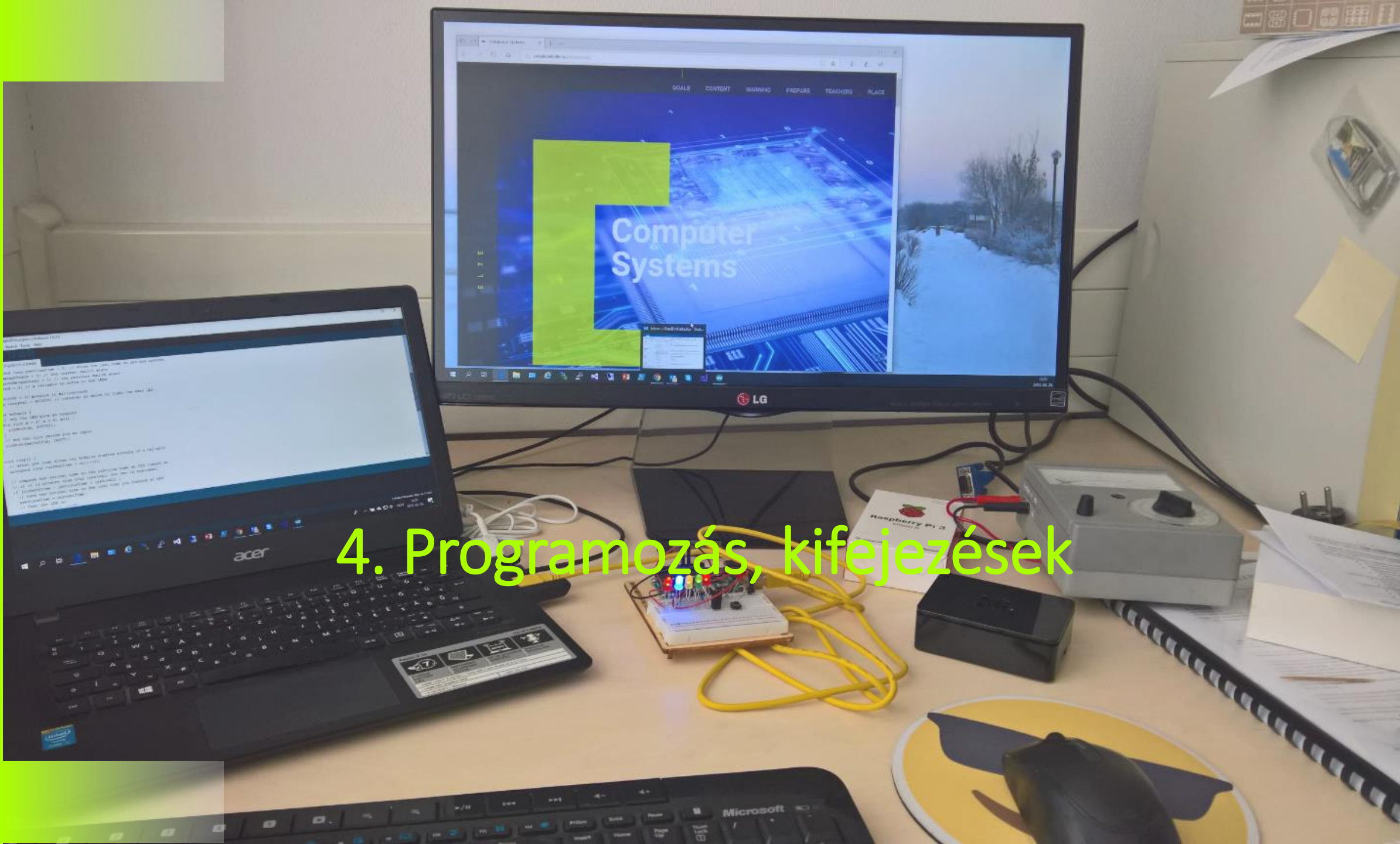
ELTE INFORMATIKA



4. Programozás, kifejezések

Visszatekintés

- Számítógépek, jelek, információk, számábrázolás, kódolás
- Felépítés, operációs rendszer, kliens-szerver szerep
- Grafikus, karakteres kapcsolat, fájlrendszerek
- Alapvető parancsok, folyamatok előtérben, háttérben
- I/O átirányítás
- Szűrők
- Reguláris kifejezések

Mi jön ma?

