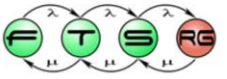
Intelligens rendszerfelügyelet (VIMIA370)

Szkriptelés és Python alapok

Horányi Gergő, Micskei Zoltán,
Szatmári Zoltán, Tóth Dániel

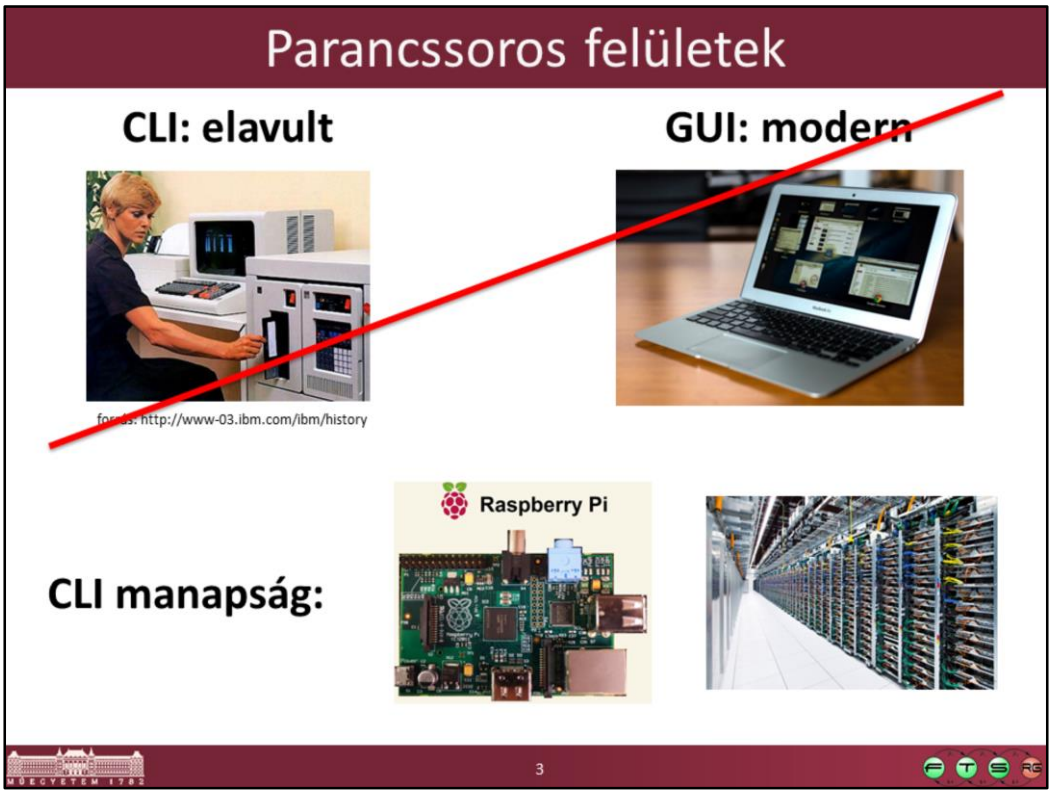


Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Utolsó módosítás: 2015. február 23.

Tartalom

- **Motiváció: szkriptelés**
- Linux alapok
- Python alapok
- Windows PowerShell (következő óra)

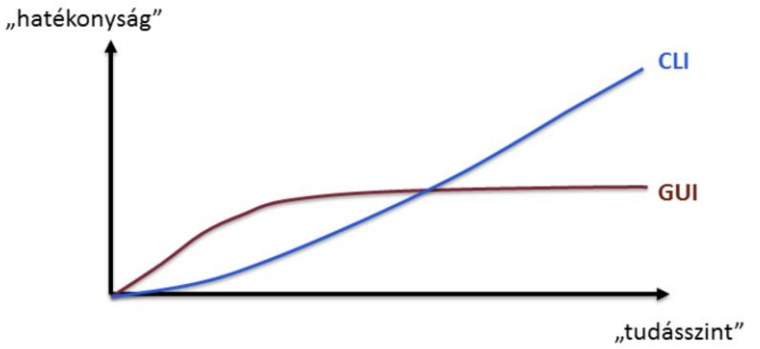


CLI: Command Line Interface
GUI: Graphical User Interface

A parancssoros felületek nem valami régi, múlt századi technológiák, manapság is megvan a nagyon fontos szerepük. Mint látni fogjuk, bizonyos feladatokra sokkal alkalmasabbak, mint a grafikus felületek.

Parancssoros és grafikus felületek

	GUI	CLI
Tanulhatóság	Könnyű	Nehezebb
Automatizálható	Nehezen	Könnyen
Hasznos	Kezdő / alkalmi felhasználó	Szakértő / gyakori használat



- Egy GUI sokkal könnyebben tanulható, könnyebb felfedezni benne a funkciókat. Ugyanakkor nehezen automatizálható, az ismétlődő feladatokat végig kell újra és újra kattintani benne. Nehéz olyan jó GUI-t készíteni, amiben az összetett, sok lépésből álló vagy sok elemet érintő műveleteket is hatékonyan lehet elvégezni.
- Egy CLI nehezebben tanulható, könnyebb elveszni benne az elején. De ha valaki elért egy tudásszintet, onnan már produktívan tud dolgozni, könnyű a visszatérő feladatokat automatizálni.

(Az ábra csak illusztrációs célokat szolgál, nem a pontos komplexitásokat jelzi.)

