ');
Console.WriteLine(t.Substring(0, space));
Console.WriteLine(t[space + 1]);
```

הפלט:

Dan
C

אורך ותו במיקום

`Length` מחזירה את אורך המחרוזת. סוגריים מרובעים אחרי שם המחרוזת מחזירים את התו שבמיקום. המיקומים מתחילים ב-0, ולכן המיקום האחרון הוא `Length - 1`. גם רווח הוא תו: הוא נספר ותופס מיקום.

```
string s = "Hello";
Console.WriteLine(s.Length);
Console.WriteLine(s[0]);
Console.WriteLine(s[1]);
Console.WriteLine(s[s.Length - 1]);
```

הפלט:

5
H
e
o

מעבר על כל התווים

`s[i]` הוא `char`, ולכן משווים אותו לתו בגרש בודד.

```
int count = 0;
for (int i = 0; i < s.Length; i++)
{
    if (s[i] == 'l')
    {
        count++;
    }
}
Console.WriteLine(count);
```

הפלט:

2

מעבר מהסוף להתחלה:

```
for (int i = s.Length - 1; i >= 0; i--)
```

לתרגול אדפטיבי, בדיקת תשובות אוטומטית, ועוזר בינה מלאכותית חכם, היכנסו ל-www.TIRGUL.ai

```
static void PrintAvg(double a, double b)
{
    double avg = (a + b) / 2;
    Console.WriteLine(avg);
}
```

```
PrintAvg(6, 7.2);
```

הפלט:

```
6.6
```

פעולה שמחזירה ערך

במקום `void` כותבים את טיפוס הערך המוחזר, ובגוף הפעולה `return` מסיים את הפעולה מיד, גם מתוך לולאה.

```
static int Bigger(int n1, int n2)
{
    if (n1 > n2)
    {
        return n1;
    }
    else
    {
        return n2;
    }
}
```

שימוש בערך המוחזר

הקריאה לפעולה "הופכת" לערך שהוחזר. אפשר לשמור אותו במשתנה, להדפיס אותו, או להשתמש בו בחישוב ובתנאי.

```
int m = Bigger(6, 7);
Console.WriteLine(m);
Console.WriteLine(Bigger(2, 1) * 10);
```

הפלט:

```
7
20
```

בקריאה כמו `Bigger(6, 7)` לבדה, הערך המוחזר לא נשמר ולא מודפס, והוא הולך לאיבוד.

פעולה ראשונה: כתיבה וקריאה

פעולה נכתבת מחוץ ל-`Main`. בכותרת: `static`, טיפוס הערך המוחזר, שם, וסוגריים. `void` אומר שהפעולה לא מחזירה כלום. התוכנית מתחילה תמיד ב-`Main`, ופעולה מתבצעת רק כשקוראים לה: שם הפעולה, סוגריים ונקודה-פסיק. כשהפעולה מסתיימת, הקוד ממשיך מהשורה שאחרי הקריאה.

```
class Program
{
    static void PrintHi()
    {
        Console.WriteLine("Hi");
    }

    static void Main(string[] args)
    {
        PrintHi();
        Console.WriteLine("Middle");
        PrintHi();
    }
}
```

הפלט:

```
Hi
Middle
Hi
```

ארבעת סוגי הפעולות

- לא מקבלת, לא מחזירה: `static void A()`
- מקבלת, לא מחזירה: `static void B(int x)`
- לא מקבלת, מחזירה: `static int C()`
- מקבלת ומחזירה: `static int D(int x)`

פעולה שמקבלת ערכים

בסוגריים של הכותרת כותבים פרמטרים: טיפוס ושם לכל אחד, ופסיק ביניהם. בקריאה שולחים ערכים, והם נכנסים לפרמטרים לפי הסדר. אפשר לשלוח מספר, משתנה או חישוב.

משתנים ופרמטרים

משתנה שנוצר בתוך פעולה קיים רק בה: `Main` לא מכיר אותו, והפעולה לא מכירה את המשתנים של `Main`. פרמטר הוא עותק של הערך שנשלח, ולכן שינוי שלו בתוך הפעולה לא משנה את המשתנה המקורי. מידע עובר רק דרך הפרמטרים ודרך `return`.

מילים שחובה להבחין ביניהן

- **מדפיסה:** `Console.WriteLine`. **מחזירה:** `return`.
- **קולטת:** `Console.ReadLine()` מהמקלדת. **מקבלת:** פרמטר בסוגריים של הכותרת.
- **כתבו תוכנית:** כותבים בתוך `Main`. **כתבו פעולה:** פעולה שמתחילה ב- `static`, מחוץ ל- `Main`.

פעולה שמחזירה bool

"מחזירה האם..." פירושו פעולה שמחזירה `bool`: `א` או `true` `א` `false`. משתמשים בה בתנאי: `.if (IsPositive(x) == true)`

