ELTE | IK

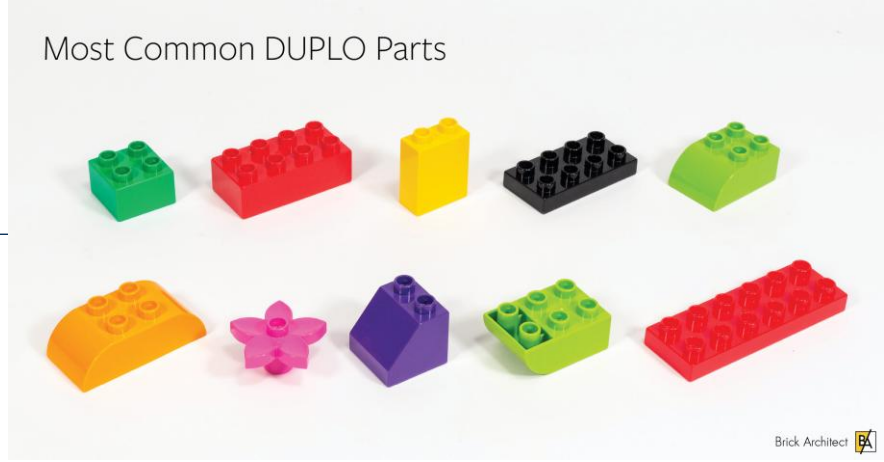
PROGRAMOZÁS

Függvények

Horváth Győző

Ismétlés

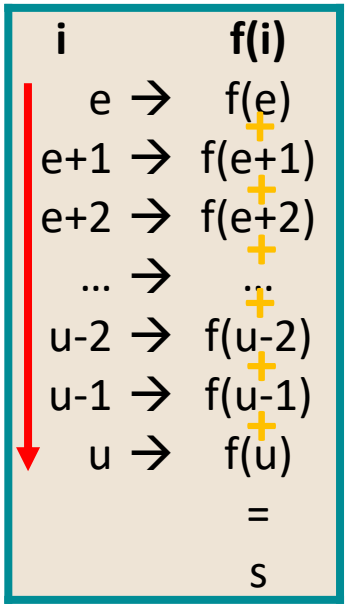
Programozási minták



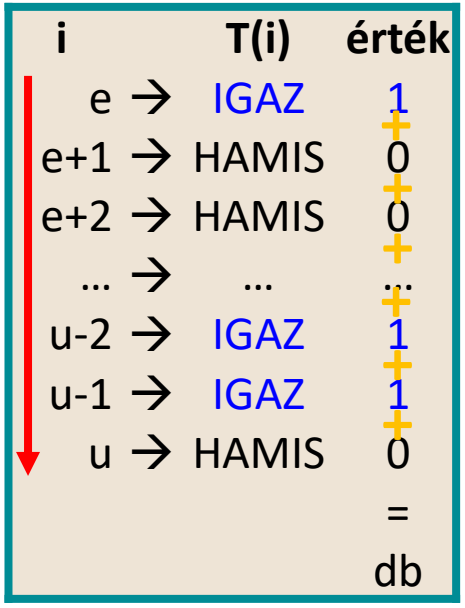
1. Összegzés
2. Megszámolás
3. Maximumkiválasztás
   a. Minimumkiválasztás
4. Feltételes maximumkeresés
5. Keresés
6. Eldöntés
   a. Mind eldöntés
7. Kiválasztás
8. Másolás
9. Kiválogatás

