14. ÓRA

Dinamikus képek kezelése

Az ezen az órán vett függvények a GD nevű nyílt forráskódú programkönyvtáron alapulnak.

A GD könyvtár olyan eszközcsoport, melynek segítségével futásidőben képeket hozhatunk létre és kezelhetjük azokat. A GD-ről a `http://boutell.com/gd/` weboldal szolgál bővebb információval. Ha a könyvtárat telepítettük és a PHP-t is lefordítottuk, elkezdhetünk dinamikus képeket létrehozni a PHP grafikus függvényeivel. Sok rendszer még mindig a könyvtár egy régebbi változatát futtatja, amely a képeket GIF formátumban hozza létre. A könyvtár későbbi változatai licenszokok miatt nem támogatják a GIF formátumot. Ha rendszerünkre a könyvtár egy újabb változatát telepítettük, úgy is lefuttathatjuk a PHP-t, hogy a képalkotó függvények kimeneti formátuma PNG legyen, amelyet a legtöbb böngésző támogat.

A GD könyvtár segítségével menet közben hozhatunk létre bonyolult alakzatokat a PHP grafikus függvényeivel. Az óra során a következőket tanuljuk meg:

- Hogyan hozzunk létre és jelenítsünk meg egy képet?
- Hogyan dolgozzunk a színekkel?
- Hogyan rajzoljunk alakzatokat, például köríveket, téglalapokat és sokszögeket?

- Hogyan töltsünk ki színnel alakzatokat?
- Hogyan kezeljük a TrueType betűtípusokat?

Képek létrehozása és megjelenítése

Mielőtt elkezdhetnénk a képekkel foglalkozni, először szert kell tennünk egy képazonosítóra. Ezt az `imagecreate()` függvény segítségével tehetjük meg. Ennek két paramétere van, az egyik a kép magasságára vonatkozik, a másik a szélességére. Ekkor egy képazonosítót kapunk vissza, amelyet az órán tárgyalt majdnem mindegyik függvény esetében használni fogunk.

Miután megkaptuk az azonosítót, már majdnem készen állunk arra, hogy megjelenítsük első képünket a böngészőben. Ehhez az `imagegif()` függvényre van szükségünk, amelynek paramétere a képazonosító. A 14.1. programban a kép létrehozásához és megjelenítéséhez használt függvényeket láthatjuk.

14.1. program Dinamikusan létrehozott kép

```
1: <?php
2: header("Content-type: image/gif");
3: $kep = imagecreate( 200, 200 );
4: imagegif($kep);
5: ?>
```

Fontos, hogy a `Content-type` fejlécsort minden egyéb előtt küldtük a böngészőnek. Közölnünk kell a böngészővel, hogy egy képet küldünk, különben a program kimenetét HTML-ként kezeli. A programot a böngésző már közvetlenül vagy egy `IMG` elem részeként hívhatja meg.

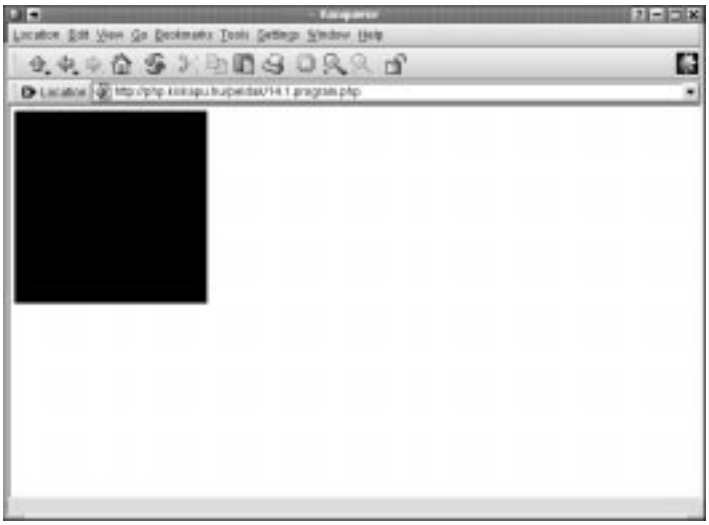
```
<img src="14.1.program.php" alt="egy PHP készítette kép">
```

A 14.1. ábrán láthatjuk a 14.1. program kimenetét.

Egy téglalapot hoztunk létre, de egyelőre nem tudjuk annak színét beállítani.

14.1. ábra

Dinamikusan létrehozott alakzat



A szín beállítása

A szín beállításához egy színazonosítóra van szükségünk. Ezt az `imagecolorallocate()` függvénnyel kaphatjuk meg, amelynek paramétere a képazonosító, illetve három `0` és `255` közötti egész szám, amelyek a vörös, zöld és kék színösszetevőket jelentik. Egy színazonosítót kapunk vissza, amellyel később alakzatok, kitöltések vagy szövegek színét határozhatjuk meg.

```
$piros = imagecolorallocate($kep, 255,0,0);
```

Az `imagecolorallocate()` függvény első meghívásakor egyúttal átállítja a paraméterül kapott alakzat színét is. Így ha az előző kódrészletet hozzáadjuk a 14.1. programhoz, egy vörös téglalapot kapunk.

Vonalak rajzolása

Mielőtt egy képre vonalat rajzolhatnánk, meg kell adnunk annak két végpontját.

A képet úgy képzeljük el, mint képernyőpontokból álló tömböt, melynek mindkét tengelye 0-tól számozódik. A számozás kezdőpontja a kép bal felső sarka.

Más szóval az (5, 8) koordinátájú képpont a hatodik képpont fentről lefelé és a kilencedik képpont balról jobbra számolva.

Az `imageline()` függvény két képpont közé rajzol egy egyenest. A függvény paraméterként egy képazonosítót vár, négy egész számot, amelyek az egyenes két végpontjának koordinátáit jelentik, valamint egy színazonosítót.

Ha a 14.2. programban megrajzoljuk a téglalap egy átlóját.

