מבוא לתכנות

הרצאה 5 – פונקציות I וטווח

סמסטר 1

מסתוריין

- מה קטע קוד ה#C הבא עושה?

```
static string mystery(int a, int b, int c)
{
        int res = 0;
        if (a > 0)
                res = a;
        if (b > 0)
                res += b;
        if (c > 0)
                res += c;
        return "the result is " + res;
}
```

מסתוריין

- הקוד מגדיר *פונקציה*
- כיצד נשתמש בו?
- בואו נריץ אותו בVS ונראה:

```
static void Main(string[] args)
{
    Console.WriteLine(mystery(1, 2, 4));
    //prints "the result is 7"
    Console.WriteLine( mystery(1, 2, -4));
    //prints "the result is 3"
}

static string mystery(int a, int b, int c)
{
    int res = 0;
    if (a > 0)
        res = a;
    if (b > 0)
        res += b;
    if (c > 0)
        res += c;
    return "the result is " + res;
}
```

פונקציות

- *פונקציה* היא רצף הוראות שמבצעות פעולות כלשהן. יש לה שם, וניתן להשתמש בה שוב ושוב.
- ל*פונקציה* בדרך כלל יש *פרמטרים* ו*סוג החזר*.
  - הפרמטרים וקביעת סוג ההחזר הינם אופציונליים.
  - יכולה להיות יותר מפקודת "החזר" אחת, אך רק אחת תבוצע. כאשר מתבצעת פעולה החזרה, הפונקציה מפסיקה לעבוד, ורק הערך המוחזר נשלח חזרה לקורא הפונקציה.

פונקציות - הגדרה

- הפונקציה מורכבת מ4 חלקים

סוג ההחזרה של הפונקציה. יכול להיות ריק = void (לא מחזיר כלום) או כל סוג אחר.

שם הפונקציה

הפרמטרים של הפונקציה. יתקבלו מהקורא לפונקציה. יכולים להיות 0 פרמטרים ומעלה.

```
static string mystery(int a, int b, int c)
{
        …code…
        return res //return a string
        …code…
}
```

בלוק הקוד של הפונקציה

קריאה לפונקציה

- הגדרת הפונקציה רק מגדירה מה הפונקציה תקבל כפרמטר, מה היא תחזיר ומה היא עושה בפועל.
- כדי להריץ את הפונקציה, נצטרך לקרוא לה.
- נכתוב את שמה, את הפרמטרים ונשמור את הערך המוחזר (במידה ויש). לא לשכוח את הסוגריים בסוף.

```
string result1 = mystery(1, 2, 4);
Console.WriteLine(result1);
//prints "the result is 7"
Console.WriteLine( mystery(1, 2, -4));
//prints "the result is 3"
```

פרמטרים וארגומנטים

- עיינו בפונקציה הבאה:

```
static string mystery(int a, int b, int c)
{
        …code…
        return res //return a string
        …code…
}
```

- יש לה 3 <u>פרמטרים</u>: a, b ,c.
- נוכל לקרוא לה עם <u>ארגומנטים</u> שונים:

```
Console.WriteLine(mystery(1, 2, 4));
Console.WriteLine(mystery(1, x, y));
Console.WriteLine(mystery(z, x*2, 4+y));
```

לפי ערך

- הארגומנטים נשלחים לפונקציה לפי ערכם. הכוונה היא שרק הערך משתייך לפרמטר ולא המשתנה עצמו שנשלח במקור. לדוגמה:

```
static void Main(string[] args)
{
    int x=7;
    Console.WriteLine(x); //print 7
    AddOne(x);
    Console.WriteLine(x); //print 7
}

static void AddOne(int num)
{
    Console.WriteLine(num); //print 7
    num++;
    Console.WriteLine(num); //print 8
}
```

דוגמה עבור פונקציה שלא מחזירה ערך

```
static void Main(string[] args)
{
    PrintTwice("dudu");
}


static void PrintTwice(string str)
{
    Console.WriteLine("**************");
    Console.WriteLine(str);
    Console.WriteLine(str);
    Console.WriteLine("**************");
}
```

```
**************
dudu
dudu
**************
```