Programozási minták

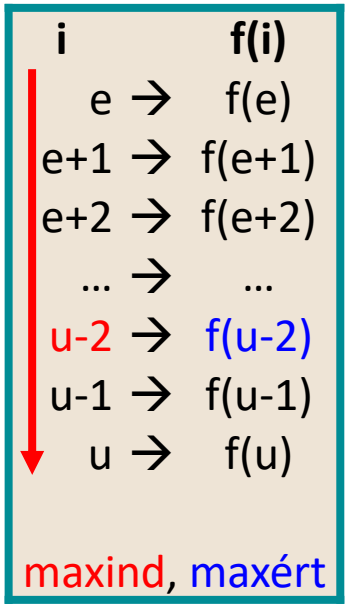
Összegzés



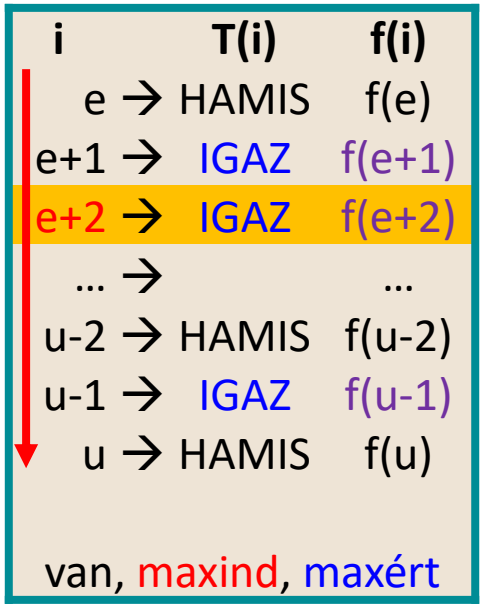
Megszámolás



Maximum kiválasztás

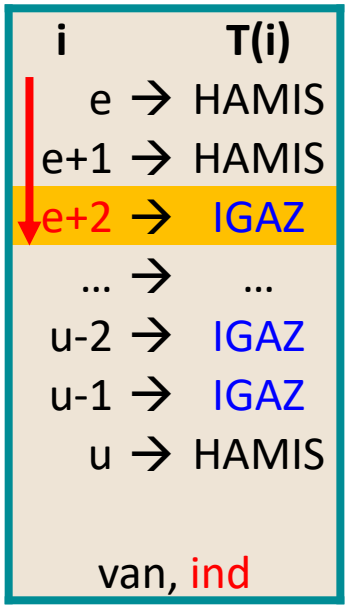


Feltételes maximumkeresés

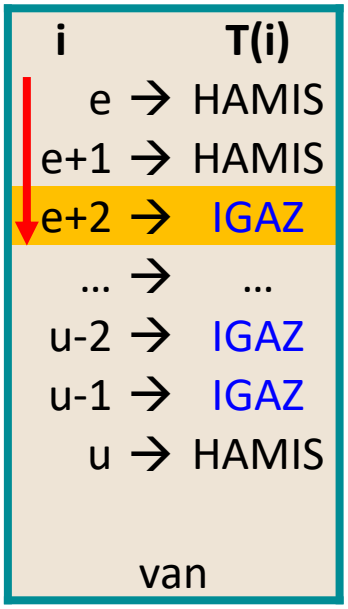


Programozási minták

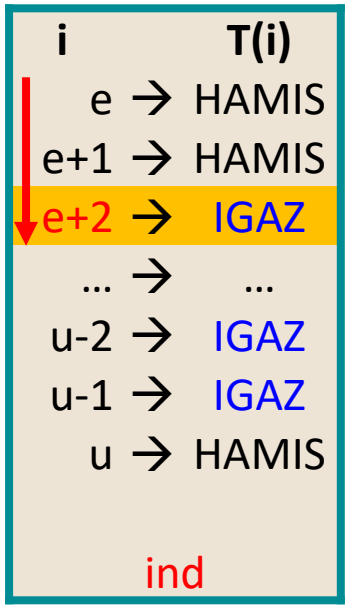
Keresés



Eldöntés

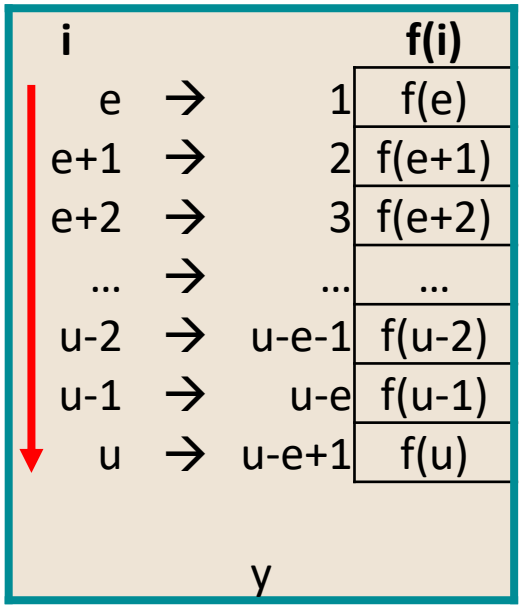


Kiválasztás

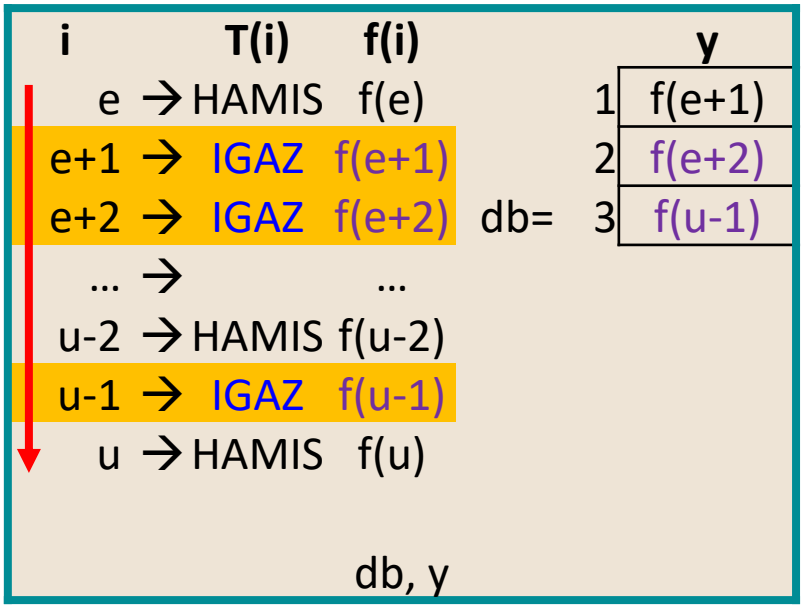


Programozási minták

Másolás



Kiválogatás



Függvények

Négyzet

Példa:
x=3,3 → y=10,89

Feladat:

Adjuk meg egy szám négyzetét!

Specifikáció:

```
Be: x∈R
Ki: y∈R
Ef: -
Uf: y=x*x
```

Mi van akkor, ha ezt a „bonyolult” részfeladatot el szeretném különíteni általánosan megfogalmazva?

Algoritmus:

```
y:=x*x
```

Négyzet

Példa:
x=3,3 → y=10,89

Feladat:

Adjuk meg egy szám négyzetét!

Specifikáció:

```
Be: x∈R
Ki: y∈R
Ef: -
Uf: y=négyzet(x)
```

Egy függvény mögé rejtettük a négyzetre emelést.

Tekintsük a **négyzet** függvény kiszámítását **önálló feladatnak**!

Algoritmus:

```
y:=négyzet(x)
```

Négyzet

Matematika:
$x \mapsto x^2$
Másként:
$f: \mathbb{R} \to \mathbb{R}, f(x) = x^2$
Esetünkben:
$négyzet: \mathbb{R} \to \mathbb{R}, négyzet(n) = n^2$

Feladat:

Add meg a **négyzet függvény** (részfeladat) működését!

Specifikáció:

```
Be: n∈R
Ki: négyzet(n)∈R
Ef: -
Uf: négyzet(n)=n*n
```

A bemenetben a függvény **paraméter**ei (∈értelmezési tartomány),
a kimenetben a függvény paraméteres **érték**e szerepel (∈értékkészlet),
az utófeltételben az **összefüggés**.

Algoritmus:

```
Function
négyzet(n:Valós): Valós
    négyzet:=n*n
```

Visszatérési érték meghatározása a függvénynévhez való értékadással

Négyzet

Példa:
x=3,3 → y=10,89

Feladat:

Adjuk meg egy szám négyzetét!