```
static bool IsPositive(int n)
{
    if (n > 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

לתרגול אדפטיבי, בדיקת תשובות אוטומטית, ועוזר בינה מלאכותית חינם, היכנסו ל-www.TIRGUL.ai

יצירת מערך

מערך הוא שם אחד שמצביע על כמה תאים מאותו סוג. ב-`new int[5]` נוצרים 5 תאים, ובכולם 0. אפשר גם לאתחל ברשימת ערכים בסוגריים מסולסלים. הגודל יכול להיות משתנה שנקלט: `new int[n]`.

```
int[] grades = new int[5];
int[] arr = {9, 4, 7, 1, 3};
string[] names = {"Dana", "Omer", "Noa"};
```

גישה לתא ו-Length

המיקומים מתחילים ב-0. `arr.Length` הוא מספר התאים, ולכן המיקום האחרון הוא `Length - 1`. תא במערך מתנהג כמו משתנה רגיל.

```
int[] arr = {9, 4, 7, 1, 3};
arr[0] = 10;
int x = arr[1];
Console.WriteLine(arr[0] + x);
Console.WriteLine(arr[arr.Length - 1]);
```

הפלט:

```
14
3
```

מעבר על מערך בלולאה

מילוי מקלט, ואחריו סריקה. מונה, סוכם, מקסימום ומינימום עובדים כמו בנושא 7, כש-`arr[i]` מחליף את הקלט. למקסימום ולמינימום נוח לאתחל לפי `arr[0]`.

```
int[] arr = new int[5];
for (int i = 0; i < arr.Length; i++)
{
    arr[i] = int.Parse(Console.ReadLine());
}
int sum = 0;
for (int i = 0; i < arr.Length; i++)
{
    sum += arr[i];
}
```

מערך מונים

כל תא במערך הוא מונה נפרד, והערך עצמו קובע איזה תא גדל. לערכים מ-1 עד 6 יוצרים 7 תאים, כדי שיהיה מיקום 6.

```
int[] counters = new int[7];
for (int i = 0; i < rolls.Length; i++)
{
    int r = rolls[i];
    counters[r] = counters[r] + 1;
}
```

פעולה שמקבלת מערך

בכותרת כותבים `int[]` ושם. בקריאה שולחים רק את שם המערך, בלי סוגריים מרובעים: `SumArray(arr)`.

```
static int SumArray(int[] array1)
{
    int sum = 0;
    for (int i = 0; i < array1.Length; i++)
    {
        sum += array1[i];
    }
    return sum;
}
```

מערך שנשלח לפעולה אינו עותק. פעולה שמשנה תא במערך משנה את המערך המקורי, גם בלי `return`.

העתקת מערך

`int[] arr2 = arr;` לא יוצרת מערך חדש, אלא מצביע נוסף על אותו מערך. שינוי דרך `arr2` נראה גם דרך `arr`. להעתקה אמיתית יוצרים מערך חדש ב-`new` ומעתיקים תא אחרי תא.

```
int[] copy = new int[arr.Length];
for (int i = 0; i < arr.Length; i++)
{
    copy[i] = arr[i];
}
```

העתקה בסדר הפוך:

```
.copy[i] = arr[arr.Length - 1 - i];
```

מערך דו-ממדי: יצירה וגישה לתא

טבלה של שורות ועמודות. ב- `new int[3, 4]` יש 3 שורות ו-4 עמודות. בגישה לתא כותבים קודם את מספר השורה ואז את מספר העמודה, ושניהם מתחילים ב-0. `GetLength(0)` מחזירה את מספר השורות, `GetLength(1)` - את מספר העמודות.

```
int[,] m = new int[3, 4];
int[,] a = {{2, 4, 6}, {1, 3, 5}};
m[2, 1] = 10;
Console.WriteLine(a[1, 2]);
Console.WriteLine(m.GetLength(0));
Console.WriteLine(m.GetLength(1));
```

הפלט:

```
5
3
4
```

מערך דו-ממדי: מעבר על כל התאים

לולאה חיצונית על השורות ולולאה פנימית על העמודות. למילוי מקלט כותבים בפנים

```
.m[i, j] = int.Parse(Console.ReadLine());
```

```
for (int i = 0; i < m.GetLength(0); i++)
{
    for (int j = 0; j < m.GetLength(1); j++)
    {
        Console.Write(m[i, j] + " ");
    }
    Console.WriteLine();
}
```