דוגמה לפונקציה ללא פרמטרים

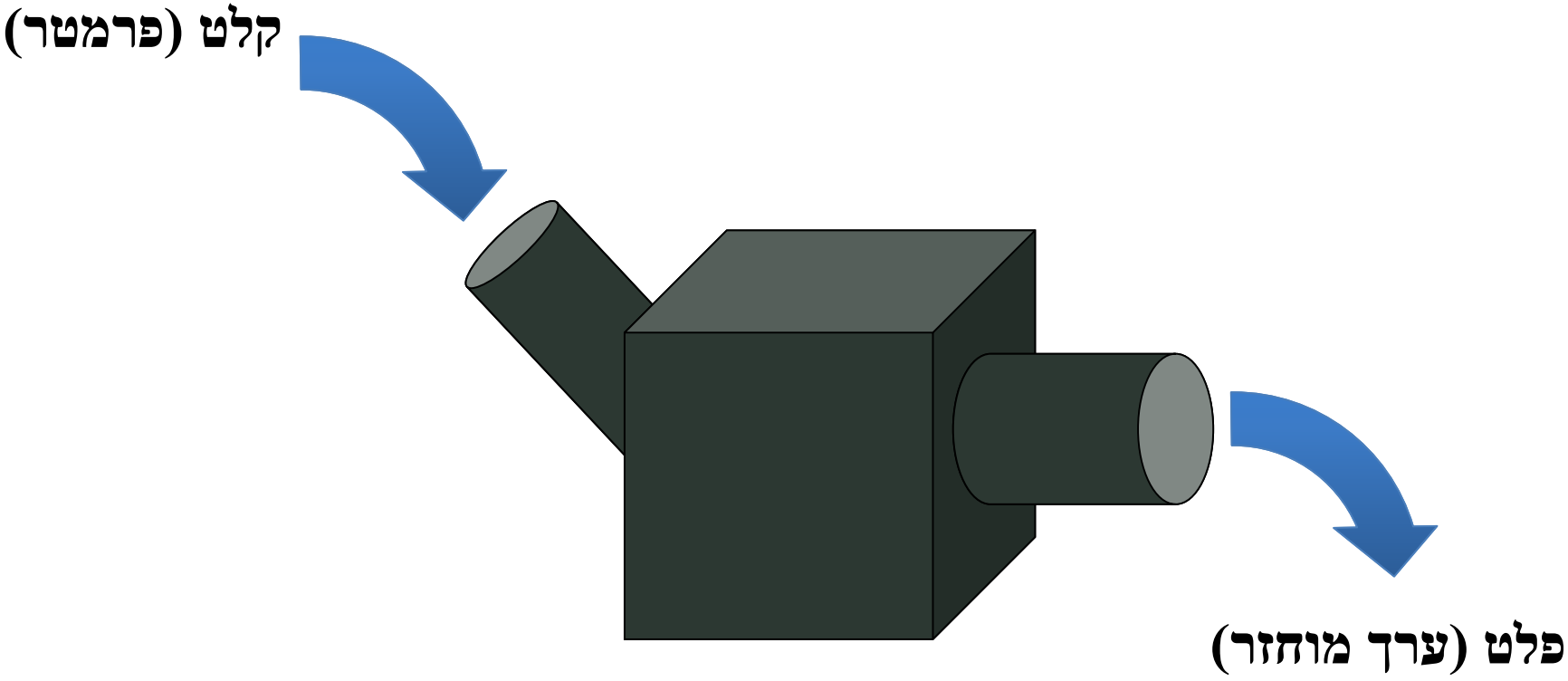
```
static void Main(string[] args)
{
    Console.WriteLine(HelloNice());
}


static string HelloNice()
{
    return " H E L L O :)";
}
```

H E L L O :)

פונקציות

- שמות פונקציות נהוג להתחיל באות גדולה, ולהשתמש ב PascalCase.
- ראינו כבר כמה פונקציות מובנות:
  - `int.Parse()`
  - `Console.ReadLine()`
    - מהם הפרמטרים של: `Console.WriteLine()` ?
    - מהו הערך המוחזר מ `Console.ReadLine()` ?

פונקציות



פונקציות

- *<u>ממשק\חתימה:</u>*
  ה"מה": הקלטים והפלטים של הפונקציה – הפרמטרים והערך המוחזר.
- *<u>מימוש:</u>*
  ה"איך": המימוש בפועל של הפונקציה.