Specifikáció:

```
Be: x∈R
Ki: y∈R
Fv: négyzet:R->R, négyzet(n)=n*n
Ef: -
Uf: y=négyzet(x)
```

Matematika:
$x \mapsto x^2$
Másként:
$f: \mathbb{R} \to \mathbb{R}, f(x) = x^2$
Esetünkben:
$négyzet: \mathbb{R} \to \mathbb{R}, négyzet(n) = n^2$

Formális paraméter

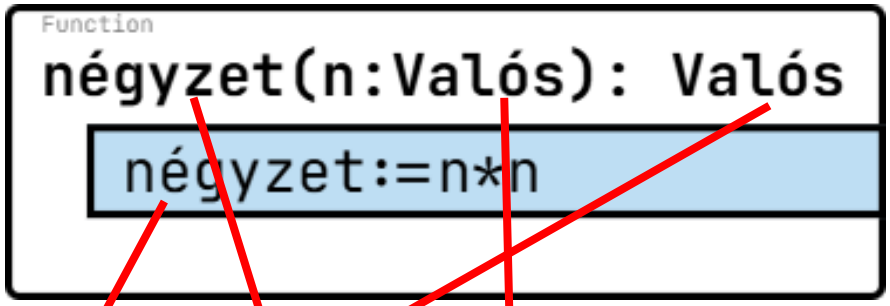
Aktuális paraméter

Algoritmus:

```
y:=négyzet(x)
```

```
Function
négyzet(n:Valós): Valós
    négyzet:=n*n
```

Négyzet
kód



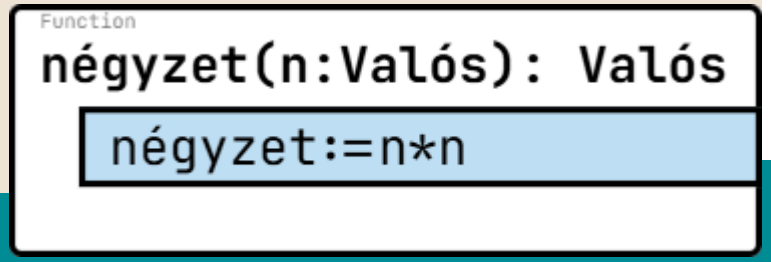

```
// négyzet:R->R, négyzet(n)=n*n
static double negyzet(double n) {
  return n * n;
}
```

Négyzet
kód

```
static void Main(string[] args) {
  Console.WriteLine(3.3 * 3.3);

  // Általánosítsuk a feladatot!
  Console.WriteLine(negyzet(3.3));

  // Függvényhívás helyettesíthető a visszatérési értékkel:
  Console.WriteLine(10.89);

  // Aktuális paraméter: konstans
  double a = negyzet(3.3);

  // Aktuális paraméter: változó értéke
  double b = negyzet(a);
}
// négyzet:R->R, négyzet(n)=n*n
static double negyzet(double n) {
  return n * n;
}
```

Aktuális paraméter

Formális paraméter



Maximum

Példa:
a=10, b=8 → m=10

Feladat:

Adjuk meg két szám közül a nagyobbat!

Specifikáció:

```
Be: a∈Z, b∈Z
Ki: m∈Z

Ef: -
Uf: m=max(a,b)
```

Mi derül ki a használatból?

1. Függvény neve = max
2. Paraméterek száma = 2
3. Paraméterek típusa = Egész,Egész
4. Visszatérési érték(ek) száma = 1
5. Visszatérési érték(ek) típusa = Egész

Maximum

Feladat:

Példa:
a=10, b=8 → m=10

Adjuk meg két szám közül a nagyobbat!

Specifikáció:

```
Be: a∈Z, b∈Z
Ki: m∈Z
Fv: max:Z x Z->Z, max(a,b)={a, ha a>=b;
                            b egyébként}
Ef: -
Uf: m=max(a,b)
```

Mi derül ki a használatból?

1. Függvény neve = max
2. Paraméterek száma = 2
3. Paraméterek típusa = Egész,Egész
4. Visszatérési érték(ek) száma = 1
5. Visszatérési érték(ek) típusa = Egész

Maximum

Feladat:

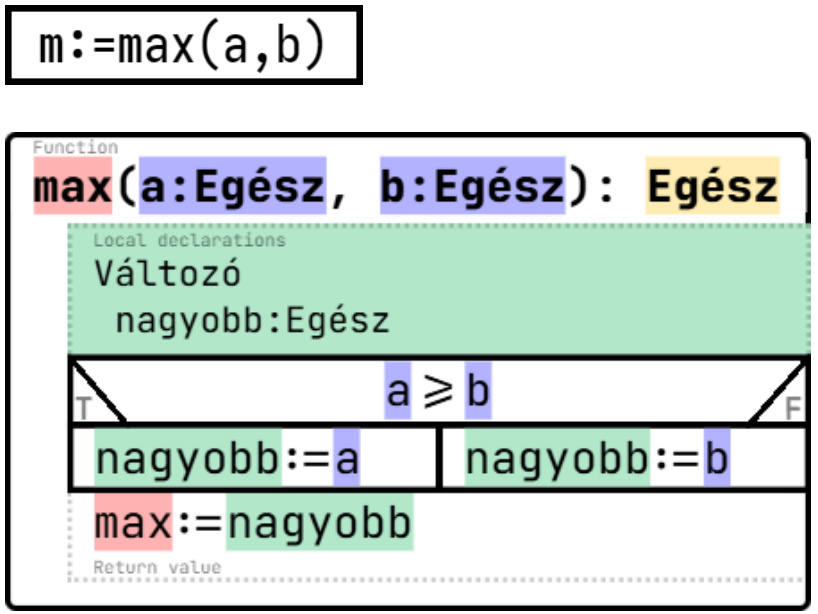
Példa:
a=10, b=8 → m=10

Adjuk meg két szám közül a nagyobbat!

Specifikáció:

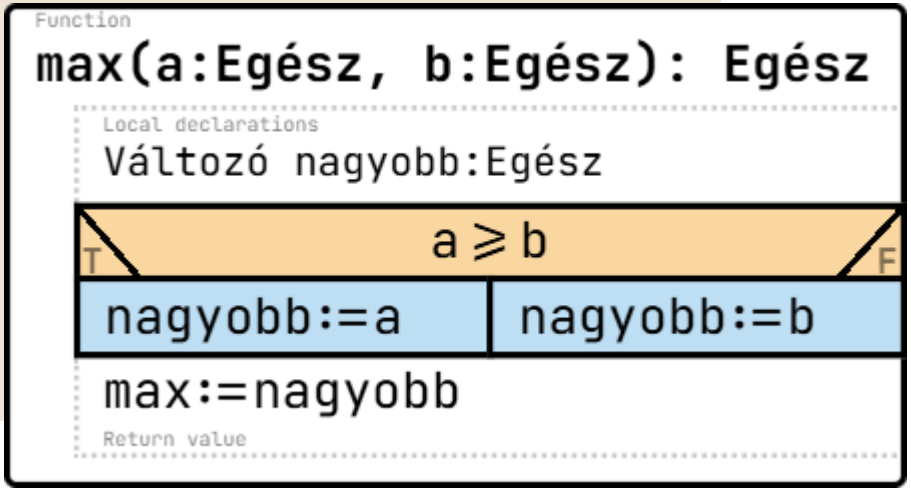
```
Be: a∈Z, b∈Z
Ki: m∈Z
Fv: max:Z x Z->Z,
    max(a,b)={a, ha a>=b;
              b egyébként}
Ef: -
Uf: m=max(a,b)
```

