Függvények

A függvények (function) névvel rendelkező utasításcsoportok, melyeknek információkat adhatunk át, és van egy visszatérési értékük.

Mint egy dzsinn:

Hogyan is “használunk” egy dzsinnt?

- megszólítjuk
- megmondjuk, hogy mit akarunk tőle
- megvárjuk a választ

Képzeljük el, hogy a csodalámpából egy selejtes dzsinn bújik elő, aki csak egyetlen dolgot tud csinálni, mondjuk meghatározni egy szám négyzetgyökét. Játsszuk le a beszélgetést:

- Ó, Hatalmas Négyzetgyökszámoló Dzsinn!
- Miért zavartál fel évezredes álmomból, halandó?
- Mondd meg kérlek, mennyi 5 négyzetgyöke?
- 2,23606797749978969640917366873…

Ha a programozási függvényeket nézzük, hasonlítanak egy dzsinnre, mivel nekik is van:

- nevük (négyzetgyökszámítás, Javásan: `Math.sqrt()`)
- paraméterük vagy paramétereik, azaz adatok, amik kellenek a működéshez (a szám, aminek a négyzetgyökét vettük)
- visszatérési értékük (a 2,23606…)

Függvény meghívása

Összegezve Javásan: `double root = Math.sqrt(5);`

Ebből a kifejezésből az elejét ismerjük: egy `root` nevű változó létrehozása. Az egyenlőségjel jobb oldalán álló kifejezésből a `Math.sqrt` a függvény neve, az 5 a paramétere.

> *A* `Math.sqrt` *pedig úgy áll össze, hogy a* `Math` *egy ún. osztály neve, amiben helyet kapnak a függvények. Az osztály fogalmát majd bevezetjük a 12. fejezetben. Most tekinthetsz úgy a* `Math.sqrt`*-re, mintha egyben lenne a függvény neve, de tekinthetsz úgy is a* `Math`*-ra, mintha a matematikai függvények tárolója lenne. Ha ugyanis odáig írod be a fejlesztőkörnyezetben a kifejezést, hogy* `Math.` *(pontot is!), akkor megjelenik egy csomó más matematikai függvény, amiből válogathatsz.*

Amikor a fenti kifejezést értelmezi a számítógép, akkor előbb az értékadás jobb oldalán álló kifejezést veszi és hajtja végre: `Math.sqrt(5)`. A függvény visszatérési értékét ennek helyére írja be, így az első lépés után olyan, mintha az eredeti kifejezésünk ez lett volna:

```
double root = 2.23606;
```

Ezt pedig már tudjuk értelmezni.

Lásd még: `int x = sc.nextInt();`, ahol a `nextInt()` a függvény, aminek nincs paramétere, az `sc.` pedig azt mondja meg, hogy ez az `sc` nevű `Scanner` objektumhoz tartozik. (Az objektumokkal is a 12. fejezetben kezdünk majd el foglalkozni).

Függvény írása

A függvényekkel való programírásban az az igazán nagyszerű, hogy mi is tudunk készíteni függvényeket.

```
public class Main {
    public static void main(String[] args) {
        double root = squareRoot(4.0);
    }

    public static double squareRoot(double param) {
       // megvalósítás
    }
}
```

Fontos, hogy a saját függvényeket az osztályon (`public class Main`) belülre, de a `main()` függvényen kívülre tegye az ember, ahogyan a példában is van.

Javában a függvények egymás „testvérei”, azaz egyik sincs a másik területén belül. Tehát nem lehetséges ilyesmi:

```
public class Main {
    public static void main(String[] args) {
        public static double squareRoot(double param) {
            // megvalósítás
        }
        double root = squareRoot(4.0);
    }
}
```

A függvények az osztályon belül bármilyen sorrendben lehetnek. (Bár az áttekinthetőség kedvéért érdemes a legkorábbi meghívást tartalmazó függvény alá tenni a hívott függvényt).

Nézzük az egyes részeket:

- `public static`: később megmagyarázom
- az első `double`: a függvény visszatérési értékének típusa
- `squareRoot`: a függvény neve
- `double param`: a függvénye paramétere. Itt most egy van, ha több lenne, vesszővel kellene elválasztani őket egymástól.

A négyzetgyök kiszámítása első feladatnak egy kicsit meredek lenne (a következő fejezetben megmutatom a megoldást).

A példa kedvéért legyen egy három szám közül a legnagyobbat kiszámító függvény írása a feladat:

```
public static int max(int a, int b, int c) {
   int m = a;
   if (b > m) {
      m = b;
   }
   if (c > m) {
      m = c;
   }
   return m;
}
```

A `return m;` két dolgot csinál:

- beállítja a függvény visszatérési értékét – ebben az esetben ami m változóban van, arra lesz „kicserélve” a hívás pontján az, hogy max(2,3,4). Lásd még ezzel kapcsolatban a Függvény meghívása részt.
- kilép a függvényből

A `return` után bármilyen kifejezést írhatnánk, tehát nem muszáj egy változót felhasználni. Lehetne `return 2+3;` vagy `return 2*m+5;` vagy boolean függvény esetén `return m > 2;` is.

Hívjuk meg az elkészült függvényünket:

```
int f = 5;
int g = 8;
int z = 2;
int mx = max(f, g, z);
System.out.println("A legnagyobb: "+mx);
```

A függvény hívásakor rendre behelyettesítjük az `f`, `g` és `z` változók **értékeit**, azaz gyakorlatilag ez lesz a függvény meghívása sorból:

`int mx = max(5, 8, 2);` → az, hogy a számok eredetileg `f`, `g` és `z` változókban voltak, az már ekkor nem is tudható, csak az érték számít, amit át akarunk adni.