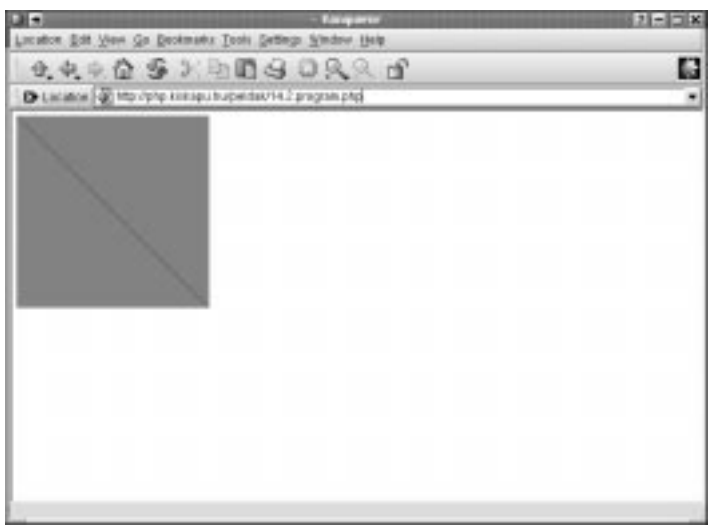
14.2. program Egyenes rajzolása az imageline() függvénnyel

```
1: <?php
2: header("Content-type: image/gif");
3: $kep = imagecreate( 200, 200 );
4: $piros = imagecolorallocate($kep, 255,0,0);
5: $kek = imagecolorallocate($kep, 0,0,255 );
6: imageline( $kep, 0, 0, 199, 199, $kek );
7: imagegif($kep);
8: ?>
```

Két színazonosítót kapunk, egyet a piros és egyet a kék színnek. Ezután a `$kek` változóban tárolt azonosítót használjuk a vonal színének megadására. Fontos megjegyeznünk, hogy a vonal a (199, 199) és nem a (200, 200) koordinátákban végződik, tekintve, hogy a képpontokat 0-tól kezdve számozzuk. A 14.2. ábrán a 14.2. program eredményét láthatjuk.

14.2. ábra

Egyenes rajzolása az imageline() függvénnyel



Alakzatok kitöltése

A PHP segítségével ugyanúgy kiszínezhetünk alakzatokat, mint kedvenc grafikai programunkkal. Az `imagefill()` függvény bemenete egy képazonosító, a kitöltés kezdőkoordinátái, valamint egy színazonosító. A függvény a kezdő képponttal megegyező színű szomszédos képpontokat a kijelölt színre állítja. A 14.3. program a képet az `imagefill()` függvény meghívásával teszi egy kicsit érdekesebbé.

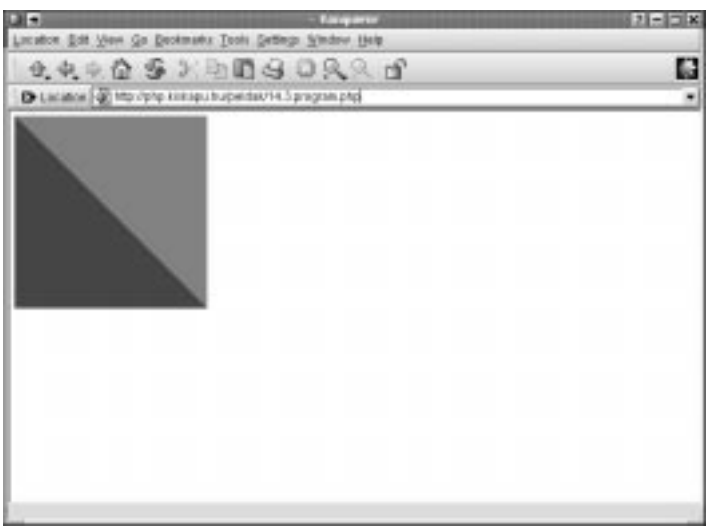
14.3. program Az imagefill() függvény használata

```
1: <?php
2: header("Content-type: image/gif");
3: $kep = imagecreate( 200, 200 );
4: $piros = imagecolorallocate($kep, 255,0,0);
5: $kek = imagecolorallocate($kep, 0,0,255 );
6: imageline( $kep, 0, 0, 199, 199, $kek );
7: imagefill( $kep, 0, 199, $kek );
8: imagegif($kep);
9: ?>
```

A 14.3. ábrán a 14.3. program kimenetét láthatjuk.

14.3. ábra

Az imagefill() függvény használata



Körív rajzolása

Az `imagearc()` függvény segítségével köríveket rajzolhatunk. Bemenete egy képazonosító, a középpont koordinátája, a szélességet meghatározó egész szám, a magasságot meghatározó egész szám, egy kezdő- és egy végpont (fokban mérve), valamint egy színazonosító. A köríveket az óramutató járásának megfelelően, 3 órától kezdve rajzoljuk. A következő kódrészlet egy negyed körívet rajzol ki:

```
imagearc( $kep, 99, 99, 200, 200, 0, 90, $kek );
```

Ez egy olyan körívrészletet jelenít meg, melynek középpontja a (99, 99) koordinátájú pontban van. A teljes magasság és szélesség 200-200 képpontnyi. A körív 3 óránál kezdődik és 90 fokot rajzolunk (azaz 6 óráig).

A 14.4. program teljes kört rajzol és kék színnel tölti ki.

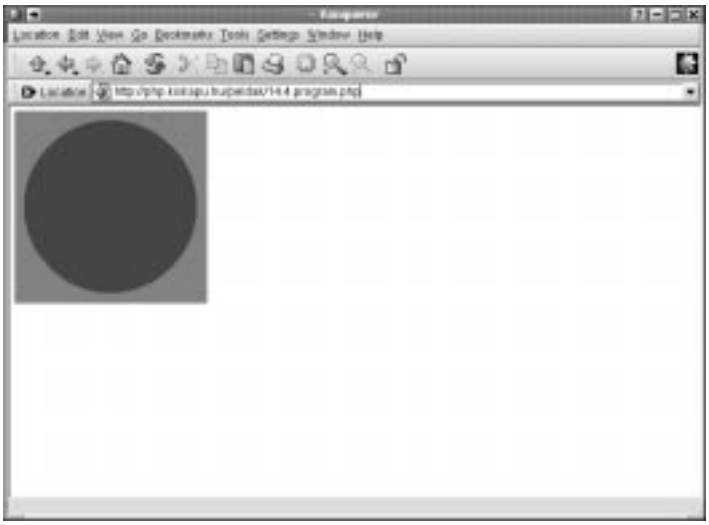
14.4. program Kör rajzolása az imagearc() függvénnyel

```
1: <?php
2: header("Content-type: image/gif");
3: $kep = imagecreate( 200, 200 );
4: $piros = imagecolorallocate($kep, 255,0,0);
5: $kek = imagecolorallocate($kep, 0,0,255 );
6: imagearc( $kep, 99, 99, 180, 180, 0, 360, $kek );
7: imagefill( $kep, 99, 99, $kek );
8: imagegif($kep);
9: ?>
```

A 14.4. program kimenetét a 14.4. ábrán láthatjuk.