Algoritmus:



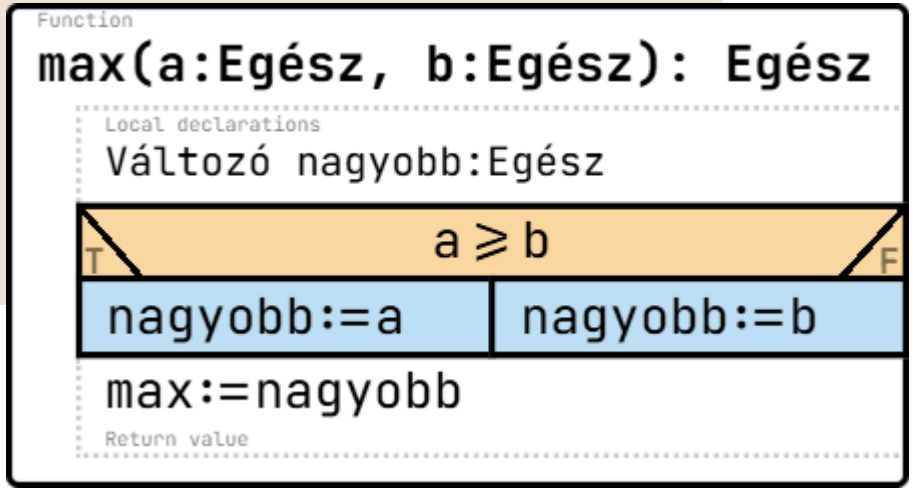
Maximum
kód

```
static void Main(string[] args) {
  // A használatból kiderül a függvény szignatúrája
  // max:ZxZ->Z, max(a, b)={a, ha a>=b; b egyébként}
  Console.WriteLine(max(3, 7));
}
// max:ZxZ->Z, max(a, b)={a, ha a>=b; b egyébként}
static int max(int a, int b) {
  int nagyobb;
  if (a >= b) {
    nagyobb = a;
  }
  else {
    nagyobb = b;
  }
  return nagyobb;
}
```



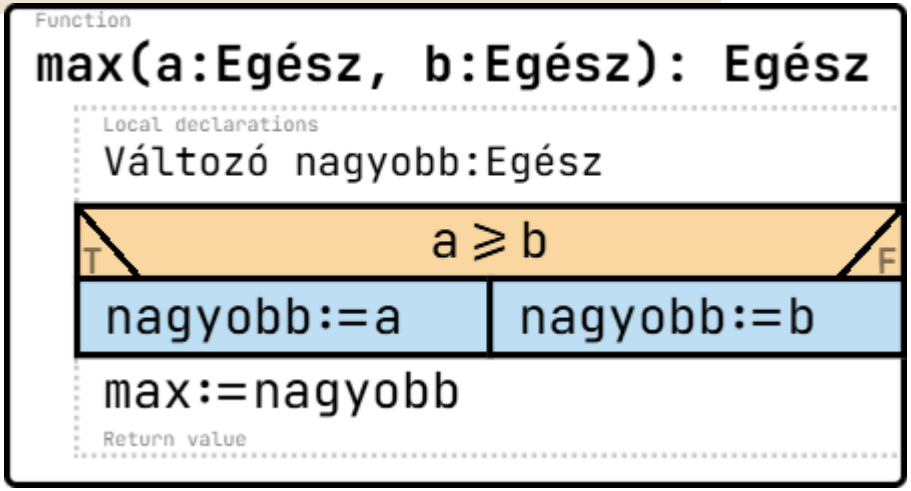
Maximum
kód

```
static void Main(string[] args) {
  // A használatból kiderül a függvény szignatúrája
  // max:ZxZ->Z, max(a, b)={a, ha a>=b; b egyébként}
  Console.WriteLine(max(3, 7));
}
// max:ZxZ->Z, max(a, b)={a, ha a>=b; b egyébként}
static int max(int a, int b) {
  if (a >= b) {
    return a;
  }
  else {
    return b;
  }
}
```



Maximum
kód

```
static void Main(string[] args) {
  // A használatból kiderül a függvény szignatúrája
  // max:ZxZ->Z, max(a, b)={a, ha a>=b; b egyébként}
  Console.WriteLine(max(3, 7));
}
// max:ZxZ->Z, max(a, b)={a, ha a>=b; b egyébként}
static int max(int a, int b) {
  return a >= b ? a : b;
}
```



Növelés

```
// Példa
int a = 10;
// így szeretném növelni:
novel(a, 8);
// és nem így:
a = novel(a, 8);
```

Feladat:

Példa:
n=10, d=8 → n'=18

Növeljünk meg egy változót egy előre megadott d értékkel!

Specifikáció:

Be: n∈Z, d∈Z
Ki: n'∈Z

Ef: -
Uf: n'=növel(n,d)

Mi derül ki a használatból?

1. Függvény neve = növel
2. Paraméterek száma = 2
3. Paraméterek típusa = Egész,Egész
4. Visszatérési érték(ek) száma = 1
5. Visszatérési érték(ek) típusa = Egész

Növelés

```
// Példa
int a = 10;
// így szeretném növelni:
novel(a, 8);
// és nem így:
a = novel(a, 8);
```

Feladat:

Példa:
n=10, d=8 → n'=18

Növeljünk meg egy változót egy előre megadott d értékkel!

Specifikáció:

```
Be: n∈Z, d∈Z
Ki: n'∈Z
Fv: növel:Z x Z->Z,
    növel(szám,mivel)=szám+mivel
Ef: -
Uf: n'=növel(n,d)
```

Mi derül ki a használatból?

1. Függvény neve = növel
2. Paraméterek száma = 2
3. Paraméterek típusa = Egész,Egész
4. Visszatérési érték(ek) száma = 1
5. Visszatérési érték(ek) típusa = Egész

Növelés

```
// Példa
int a = 10;
// így szeretném növelni:
novel(a, 8);
// és nem így:
a = novel(a, 8);
```

Feladat:

Példa:
n=10, d=8 → n'=18

Növeljünk meg egy változót egy előre megadott d értékkel!