המרת סוג - Cast

- קיימות פונקציות ואופרטורים אשר מתוכננים להמיר בין סוגים שונים של מידע:

```
int num = int.Parse("100");      //converting
string str = num.ToString();     //converting
double d = 7.7;
num = (int)d;                    //casting!
d = num;                         //no need to cast
float f = num;                   //no need to cast
d = 3 / 2;                       //int devision
Console.WriteLine(d);            //1
d = 3 / (double)2;               //in order to get the decimal
point!
Console.WriteLine(d);            //1.5
```

פונקציות מתמטיות math

- פונקציות מתמטיות רגילות:

שם קבוע

```
int radius = 5;
double circumference = 2 * Math.PI * radius;
double area = Math.PI * Math.Pow(radius, 2);

Console.WriteLine(Math.PI);
Console.WriteLine(area);
Console.WriteLine(Math.Sin(40));
Console.WriteLine(Math.Sqrt(9));
```

שם פונקציה

יצירה של פונקציות משלנו

- הכוח האמיתי של הפונקציות הוא שאתם מסוגלים ליצור אותן בעצמכם!

```
static void Main(string[] args)
{
    Console.WriteLine(get_area(5));
    Console.WriteLine(get_area(12));
    Console.WriteLine(get_area(213));

}

static double get_area(int radius)
{
    double area = Math.PI * Math.Pow(radius, 2);
    return area;
}
```

הייזכרו: מסתורין

```
static string mystery(int a, int b, int c)
{
        int res = 0;
        if (a > 0)
                res = a;
        if (b > 0)
                res += b;
        if (c > 0)
                res += c;
        return "the result is " + res;
}
```

- אז הקוד הנ"ל הוא למעשה פונקציה!
- מהם הפרמטרים של הפונקציה?
- מהו הערך המוחזר?

טווח הפונקציה

```
static double get_area(int radius)
{
    double area = Math.PI * Math.Pow(radius, 2);
    return area;
}
```

- בפונקציה הנ"ל, המשתנה *area* הוא ***משתנה לוקאלי***
- הוא מוכרז בתוך הפונקציה ומשומש אך ורק בה
- אם נשנה את ערכו, זה ישפיע רק על הקוד בפונקציה
- לא נוכל לפנות למשתנה מחוץ לגבולות הפונקציה
- המשתנה radius גם הוא לוקאלי. באופן כללי, ***פרמטר*** הוא משתנה לוקאלי אשר ערכו מאותחל מבחוץ, למשל ע"י הקורא לפונקציה.

סגירות פונקציה (טווח)

מה שקורה בפונקציה, נשאר בפונקציה...

(חוץ מהערך המוחזר)

- האם הקוד הבא תקין?

```
static double get_area(int radius)
{
    double area = Math.PI * Math.Pow(radius, 2);
    return area;
}

static void Main(string[] args)
{
    double x= get_area(5);
    Console.WriteLine(x);
    Console.WriteLine(area);
    Console.WriteLine(radius);

}
```

לא מוגדר בטווח

סגירות פונקציה (טווח)

- מה בנוגע לקוד הבא?

```
static void Main(string[] args)
{
    Console.WriteLine(foo2(foo1(3)));
}

static int foo1(int x)
{
    int y = x * 2 + 8;
    return y;
}

static int foo2(int z)
{
    int w = z * y - x;
    return w;
}
```

טווח באופן כללי

- באופן כללי, כל משתנה נגיש בתוך הבלוק שבו הוא הוגדר (כולל, כמובן, כל הבלוקים שבתוך אותו הבלוק).

```
int outside = 7;
if (outside == 7)
{
    int inside = 10;
    outside = inside;
    if (outside == inside)
    {
        int ininside = 100;
        inside = ininside;
        for (int i = 0; i < outside; i++)
        {
            Console.WriteLine(outside*10 + i);
        }
    }
    //ininside = 200; //ERROR!
}
//inside = 8; //ERROR!
outside++;
//int i = 10; ERROR because i in the child scope
for (int i = 0; i < outside; i++)
{
    Console.WriteLine(outside + i);
}
//int i = 10; //ERROR because i in the child scope
//i=10; //ERROR!
outside++;
```

למה להשתמש בפונקציות?

- מקל על קריאת ודיבוג הקוד על ידי שבירת הלוגיקה לבלוקים נפרדים.
- מונע כפילויות בקוד – עוזר להקטין את גודל הקוד ותיקון בעיות בו.
- ועוד סיבות שתמצאו בעצמכם, בהמשך...

תרגיל 1

- בהינתן כמות האירו, והערך של 1 אירו ב$, חשבו את השווי של האירו ב$.
- כתבו פונקציה המממשת את החישוב הנ"ל.

```
static double Euro2Dollar(int euros, double oneEeuroValue)
{
      return euros * one_euro_value;
}
```

תרגיל 2

- כתבו פונקציה שמקבלת את הפרמטרים הבאים:
  - מחיר מניה (ב$)
  - מספר מניות
  - מחיר מניה בשוק (ב$)
- הפונקציה תחזיר את הרווח/הפסד, בתור אחוז.