- Változók, használatuk
- Változó behelyettesítés
- Parancs behelyettesítés
- Elemi műveletek (aritmetikai, szöveges)
- Logikai kifejezések
- …

Változók UNIX shellben I.

- Szöveges változó létrehozására van lehetőség.
  - Környezeti vagy shell változóról beszélhetünk.
- Változónév betű lehet, majd szám és aláhúzás karakterek használhatók.
- Környezeti változók, mind a környezetben, mind az abból indított parancsokban láthatók. (globális)
  - env – program futása a definiált környezetben
    - Ha nincs paraméter akkor a környezeti változókat írja ki!
    - Különböző rendszerekben eltér a kiírt változók köre!
  - PATH (.profile), elválasztó karakter :,PS1, LOGNAME,stb.

Változók UNIX shellben II.

- Változó definiálás: = jel
  - Példa: csapat=Fradi; szam=5;
  - mondat=”Alma a fa alatt.”
  - spec_mondat=’Ez 5$ és 3%\10% lesz!’
- Változó tartalma mindig karakter, még a szám se szám!
- set parancs: összes változó kiírása (környezeti is)
- unset parancs: változó törlése
  - unset csapat # set eredmény: _=alma

Változó tartalma, hatóköre

- Példa: hajra=”Hajrá Fradi!”
- Tartalom: $hajra
  - echo $hajra # Hajrá Fradi!
  - echo Mondjuk: $hajra #behelyettesítés
- export hajra: A hajra változó környezeti lesz!
  - Ettől az unset ezt is törli!
- Környezeti változót mindenki látja!
  - Sima változót csak az aktuális shell!

Változó behelyettesítés

- csapat=Fradi; echo $csapatka # Mi lesz az eredmény?
- echo $csapat a legjobb!; # Természetesen!
- echo ${csapat}a legjobb!
- unset csapat; x=${csapat-Újpest} # ha csapat nincs, x=Újpest
  - inicializálás ha nem létezik
  - x=${csapat=Újpest} # csapat is változik!
  - echo $x a legjobb! # ???
- y=${csapat?Szia} # ha csapat nem definiált, a Szia kiírásra kerül, majd kilép a shell, y nem kap új értéket!
- y=${csapat+Újpest} # ha csapat definiált, y=Újpest

Parancsbehelyettesítés

- `parancs` parancsot végrehajtja, parancs kimenete kerül a helyére
  - Bash shellben: $(parancs)
  - ki_vagyok=`whoami` # $(whoami)
  - a=`date` ; b='date' #
  - echo $a # ???
  - echo $b # ???
    - A date szó lesz az eredmény.
  - eval $b #man eval,
    - Ugyanaz az eredmény, mintha a parancs $b lenne eval nélkül!

Első shell scriptem

- A fájl neve: elso

```
illes@panda:~$ vi elso
illes@panda:~$ chmod +x elso
illes@panda:~$ cat elso
echo Ez az első shell scriptem!

illes@panda:~$ ./elso
Ez az első shell scriptem!
illes@panda:~$ elso
Ez az első shell scriptem!
illes@panda:~$
```

- Egy kis módosítás:

```
illes@panda:~$ cat elso
#!/bin/sh
#
echo Ez az első shell scriptem!
```

Műveletek shellben

- Egyetlen művelet létezik: szöveg összefűzés
  - Shellben minden változó szöveg!
- x=piros; echo Nyári $x alma! # ?
  - Aritmetikai műveletek közvetlenül nem használhatók.
  - x=alma; y=$x+fa; echo $y # alma+fa
  - a=5; b=$a+1; echo $b #5+1
- A szövegösszefűzésen kívül közvetlenül se egyéb szöveges, se aritmetikai műveleteket nem támogat!