Specifikáció:

```
Be: n∈Z, d∈Z
Ki: n'∈Z
Fv: növel:Z x Z->Z,
    növel(szám,mivel)=szám+mivel
Ef: -
Uf: n'=növel(n,d)
```

Algoritmus:

```
növel(a, 8)
```

```
Function
növel(Vált szám:Egész,
      mivel:Egész)
    szám:=szám+mivel
```

Változó paraméterként határozzuk meg!

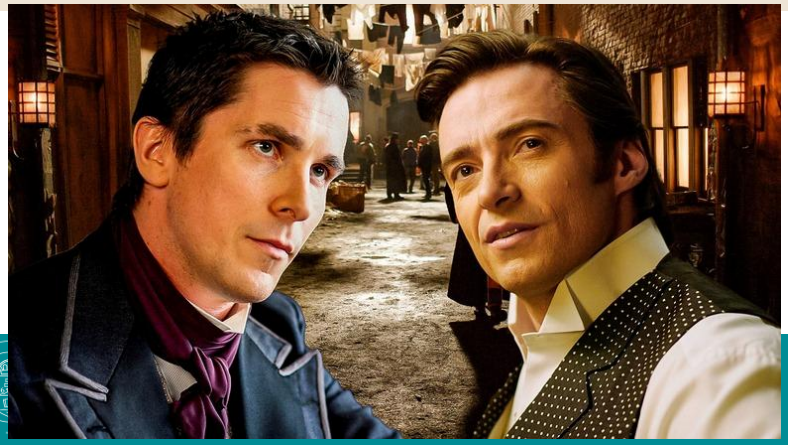
Növelés
kód

Function
növel(Vált szám:Egész, mivel:Egész)

```
szám:=szám+mivel
```

```
static void Main(string[] args) {
  int a = 10;
  Console.WriteLine("Előtte: {0}", a);
  novel(a, 8);
  Console.WriteLine("Utána: {0}", a);
}

static void novel(int szam, int mivel) {
  szam = szam + mivel;
}
```

```
Előtte: 10
Utána: 10
```



Rossz!

Érték szerinti paraméterátadás van: a változó értékéről (10) másolat készül, ez kerül a formális paraméternek átadásra (szám), a másolat módosul, és szűnik meg a függvény végén.

Növelés
kód

Function
növel(Vált szám:Egész, mivel:Egész)

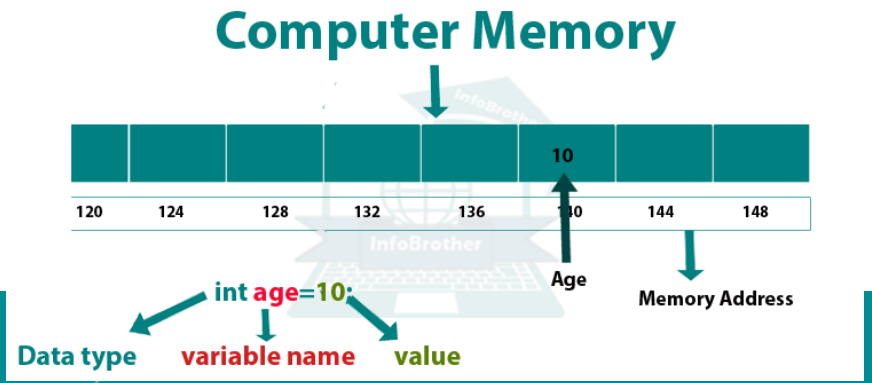
```
szám:=szám+mivel
```

```
static void Main(string[] args) {
  int a = 10;
  Console.WriteLine("Előtte: {0}", a);
  novel(ref a, 8);
  Console.WriteLine("Utána: {0}", a);
}

static void novel(ref int szam, int mivel) {
  szam = szam + mivel;
}
```

```
Előtte: 10
Utána: 18
```

Helyes!

Referencia szerinti paraméterátadás: a változó referenciája (memóriacíme) kerül átadásra, így használatkor a függvény a külső változó értékével dolgozik.



Csere

```
// Példa
int a = 10, b = 20;
// Csere:
Console.WriteLine("Előtte: {0}, {1}", a, b);
csere(a, b);
Console.WriteLine("Utána: {0}, {1}", a, b);
```

Feladat:

Cseréljük fel két változó értékét!

Példa:
a=10, b=8 → a'=8, b'=10

Specifikáció:

```
Be: a∈Z, b∈Z
Ki: a'∈Z, b'∈Z

Ef: -
Uf: a'=b és b'=a
```

Csere

```
// Példa
int a = 10, b = 20;
// Csere:
Console.WriteLine("Előtte: {0}, {1}", a, b);
csere(a, b);
Console.WriteLine("Utána: {0}, {1}", a, b);
```

Feladat:

Cseréljük fel két változó értékét!

Példa:
a=10, b=8 → a'=8, b'=10

Specifikáció:

```
Be: a∈Z, b∈Z
Ki: a'∈Z, b'∈Z
Fv: csere:Z x Z->Z x Z,
    csere(x,y)=(y, x)
Ef: -
Uf: (a',b')=csere(a,b)
```

Mi derül ki a használatból?

1. Függvény neve = csere
2. Paraméterek száma = 2
3. Paraméterek típusa = Egész,Egész
4. Visszatérési érték(ek) száma = 2
5. Visszatérési érték(ek) típusa = Egész, Egész

Csere

```
// Példa
int a = 10, b = 20;
// Csere:
Console.WriteLine("Előtte: {0}, {1}", a, b);
csere(a, b);
Console.WriteLine("Utána: {0}, {1}", a, b);
```

Feladat:

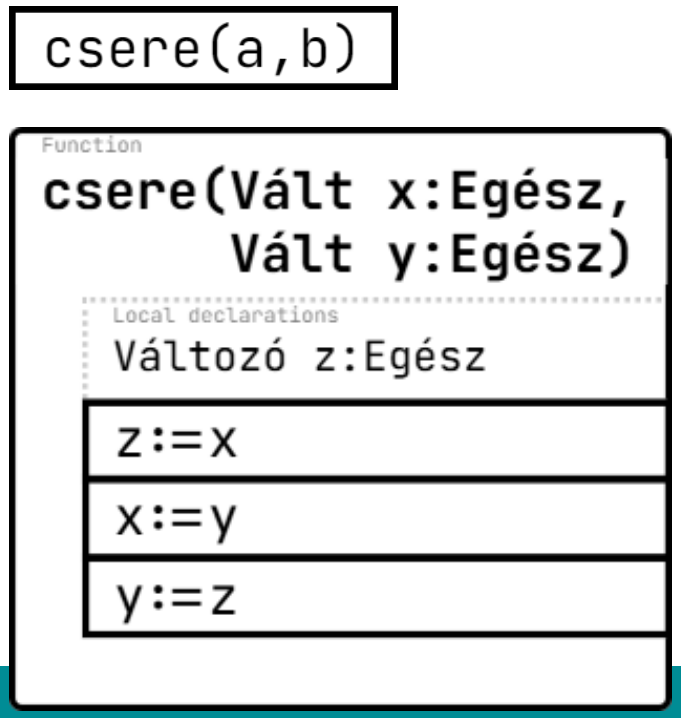
Cseréljük fel két változó értékét!

Példa:
a=10, b=8 → a'=8, b'=10

Specifikáció:

Be: a∈Z, b∈Z
Ki: a'∈Z, b'∈Z
Fv: csere:Z x Z->Z x Z,
    csere(x,y)=(y, x)
Ef: -
Uf: (a',b')=csere(a,b)

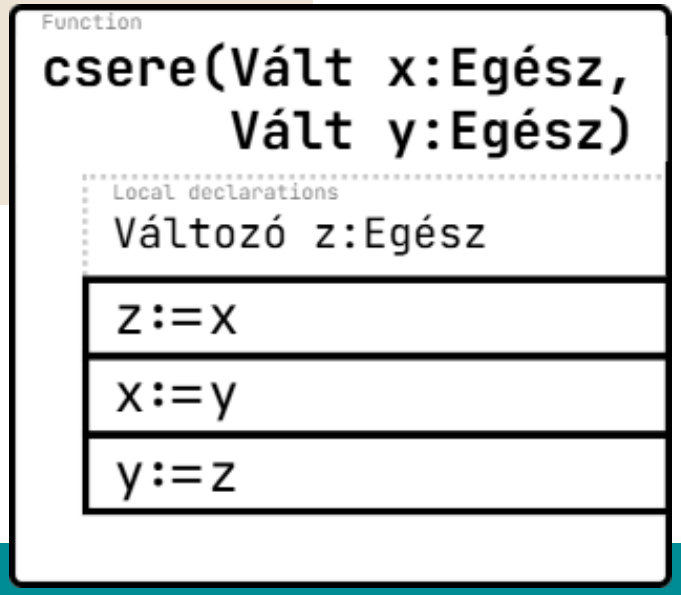
Algoritmus:



Csere
kód

```
static void Main(string[] args) {
  int a = 10, b = 8;
  Console.WriteLine("Előtte: {0}, {1}", a, b);
  csere(ref a, ref b);
  Console.WriteLine("Utána: {0}, {1}", a, b);
}

static void csere(ref int x, ref int y) {
  int z = x;
  x = y;
  y = z;
}
```

```
Előtte: 10, 8
Utána: 8, 10
```



Mintamegoldás

Feladatmegoldási minta
gyorsabb vonat az előzőnél

Feladat a Mesterről

Gyorsabb vonat az előzőnél

Ismerjük N vonat menetidejét Budapestről Siófokra.

Írj programot, amely megad egy vonatot, amely gyorsabb, mint az előző!

Bemenet

A *standard bemenet* első sorában a vonatok száma van (1≤N≤100). A következő N sor mindegyike egy-egy egész számot tartalmaz, az egyes vonatok menetidejét (1≤M≤1000).

Kimenet

A *standard kimenet* első sorába egy az előzőnél gyorsabb vonat sorszámát kell írni (ha több ilyen is van, akkor az elsőt)! Ha nincs ilyen vonat, akkor -1-et kell írni!

Példa

Bemenet	Kimenet
6 118 200 199 116 200 122	3

Feladatmegoldási minta
gyorsabb vonat az előzőnél

Részletekért ld. a mintamegoldásról szóló előadást!

Feladatsablon

(mintafeladat)

```
Be: e∈Z, u∈Z
Ki: van∈L, ind∈Z
Ef: -
Uf: (van,ind)=KERES(i=e..u,
                    T(i))
```

Előzőnél gyorsabb vonat

(konkrét feladat)

```
Be: n∈N, midő∈N[1..n]
Ki: van∈L, melyik∈N
Ef: -
Uf: (van,melyik)=KERES(i=2..n,
                       midő[i]<midő[i-1])
```

```
ind  ~ melyik
e..u ~ 2..n
T(i) ~ midő[i]<midő[i-1]
```

```
ind:=e
ind⩽u és nem T(ind)
    ind:=ind+1
van:=ind⩽u
```

```
melyik:=2
melyik⩽n és
nem midő[melyik]<midő[melyik-1]
    melyik:=melyik+1
van:=melyik⩽n
```

Kódszervezés függvényekkel
előzőnél gyorsabb vonat

```
static void Main(string[] args) {
  // Deklarálás (változók, specifikáció be,ki)
  int n;
  int[] mido;
  bool van;
  int melyik;
  // Beolvasás (specifikáció be)
  Console.Write("n = ");
  int.TryParse(Console.ReadLine(), out n);
  mido = new int[n];
  for (int i = 1; i <= n; i++) {
    Console.Write("{0}. menetido = ", i);
    int.TryParse(Console.ReadLine(), out mido[i - 1]);
  }
  // Feldolgozás (algoritmus, stuki)
  melyik = 2;
  while (melyik <= n && !(mido[melyik - 1] < mido[melyik - 1 - 1])) {
    melyik = melyik + 1;
  }
  van = melyik <= n;
  // Kiírás (specifikáció ki)
  if (van) {
    Console.WriteLine("Van, a(z) {0}. vonat gyorsabb az előzőnél.", melyik);
  }
  else {
    Console.WriteLine("Nincs gyorsabb vonat az előzőnél.");
  }
}
```

