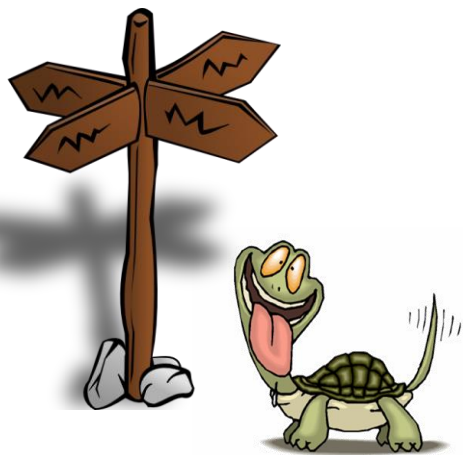




Változók és feltételek



Kezdődik az igazi programozás

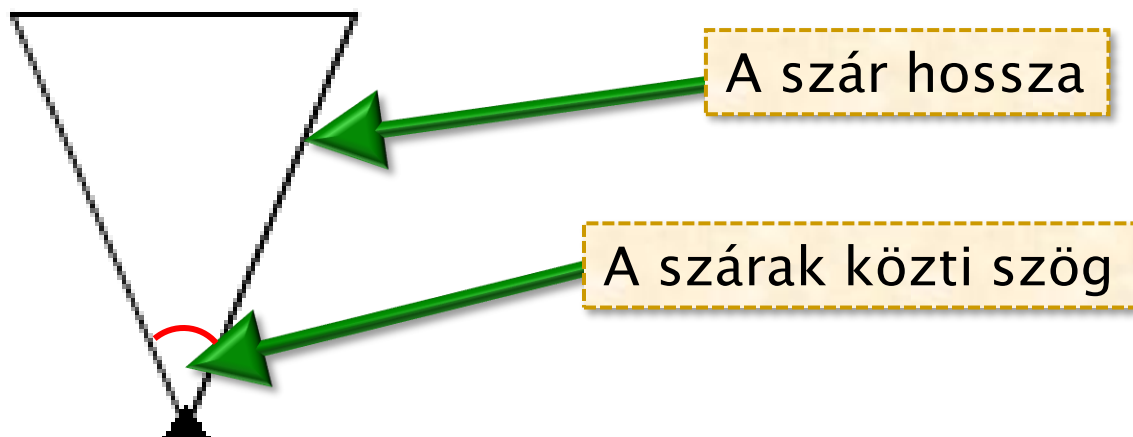




Egyenlőszárú háromszög

- Hogyan tanítsuk meg a teknőseket **egyenlőszárú háromszöget rajzolni** adott szárhosszal és adott szöggel a szárak között?

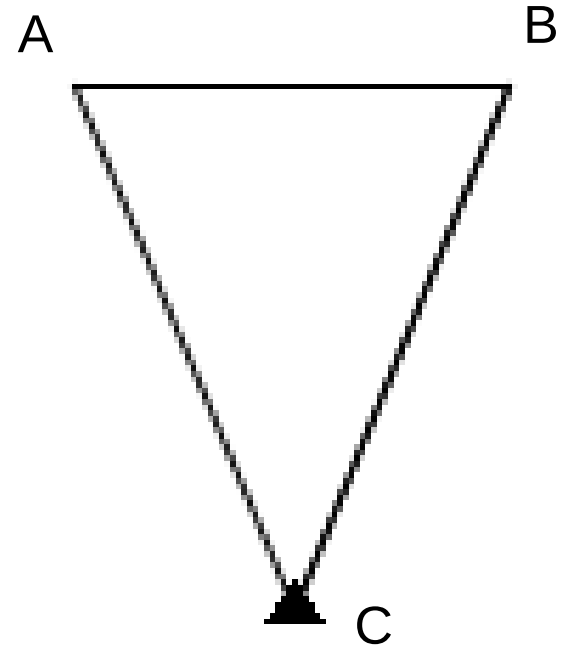
```
public void egyenloszaru(double szar, double szog)
```