Egy jó összefoglaló a GUI és CLI előnyeiről és hátrányairól: Douglas Bell. „Discussion: GUI versus Command Line”, In: Software Engineering for Students, URL: http://wps.pearsoned.co.uk/ema_uk_he_bell_softeng_4/103/26561/6799810.cw/content/index.html

Szkriptelés motivációja: automatizálás

- Gyakran ismétlődő feladatok
  - Fájlok csoportos átnevezése
  - MP3 csoportos átkódolás
  - Több fejlesztési projekt együttes fordítása
  - Felhasználók csoportos felvétele
  - …

Power user

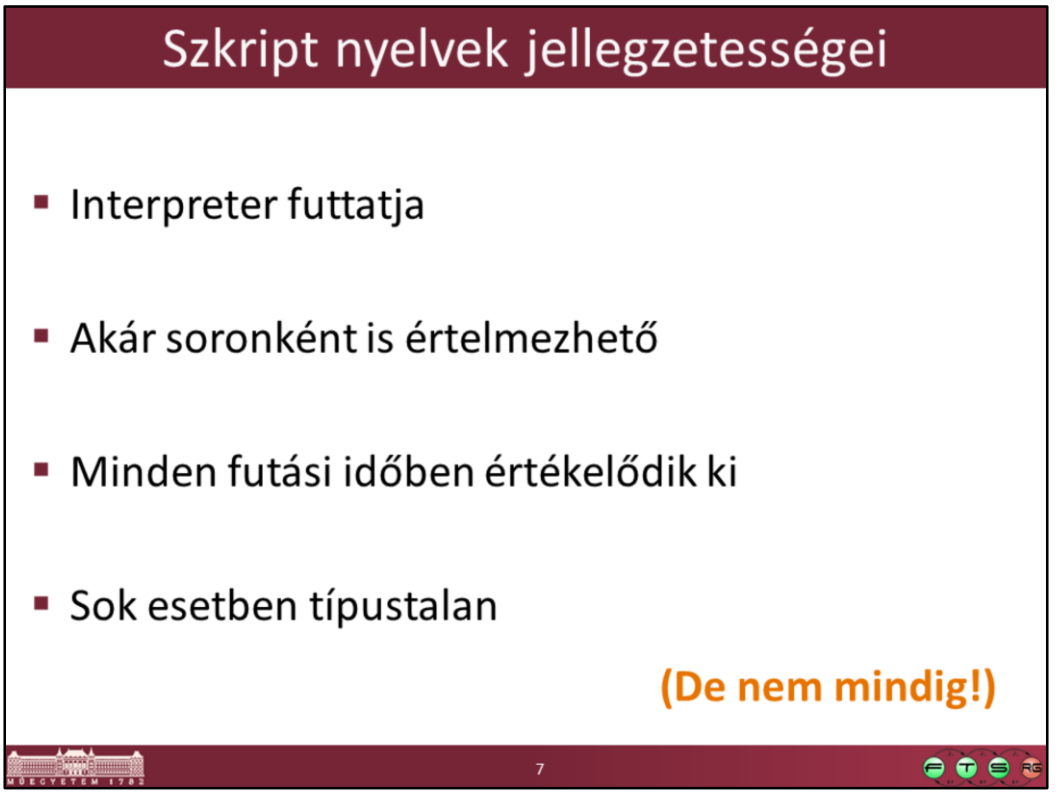
Development

Operations

- Java/C# programot is írhatunk rá
  - **DE**: biztos ez a leghatékonyabb eszköz?

Motiváció: szkript nyelvek

- Nem szükséges speciális fejlesztői környezet
- Legtöbb gépen elérhető a futtatókörnyezet
- Gyors és hatékony eszköz

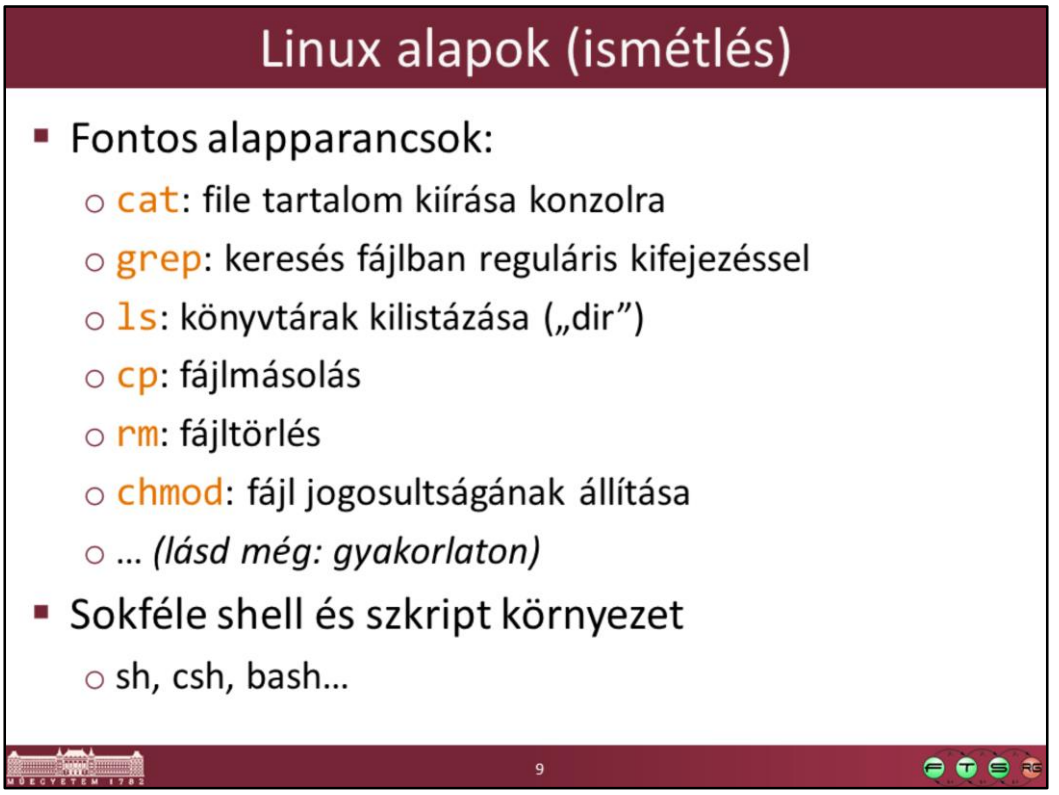


Ezek miatt a jellegzetességeik miatt nagyon alkalmasak a gyors, rövid feladatok megoldására (pár perces feladatok, „eldobható” kód).