14.4. ábra

Kör rajzolása az imagearc() függvénnyel



Téglalap rajzolása

A PHP `imagerectangle()` függvényével téglalapot rajzolhatunk. Az `imagerectangle()` bemenete egy képazonosító, a téglalap bal felső és jobb alsó sarkának koordinátája, valamint egy színazonosító. A következő kódrészlet olyan téglalapot rajzol, melynek bal felső és jobb alsó koordinátái rendre (19, 19) és (179, 179):

```
imagerectangle( $kep, 19, 19, 179, 179,  $kek );
```

Az alakzatot ezután az `imagefill()` függvénnyel tölthetjük ki. Mivel ez meglehetősen gyakori művelet, a PHP-ban létezik az `imagefilledrectangle()` függvény, amely ugyanazokat a bemeneti értékeket várja, mint az `imagerectangle()`, de az általunk meghatározott színnel ki is tölti a téglalapot. A 14.5. program egy kiszínezett téglalapot hoz létre és megjeleníti a böngészőben.

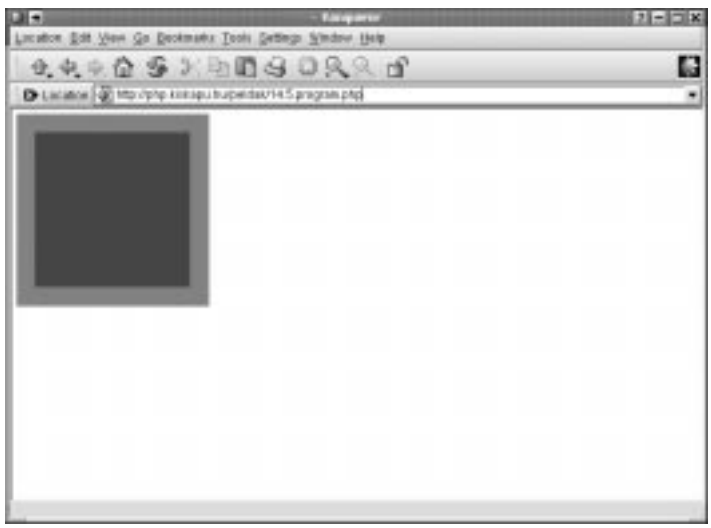
14.5. program Téglalap rajzolása az imagefilledrectangle() függvénnyel

```
1: <?php
2: header("Content-type: image/gif");
3: $kep = imagecreate( 200, 200 );
4: $piros = imagecolorallocate($kep, 255,0,0);
5: $kek = imagecolorallocate($kep, 0,0,255 );
6: imagefilledrectangle( $kep, 19, 19, 179, 179,  $kek );
7: imagegif( $kep );
8: ?>
```

A 14.5. ábrán a 14.5. program által előállított képet láthatjuk.

14.5. ábra

Téglalap rajzolása az imagefilledrectangle() függvénnyel



Sokszög rajzolása

Az `imagepolygon()` függvény segítségével kifinomultabb alakzatokat is rajzolhatunk. E függvény bemenete egy képazonosító, a pontok koordinátáiból álló tömb, az alakzat pontjainak számát jelző egész szám és egy színazonosító. Az `imagepolygon()` bemenetét képező tömbnek számmal indexeltnek kell lennie. Az első két elem az első pont koordinátáit adja meg, a második kettő a második pontét és így tovább. Az `imagepolygon()` függvény megrajzolja a pontok közötti vonalakat és az utolsó pontot automatikusan összeköti az elsővel. Az `imagefilledpolygon()` függvény segítségével színnel kitöltött sokszögek hozhatók létre.

A 14.6. program egy kitöltött sokszöget rajzol és jelenít meg a böngészőben.

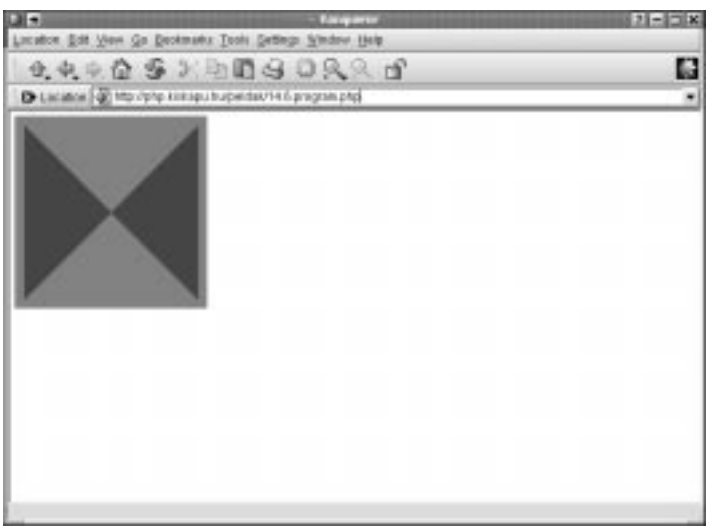
14.6. program Sokszög rajzolása az imagefilledpolygon() függvénnyel

```
 1: <?php
 2: header("Content-type: image/gif");
 3: $kep = imagecreate( 200, 200 );
 4: $piros = imagecolorallocate($kep, 255,0,0);
 5: $kek = imagecolorallocate($kep, 0,0,255 );
 6: $pontok = array (10, 10,
 7:                  190, 190,
 8:                  190, 10,
 9:                  10, 190
10:                  );
11: imagefilledpolygon( $kep, $pontok, count
                          ( $pontok )/2, $kek );
12: imagegif($kep);
13: ?>
```

Vegyük észre, hogy az összekötendő pontok száma az `imagefilledpolygon()` függvény hívásakor megegyezik a `$pontok` tömb elemei számának felével. A 14.6. ábrán a 14.6. program eredményét láthatjuk.

14.6. ábra

Sokszög rajzolása az imagefilledpolygon() függvénnyel



A színek átlátszóvá tétele

A PHP az `imagecolortransparent()` függvénnyel lehetővé teszi, hogy a kiválasztott színeket az ábrán belül áttetszővé tegyük. A függvény bemenete egy kép- és egy színazonosító. Ha a képet megjelenítjük egy böngészőben, az `imagecolortransparent()` függvénynek átadott szín áttetsző lesz. A 14.7. program kimenete a sokszöget rajzoló programot úgy változtatja meg, hogy az alakzat „úszni” fog a böngészőben, ahelyett, hogy egy háttérszín előtt jelenne meg.