Egyenlőszárú háromszög

1. Fordulj el **fok/2** balra (C-ben)
2. Lépjél szárhossznyit előre (A-ba)
3. Lépj vissza (C-be)
4. Fordulj jobbra **fok** fokot (C-ben)
5. Lépjél előre szárhossznyit (B-be)
6. Fordulj A felé és lépj előre (A-ba)

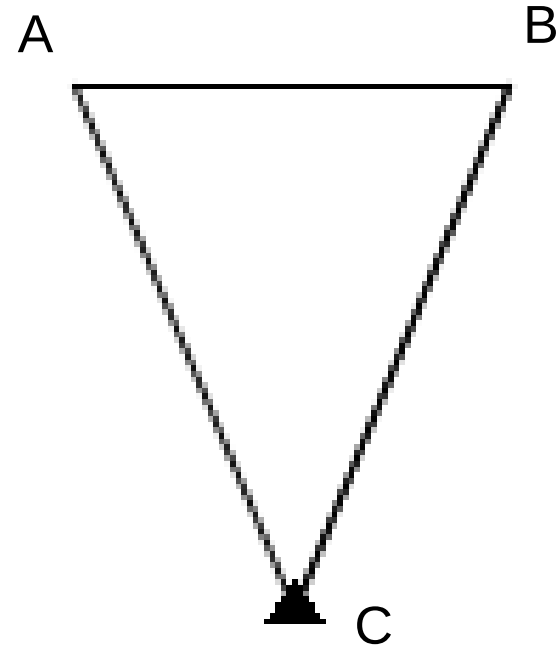


Hogy tudunk az A pontba lépni,
ha nem ismerjük a koordinátáit?



Egyenlőszárú háromszög

1. Fordulj el **fok/2** balra (C-ben)
2. Lépjél szárhossznyit előre (A-ba)
 - Az A pontban vagyok és valahogy megjegyelem a X és Y koordinátáimat
 - `this.getX()`
 - `this.getY()`
3. Lépj vissza (C-be)
4. Fordulj jobbra **fok** fokot (C-ben)
5. Lépjél előre szárhossznyit (B-be)
6. Fordulj A felé és lépj előre (A-ba)
 - `this.moveTo(?, ?)` azokra a koordinátákra, amelyeket a (2) pontban jegyeztem meg



Szükségünk van valamire, amivel meg tudunk jegyezni valamilyen értéket, majd később hozzájutni



Hogyan emlékezzünk?





Hogyan emlékezzünk...

- Értékek megjegyzésére **változókat** használunk...

- Változó:

