Automation part 1

Web link

Main takeaways

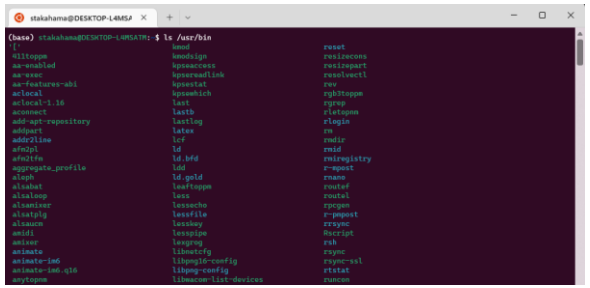
- You can use the shell to perform common operations (copy / move files) on your computer via text input
- The shell can serve as “glue” among the different programs
- You can use Python in place of the shell (or MATLAB or C)

Command line interface (CLI)

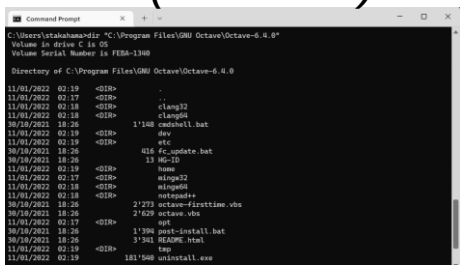
Shells

- CLIs provide an interface to one of several shells
- shells provide instructions to the operating system

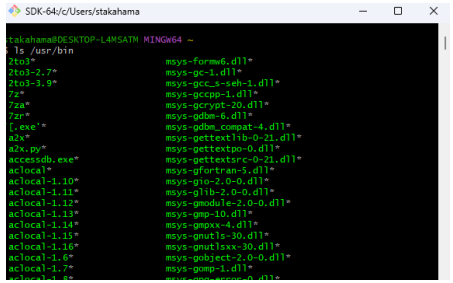
bash shell (Linux)



CMD shell (Windows)



bash shell (Windows)



don’t mind the coloring – these are just user configurations

user

CLI

terminal

executables *operators*

shell

operating system

machine

Spectrum of automation

runs on one computer ←→ runs on many computers

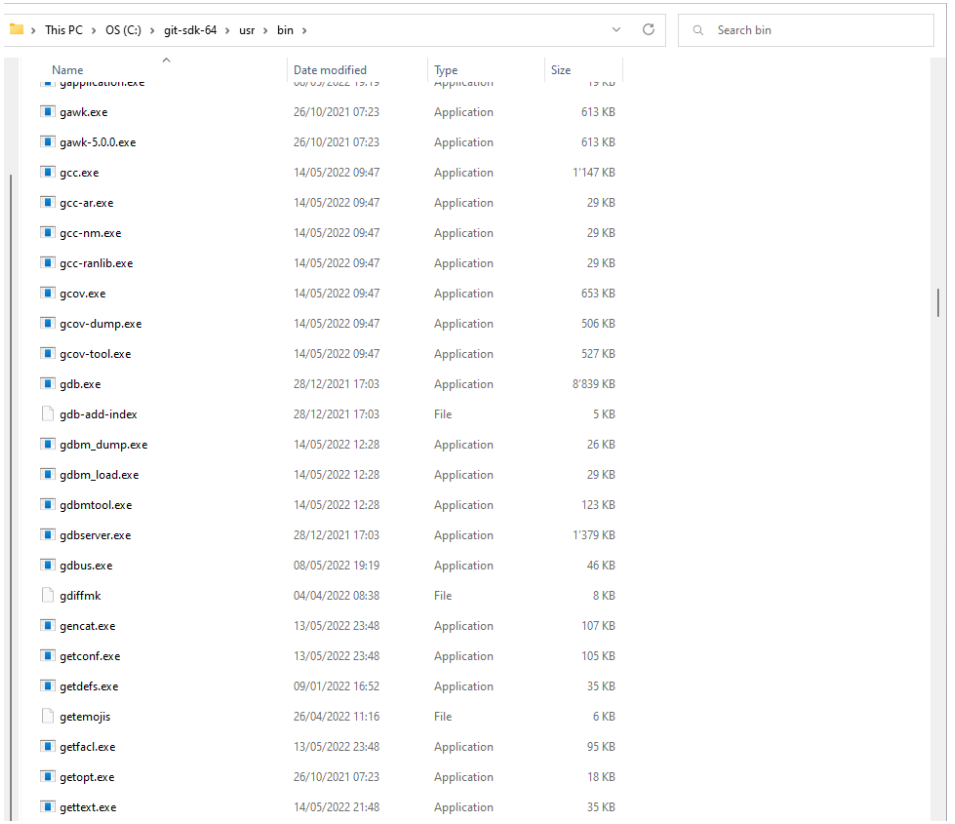
- POSIX-compliant shell commands keep things standardized across platforms (e.g., not CMD or Powershell in Windows)