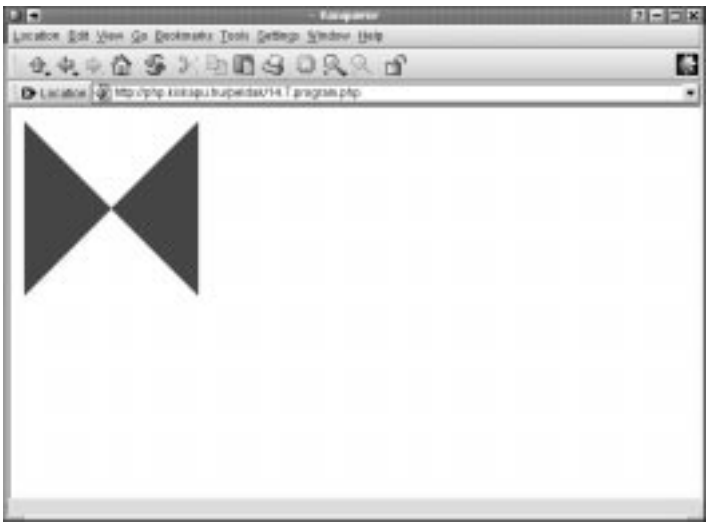
14.7. program A színek átlátszóvá tétele az imagecolortransparent() függvénnyel

```
 1: <?php
 2: header("Content-type: image/gif");
 3:
 4: $kep = imagecreate( 200, 200 );
 5: $piros = imagecolorallocate($kep, 255,0,0);
 6: $kek = imagecolorallocate($kep, 0,0,255 );
 7:
 8: $pontok = array (10, 10,
 9:              190, 190,
10:              190, 10,
11:              10, 190
12:              );
13:
14: imagefilledpolygon( $kep, $pontok, count
                       ( $pontok )/2, $kek );
15: imagecolortransparent( $kep, $piros );
16: imagegif($kep);
17: ?>
```

A 14.7. ábrán a 14.7. program kimenetét láthatjuk.

14.7. ábra

A színek átlátszóvá tétele az image-colortransparent() függvénnyel



Szövegek kezelése

Ha rendszerünkön vannak TrueType betűk, a képekre írhatunk is. A GD programkönyvtár mellett szükségünk lesz a FreeType programkönyvtár telepítésére is. Ha ezek rendelkezésünkre állnak, egy képben megjelenő diagramokat és navigációs elemeket is létrehozhatunk. A PHP biztosít számunkra egy olyan eszközt is, amellyel ellenőrizhetjük, hogy a beírandó szöveg elfér-e a rendelkezésre álló helyen.

Szövegírás az imageTTFtext() függvénnyel

Az `imageTTFtext()` függvénnyel az ábrákra szöveget írhatunk. A függvény nyolc bemenő paramétere a következő: képazonosító, méret, amely a kiírandó szöveg magasságára utal, szög, a kezdő `x` és `y` koordináták, színazonosító, a TrueType betűtípus elérési útvonala és a kiírandó szöveg.

A kiírandó szöveg kezdőkoordinátája határozza meg, hol lesz a szöveg első karakterének alapvonala.

A 14.8. program egy karakterláncot ír az ábrára, majd az eredményt megjeleníti a böngészőben.

14.8. program Szövegírás az imageTTFtext() függvénnyel

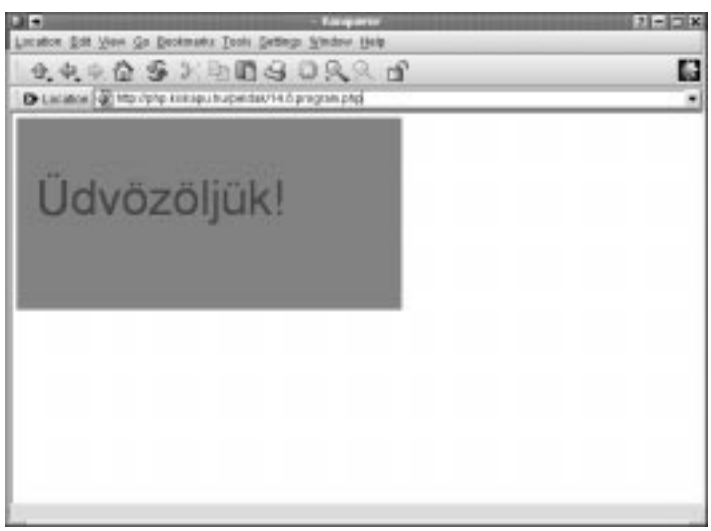
```
 1: <?php
 2: header("Content-type: image/gif");
 3:
 4: $kep = imagecreate( 400, 200 );
 5: $piros = imagecolorallocate($kep, 255,0,0);
 6: $kek = imagecolorallocate($kep, 0,0,255 );
 7: $betukeszlet = "/usr/local/office52/share/fonts/
                    /TrueType/arial.ttf";
 8:
 9: imageTTFtext( $kep, 50, 0, 20, 100, $kek,
                  $betukeszlet, "Üdvözöljük!" );
10:
11: imagegif($kep);
12: ?>
```

Létrehozunk egy 400 képpontnyi széles és 200 képpontnyi magas vásznat, megadunk két színt és a `$betukeszlet` változóban tároljuk a TrueType betűtípus elérési útját. Ezután ráírjuk az ábrára a "Üdvözöljük!" szöveget. Figyelem, a betűtípusok a kiszolgálón feltehetőleg más könyvtárban találhatók! Ha nem vagyunk biztosak a helyben, keressük a `.ttf` kiterjesztésű fájlokat.

Az `imageTTFtext()` meghívásához az 50-es magasságot, 0 fokos szöget és (20, 100) kezdőkoordinátát adtuk át. Emellett a függvény megkapja még a `$kek` változóban tárolt színazonosítót és a `$betukeszlet` változóban tárolt betűtípust, végül a kiírandó szöveget. Az eredményt a 14.8. ábrán láthatjuk.