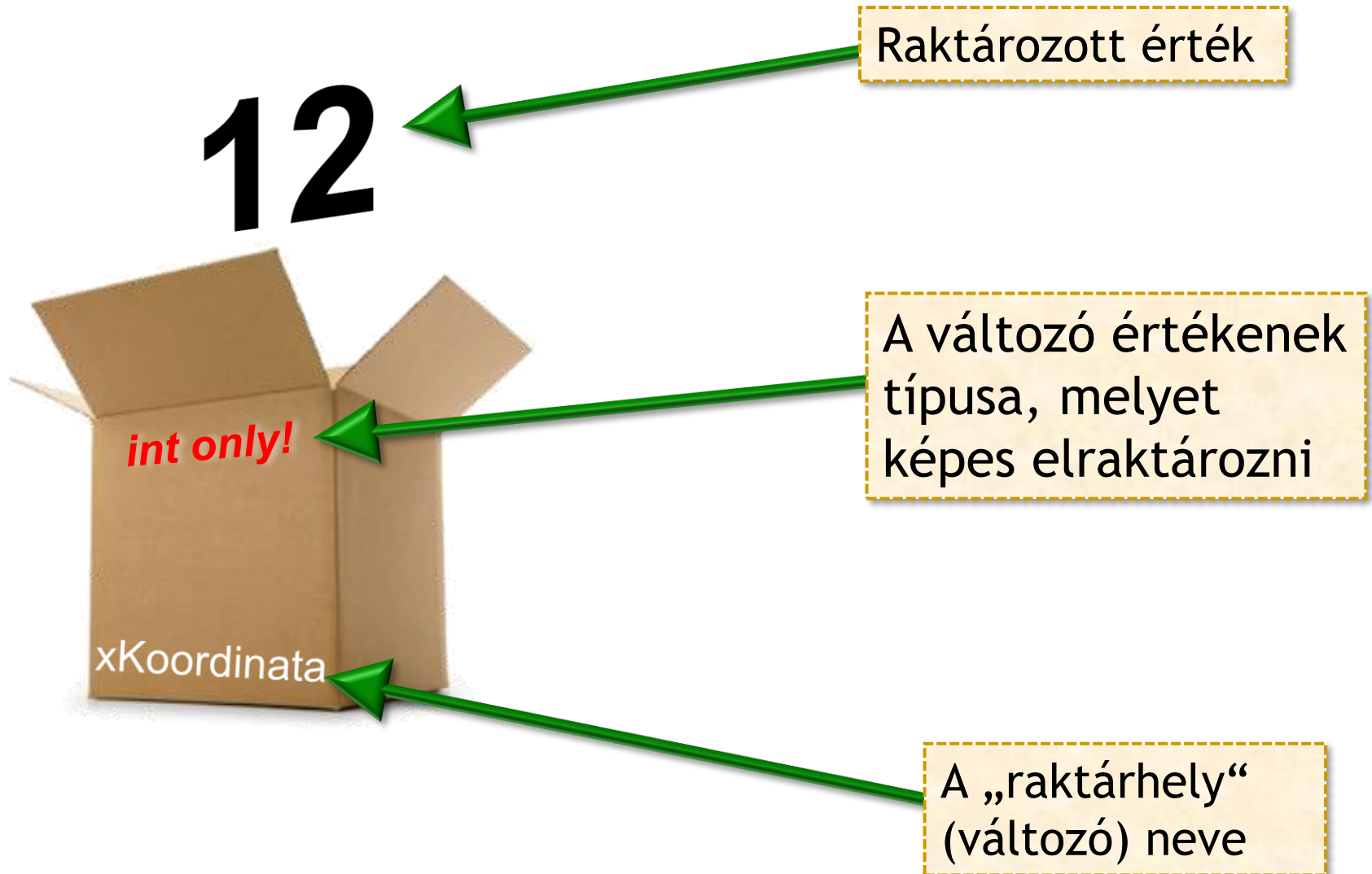
Vigyázat: matematikában a változók mást jelentenek.

- van **neve** (megjelölés, megnevezés)
- van **típusa** - megengedett értékek, melyeket meg tudnak jegyezni (úgy mint a paramétereknél)

**Adott típusú érték megnevezett helye
a számítógép memóriájában**



Változók - elképzelés





Mit kell tudunk?

- Alapkérdések a változókról...
 - Hogyan hozunk létre változót?
 - Hogyan nevezzük el a változót?
 - Milyen értékeket tud a változó raktározni?
 - Hogyan mentünk el a változóba valamilyen értéket?
 - Hogyan tudjuk a változóba elmentett értéket kiolvasni?
 - Mikor végződik a változó élete?





Változó létrehozása

- A változó létrehozására szolgáló parancs:

double xPozicioA;



A megengedett értékek típusa, melyeket a változóba „tehetünk”

A változó neve

- A létrehozott változó **nem inicializált**, nem hordoz semmilyen értéket.



Változó értékének beállítása

- A változó értékének beállítására szolgáló parancs:

```
xPozicioA = 30.5;
```



A változó neve, melynek beállítjuk az értéket



Az érték amelyet be akarunk állítani

- Értékként beállíthatjuk egy metódus eredményét:

```
xPozicioA = this.getX();
```

- Inicializatlan változó ezután a művelet után **inicializálttá** válik.