Azonban a mai modern szkriptnyelvekben használhatóak komplex feladatokra is, bonyolult szoftverfejlesztési projektekben. A fenti jellegzetességek se érvényesek minden nyelvre mindig (pl. Pythonhoz is léteznek JIT fordítók, Python és PowerShell is támogat típusokat…).

Tartalom

- Motiváció: szkriptelés
- **Linux alapok**
- Python alapok
- Windows PowerShell (következő óra)



A tárgyban alapvető Linux felhasználói ismeretek szükség lesz ahhoz, hogy a házi feladatokat meg tudjuk oldani. A fenti parancsokat biztos kell majd használni.

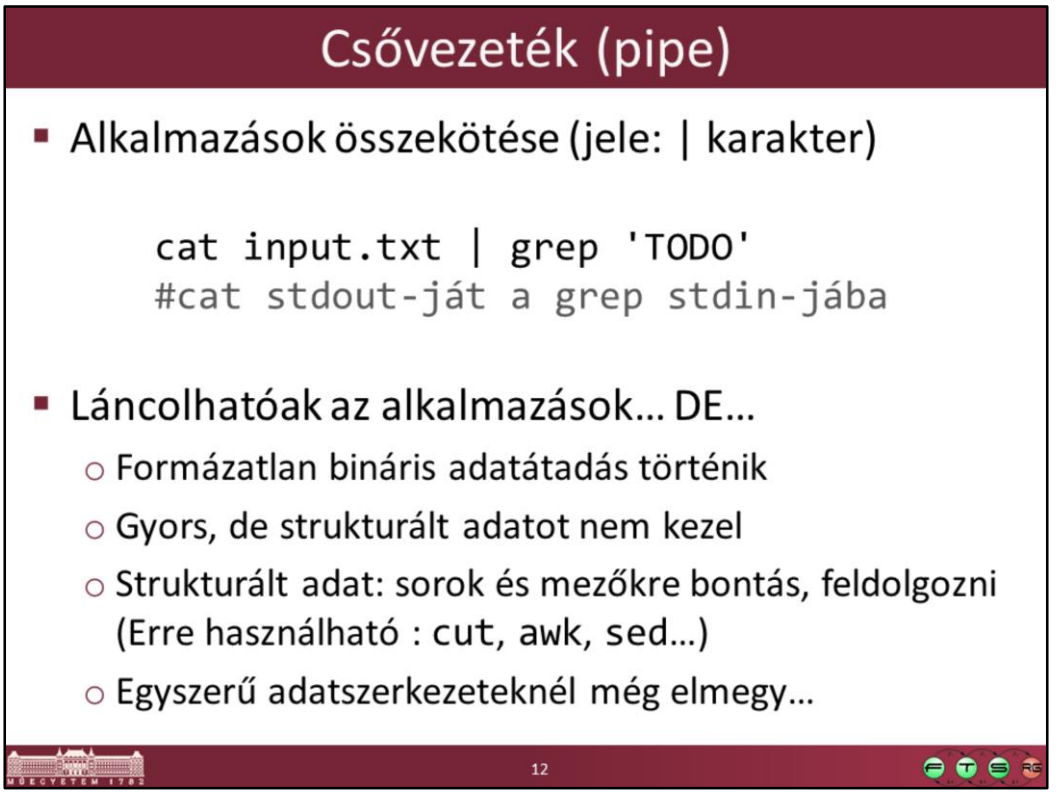
Az *Operációs rendszerek* kapcsolódó előadásait érdemes átismételni, és a *Szkriptelés* gyakorlat anyaga is tartalmaz egy gyorstalpalót. (Sajnos a Mérés labor 4. kapcsolódó mérése párhuzamos az IRF-fel.)

Bash shell (alapvető funkciók)

- Automatikus kiegészítés: **TAB billentyű**
- Parancs előzmények tárolása
  - **Fel** és **Le** gombokkal navigálás
  - **CTRL+R** kombinációval keresés
  - `history` parancs
- Terminál gyors bezárása: CTRL+D