14.8. ábra

Szöveg írása az imageTTFtext() függvénnyel



Persze most még csak találgatunk, hová tegyük a szöveget. A betűméret nem árul el sokat a szöveg magasságáról, a szélességéről még kevesebbet.
Az `imageTTFtext()` függvény persze visszaad információt a szöveg kiterjedéséről, de akkorra már minden eldőlt. Szerencsére a PHP lehetőséget nyújt arra, hogy próbálkozzunk, mielőtt még élesben elvégeznénk a műveletet.

Szöveg kiterjedésének ellenőrzése az imageTTFbox() függvénnyel

A szöveg kiterjedéséről az `imageTTFbox()` függvénnyel szerezhetünk információt, amely onnan kapta nevét, hogy a szöveget határoló dobozt írja le. A függvény bemenete a betűméret, a szög, a betűtípus elérési útja és a kérdéses szöveg.
Azon kevés függvények egyike, melyekhez nem kell a képazonosító. Egy nyolcelemű tömböt ad vissza, amely a szöveg kiírásához szükséges terület paramétereit (koordinátáit) tartalmazza. A 14.1. táblázatban a függvény által visszaadott tömböt értelmezzük.

14.1. táblázat Az imageTTFbox() függvény által visszaadott tömb

Sorszám	*Leírás*
0	bal alsó (vízszintes tengely)
1	bal alsó (függőleges tengely)
2	jobb alsó (vízszintes tengely)
3	jobb alsó (függőleges tengely)
4	jobb felső (vízszintes tengely)
5	jobb felső (függőleges tengely)
6	bal felső (vízszintes tengely)
7	bal felső (függőleges tengely)

A függőleges tengelyhez visszaadott számok (1-es, 3-as, 5-ös, és 7-es tömbelemek) a szöveg alapvonalához viszonyulnak, amelynek koordinátája 0. A szöveg tetejének a függőleges tengelyen mért értékei ehhez képest általában negatív számok, míg a szöveg aljának itt mért értékei általában pozitívak. Így kapjuk meg, hány képponttal nyúlik a szöveg az alapvonal alá. Az `imageTTFbox()` által figyelembe vett koordinátarendszert úgy kell elképzelnünk, mintha a szöveg alapvonala lenne az `y` irányban vett `0` érték és a negatív koordináták attól felfelé, a pozitívak pedig attól lefelé helyezkednének el. Így például ha az `imageTTFbox()` függvénnyel ellenőrizendő szövegben van egy `"y"`, a visszakapott tömbben az `1`-es indexű elem lehet, hogy `3` lesz, mivel az `"y"` alsó szára három képponttal nyúlik

az alapvonal alá. Lehetséges, hogy a 7-es elem –10 lesz, mivel a szöveg 10 képponttal van az alapvonal fölé emelve. A pontos értékek betűtípustól és betűmérettől függnek.

Csak hogy bonyolítsuk a helyzetet, úgy tűnik, hogy az imageTTFbox() függvény által visszaadott alapvonal és a rajzolás közben látható között 2 képpontos eltolás van. Ezt úgy ellensúlyozhatjuk, hogy az alapvonalat két képpontnyival magasabbnak képzeljük, mint amilyennek az imageTTFbox() függvény mutatja.

A vízszintes tengelyen a bal oldali számok (a 0-ás és a 6-os elem) esetében az imageTTFbox() függvény az adott kezdőpont előtt kezdődő szöveget negatív számmal kifejezett eltolással jelöli. Ez általában alacsony szám, mivel a betűk jellemzően jelentősebb eltérést mutatnak az alapvonaltól, mint az x irányú kezdőkoordinátától (gondoljunk a p, g vagy j betűkre). A vízszintes tengelyen így már csak az elvárt pontosság kérdése, hogy szükségünk van-e a koordináták kiigazítására.

Az imageTTFbox() függvény által visszaadott értékeket arra is használhatjuk, hogy az ábrán belül a szöveget igazítsuk. A 14.9. program dinamikusan küldi a böngészőnek a szöveget, úgy, hogy mind a vízszintes, mind a függőleges tengelyen középre helyezi azt.

14.9. program Szöveg elhelyezése az imageTTFbox() függvény használatával

```
 1: <?php
 2: header("Content-type: image/gif");
 3: $magassag = 100;
 4: $szelesseg = 200;
 5: $betumeret = 50;
 6: if ( ! isset ( $szoveg ) )
 7:      $szoveg = "Változtasson meg!";
 8: $kep = imagecreate( $szelesseg, $magassag );
 9: $piros = imagecolorallocate($kep, 255,0,0);
10: $kek = imagecolorallocate($kep, 0,0,255 );
11: $betukeszlet = "/usr/local/office52/share/fonts/
                    /TrueType/arial.ttf";
12: $szovegszelesseg = $szelesseg;
13: $szovegmagassag;
14: while ( 1 )
15:      {
16:      $doboz = imageTTFbox( $betumeret, 0,
                   $betukeszlet, $szoveg );
17:      $szovegszelesseg = abs( $doboz[2] );
18:      $szovegmagassag = ( abs($doboz[7]) )-2;
```

14.9. program (folytatás)

```
19:      if ( $szovegszelesseg < $szelesseg - 20 )
20:          break;
21:      $betumeret--;
22:      }
23: $gifXkozep = (int) ( $szelesseg/2 );
24: $gifYkozep = (int) ( $magassag/2 );
25: imageTTFtext(     $kep, $betumeret, 0,
26:          (int) ($gifXkozep-($szovegszelesseg/2)),
27:          (int) ($gifYkozep+($szovegmagassag/2)),
28:          $kek, $betukeszlet, $szoveg );
29: imagegif($kep);
30: ?>
```

A kép magasságát és szélességét a `$magassag` és `$szelesseg` változókban tároljuk, a betűtípus alapbeállítású méretét pedig 50-re állítjuk. Megvizsgáljuk, hogy létezik-e már a `$szoveg` nevű változó; ha nem, beállítjuk egy alapértékre. Ezzel az eljárással egy ábrának is lehetnek paraméterei, fogadhat adatot egy weboldalról – akár egy URL keresőkifejezéséből, akár egy kitöltött űrlapról. A képazonosító előállításához az `imagecreate()` függvényt használjuk. A színazonosítókat a szokásos módon állítjuk elő, a `$betukeszlet` nevű változóban pedig a TrueType betűtípus elérési útját rögzítjük.