Kettő az egyben

- Változó létrehozása inicializálással együtt

```
double xPozicioA = 30.5;
```

Rövidítés az alábbi helyett:

```
double xPozicioA;  
xPozicioA = 30.5;
```

- **Tanács:**

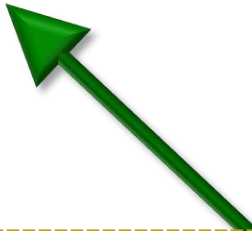
- Érdemes inicializálni a változókat rögtön a létrehozás során.
- **Iniciálatlan** változóval **nem lehet dolgozni**, a Java ezt szigorúan kontrolálja!



Változó értékének olvasása

- A változó neve azt **az értékét képviseli**, mely az adott pillanatban el van benne mentve.

```
double lepesHossz = 30;  
tekno.step(lepesHossz);
```



A teknős annyit lép előre, amilyen érték van **aktuálisan** elementve a *lepesHossz* változóban, tehát 30.



És végre a háromszög...

```
public void egyenloszaru(double szar, double szog) {  
    this.turn(-szog/2);  
    this.step(szar);  
    double xPozicioA = this.getX();  
    double yPozicioA = this.getY();  
    this.step(-szar);  
    this.turn(szog);  
    this.step(szar);  
    this.moveTo(xPozicioA, yPozicioA);  
}
```

Az *xPozicioA* és az *yPozicioA* változókba elmentettük a teknős koordinátáinak aktuális értékeit (A pont)

Felhasználjuk a változókba elmentett értékeket mint a paraméterek értéke a metódus hívásánál.



Az „=” jobb oldaláról

- Hozzárendelésnél az egyenlőségjel jobb oldalára írhatunk pl. tetszőleges **számtani kifejezést**...
- A hozzárendelés előtt először kiértékelődik a kifejezés az = jobb oldalán és ez az érték rendelődik hozzá a változóhoz

```
double szam = 10;
```

```
double hatvany = szam * szam;
```

```
hatvany = szam * szam * szam;
```



Mi történik?

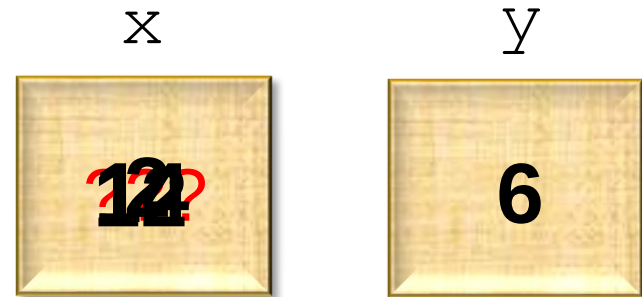
➔ **int** x;

x = 2;

int y = 6;

x = 2 * y; x ◀ 2 * 6;

x = x + 2; x ◀ 12 + 2;



- Mindig először az = jobb oldala számítódik ki!



Az „=” jobb oldaláról

- Ha egy double (int) típusú változóhoz értéket rendelünk, az = jobb oldalára tetszőleges, **számtani kifejezést** írhatunk, melynek eredménye egy valós (egész) szám - **kompatibilis érték**.
- **Megjegyzés:** számtani kifejezést írhatunk a hívott metódus paramétereinek helyére is

```
this.setDirection(2*elfordulás);
```

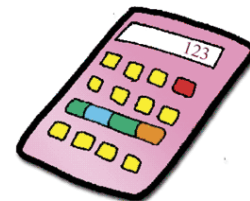


A metódus hívása előtt kiszámítódik a kifejezés értéke.



Számtani kifejezés

- Egy számtani kifejezésben használhatunk:



- **Matematikai műveletek** jeleit :
 - $*$ (szorzás), $+$ (összeadás), $-$ (kivonás), $/$ (osztás),
 - $\%$ (maradékos osztás): $10 \% 3 = 1$, $6 \% 8 = 6$, $12 \% 4 = 0$
- Kerek zárójeleket ($()$)
- Változók neveit, melyek a bennük lévő értéket képviselik
- számokat
- Értékeket visszaadó metódusokat:

```
double változás =  
    3 * this.distanceTo(100, 200);
```