Kódszervezés függvényekkel
előzőnél gyorsabb vonat

Elvárások (pszeudo-kód)

```
static void Main(string[] args) {
  // Deklarálás (változók, specifikáció be,ki)
  int n;
  int[] mido;
  bool van;
  int melyik;
  // Beolvasás (specifikáció be)
  beolvas(n, mido);




  // Feldolgozás (algoritmus, stuki)
  keres_vonat(n, mido,       // bemeneti adatok
              van, melyik);  // kimeneti adatok
  // vagy:
  (van, melyik) = keres_vonat(n, ido); // kimeneti adatok ← bemeneti adatok

  // Kiírás (specifikáció ki)
  kiir(van, melyik);




}
```

Kódszervezés függvényekkel
előzőnél gyorsabb vonat

Beolvasás függvény

```
static void Main(string[] args) {
  // Deklarálás (változók, specifikáció be,ki)
  int n;
  int[] mido;
  bool van;
  int melyik;

  beolvas(out n, out mido);

  // ...
}
static void beolvas(out int n, out int[] mido) {
  // Beolvasás (specifikáció be)
  Console.Error.Write("n = ");
  int.TryParse(Console.ReadLine(), out n);
  mido = new int[n];
  for (int i = 1; i <= n; i++) {
    Console.Error.Write("{0}. menetido = ", i);
    int.TryParse(Console.ReadLine(), out mido[i - 1]);
  }
}
```

Kódszervezés függvényekkel
előzőnél gyorsabb vonat

Feldolgozás függvény

```
static void Main(string[] args) {
  // Deklarálás (változók, specifikáció be,ki)
  int n;
  int[] mido;
  bool van;
  int melyik;

  beolvas(out n, out mido);
  keres_vonat1(n, mido,
               out van, out melyik);
  // ...
}
static void keres_vonat1(int n, int[] mido, out bool van, out int melyik) {
  // Feldolgozás (algoritmus, stuki)
  melyik = 2;
  while (melyik <= n && !(mido[melyik - 1] < mido[melyik - 1 - 1])) {
    melyik = melyik + 1;
  }
  van = melyik <= n;
}
```

Kódszervezés függvén
előzőnél gyorsabb vonat

Specifikáció:

```
Be: n∈N, midő∈N[1..n]
Ki: van∈L, melyik∈N
Ef: -
Uf: (van,melyik)=KERES(i=2..n,
                       midő[i]<midő[i-1])
```

A megoldás tekinthető olyan függvénynek, ami
`(n, mido) → (van, melyik)`
Másképpen:

```
keres_vonat:NxN[] -> LxN
(van,melyik):=keres_vonat(n,mido)
```

```
static void Main(string[] args) {
  // Deklarálás (változók, specifikáció be,ki)
  int n;
  int[] mido;
  bool van;
  int melyik;

  beolvas(out n, out mido);
  (van, melyik) = keres_vonat2(n, mido);
  // ...
}
static (bool van, int melyik) keres_vonat2(int n, int[] mido) {
  // Feldolgozás (algoritmus, stuki)
  int melyik;
  bool van;

  melyik = 2;
  while (melyik <= n && !(mido[melyik - 1] < mido[melyik - 1 - 1])) {
    melyik = melyik + 1;
  }
  van = melyik <= n;

  return (van, melyik);
}
```

Kódszervezés függvényekkel
előzőnél gyorsabb vonat

Kiírás függvény

```
static void Main(string[] args) {
  // Deklarálás (változók, specifikáció be,ki)
  int n;
  int[] mido;
  bool van;
  int melyik;

  beolvas(out n, out mido);
  (van, melyik) = keres_vonat2(n, mido);
  kiir(van, melyik);
}
static void kiir(bool van, int melyik) {
  // Kiírás (specifikáció ki)
  if (van) {
    Console.WriteLine(melyik);
  }
  else {
    Console.WriteLine(-1);
  }
}
```

Kódszervezés függvényekkel

Teljes megoldás

```
static void Main(string[] args) {
  // Deklarálás (változók, specifikáció be,ki)
  int n;
  int[] mido;
  bool van;
  int melyik;

  beolvas(out n, out mido);
  keres_vonat1(n, mido, out van, out melyik);
  kiir(van, melyik);
}
static void beolvas(out int n, out int[] mido) {
  // Beolvasás (specifikáció be)
  Console.Error.Write("n = ");
  int.TryParse(Console.ReadLine(), out n);
  mido = new int[n];
  for (int i = 1; i <= n; i++) {
    Console.Error.Write("{0}. menetido = ", i);
    int.TryParse(Console.ReadLine(), out mido[i - 1]);
  }
}
static void keres_vonat1(int n, int[] mido, out bool van, out int melyik) {
  // Feldolgozás (algoritmus, stuki)
  melyik = 2;
  while (melyik <= n && !(mido[melyik - 1] < mido[melyik - 1 - 1])) {
    melyik = melyik + 1;
  }
  van = melyik <= n;
}
static void kiir(bool van, int melyik) {
  // Kiírás (specifikáció ki)
  if (van) {
    Console.WriteLine(melyik);
  }
  else {
    Console.WriteLine(-1);
  }
}
```

Kódszervezés függvényekkel

```
static void Main(string[] args) {
  // Deklarálás (változók, specifikáció be,ki)
  int[] mido;
  bool van; int melyik;

  beolvas(out mido);
  (van, melyik) = keres_vonat?(mido);
  // ...
}
static void beolvas(out int[] mido) {
  // Beolvasás (specifikáció be)
  int n;
  Console.Error.Write("n = ");
  int.TryParse(Console.ReadLine(), out n);
  mido = new int[n];
  for (int i = 1; i <= n; i++) {
    Console.Error.Write("{0}. menetido = ", i);
    int.TryParse(Console.ReadLine(), out mido[i - 1]);
  }
}
static (bool van, int melyik) keres_vonat?(int[] mido) {
  // Feldolgozás (algoritmus, stuki)
  int melyik;
  bool van;

  int n = mido.Length;
  melyik = 2;
  while (melyik <= n && !(mido[melyik - 1] < mido[melyik - 1 - 1])) {
    melyik = melyik + 1;
  }
  van = melyik <= n;
  return (van, melyik);
}
```

Tömb tartalmazza a hosszát, n elhagyható, lekérdezhető

Függvények a programozási minták sablonjaiban

Példa – jövedelmek
visszavezetés

Ismerjük egy ember havi bevételeit és kiadásait. Adjuk meg, hogy év végére **mennyivel** nőtt a vagyona!