Azt szeretnénk, hogy a `$szoveg` változóban tárolt karaktersorozat elférjen a rendelkezésére álló helyen, de egyelőre nem tudhatjuk, hogy ez sikerülni fog-e. A `while` utasításon belül átadjuk a betűtípus elérési útját és a szöveget az `imageTTFbox()` függvénynek, az eredményül kapott tömböt pedig a `$doboz` változóba helyezzük. A `$doboz[2]` tartalmazza a jobb alsó sarok helyét a vízszintes tengelyen. Ezt vesszük a karaktersorozat szélességének és a `$szovegszelesseg` változóban tároljuk.

A szöveget függőlegesen is középre szeretnénk helyezni, de a szöveg alapvonala alatt lévő részt figyelmen kívül hagyva. Az alapvonal feletti magasság meghatározásához használhatjuk a `$doboz[7]` változóban található szám abszolút értékét, bár ezt még ki kell igazítanunk a fent említett két képponttal. Ezt az értéket a `$szovegmagassag` változóban tároljuk.

Most, hogy kiszámítottuk a szöveg szélességének értékét, összevethetjük az ábra szélességével (tíz képpontnyival kell kevesebbnek lennie, a keret miatt). Ha a szöveg keskenyebb, mint a kép szélessége, befejezzük a ciklust. Ellenkező esetben csökkentjük a betűméretet és újra próbálkozunk.

A `$magassag` és a `$szelesseg` változók értékét elfelezve megkapjuk az ábra hozzávetőleges középpontját. A szöveget az ábra középpontjából és a szöveg magasságából, illetve szélességéből kiszámított eltolással az ábrára írjuk, végül megjelenítjük a képet a böngészőben. A 14.9. ábrán láthatjuk a 14.9. program kimenetét.

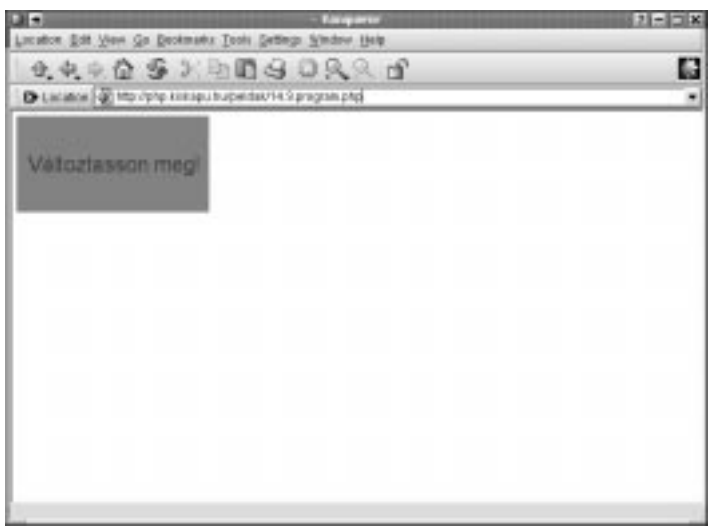
Ezt a kódot egy másik oldal is meghívhatja, egy `IMG` elem részeként. A következő részlet egy olyan programot hoz létre, amely lehetővé teszi az egyes felhasználóknak, hogy saját szöveget jelenítsenek meg az ábrán:

```
1: <?php
2: if ( ! isset( $szoveg ) )
3:      $szoveg = "Dinamikus szöveg!";
4: ?>
5: <form action="<? print $PHP_SELF ?>" method="POST">
6: <input type="text" name="szoveg">
7: </form>
8: <p>
9: <img src="14.9.program.php?<? print
              "szoveg=".urlencode($szoveg) ?>">
```

Amikor a 14.9. programot meghívjuk, egy olyan keresőkifejezéssel egészítjük ki a hívást, amely tartalmazza a megjelenítendő szöveget. A tizenkilencedik órában többet is tanulhatunk az információk programról programra való átadásáról.

14.9. ábra

Szöveg elhelyezése az imageTTFbox() függvény használatával



A fenti elemek összegyúrása

Készítsünk egy példát az ezen az órán tanult függvények használatával! Tegyük fel, hogy arra kértek bennünket, hogy készítsünk egy dinamikus oszlopgrafikont, amely több címkézett számot hasonlít össze. Az egyes oszlopok alatt a megfelelő címkének kell lennie. Ügyfelünknek tudnia kell módosítani a grafikon oszlopainak számát, a kép szélességét és magasságát és a grafikont körülvevő keret méretét. A grafikont fogyasztói szavazáshoz fogják használni, így az adatokat csak látszólagos pontossággal kell megjeleníteni. Egy sokkal részletesebb lebontást az ábrát tartalmazó oldal HTML részében találhatunk.

Az oszlopokat és értékeiket a legegyszerűbb módon egy asszociatív tömbben tárolhatjuk. Miután megvan ez a tömb, ki kell számolnunk az oszlopok számát és a tömb legnagyobb értékét:

```
$cellak = array ( "tetszik"=>200, "nem tetszik"=>400,
                  "közömbös"=>900 );
$max = max( $cellak );
$cellaszam = count( $cellak );
```

Létre kell hoznunk néhány változót, így ügyfelünk egyéni igényeihez igazíthatja az ábrát:

```
$teljesszelesseg = 400;
$teljesmagassag = 200;
$xmargo = 20; // jobb és bal margó
$ymargo = 20; // felső és alsó margó
$oszlopkoz = 5; // az oszlopok közötti hely
$alsokeret = 40; // az ábra alján kimaradó hely
 ➥ (a margót nem számolva)
$betukeszlet = "/usr/local/office52/share/fonts/
 ➥ /TrueType/arial.ttf";
```

Ügyfelünk a változók módosításával meghatározhatja az ábra magasságát és szélességét. Az `$xmargo` és `$ymargo` a grafikon körüli margóvastagságot adja meg. Az `$oszlopkoz` az oszlopok közötti távolságot határozza meg, az `$alsokeret` változó pedig az oszlopok alatti címkéknek szánt hely nagyságát.