מערך דו-ממדי: שורה, עמודה ואלכסונים

לשורה אחת או לעמודה אחת מספיקה לולאה אחת: בשורה `r` עוברים על `m[r, j]`, ובעמודה `c` על `m[i, c]`. אלכסונים יש רק במערך ריבועי: באלכסון הראשי התא הוא `m[i, i]`, ובאלכסון המשני `m[i, n - 1 - i]`.

```
int n = m.GetLength(0);
int sum1 = 0;
int sum2 = 0;
for (int i = 0; i < n; i++)
{
    sum1 += m[i, i];
    sum2 += m[i, n - 1 - i];
}
```

לתרגול אדפטיבי, בדיקת תשובות אוטומטית, ועוזר בינה מלאכותית חינם, היכנסו ל-www.TIRGUL.ai

```
Player p1 = new Player("Dana", 40);
Player p2 = new Player("Omer", 25);
p1.AddPoints(10);
Console.WriteLine(p1.GetScore());
Console.WriteLine(p1);
```

הפלט:

```
50
Dana: 50
```

פעולות Get ו-Set

Get מחזירה את ערך התכונה, ו-Set מקבלת ערך חדש ומכניסה אותו לתכונה. פעולה שרק משנה תכונה היא void.

```
public int GetScore()
{
    return this.score;
}

public void SetScore(int score)
{
    this.score = score;
}
```

פעולות פנימיות

פעולה פנימית משנה תכונה, או מחשבת ומחזירה ערך. פעולה בוליאנית מחזירה true או false ב-1 if, ו-0 else.

מחלקה ותכונות

מחלקה היא תבנית לעצמים: היא קובעת אילו תכונות יש לכל עצם ואילו פעולות הוא יודע לבצע. התכונות נכתבות בראש המחלקה, תמיד private, ולכן רק הפעולות של המחלקה ניגשות אליהן ישירות.

```
public class Player
{
    private string name;
    private int score;
    private bool active;
}
```

פעולה בונה ו-this

הפעולה הבונה נקראת בשם המחלקה, בלי טיפוס מוחזר, ורצה פעם אחת כשיוצרים עצם ב-new. this.name היא התכונה של העצם הנוכחי, ו-name לבדו הוא הפרמטר. כשלשניהם אותו שם, this חובה. תכונה יכולה לקבל ערך קבוע, ואפשר לבדוק תקינות לפני ההשמה.

```
public Player(string name, int score)
{
    this.name = name;
    if (score < 0)
    {
        this.score = 0;
    }
    else
    {
        this.score = score;
    }
    this.active = true;
}
```

יצירת עצם ושימוש בו

בתוכנית הראשית יוצרים עצם ב-new עם הארגומנטים לפעולה הבונה, ומפעילים פעולות בעזרת נקודה אחרי שם העצם. לתכונה private אי אפשר לגשת מבחוץ (p1.score היא שגיאה), רק דרך הפעולות.

```

Player a = new Player("Noa", 10);
Player b = a;
b.AddPoints(5);
Console.WriteLine(a.GetScore());
Player c = null;
if (c == null)
{
    Console.WriteLine("No player");
}

```

הפלט:

```

15
No player

```

פעולה חיצונית שמקבלת עצמים

פעולה חיצונית (static, מחוץ למחלקה) מקבלת עצם כפרמטר ומשתמשת בפעולות שלו. אין בה `this`. היא יכולה לשנות את העצם שקיבלה, להשוות שני עצמים, או להחזיר אחד מהם. בקריאה שולחים את שם העצם: `Player top = Better(p1, p2);`

```

static Player Better(Player a, Player b)
{
    if (a.GetScore() > b.GetScore())
    {
        return a;
    }
    else
    {
        return b;
    }
}

```

פעולה פנימית עם עצם אחר (other)

פעולה פנימית יכולה לקבל עצם נוסף מאותה מחלקה, בדרך כלל בשם `other`. בקריאה `p1.Beats(p2)`, `this` הוא `p1`, העצם שעליו הופעלה הפעולה, ו-`other` הוא `p2`, העצם שנשלח בסוגריים. בתוך המחלקה מותר לגשת ישירות גם לתכונות של `other`.

```

public bool Beats(Player other)
{
    if (this.score > other.score)
    {
        return true;
    }
    else
    {
        return false;
    }
}

```

```

public void AddPoints(int points)
{
    this.score = this.score + points;
}

public bool IsHigh()
{
    if (this.score >= 50)
    {
        return true;
    }
    else
    {
        return false;
    }
}