Spirál

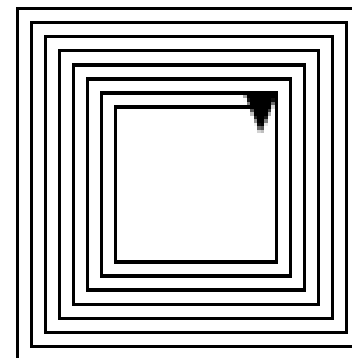
- Tanítsuk meg a teknőst a spirál metódusra:

```
public void negyzetesSpiral (int oldalszam)
```

- rajzolj **négyszet alakú spirált** a megadott oldalak számával
- Az első oldal hossza 150
- Minden lépéssel az oldal hossza 2-vel csökken



150, 148,
146, 144, ...





Típusszabályok

- Egy egész számú változóba (int típusú) csak egész számot menthetünk el
- Eredmény típusa:
 - $int + double \gg double$
 - $int + int \gg int$



Meddig él egy változó? (1)

- Az egymáshoz tartozó kapcsos zárójelek {} határozzák meg a **parancsok blokkjait**
- A blokkok további blokkokat tartalmazhatnak:
 - A metódus blokkja alblokként tartalmazhatja a *for* ciklus blokkját
- A változó akkor szűnik meg létezni, ha befejeződik azon blokk végrehajtása amelyben létrejött!
- A változó élettartama - a változó létrejöttétől (deklarálásától) az adott blokk csukó kapcsos zárójeléig.



Meddig él egy változó? (2)

```
for (int i=0; i<20; i++) {  
    double xPozicio = this.getX();  
    this.step(10);  
    this.turn(10);  
    this.setX(xPozicio + 8);  
}
```

Az xPozicio
20-szor jön létre
és szűnik meg.

Ennél a zárójelnél
szűnik meg az
xPozicio változó





A primitív változók típusai

● Egész számú értékek:

- **byte** (-128-tól 127-ig) 
- **short** (-32 768-tól 32 767-ig) 
- **int** (-2 147 483 648-tól 2 147 483 647-ig) 
- **long** (-9 223 372 036 854 775 808-tól 9 223 372 036 854 775 807-ig) 

Különböző típusok =
különféle **terjedelem** és
memóriakövetelmény

● Valós számok:

- **double** - pontosabb, több helyet foglal
- **float** - kevésbé pontos, kevesebb helyet foglal

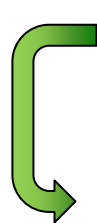


primitív = egyszerű



Véletlenek ...

- Javaban létezik egy metóduş, amely képes kigenerálni egy **véletlen valós számot** 0 és 1 között:

 `Math.random()`
`<0, 1)`



- Példa:

- Teknős elfordítása véletlen értékkel:

```
teknoş.turn(Math.random()*360);
```

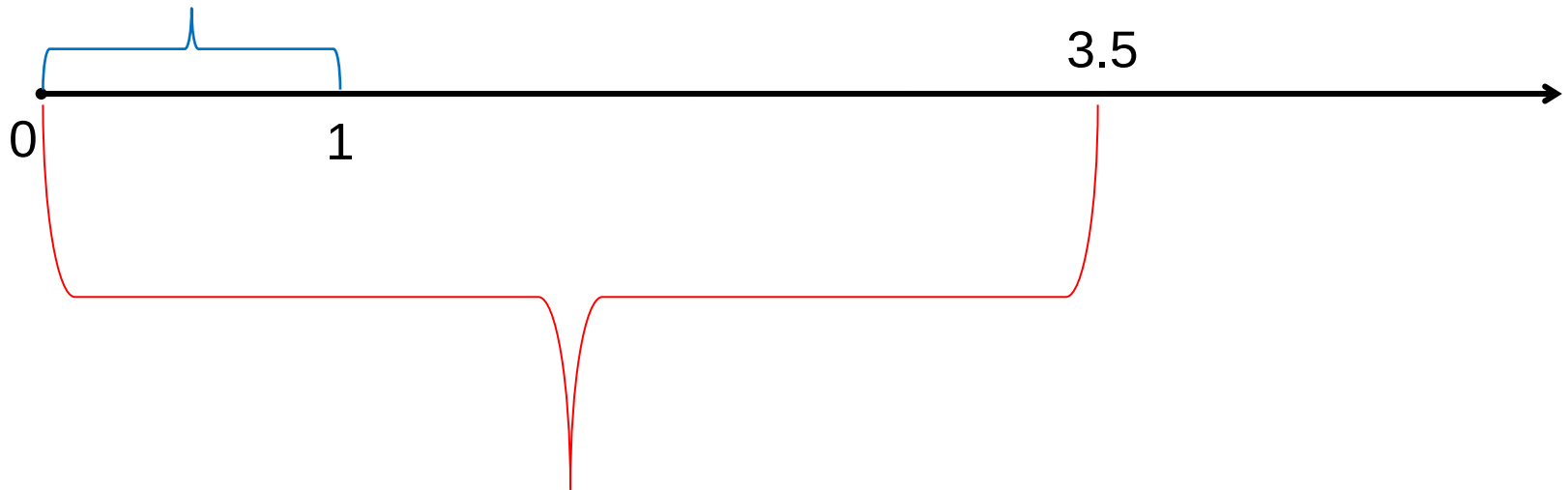
- Véletlen hosszúságú lépés 10 és 25 között:

```
teknoş.step(10 + Math.random()*15);
```



Véletlen számok

`Math.random()`



`Math.random() * 3.5`

- Véletlen szám az $<a, b)$ intervallumból :

$$a + \text{Math.random()} * (b - a)$$



Kerge teknősök

- Programozzuk meg a teknősök véletlenszerű lassított mozgását ...

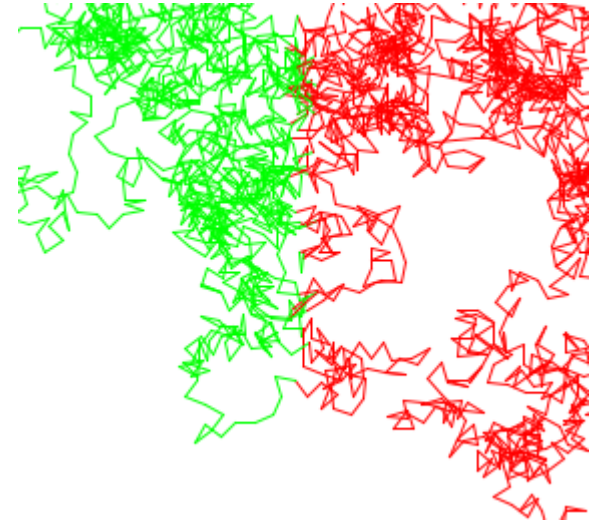
```
public void veletlenSeta(int lepesSzam) {  
    for (int i=0; i<lepesSzam; i++) {  
        this.setDirection(Math.random() * 360);  
        this.step(10);  
        JPAZUtilities.delay(30);  
    }  
}
```





Kerge teknős és a színek

- Változtassa meg a teknős a tolla színét aszerint, hogy a felület melyik oldalán áll:
 - **Ha** az x-koordinátád 150-től kisebb, **akkor** rajzolj zöld színnel, **egyébként** rajzolj piros színnel.





Feltételes parancs (1)

Feltétel

```
if (this.getX() < 150) {  
    this.setPenColor(Color.green);  
} else {  
    this.setPenColor(Color.red);  
}
```

A végrehajtandó
parancsok ha a feltétel
igaz

A végrehajtandó
parancsok ha a feltétel
hamis



Feltételes parancs (2)

```
if (feltétel) {  
    parancsok ha a feltétel igaz  
}  
else {  
    parancsok ha a feltétel hamis  
}
```





Logikai kifejezés (1)

- **A feltétel** mindig egy **logikai kifejezés**
- A logikai kifejezés egy olyan kifejezés, melynek eredménye egy **logikai érték**: igaz/hamis
- A logikai kifejezés tartalmazhat:
 - Számtani értékeket összehasonlító műveletjeleket
 - Egyenlőtlenség $>$, $<$, $>=$, $<=$
 - **Egyenlőség** $==$
 - Nem egyenlőség $!=$
 - Logikai műveletek (és, vagy, ...)