number of steps to get results (typically “build” and “run” are two separate steps)

- shell scripts are key to automating this process

Executables

- To run a program, type its name at the command line
- *How does my computer know where the program is stored?*
  - type echo $PATH to get a list of directories in which your shell will look for executables
  - for an executable in the *current directory* (e.g., generated by gcc), prefix the executable name with "./"



Executables

Name	Description
cd	change directory
pwd	present working directory
mv	move or rename file/directory
cp	copy file/directory
cat	print file to terminal
head	show first 6 lines in terminal
tail	show last 6 lines in terminal
./{executable}	run {executable} found in this directory

https://stakahama.gitlab.io/sie-eng270/C_intro.html#org491705c

Shell operations

- Redirection <, >, >>
- Pipe |
- See [sieprog.ch]

Redirection

Lorsqu'on lance une commande (un programme) depuis le terminal, sa sortie (les printf's) est affiché sur l'écran.

```
$ ./lacDeThoune
1       557.325997      19.000000       19.044000
2       557.325993      19.000000       19.043542
3       557.325990      19.000000       19.043088
...
46      557.325878      19.000000       19.027469
47      557.325876      19.000000       19.027183
48      557.325874      19.000000       19.026900
```

Rediriger la sortie dans un fichier

Avec l'opérateur `>`, on peut rediriger cette sortie dans un fichier:

```
$ ./lacDeThoune > resultat
```

Plus rien n'est affiché sur l'écran. Mais si la simulation prend un moment, on peut suivre le fichier depuis un autre terminal:

```
$ tail -f resultat
```

Passer la sortie à un autre programme

Dans le même style, on peut passer la sortie à un autre programme:

```
$ ./lacDeThoune | ./analyse.py
```

La sortie (stdout) de *lacDeThoune* est connecté à l'entrée d'*analyse.py*.

De cette manière, on peut mettre en place des chaînes de traitement:

```
$ ./lacDeThoune | grep 'inondation' | ./analyse.py > resultat-final
```

Passer un fichier à un programme

Dans le sens inverse, on peut passer un fichier à l'entrée d'un programme:

```
$ ./analyse.py < resultat
```

Replace the shell with Python

- Many shell commands can be replaced with Python functions (e.g., from the Path module)
- Shell commands can be passed from Python to the shell – useful for constructing shell commands from character strings
- Python can also pipe data among programs

Function	Description
Path.mkdir	create directory
Path.rmdir	remove directory
Path.unlink	remove file
Path.rename	rename files
Path.cwd	get current working directory
Path.exists	check if file exists
Path.parent	"dirname"
Path.name	"basename"
PurePath.joinpath	join paths
Path.stem	sans file extension
Path.suffix	file extension