Aritmetikai műveletek

- let utasítás, a bash része, régi sh-ban nincs
  - a=2; let b=a+1; echo $b # eredmény 3
- expr utasítás
  - expr $a op $b PL: expr 3 * 4 !!!
  - op: <,<=,>,>=,!=,=,+,-,*,/,% (mod)
  - Javasoljuk az expr használatát kompatibilitás miatt!
- bc parancs (szűrő)
  - C nyelv jellegű bemeneti szöveget vár
    - Létezik ciklus, szögfüggvények, fv definíció,stb
  - Példa: echo 2*16 | bc #32

BC példa

- Összetettebb példa:
  - Függvény definíció
  - Négyzetgyök, szögfüggvény használatra példa!
- Fájl név: bcpelda
- Futtatás: chmod +x bcpelda
  - ./bcpelda

```
#!/usr/bin/bc -lq
#a –l kapcsoló a math könyvtárat használja
# ez kell s szinusz fv-hez (s)
define fakt (x) {
if (x <= 1) return (1);
return (fakt(x-1) * x);
}
print "Az 5 faktorialis erteke: ";
print fakt(5);
print " !\n";
print "A 25 négyzetgyöke: ";
print sqrt(25);
print "\n";
print "Színusz PI/2 értéke:";
print s(3.1415/2);          #színusz
print "\n";
quit
```

Parancs paraméterek

- Példa: legjobb csapat a Fradi! #ez parancs ☺
  - Mi a parancsnév?
- $1, $2, $3, … # paraméter változók
  - echo $1 # csapat
- $0 # parancs neve (legjobb)
- $# # paraméterek száma
- $* # összes paraméter, idézőjel hatás nincs!
- „$@” # külön a paraméterek, példa: param

Paraméter példa

- Hívása: param barack fa ’alma fa’

```
echo paraméter használat
echo "adunk összetett paramétert is 'alma
fa’”
#minden külön
echo '$* használat'
for i in $*
do
echo $i   # barack, fa, alma, fa külön sorban
done
```

```
#minden egyben
echo '"$*" használat for ciklusban'
for i in "$*"
do
echo $i   # eredmény egy sorban!
# ''között”vannak a paraméterek
done
echo '"$@" használat'
# az alábbi sor a for i in $@ alak rövid változata
for i
do
echo $i
done
```

Param példa futása

```
illes@panda:~$ param fű fa 'alma fa'
paraméter használat
adunk összetett paramétert is ('alma fa')

$* használat
fű
fa
alma
fa
"$*" használat for ciklusban
fű fa alma fa
"$@" használat
fű
fa
alma fa
illes@panda:~$
```

Még több parancs paraméter

- Ismétlés:
  - $0 – parancs név
  - $1,…$9 paraméterek
- Csak 9 paraméter lehet?
- Nem, használhatjuk a {} változó megadó karaktereket!
- Igaz, elég ritka a nagy paraméterszám!
  - Nem ajánlott a túl sok (4 vagy több) paraméter használat!

Nagyszámú paraméter használat

- ${10}
  - Tizedik paraméter, zárójelek nélkül $10, a $1-hez fűzné a 0 karaktert!
  - ${100} # 100. paraméter…
- Shift utasítás
  - A paramétereket eggyel balra lépteti.
    - Ez $1-et kidobja, majd balra lépnek egyet a paraméterek,$2->$1,… $10->$9 ($10 –be $11 kerül, ha van)
  - Jellemző feldolgozás ciklus segítségével.

Egyéb hasznos változók

- $$ - A futó program PID értéke!
- $! - A háttérben utoljára végrehajtott program PID-je!
- $- - A shell kapcsolói . ?????

Program befejezése,eredménye

- Egy shell script program az utolsó utasítás elvégzésével befejezi működését, visszatér a hívó félhez, a shellbe!
- Van-e ennek eredménye?
  - Igen!
- Minden utasításnak van befejezési eredménye!
  - Ez az eredmény a program sikerességét, sikertelenségét mutatja!
- Egy program sikeresen lefut, ha végre tudja „rendesen” hajtani a feladatát!
  - Ez programonként más és más!