Most hogy megvannak ezek az értékek, a némi számolgatás árán belőlük kapott hasznos számokat változókba tesszük:

```
$vaszonszelesseg = ( $teljesszelesseg - $xmargo*2 );
$vaszonmagassag = ( $teljesmagassag - $ymargo*2 - $alsokeret
);
$xPoz = $xmargo; // a rajzolás kiindulópontja az x tengelyen
$yPoz = $teljesmagassag - $ymargo - $alsokeret; //
 ➥ a rajzolás kiindulópontja az y tengelyen
$cellaszelesseg = (int) (( $vaszonszelesseg - ( $oszlopkoz *
 ➥ ( $cellaszam-1 ) )) / $cellaszam) ;
$szovegmeret = (int)($alsokeret);
```

Kiszámítjuk a rajzfelületet (azt a területet, ahová majd az oszlopok kerülnek). A vízszintes összetevőt úgy kapjuk meg, hogy a teljes szélességből kivonjuk a margó kétszeresét. A függőleges mérethez az `$alsokeret` változót is számításba kell vennünk, hogy a címkéknek helyet hagyjunk. Az `$xPoz` tárolja azt az `x` koordinátát, amelytől az oszlopok rajzolását kezdjük, így ennek értékét az `$xmargo` változó értékére állítjuk, amely a margó értékének vízszintes összetevőjét tárolja. Az oszlopok talppontját tároló `$yPoz` változó értékét úgy kapjuk meg, ha az ábra teljes magasságából kivonjuk a margót és a címkéknek kihagyott hely nagyságát, amelyet az `$alsokeret` változóban tároltunk.

A `$cellaszelesseg` az egyes oszlopok szélességét tárolja. Ahhoz, hogy az értékét megkapjuk, számoljuk ki az oszlopok közötti helyek összegét, ezt vonjuk ki a diagram szélességéből, majd az eredményt osszuk el az oszlopok számával.
Kezdetben úgy állítjuk be a szöveg méretét, hogy egyezzen a szöveg számára szabadon maradt terület magasságával (amit az `$alsokeret` változóban tárolunk).

Mielőtt megkezdenénk a munkát az ábrával, meg kell határoznunk a szöveg méretét. Viszont nem tudjuk, milyen szélesek lesznek a címkék és mindenképpen azt szeretnénk, hogy az egyes címkék elférjenek a felettük levő oszlop szélességén belül. A `$cellak` tömbön egy ciklus segítségével meghatározzuk a legnagyobb címke méretét:

```
forach( $cellak as $kulcs => $ertek )
     {
     while ( 1 )
          {
          $doboz = ImageTTFbBox( $szovegmeret, 0,
                    $betukeszlet, $kulcs );
          $szovegszelesseg = $doboz[2];
          if ( $szovegszelesseg < $cellaszelesseg )
               break;
          $szovegmeret—;
          }
     }
```

Minden elem esetében egy ciklust kezdve, az `imageTTFbox()` segítségével információt szerzünk a címke méretét illetően. A kapott értéket a `$doboz[2]` változóban találhatjuk meg, amit összevetünk a `$cellaszelesseg` változóval, amely egy oszlop szélességét tartalmazza. A ciklust abban az esetben állítjuk le, ha a szöveg szélessége kisebb, mint az oszlopszélesség. Egyébként pedig csökkentjük a `$szovegmeret` értékét és újra ellenőrizzük a méreteket. A `$szovegmeret` addig csökken, míg a tömb összes címkéje kisebb nem lesz az oszlopszélességnél.

Most már végre létrehozhatunk egy képazonosítót és elkezdhetünk dolgozni vele.

```
$kep = imagecreate( $teljesszelesseg, $teljesmagassag );
$piros = ImageColorAllocate($kep, 255, 0, 0);
$kek = ImageColorAllocate($kep, 0, 0, 255 );
$fekete = ImageColorAllocate($kep, 0, 0, 0 );
forach( $cellak as $kulcs => $ertek )
     {
     $cellamagassag = (int) (($ertek/$max) * $vaszonmagassag);
     $kozeppont = (int)($xPoz+($cellaszelesseg/2));
     imagefilledrectangle( $kep, $xPoz, ($yPoz-$cellamagassag),
        ($xPoz+$cellaszelesseg), $yPoz, $kek );
     $doboz = ImageTTFBox( $szovegmeret, 0, $betukeszlet,
              $kulcs );
     $szovszel = $doboz[2];
     ImageTTFText( $kep, $szovegmeret, 0, ($kozeppont-
                  ($szovszel/2)),
              ($teljesmagassag-$ymargo), $fekete, $betukeszlet,
               $kulcs );
     $xPoz += ( $cellaszelesseg + $oszlopkoz);
     }
imagegif( $kep );
```

Ezt az `imagecreate()` függvénnyel tesszük meg, majd kiosztunk néhány színt is. Újra végiglépkedünk a `$cellak` tömbön. Kiszámítjuk az oszlop magasságát és az eredményt a `$cellamagassag`-ba helyezzük. Kiszámoljuk a középpont x koordinátáját, amely megegyezik az `$xPoz` és a fél oszlop szélességének összegével.

Az `imagefilledrectangle()` függvénynek az `$xPoz`, `$yPoz`, `$cellamagassag` és `$cellaszelesseg` változókat átadva megrajzoljuk az oszlopot.

A szöveg középre igazításához megint szükségünk van az `imageTTFbox()` függvényre, amelynek eredménytömbjét a `$doboz` változóban tároljuk. A `$doboz[2]` értéket fogjuk munkaszélességnek választani, ezt bemásoljuk a `$szovszel` átmeneti változóba. Most már elegendő információ áll rendelkezésünkre ahhoz, hogy kiírassuk a címkét. Az `x` koordináta a `$kozeppont` változó és a szövegszélesség felének különbsége lesz, az `y` koordináta pedig az alakzat magasságának és a margónak a különbsége.

Megnöveljük az `$xPoz` változót, felkészülve ezzel a következő oszlop feldolgozására.

A ciklus végeztével megjelenítjük az elkészült képet.

A teljes programot a 14.10. példában láthatjuk, a végeredményt pedig a 14.10. ábrán.