```

פעולה שנעזרת בפעולה אחרת

פעולה יכולה להפעיל פעולה אחרת של אותו עצם. גם לפני שם הפעולה כותבים `this`, וכך לא חוזרים על החישוב שכבר כתוב ב-`IsHigh`.

```

public string Level()
{
    if (this.IsHigh() == true)
    {
        return "Pro";
    }
    else
    {
        return "Rookie";
    }
}

```

ToString והדפסת עצם

`ToString` מחזירה מחרוזת שמתארת את העצם. הכותרת שלה קבועה, כולל `override`. כשמדפיסים עצם, `Console.WriteLine(p1)`, מודפס מה ש-`ToString` מחזירה.

```

public override string ToString()
{
    return this.name + ": " + this.score;
}

```

שני שמות לאותו עצם, ו-null

ההשמה `b = a` אינה יוצרת עצם חדש: שני השמות מצביעים על אותו עצם, ושינוי דרך `b` נראה גם דרך `a`. משתנה מטיפוס מחלקה שלא הוכנס אליו עצם מכיל `null`, ואי אפשר להפעיל עליו פעולה.

מערך של עצמים

`new Player[3]` יוצר שלושה תאים ריקים, ובכל אחד `null`, עד שמכניסים לתא עצם ב-`new`. על תא `arr[i].GetScore()` מפעילים פעולה כמו על עצם: `arr.Length` עוברת על כל התאים.

```
Player[] arr = new Player[3];
arr[0] = new Player("Dana", 40);
arr[1] = new Player("Omer", 25);
arr[2] = new Player("Noa", 60);
for (int i = 0; i < arr.Length; i++)
{
    Console.WriteLine(arr[i]);
}
```

פעולות על מערך של עצמים

הצורות מנושא 12 עובדות גם כאן: סכום, ספירה לפי תנאי, "האם קיים" (מחזירים `true` בתוך הלולאה ו-`false` אחריה), וקיצוני. משווים את מה שהפעולות של `arr[i]` מחזירות, ואת העצם הקיצוני שומרים במשתנה מטיפוס המחלקה שמתחיל מ-`arr[0]`. פעולה יכולה גם לבנות ולהחזיר מערך חדש.

```
static Player Best(Player[] arr)
{
    Player best = arr[0];
    for (int i = 1; i < arr.Length; i++)
    {
        int s = arr[i].GetScore();
        if (s > best.GetScore())
        {
            best = arr[i];
        }
    }
    return best;
}
```

הכלה: תכונה שהיא מערך

תכונה יכולה להיות מערך. הפעולה הבונה מקבלת את המערך ושומרת אותו, והפעולות הפנימיות עוברות עליו עם `this.points[i]` ו-`this.points.Length`.

```
public class Quiz
{
    private string topic;
    private int[] points;

    public Quiz(string topic, int[] points)
    {
        this.topic = topic;
        this.points = points;
    }

    public double Average()
    {
        int sum = 0;
        int n = this.points.Length;
        for (int i = 0; i < n; i++)
        {
            sum += this.points[i];
        }
        return (double)sum / n;
    }
}
```

הכלה: תכונה שהיא עצם

תכונה יכולה להיות עצם ממחלקה אחרת. הפעולה הבונה מקבלת עצם מוכן, או שיוצרים אותו בתוך הקריאה, `new` בתוך `new`. לתכונות של העצם הפנימי ניגשים דרך הפעולות שלו, ואפשר לשרשר קריאות: `c.GetOwner().GetName()` ב-`c.ToString()`, הצמדת `this.owner` למחרוזת מפעילה את ה-`ToString` של הבעלים.

```
public class Car
{
    private string model;
    private Owner owner;

    public Car(string model, Owner owner)
    {
        this.model = model;
        this.owner = owner;
    }

    public Owner GetOwner()
    {
        return this.owner;
    }

    public override string ToString()
    {
        return this.model + ", " + this.owner;
    }
}
```

```
private Player[] squad;

public int Total()
{
    int sum = 0;
    int n = this.squad.Length;
    for (int i = 0; i < n; i++)
    {
        sum += this.squad[i].GetScore();
    }
    return sum;
}
```

```
Owner o = new Owner("Dana", 30);
Car c = new Car("Mazda", o);
Car d = new Car("Kia", new Owner("Tom", 45));
Console.WriteLine(c.GetOwner().GetName());
```

מחלקה שמחזיקה מערך של עצמים

השילוב של השניים: תכונה שהיא מערך של עצמים, ופעולות שעוברות עליו ומפעילות את הפעולות של העצמים שבתוכו. הפעולה הבונה מקבלת את המערך ושומרת אותו, כמו ב- Quiz.