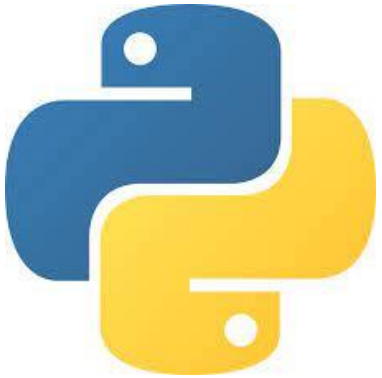
https://stakahama.gitlab.io/sie-eng270/python_shell_string.html#org95d4643

https://stakahama.gitlab.io/sie-eng270/C_intro.html#orge70ce4f

Shell can be a glue among different programs

Python strengths

- File I/O
- String processing
- Graphics, plotting

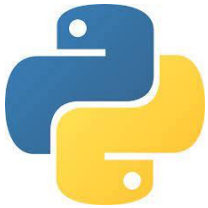


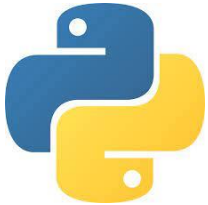
C strengths

- Speed (iterative calculations)
- Explicit memory management

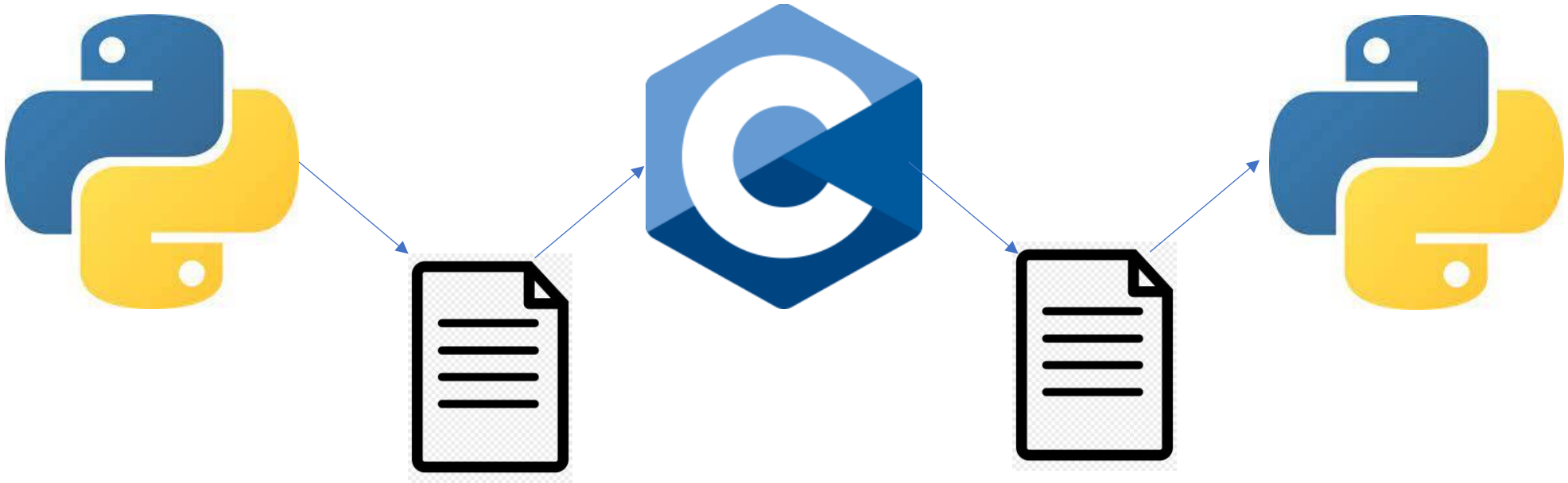
Example

1. Generate random numbers
2. Square the values
3. Plot the results





Method 1 – pass data through files



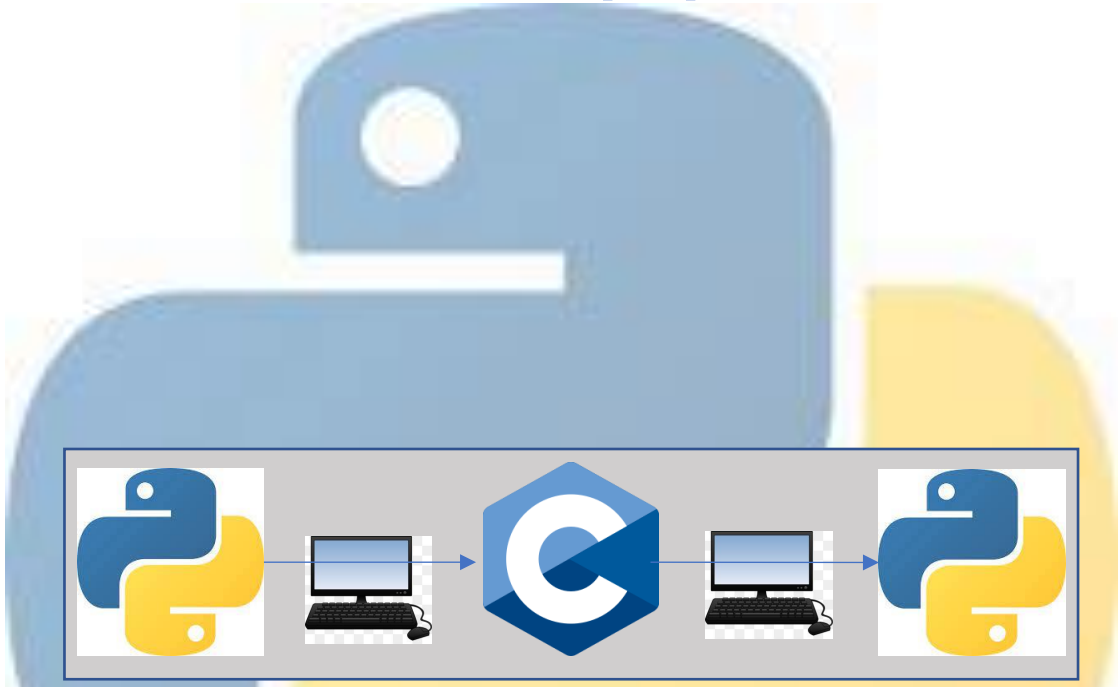
Method 2 – pipes / redirection

```
./gennum.py | ./c_square | ./analyze.py
```

```
./gennum.py | ./c_square > file2.txt
```

```
./analyze.py < file2.txt
```

Method 3 – pipes, but through Python



```
from subprocess import Popen, PIPE, STDOUT

my_list = range(1000)
my_list_str = '\n'.join(map(str, my_list))

p = Popen(['./c_square'], stdin = PIPE, stdout = PIPE, stderr = STDOUT)
output, errmssg = p.communicate(input = my_list_str.encode())

squared_list = list(map(float, output.decode().rstrip('\n').split('\n')))