Parancsok eredménye

- Ilyen parancs eredmény minden esetben létrejön!
- A parancs eredményét a $? változó tartalmazza!
- echo szia! ; echo $? # 0 lesz az echo parancs eredménye!
- Ha az eredmény 0, akkor a parancs sikeresen lefutott!
- Ha pozitív az eredmény (legtöbbször 1), a parancs sikertelen

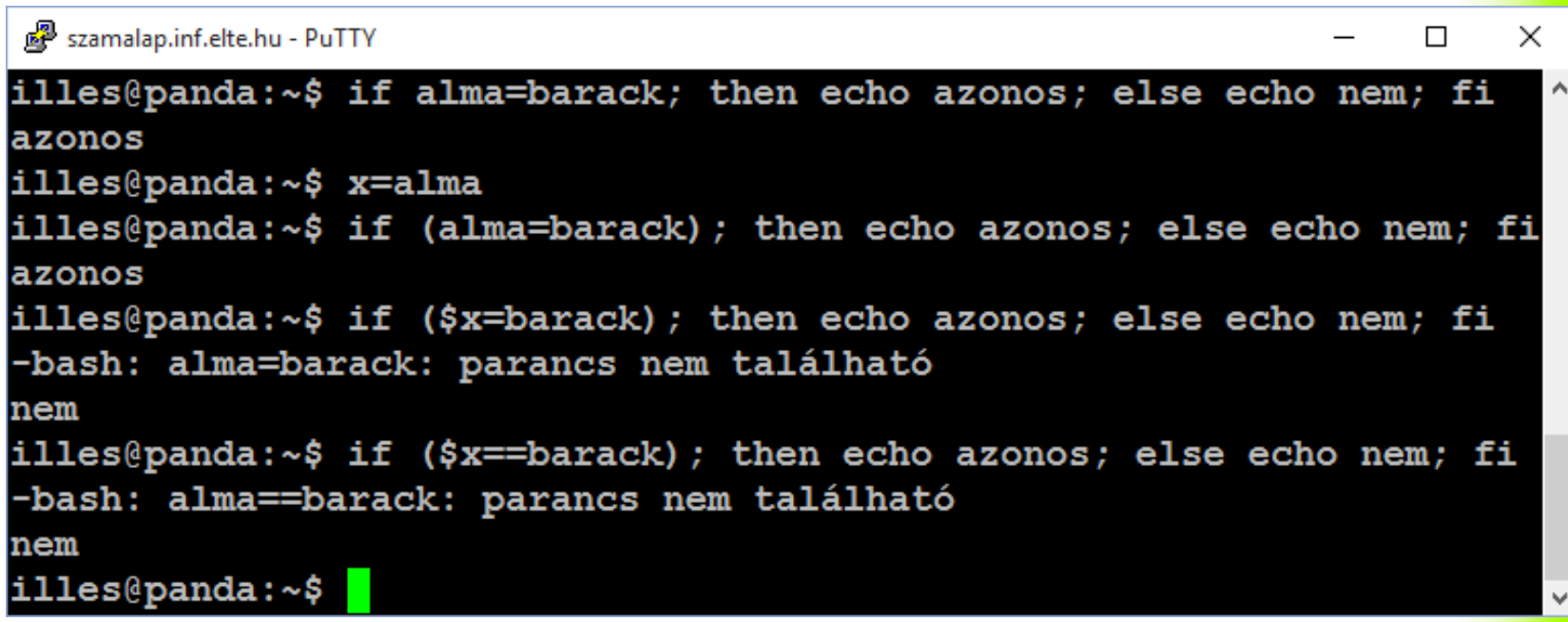
```
illes@panda:~$ cp beka peti
cp: stat "beka" sikertelen: Nincs ilyen fájl vagy könyvtár
illes@panda:~$ echo $?
1
illes@panda:~$
```