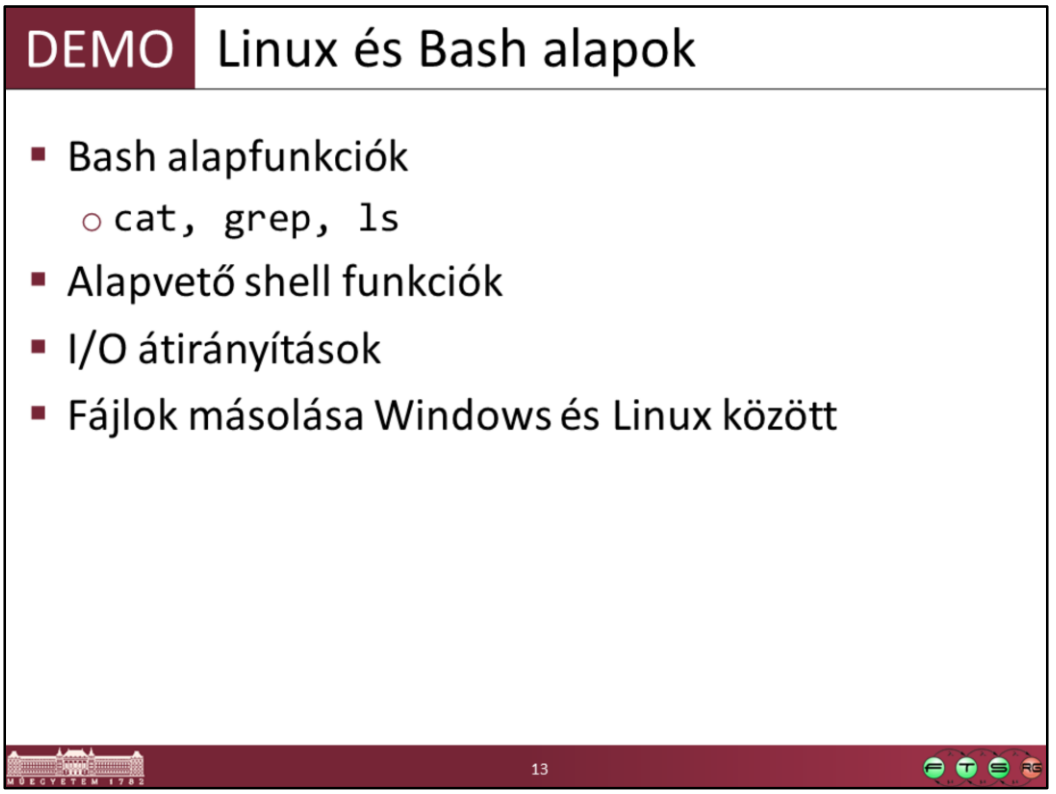
Átirányítások

- Standard I/O, minden programnak
  - 0 – stdin
  - 1 – stdout
  - 2 – stderr
- Átirányítás
  - `cat fájlnév` #fájl → stdout
  - `cat fájlnév 2>&1` #stderr → stdout
  - `cat fájlnév > másikfájl` #fájl → stdout → másikfájl
  - `cat fájlnév >> másikfájl` #fájl → stdout → másikfájl (append)
  - `cat fájlnév 2> másikfájl` #fájl → stdout, stderr → másikfájl
  - `cat fájlnév &> másikfájl` #minden a fájlba ömlesztve



Csővezetékből érkező adat soronkénti feldolgozása bash segítségével:

- Módszer1: `pipecmd | while read`
- Módszer2: `for VAR in $(pipecmd)`



pwd
ls
ls -l
mkdir test
cd test
kozben fel/le gomb, TAB hatasanak demonstralasa
echo "hello"
echo "hello" > hello.txt
nano hello.txt
cat hello.txt
cat hello.txt | grep he

Tartalom

- Motiváció: szkriptelés
- Linux alapok
- **Python alapok**
- Windows PowerShell (következő óra)

Miért éppen Python?

Számos elterjedt szkript nyelv létezik:

- Python
  - Hasonlít a már tanult nyelvekhez (C, Java, C#, …)
  - Nagyon elterjedt, aktívan fejlesztik
  - Jól dokumentált, rengeteg kiegészítéssel

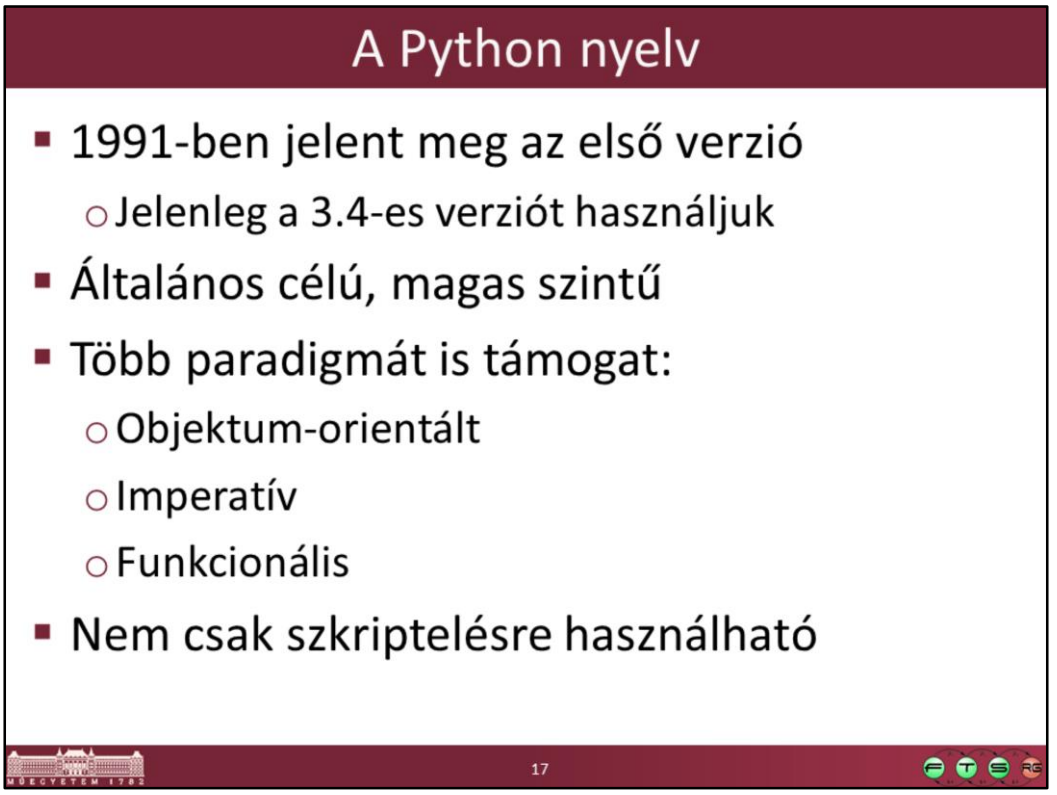
Megjelenések éve: Bash (1977/1989), Perl (1987), Python (1991), Ruby (1995)