Vigyázat, nem „=”
(gyakori hiba!)



Logikai kifejezés (2)

● Logikai műveletek:

- $(\text{kifejezés1}) \ \&\& \ (\text{kifejezés2}) \ \dots$ $\wedge$
igaz a kifejezés1 és ugyanakkor a kifejezés2
- $(\text{kifejezés1}) \ || \ (\text{kifejezés2}) \ \dots$ $\vee$
igaz a kifejezés1 vagy a kifejezés2
- $!(\text{kifejezés})$ éppen akkor igaz, amikor a kifejezés hamis $\neg$

● Példák:

```
x == 2
```

```
this.getX() > 100
```

```
(x >= 100) && (x <= 200)
```

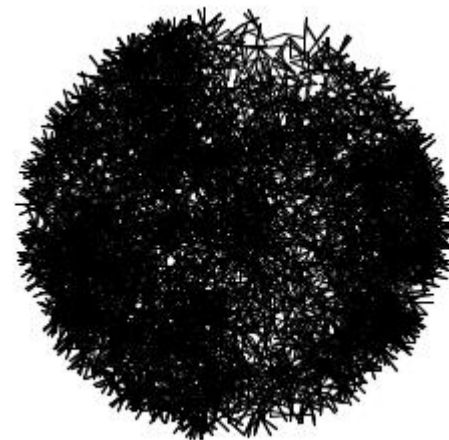
```
(x < 30) || (x > 100)
```

```
this.distanceTo(100, 100) > 300
```



Kerge teknősök láncához kötve

- Írjunk egy metódust, amelyben a teknős adott számú **véletlen lépést** fog végrehajtani úgy, hogy soha ne jusson el messzebb a kiindulópontjától mint 100
- Egy lépés:
 - Fordulj el véletlen fokkal
 - Tegyél egy 10 hosszú lépést
 - **Ha** távolabb vagy a kiindulóponttól mint 100,
akkor lépj vissza





Feltételes parancs else nélkül

- Néha csak akkor akarunk végrehajtani egy utasítást ha teljesül a feltétel, de **ellenkező esetben** nem csinálunk **semmit**

- Üres *else* ág írása helyett kihagyhatjuk:

```
if (feltétel) {  
    parancsok ha a feltétel igaz  
}
```



boolean

- Logikai érték elmentésére a **Boolean** adatípust használjuk melynek két értéke van **true** (igaz) és **false** (hamis):

```
boolean kozel = (this.distanceTo(100, 100) < 30);
```

```
boolean igazVagyok = true;
```

```
boolean hamisVagyok = false;
```



„Titkos rövidítések“

- Ha x egész számú változó, akkor:

- $x++$; ugyanaz mint $x = x + 1$;
- $x--$; ugyanaz mint $x = x - 1$;
- $x += 5$; ugyanaz mint $x = x + 5$;
- $x *= y$; ugyanaz mint $x = x * y$;
- ... és sok más ...



- Az $x++$ és $x-$ kivételével a rövidítéseket elővigyázatosan használjuk! Nem művészet olyan programot írni amelyet nehéz olvasni és érteni.



Amit (nem) tudunk...

- A metódus **paraméterei** nem más, mint **egyszerű változók**, melyek értéke azokkal az értékkel van inicializálva amilyenennel a metódust hívjuk...
- A változó **nevének** választásánál ajánlott betartani néhány **szabályt**:
 - Nem szabad számmal kezdődnie, nem tartalmazhat szóközt
 - Nem használunk ékezeteket
 - Csak kisbetűket használunk. Nagybetűt csak abban az esetben amikor a betű egy következő szónak a kezdetét jelöli.
 - `azEnNagyonHosszuNevuValtozom`