Ez a három érték illesztődik be a `max` függvény `a`, `b` és `c` paramétereibe, és `max` függvényt ezekkel a kezdőértékekkel kezdjük el futtatni.

Mikor a függvény végére érünk, `m` változóba bekerül a `8` érték.

A `return m;` ezen paraméterek esetében gyakorlatilag `return 8;`-két értelmezendő, azaz ez az érték lesz az, amivel a számítógép „gondolatban” kicseréli a meghívásnál a `max(f,g,z)` részt. Azaz:

```
int mx = 8;
```

Függvény írása - megjegyzések

A függvény meghívásakor az `f` változóból a számítógép kiveszi az értéket és átírja az `a` változóba, tehát mondhatjuk azt is, hogy az `f` változó értéke **lemásolódik**, tehát ha megváltoztatjuk a függvényben az `a` változónak az értékét, akkor annak csak `a`-ra lesz hatása, `f`-re nem, mert csak `a`, azaz a **másolat** értéke változik meg.

> *Lehet, hogy ez így végigvéve egyértelmű, de más programozási nyelvekben (pl. C++, C#) lehetséges az, hogy egy egyszerű érték (primitív típus) átadása esetén megmaradjon a kapcsolat a függvénynek átadott változó és a belül használt változó közt, azaz ha ebben az esetben `a`-t megváltoztatnánk, akkor változna `f` is. Ezt amúgy változó- vagy referencia szerinti paraméterátadásnak nevezzük.*
>
> *Javában **primitív típusok** függvénynek való átadásakor mindig megszűnik a kapcsolat a hívó függvényben lévő változó és a hívott függvényben lévő paraméter között, ahogyan fent láttuk. Ezt hívják érték szerinti paraméterátadásnak.*
>
> *Tömbök és objektumok átadása külön történet, hamarosan nézzük azt is.*

A függvény paramétereinek neve `a`, `b`, `c` nem kell, hogy megegyezzen a hívásban használt változókkal (`f`, `g`, `z`), de lehetnek azonosak is.

> *Ennek az a célja, hogy ne kelljen mindenáron a függvény változóinak nevét használni a hívó függvényben, hanem nyugodtan lehessen bármilyen változóból paramétert átadni a függvénynek.*

A függvényeken belül létrehozott változókat, az ún. *lokális változókat* (*local variables* – ilyen az `m`) egy másik függvényből csak úgy érhetjük el, ha az visszatérési értéke a függvénynek. Ha van más lokális változó egy függvényben, ami nem visszatérési értéke a függvénynek, azt lehetetlen elérni. Ilyen szempontból nézve a függvény egy „fekete doboz”, amibe be tudunk adni értékeket, és az kiad utána valamit, de nem látunk bele a belsejébe.

A lokális változók a deklaráció pontjától a függvény végéig élnek, a függvény végén megszűnnek.

Másolás következményei

Ez **hibás**:

```
public static void replaceError(int a, int b) {
   int c = a;
   a = b;
   b = c;
}
```

majd a `main()`-ben:

```
int x = 7;
int y = 8;
replaceError(x, y);
System.out.println("x="+x+"; y="+y);
```

Próbáld ki! ☺

Na jó… szóval az eredmény ez lesz: `x=7; y=8`, mert a másolás miatt az eredeti `x` és `y` változó értéke nem változott meg.

Ennek oka az, hogy amikor meghívtuk a `replaceError()` függvényt az `x` és `y` értékekkel, akkor elvált a `7` és `8` érték az őket tartalmazó változóktól, és a hívás ennyi lett:

```
replaceError(7, 8);
```

Ennek hatására a függvényt úgy kezdtük el futtatni, hogy `a` értéke `7`, `b` értéke `8` lett.

Végrehajtódik a függvény, mire `a` értéke `8`, `b` értéke `7` lesz.

Mivel már korábban elszakadt a két érték az eredeti `x` és `y` változóktól, a függvény futtatása a hívóra semmilyen formában nem hat vissza.

void

Visszatérési értékként használhatjuk a `void` kulcsszót is, amely azt jelenti, hogy az adott függvénynek nincs visszatérési értéke, csak végrehajt valamit (pl. képernyőre írás), majd kilép.

Ez azt is jelenti, hogy nem is állhat az értékadás jobb oldalán (nincs érték, amit betehetnénk a változóba):

```
public void voidFunction(double frac, int integral) {...}

int hello = voidFunction(4.2, 4);
```

`void` függvények esetében is használható a `return`, de csak így: `return;`, érték nélkül.

Megjegyzések

- két függvénynek lehet azonos neve is, ekkor a paraméterek száma, típusa eltérő kell legyen. (A számítógépnek meg kell tudni különböztetni, hogy melyikre gondoltunk. Gondolatolvasás hiányában a függvény neve és paramétereinek típusa alapján teszi meg).
- A Javás objektum-orientált terminológiában találkozhatunk a *metódus (method)* fogalommal is. Itt most – az objektum-orientált programozás koncepciójának bevezetése előtt – egyelőre függvénynek nevezem őket, adózva a strukturált programozás és a C programozás nagyjainak. A legtöbb itt megtanult ismeret érvényes a metódusok esetén is, ami nem, azt külön jelzem majd (`static`).
- Ha olyan programot írunk, amelyben egy adott függvény visszatérési értékét több helyen használjuk fel, akkor a függvény visszatérési értékének eltárolása a megoldás. Akárhányszor leírod a függvény meghívását (`fuggvenynev(parameterek);`) a függvény annyiszor meghívódik!

Végezd el a 1-5. feladatokat!