Még egy lista, hogy ki használ Pythont: http://www.python.org/success-stories/

Érdemes megnézni, hogy kik szponzorálják és kik adnak elő egy Python konferencián: https://us.pycon.org/2015/

Rengeteg nagy cég használ valamilyen formában Pythont:

- Google: https://developers.google.com/edu/python/ http://code.google.com/p/google-api-python-client/
- Dropbox: https://tech.dropbox.com/2012/12/welcome-guido/
- Amazon: http://aws.amazon.com/python/
- Twitter: https://github.com/bear/python-twitter
- Facebook: https://developers.facebook.com/blog/post/301/
- Microsoft: http://pytools.codeplex.com/
- HP: http://hp.jobs/sunnyvale-ca/software-designer-javapython-engineer/33914192/job/
- Spotify: https://ep2013.europython.eu/conference/talks/spotify-and-python-love-first-sight



Figyeljünk arra, hogy a Python hivatalos honlapja hajlamos a 2.x verziójú dokumentációkat feldobni. Mindig válasszuk ki a 3.4-es verziót!

Python filozófia

„Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Readability counts.”

The Zen of Python (PEP20) részlet

The full proposal: http://www.python.org/dev/peps/pep-0020/

Hello world példa

- Indítsuk el a Python interpretert

  $ python3

- **FIGYELEM**: nem python, hanem python3
  - A python az a 2.x verzió!

```
pi@raspberrypi ~/test $ python3
Python 3.2.3 (default, Mar  1 2013, 11:53:50)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Írjunk ki valamit:

  >>> print("Hello world!")

Linux esetén a python még a 2.x verziót indítja el, a tárgyban a 3.x verziót használjuk (python3 parancs).
A legtöbb mai Linuxon még fent van a 2.x-es verzió is, mert sok régi program azt használja. Így figyeljünk oda, hogy ne keverjük össze a verziókat!

A Python 2.x-es verziójában még működött a print nem függvény változata: print "hello", ez 3.x-nél már csak függvényként működik: print("hello")

Hello world szkript

- Kedvenc editorba írjuk be (nano, mcedit, vi, emacs…)

```
#!/usr/bin/env python3
# this is a comment
print( "Hello world" )
```

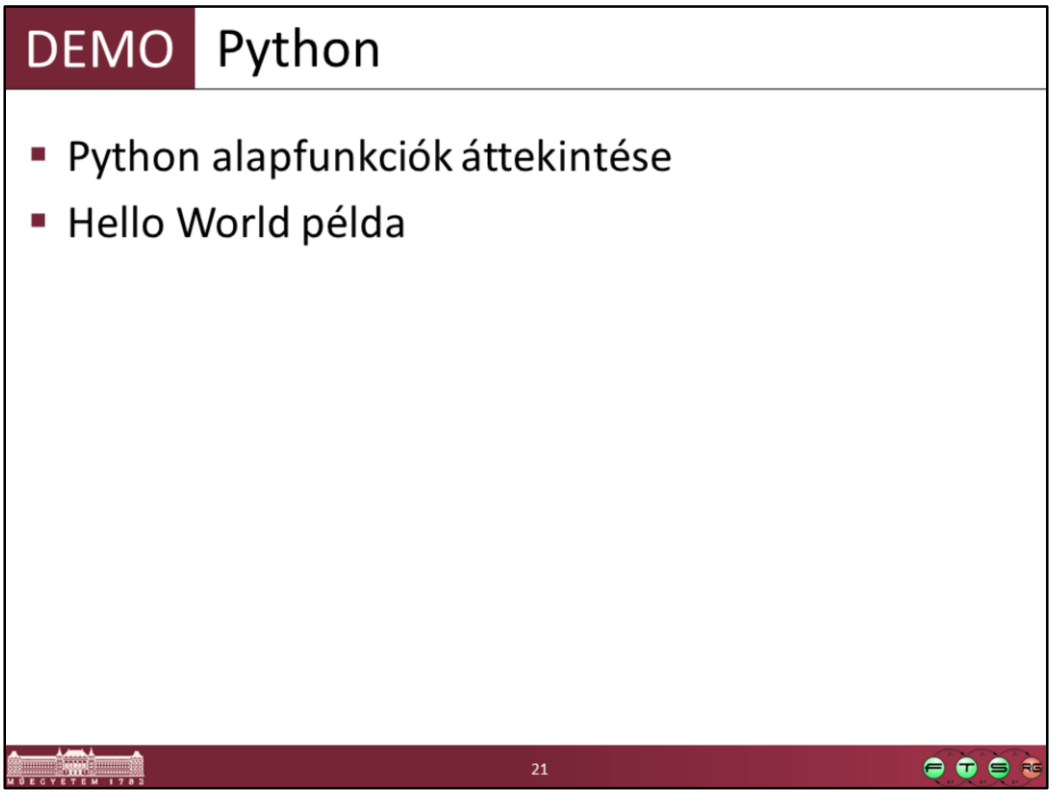
- Első sor: „shebang”
  - Egy hint, jelzi, hogy ez milyen fájl is valójában
- Adjunk neki futtatási jogot:

```
chmod +x hello.py
```

- Futtassuk:

```
./hello.py
```


(a ./ azért kell, mert az aktuális könyvtár nincs a path-ban)



- Nézzük meg az interpretert
    - print(), help, exit (először zárójelek nélkül)
- Írjuk meg a következő szkriptet:

```
#!/usr/bin/env python3
print( "Hello world" )
```

- Próbáljuk lefuttatni úgy, hogy nem adunk rá futási jogot
- chmod, majd futtassuk
- Ezután futtassuk úgy is, hogy:
    - python3 hello.py

Változókezelés

- A szokott típusok elérhetőek
  - Számok
  - Sztringek
  - Listák, …
- Szkriptnyelv → automatikus típusválasztás
  - DE: van típusellenőrzés
- Változókonvertáló függvények léteznek
  - pl.: `int("6")`
    `str(15)`

Numeric types:

- *integers:* Integers have unlimited precision
- *floating point numbers:* Floating point numbers are usually implemented using double in C
- *complex numbers:* Complex numbers have a real and imaginary part, which are each a floating point number

Változókezelés

```
irf = "Intelligens rendszerfelügyelet" #Értékadások
year = 2015
course = irf + " " + str(year)

x = y = z = 0                    # többszörös értékadás
a, b = 2, 3

> print (Course) # Nem definiált változó, hibaüzenet!
                 # (kis-, nagybetű számít!)
> print(course);
  Intelligens rendszerfelügyelet 2015
```

Sztringek kezelése

```
q1 = "Bring us a shrubbery!"
q2 = 'Brave Sir Robin ran away' # meg lehet így is adni
q3 = '''What is the air-speed velocity
 of an unladen swallow?'''      # triple ': lehet többsoros

print( q1[2] )                  # eredmény: i
```

Sztringek részeinek visszaadása (slicing):

```
   s[start:stop]                # azon s[k], ahol start <= k < stop

print( q1[6:8] )                # eredmény: us
print( q1[11:-1] )              # eredmény: shrubbery
print( q1[:4] )                 # eredmény: Brin
```


Dokumentációból:

- „Strings are immutable sequences of Unicode code points”
- „If *i* or *j* is negative, the index is relative to the end of the string: len(s) + i or len(s) + j is substituted. But note that -0 is still 0.”
- „The slice of *s* from *i* to *j* is defined as the sequence of items with index *k* such that i <= k < j. If *i* or *j* is greater than len(s), use len(s). If *i* is omitted or None, use 0. If *j* is omitted or None, use len(s). If *i* is greater than or equal to *j*, the slice is empty.”

További hasznos metódusok sztringeken:

- capitalize, find, format, isnumeric, join…

Listák

A lista is egy sorozat (sequence):

```
fruits = ["apple", "pear"]

fruits.append("peach")
len(fruits)          # 3

fruits[1]            # "plum"

"pear" in fruits     # True
```


„Lists are mutable sequences, typically used to store collections of homogeneous items (where the precise degree of similarity will vary by application).”

De nem feltétlen kötelező ugyanolyan típusú elemeket berakni a listába, pl. a következő is lefut:

list2 = [1, 'a']

Hasznos műveletek listákhoz:
- min(list), max(list), x in list

Hasznos metódus listákhoz:
- list.sort()

DEMO Változókezelés

- Változók, értékadások
- Szövegek kezelése
- Listák kezelése

Vezérlési szerkezetek: elágazás

Pythonban zárójelezés helyett blokkok behúzása van!

```
number = -1
if number < 0:
    print("Negative number")
elif number < 3:
    print("Small number")
else:
    print("Big number")
```

- Szóköz **VAGY** TAB karakterekkel, de csak az egyikkel
- Akár parancssori értelmezőben is használhatjuk
- Ne felejtsük le a kettőspontot a végéről
- Logikai műveletek: and / or / not

A behúzást (indentation) lehet TAB karakterrel vagy szóközzel jelölni.

A Python Style Guide javaslata szerint azonban érdemesebb átállítani a szövegszerkesztőkent, hogy TAB helyett szóközt rakjon, így biztosítva, hogy máshol is ugyanúgy jelenjen meg a kód. A hivatalos ajánlás 4 darab szóköz használata.

Vezérlési szerkezetek az interpreterben

… jelzi, hogy összetett utasításban vagyunk

Ide külön be kell írni a szóközöket nekünk

Végén egy üres sor jelzi, hogy lezártuk ezt a szerkezetet

```
>>> number = 3
>>> if number < 3:
...     print("Small number")
... elif number < 0:
...     print("Negative number")
... else:
...     print("Big number")
...
Big number
>>>
>>>
```

Ciklusok

For ciklus **sorozaton** iterál végig (~C# forach)

```
for x in [1, 2, "alma"]:
    print(x)

for i in range(0, 5):
    print(i)
```

while ciklus:

```
# Fibonacci
a, b = 0, 1
while b < 100:
    print(b, end=',')
    a, b = b, a+b
```


Range: „The *range* type represents an immutable sequence of numbers and is commonly used for looping a specific number of times in *for* loops.”

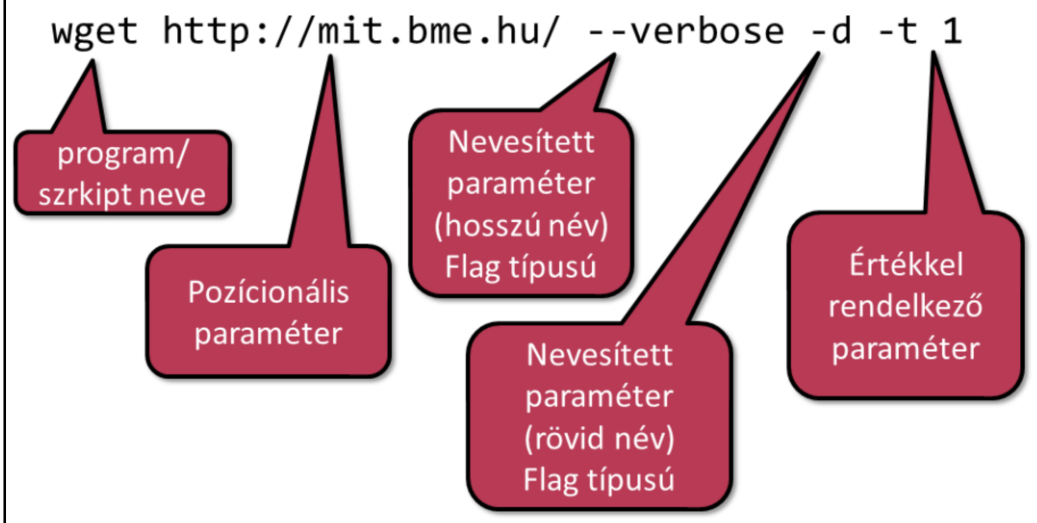
Modulok

- Előre elkészített segédmodulokat használhatunk
  - CSV kezelés (`csv`)
  - Külső parancsok hívása (`subprocess`)
  - Operációs rendszer adatai (`os`)
- Használatuk:
  - `import modulename`

Bővebben lásd: Python Tutorial. Chapter 6. Modules, URL: http://docs.python.org/3.3/tutorial/modules.html

Parancssori paraméterek

Hogyan használunk egy parancssori programot?



Argparse modul

- Paraméterek kezelése Pythonban
  - `sys.argv` listában megkapjuk
  - lehetne kézzel kezelni, de
- **`argparse:`** paraméterkezelő modul
  - nevesített paraméterek (rövid és hosszú névvel)
  - flag-ek
  - pozícionális paraméterek
  - opcionális paraméterek
  - tömbparaméterek

Documentation: http://docs.python.org/dev/library/argparse.html
Tutorial: http://docs.python.org/dev/howto/argparse.html#id1

Argparse példa

```
parser = argparse.ArgumentParser();
parser.add_argument("name",
    help="The name to be greeted.",
    type=str)
parser.add_argument("-q", "--quantity",
    help="Amount of greetings.",
    type=int, default=1)
args = parser.parse_args();

args.name
```

Így férünk hozzá a paraméter értékéhez

- A szükséges ellenőrzéseket elvégzi helyettünk
- Még [-h]elpet is generál

Visszatérési érték

- Minden parancsnak van visszatérési értéke
  - Következtethetünk belőle a lefutás eredményére
  - Ha minden rendben, akkor 0
  - Hibás esetekben különböző hibakódok visszaadása
- Pythonban: `sys.exit(return_value)`
- Főleg paraméterek ellenőrzésénél fontos

DEMO Parancssori paraméterek

- ParameterHandlingArgParse.py
  - Paraméterek definiálása
  - Nevesített paraméterek használata
  - Paraméterhibák kezelése
- Visszatérési érték

ParameterHandlingArgParse.py
ParameterHandlingArgParse.py IRF
ParameterHandlingArgParse.py IRF -g 5
ParameterHandlingArgParse.py IRF -g 5 -file tmp
cat tmp
ParameterHandlingArgParse.py IRF -g 5 -file tmp
ParameterHandlingArgParse.py IRF -g 5 -file tmp -X

Sztring darabolás

- String objektum *partition* vagy *split* metódusával

```
passwd="root:*:0:0:/bin/sh"
first, sep, remainders = passwd.partition(":")
all = passwd.split(":")
print(first)
print(remainders)
print(all)
```