print(all(map(lambda x, x2: x**2 == x2, my_list, squared_list)))

import matplotlib.pyplot as plt
plt.ion()
plt.plot(squared_list)
plt.xlabel('Point number')
plt.ylabel('Value')
```

Method 4 – call C as a Python module

```
.
├── __init__.py
├── main.py
├── mylib.c
├── mylib_linux.so
└── mylib.py
```

```python
from mylib import py_c_square

my_list = range(1000)
squared_list = py_c_square(my_list)

print(all(map(lambda x, x2: x**2 == x2, my_list, squared_list)))

import matplotlib.pyplot as plt
plt.ion()
plt.plot(squared_list)
plt.xlabel('Point number')
plt.ylabel('Value')
```

```python
from ctypes import c_double, c_int, CDLL
import sys

lib_path = f'./mylib_{sys.platform}.so'
basic_function_lib = CDLL(lib_path)
c_square = clib.c_square
c_square.restype = None # return type is 'void'

def py_c_square(list_in):
    """Call C function to calculate squares. Returns list."""
    n = len(list_in)
    c_arr_in = (c_double * n)(*list_in)
    c_arr_out = (c_double * n)()
    c_square(c_int(n), c_arr_in, c_arr_out)
    return c_arr_out
```

```c
#include <stdlib.h>

void c_square(int n, double *array_in, double *array_out) {
  int i;
  for (i = 0; i < n; i++) {
    array_out[i] = array_in[i] * array_in[i];
  }
}
```