Feladatsablon
(mintafeladat)

```
Be: e∈Z, u∈Z

Ki: s∈H
Ef: -
Uf: s=SZUMMA(i=e..u, f(i))
```

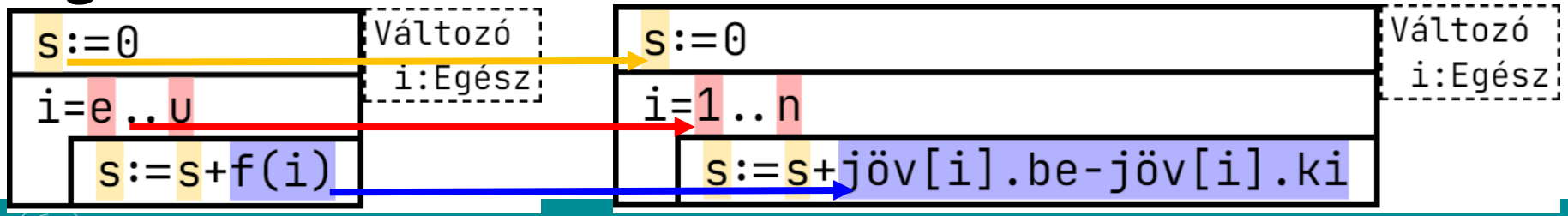
Jövedelmek
(konkrét feladat)

```
Be: n∈N, jöv∈Jövedelem[1..n],
    Jövedelem=(be:N x ki:N)
Ki: s∈Z
Ef: -
Uf: s=SZUMMA(i=1..n,
             jöv[i].be-jöv[i].ki)
```

Visszavezetés:

e..u	~	1..n
f(i)	~	jöv[i].be-jöv[i].ki

Algoritmus:



Példa – jövedelmek
visszavezetés

Ismerjük egy ember havi bevételeit és kiadásait. Adjuk meg, hogy év végére **mennyivel** nőtt a vagyona!

Feladatsablon
(mintafeladat)

```
Be: e∈Z, u∈Z

Ki: s∈H

Ef: -
Uf: s=SZUMMA(i=e..u, f(i))
```

Jövedelmek
(konkrét feladat)

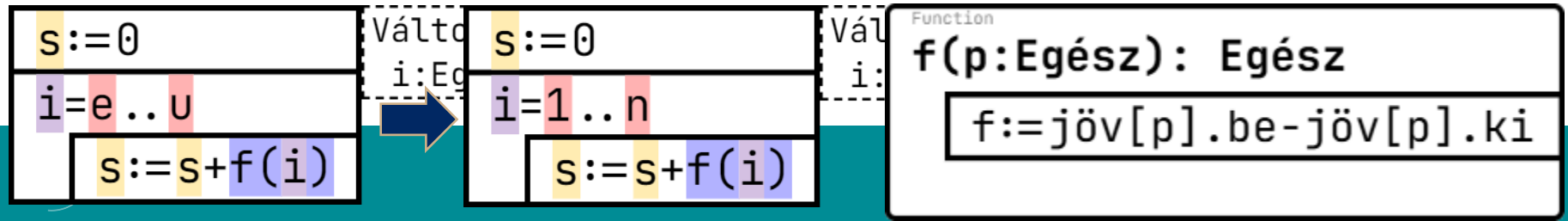
```
Be: n∈N, jöv∈Jövedelem[1..n],
    Jövedelem=(be:N x ki:N)
Ki: s∈Z
Fv: f:[1..n]->Z,
    f(p)=jöv[p].be-jöv[p].ki
Ef: -
Uf: s=SZUMMA(i=1..n, f(i))
```

Visszavezetés:

`e..u ~ 1..n`

Behelyettesítés helyett ki is fejthetjük az f(i) jelentését!

Algoritmus:



Függvényekkel kapcsolatos fogalmak

Függvények
fogalmak (c#)

Blokk. A { és a hozzá tartozó } közötti programszöveg.
Hatáskör. Egy X azonosító hatásköre az a programszöveg (nem feltétlenül összefüggő), ahol az azonosítóra hivatkozni lehet.
A hatáskört a blokkstruktúra határozza meg. Ha két blokknak van közös része, akkor az egyik teljes egészében tartalmazza a másikat.
Egy azonosító hatásköre a deklarációját követő karaktertől a blokkot lezáró } végzárójelig tart, kivéve azt a beágyazott blokkot, és ennek beágyazottjait, amelyben újra lett deklarálva.

Függvények
fogalmak (c#)

Hatáskör. Egy adott helyen hivatkozott azonosító lokális, ha a hivatkozás helyét tartalmazó legszűkebb blokkban lett deklarálva. Egy azonosító globális (az adott blokkra nézve), ha nem lokális.
Élettartam. Minden B blokkban deklarált változó élettartalma a blokkba való belépéstől a blokk utolsó utasításának befejeződéséig tart.

Függvények
fogalmak (c#)

C# tudnivalók – összefoglalás:

- **Formális** skalár paraméter:
  - bemeneti → nincs speciális kulcsszó
  - kimeneti → ref/out a speciális prefix „kulcsszó”
- **Aktuális** skalár paraméter:
  ha a megfelelő formális paraméter
  - bemeneti → akár konstans, akár változó
  - kimeneti → **csak** változó, ref/out a speciális prefix-szel

  lehet.

  ref: a paramétert meg lehet változtatni
  out: a paramétert meg kell változtatni

Pontosabban: **nem tömb**

Pontosabban: **nem tömb**

Tömb paraméter esetén mindig **hivatkozás szerinti** a paraméterátadás.

vek
c#)

C# tudnivalók – összefoglalás:

- Skalár paraméterátadás:
  - **Értékszerinti** – a formális paraméterből „keletkezett” lokális változóba másolódik a híváskor az aktuális paraméter értéke, így ennek a törzsön belüli megváltozása nincs hatással az aktuális paraméterre. Pl.:

Input-paraméterek.

```
static int max(int x, int y)
```

  - **Hivatkozás szerinti** – a formális paraméterbe az aktuális paraméter címe (*rá való hivatkozás*) kerül, a lokális néven *is* elérhetővé válik. Pl.:

In-/Output-paraméter.

Input-paraméterek.

```
static void max(int x, int y, ref int max_xy)
static void max(int x, int y, out int max_xy)
```

Input-paraméterek.

Output paraméter.

Összefoglalás

Függvények

- Függvények szerepe
  - Részfeladatok csoportosítása (alprogram)
  - Általánosítás (paraméterekkel)