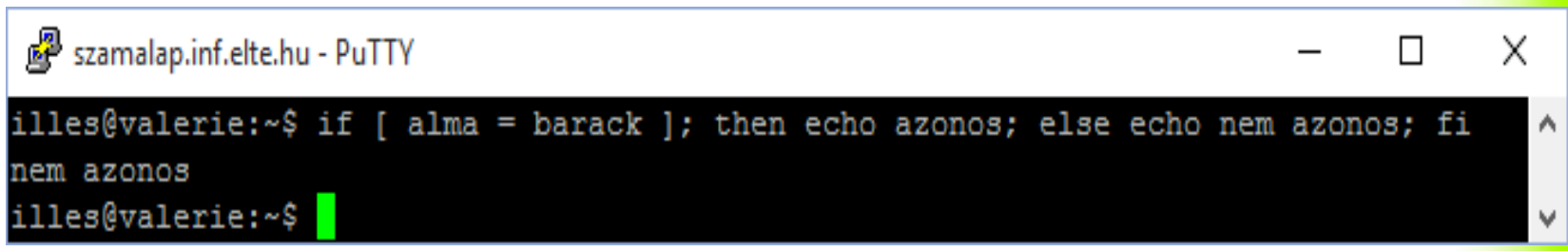
exit parancs

- Közvetlenül az exit parancs segítségével ki is léphetünk egy shell programból!
- exit érték # ahol érték egy 0-255 közti egész lehet
  - Ha az exit paramétere 0, az azt jelenti, hogy a befejezett program sikeres volt!
    - Ez a sikeresség **logikai igazként** is értelmezhető!)
  - Ha a paraméter pozitív, a futás sikertelen!
    - Ez a sikertelenség **logikai hamisként** is értelmezhető!
- Hasonlít a C nyelvhez, csak ott fordított az értelmezés!

Logikai kifejezések

- Lássuk az alábbiakat:
- Látható: Nincs önmagában logikai kifejezés!
- Értékadás van, ez mint logikai is értelmezhető! (igaz)
- Csak a [] logikai kifejezés jó!





Logikai eredményt adó utasítás

- Ahogy láttuk, logikai művelet nincs!
  - De minden utasítás eredménye logikai eredményként is értelmezhető!!!!
- Segít a **test** utasítás! Ezt az utasítást gyakran a [] karakterekkel helyettesítik!
- test operandus1 operátor operandus2 # fontos a helyköz!
  - Vagy: [operandus1 operátor operandus2] # itt is fontos a helyköz, [után és] előtt is!

Test – aritmetikai műveletek

- test, vagy […] logikai vizsgálat
- 0 – igaz, 1- hamis, echo $?
- true – igaz parancs, exit 0
- false – hamis parancs, exit 1
  - -lt,-gt.-le,-ge,-eq,-ne numerikus vizsgálat
    - [$x –lt 5]
  - -f file, -d dir file vagy könyvtár létezés
  - -o, vagy, -a az és operátor, ! a tagadás
  - -r,-w,-x fájl read,write,execute jog megléte

Test – szöveges, fájl vizsgálat

- Szöveg összehasonlítás:
  - =, != azonos, nem azonos sztring vizsgálat
    - test $a = $b # igaz, ha $a és $b azonos szöveg
    - test $a != $b # igaz, ha $a és $b nem azonos szöveg
  - test -z $X # igaz, ha $X szöveg hossza 0, üres sztring
  - test -n $X # igaz, ha $X string hossza nem 0, ha nem üres
    - [-n $X] # fontos a helyköz!!!!
- Teljes referenciához: man test

Test példa – I.