```
> root
> *:0:0:/bin/sh
> ['root', '*', '0', '0', '/bin/sh']
```

Külső parancsok hívása

- `subprocess.call()`
  - Parancsok hívása (`stdin` és `stdout` használata nélkül)
- `subprocess.check_output()`
  - Parancsok hívása az `stdin` és `stdout` felhasználásával
  - Ha szükséges a parancs kimenetének feldolgozása

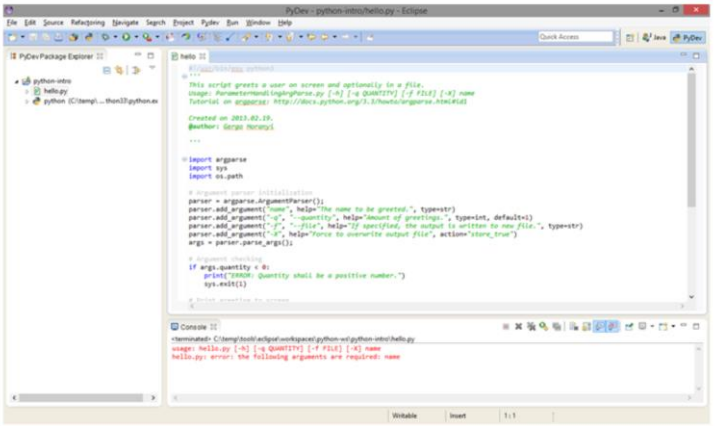
Kommentek

- Hagyományos és sorvégi kommentek
  - # karakter használatával
- Fejkommentek (docstring)
  - Függvény, osztály, modul elején
  - 3-3 idézőjel (") használatával

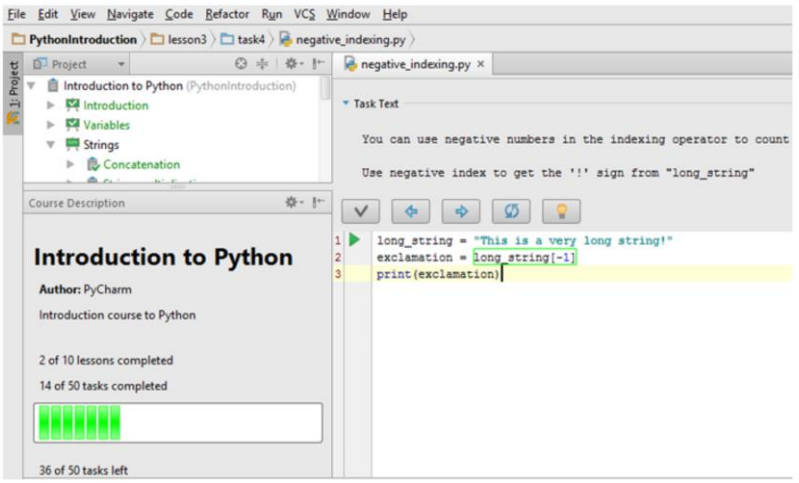
```
def sum(a, b):
    """Return the sum of a and b"""
```

Miben fejlesszünk?

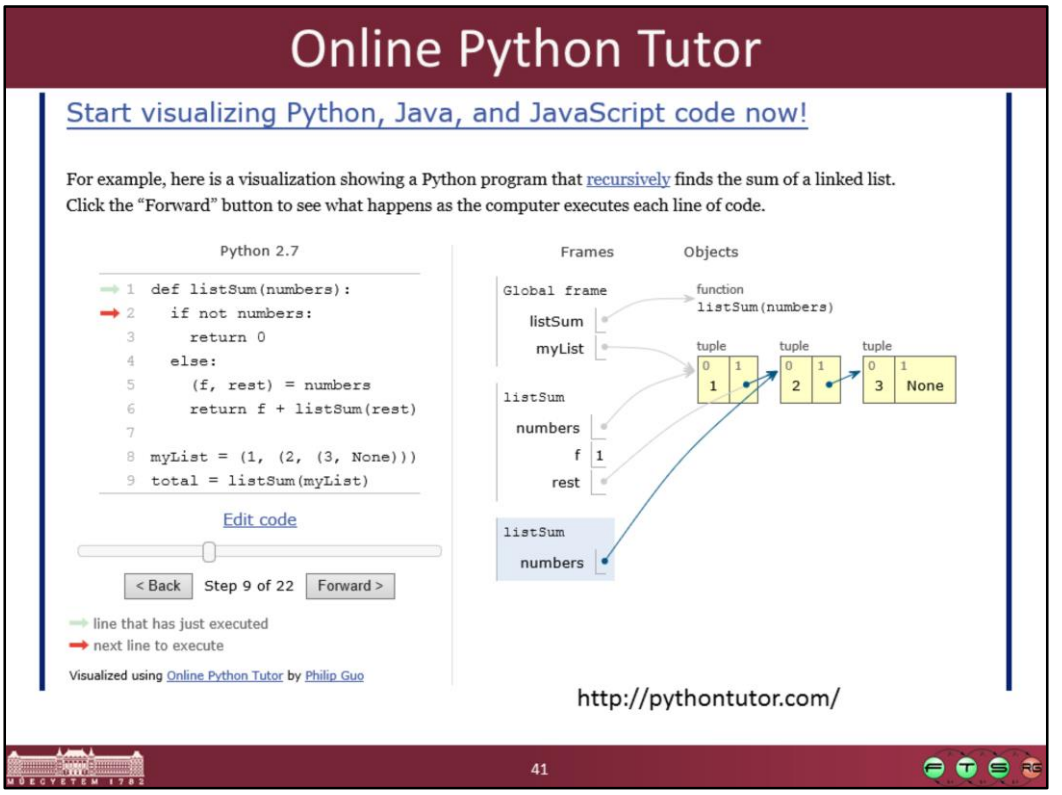
- Parancssori fejlesztőeszköz (`mcedit`, `nano`, …)
  - bármilyen szövegszerkesztő
- Integrált fejlesztőkörnyezet (IDE): PyDev



PyCharm Educational Edition



- Teljes értékű IDE
- Python oktatóanyag (task-based)



A Python Tutor segítségével a kód végrehajtását megjeleníthetjük lépésről-lépésre, és így segít megérteni az alapvető elemek működését.

Python Style Guide (PEP8)

Hogyan írjunk szép és olvasható kódot?

- Use 4-space indentation, and no tabs.
- Wrap lines so that they don't exceed 79 characters.
- Use blank lines to separate functions and classes, and larger blocks of code inside functions.
- When possible, put comments on a line of their own.
- Use spaces around operators and after commas, but not directly inside bracketing constructs: a = f(1, 2) + g(3, 4).
- Name your classes and functions consistently;
- Don't use non-ASCII characters in identifiers
- ...

Fontos, hogy érdemes már az elején megszokni, hogy szép és olvasható kódot írjunk. Ehhez ad sok hasznos tanácsot a PEP8.

A teljes ajánlás elérhető itt: http://www.python.org/dev/peps/pep-0008/

Ami kimaradt

- függvények (`def`)
- osztályok, saját modulok
- további adatstruktúrák (`dictionary`, `set`…)
- fájlok olvasása és írása (`open`,)
- hibakezelés (`try/except`)
- további beépített modulok:
  - `json, math, random, urllib, datetime, xml…`
- …

Feladat

Készítsünk egy olyan szkriptet, ami

- paraméterként kap egy könyvtárnevet
- kiírja, hogy hány alkönyvtár van benne
- kiírja, hogy melyik kiterjesztésből van a legtöbb a könyvtárban lévő fájloknál

További információ

- A Unix operációs rendszer:
  http://www.hit.bme.hu/~szandi/unix/index.html
- man bash, man sed, man cut, man sort, man grep…

- Official Python tutorial:
  http://docs.python.org/3.4/tutorial/
- Google Python class:
  https://developers.google.com/edu/python/

http://www.hit.bme.hu/~szandi/unix/index.html
http://docs.python.org/3.3/tutorial/
https://developers.google.com/edu/python/