14.10. program Egy dinamikus oszlopdiagram

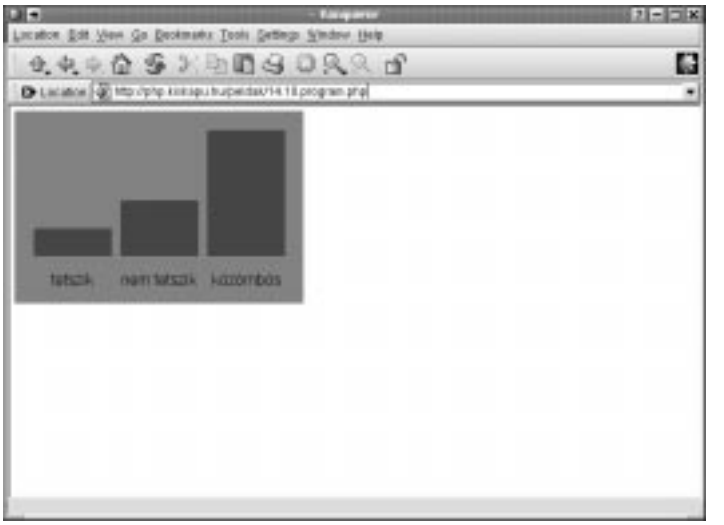
```
 1: <?php
 2: header("Content-type: image/gif");
 3: $cellak = array ( "tetszik"=>200,
                      "nem tetszik"=>400,
                      "közömbös"=>900 );
 4: $max = max( $cellak );
 5: $cellaszam = count( $cellak );
 6: $teljesszelesseg = 300;
 7: $teljesmagassag = 200;
 8: $xmargo = 20; // jobb és bal margó
 9: $ymargo = 20; // fenti és lenti margó
10: $oszlopkoz = 10; // az oszlopok közötti hely
11: $alsokeret = 30; // az ábra alján kimaradó hely
                        (a margót nem számolva)
12: $betukeszlet = "/usr/local/office52/share/fonts/
                   /TrueType/arial.ttf";
13: $vaszonszelesseg = ( $teljesszelesseg - $xmargo*2 );
14: $vaszonmagassag = ( $teljesmagassag - $ymargo*2
                        - $alsokeret );
15: $xPoz = $xmargo; // a rajzolás kiindulópontja
                        az x tengelyen
16: $yPoz = $teljesmagassag - $ymargo - $alsokeret; //
           a rajzolás kiindulópontja az y tengelyen
17: $cellaszelesseg = (int) (( $vaszonszelesseg -
     ( $oszlopkoz * ( $cellaszam-1 ) )) / $cellaszam) ;
```

14.10. program (folytatás)

```
18: $szovegmeret = (int)($alsokeret);
19: // betűméret kiigazítása
20: forach( $cellak as $kulcs => $ertek )
21:      {
22:      while ( 1 )
23:           {
24:           $doboz = ImageTTFbBox( $szovegmeret, 0,
                       $betukeszlet, $kulcs );
25:           $szovegszelesseg = abs( $doboz[2] );
26:           if ( $szovegszelesseg < $cellaszelesseg )
27:                break;
28:           $szovegmeret—;
29:           }
30:      }
31: $kep = imagecreate( $teljesszelesseg,
                        $teljesmagassag );
32: $piros = ImageColorAllocate($kep, 255, 0, 0);
33: $kek = ImageColorAllocate($kep, 0, 0, 255 );
34: $fekete = ImageColorAllocate($kep, 0, 0, 0 );
35: $szurke = ImageColorAllocate($kep, 100, 100, 100 );
36:
37: forach( $cellak as $kulcs => $ertek )
38:      {
39:      $cellamagassag = (int) (($ertek/$max) *
                          $vaszonmagassag);
40:      $kozeppont = (int)($xPoz+($cellaszelesseg/2));
41:      imagefilledrectangle( $kep, $xPoz,
                             ($yPoz-$cellamagassag),
                             ($xPoz+$cellaszelesseg),
                             $yPoz, $kek );
42:      $doboz = ImageTTFbBox( $szovegmeret, 0,
                  $betukeszlet, $kulcs );
43:      $szovszel = $doboz[2];
44:      ImageTTFText( $kep, $szovegmeret, 0,
                       ($kozeppont-($szovszel/2)),
45:                ($teljesmagassag-$ymargo), $fekete,
                   $betukeszlet, $kulcs );
46:      $xPoz += ( $cellaszelesseg + $oszlopkoz);
47:      }
48: imagegif( $kep );
49: ?>
```

14.10. ábra

Egy dinamikus oszlopdiagram



Összefoglalás

A GD programkönyvtár PHP-s támogatása lehetővé teszi, hogy dinamikus oszlopdiagramokat és navigációs elemeket hozhassunk létre, viszonylag könnyen.

Ezen az órán megtanultuk, hogyan használjuk az `imagecreate()` és az `imagegif()` függvényeket képek létrehozására és megjelenítésére. Azt is megtanultuk, hogyan szerezzünk színazonosítót és hogyan töltsünk ki színnel területeket az `imagecolorallocate()` és az `imagefill()` függvényekkel. Megnéztük, hogyan használjunk vonal- és alakzat függvényeket alakzatok körvonalainak megrajzolására és kitöltésére. Megtanultuk, hogyan használjuk a PHP által támogatott FreeType programkönyvtárat a TrueType betűtípusokkal való munkára, valamint elemeztünk egy programot, amelyben egy alakzatra szöveget írtunk és megnéztünk egy példát, amelyben a tanult eljárásokat a gyakorlatban is láthattuk.

Kérdések és válaszok

Fellépnek-e teljesítménybeli problémák dinamikus képek használata esetén?

Egy dinamikusan létrehozott kép lassabban töltődik be a böngészőbe, mint egy már létező fájl. A program hatékonyságától függően ezt a felhasználó nem feltétlenül veszi észre, de azért a dinamikus képeket módjával használjuk.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Milyen fejlécsort kell a böngészőnek küldenünk a GIF kép létrehozása és megjelenítése előtt?

2. Milyen függvényt használnánk egy képazonosító előállításához, amelyet a többi függvénynél használhatunk?

3. Milyen függvényt használnánk a létrehozott GIF kép megjelenítésére?

4. Milyen függvény segítségével szerezhetjük meg a színazonosítót?

5. Melyik függvénnyel rajzolnánk vonalat egy dinamikus képre?

6. Melyik függvénnyel töltenénk ki színnel egy dinamikus alakzatot?

7. Melyik függvényt használhatjuk körív rajzolására?

8. Hogyan rajzoljunk téglalapot?

9. Hogyan rajzoljunk sokszöget?

10. Melyik függvénnyel írnánk egy dinamikus alakzatra (a FreeType programkönyvtár segítségével)?

Feladatok

1. Írjunk egy programot, amely egy folyamatjelzőt hoz létre, amely mutatja, mennyi pénz folyt be egy gyűjtés alkalmával az adott célra szükséges pénzből.

2. Írjunk egy programot, amely egy főcímet ábrázoló képet ír ki egy bejövő űrlap vagy lekérdezés adataiból. Tegyük lehetővé, hogy a felhasználó határozza meg a rajzterületet, az előtér és a háttér színét, valamint az árnyék meglétét és annak méretét.