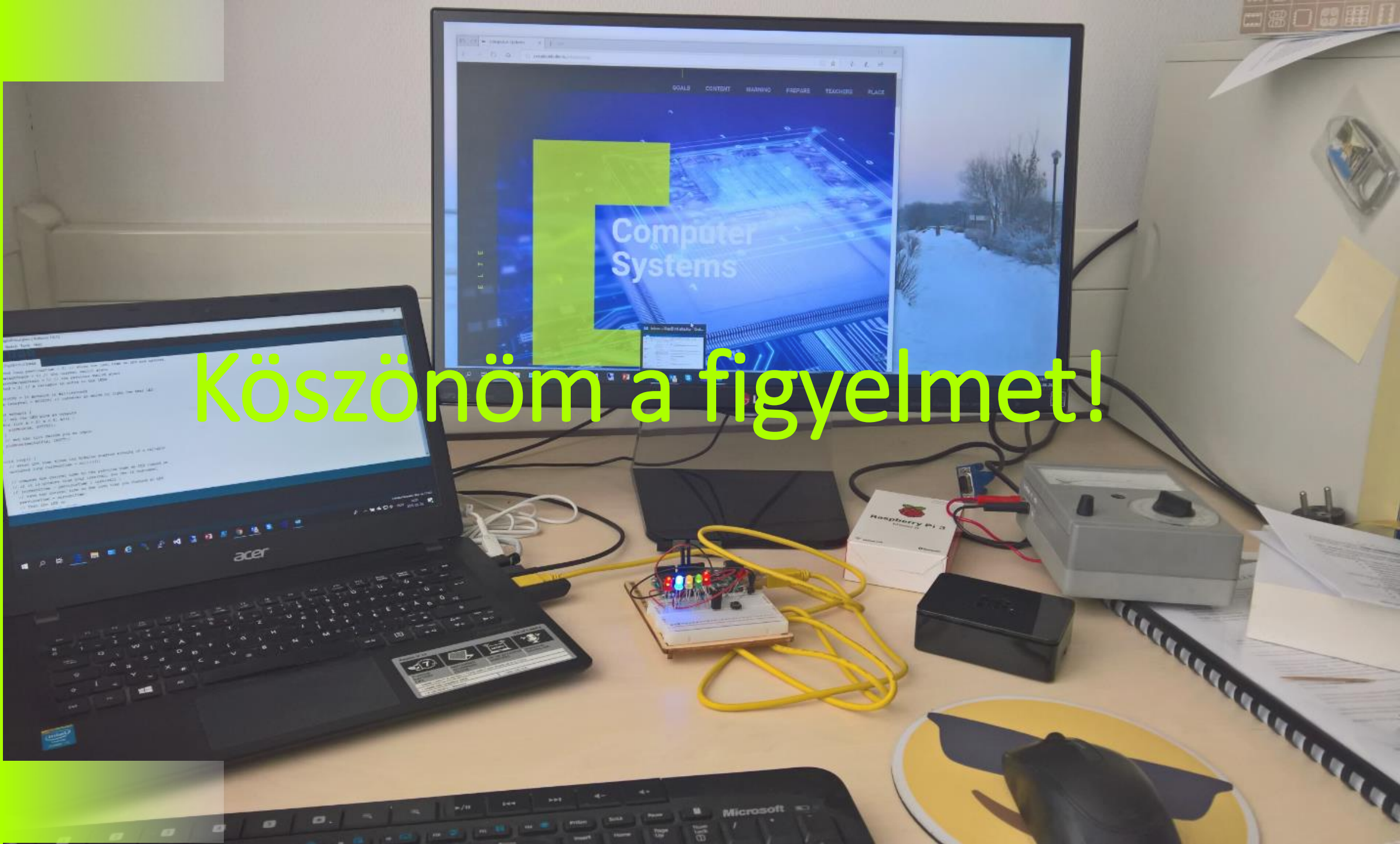
- A példa a leggyakrabban használt jellemzőket mutatja!
- Egyszerű logikai kifejezések.

```
$ x=4
$ [ $x –lt 6 ]   # test $x –lt 6
$ echo $?
0 (igaz)
$ test "alma" = "körte„ # [ „alma” = „körte” ]
$ # szöveges azonosság vizsgálat eredménye
$ echo $?
$ 1 (hamis)
$test –z „alma”  # üres string vizsgálat
$echo $?
1
```

Test példa – II.

- Összetett logikai vizsgálat
- Fájl, könyvtár vizsgálat

```
$ x=4
$ y=fradi
$ [ $x -eq 4 –a $y != „vasas” ]
$ # logikai ÉS kapcsolat -a
$ echo $?
0 (igaz)
$ [ ! $x –eq 4 ]
$ echo $?
1 (hamis)
$ test –f fradi # igaz, ha fradi fájl létezik
$ [ -d alma ]  # igaz, ha az alma könyvtár
                # létezik
```



ELTE INFORMATIKA