תרגיל 2 - המשך

- מימוש #1:

```
static double get_share_gain(double bp, int amount,
double mp)
{
    double buy_payment = bp * amount;
    double sale_price = mp * amount;
    double profit = sale_price - buy_payment;
    double percent = (profit / buy_payment) * 100;
    return percent;
}
```

- מימוש #2:

```
static double get_share_gain(double bp, int amount,
double mp)
{
    return ((mp - bp) / bp) * 100;
}
```

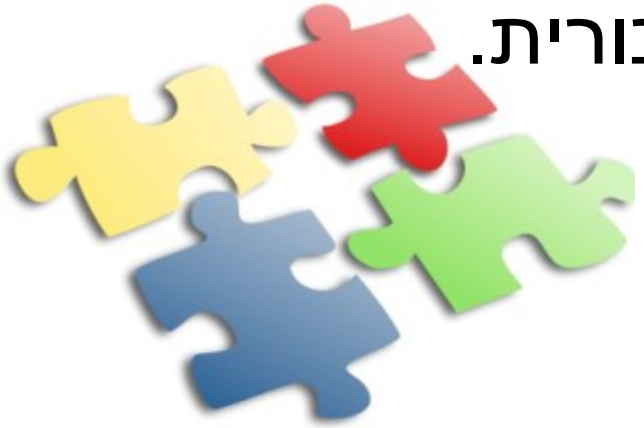
ממשק אל מול מימוש

- פונקציה מגדירה *ממשק* ספציפי.
- נוכל לחשוב על מספר רב של *מימושים* לאותו ממשק.

- ברגע שהגדרנו ממשק טוב, נוכל לשנות את המימוש שלו מבלי לשנות את הממשק.
  - זהו יתרון חשוב של שימוש בפונקציות!

ממשק

- אפליקציה בנויה מחלקים רבים; הקוד של כל חלק מתקשר עם הקוד של חלקים אחרים.
- כדי לאפשר את התקשורת הזו, אנו משתמשים בממשק המוגדר של כל חלק.
- הממשק הוא כמו חוזה מוכרז ציבורית.



ממשקים מצד שלישי

- לעתים אפליקציות שונות מתקשרות אחת עם השנייה.
- כדי לעשות זאת, כל אפליקציה מספקת ממשק, והאפליקציה האחרת משתמשת בממשק כדי לתקשר איתו.
- רוב האפליקציות הפופולאריות מספקות ממשקים כדי שיתקשרו עמן.

ממשקים פתוחים

- לפני כעשור, חברות פיתחו ממשקים עבור שימושים פנימיים בלבד. הם שמרו אותם בסוד ולא נתנו לחברות אחרות להשתמש בהם.
- כיום, ממשקים פתוחים הם המפתח להצלחה עבור חברות רבות ומוצרים:
  - ממשק פייסבוק
  - אפסטור של אפל
  - מיחשוב מעונן של אמאזון
  - תוספים לגוגל כרום

תוכנית כסדרה של פונקציות

- תוכנית מחשב לרוב מאורגנת כסדרה של פונקציות רבות שקוראות אחת לשנייה.
- נקודת ההתחלה היא פונקציית Main().

```
static void foo1(int x)
{
    foo2(x + 1);
}

static void foo2(double y)
{
    foo3(y / 2);
}

static void Main(string[] args)
{
    foo1(5);
}

static void foo3(double z)
{
    foo4(z - 3);
}

static void foo4(double w)
{
    Console.WriteLine(w);
}
```

תרגיל 3

- היזכרו בפונקציה לחישוב תשואה מהמניה:

```
static double get_share_gain(double bp, int amount,
double mp)
{
    return ((mp - bp) / bp) * 100;
}
```

- כתבו פונקציה אחרת, אשר בהינתן אותם פרמטרים, מחזירה את הרווח מהמניה:

```
static double get_share_profit(double bp, int amount,
double mp)
{
    return amount * bp * get_share_gain(bp, amount, mp);
}
```

תרגילים נוספים

1. כתבו פונקציה בשם IsPrime שמקבלת מספר שלם ומחזירה אמת/שקר אם המספר הוא ראשוני.
2. כתבו פונקציה בשם AllPrimes שמקבלת מספר שלם ומדפיסה את כל המספרים הראשוניים מ2 ועד המספר.
3. כתבו פונקציה שיש לה תפריט:
    1. חיבור 2 מספרים
    2. חיסור 2 מספרים
    3. הכפלת 2 מספרים
    4. חלוקת 2 מספרים
    5. יציאה