LECTURE 41

THE SUBPROCESS MODULE

MCS 260 Fall 2021

David Dumas

REMINDERS

- Project 4 is due Friday at 6pm central.
- Homework 14 due tomorrow 10am. It's the last one.
- Worksheet 15 is available.
- Regular lab schedule this week.

SUBPROCESS

Python's `subprocess` module contains functions for starting and managing processes, i.e. asking the OS to run other programs.

`subprocess.run(args)` runs command `args`.

```
import subprocess
subprocess.run("explorer.exe")   # windows
subprocess.run("ls") # linux / macOS
```

`args` can be a string or a list of strings (the command line arguments).

WHAT RUN() DOES

Starts an external process from the given command.

(By default, the new process can accept input and write output to the terminal.)

Wait for process to end.

Return object with info about the process.

Common options for `run()` and default values

- `check=False` — If True, raise an exception if process reports an error when it exits.
- `shell=False` — If True, first start a new shell and then ask the shell to run the program.
- `cwd=None` — If not None, sets the working directory of the new process.
- `timeout=None` — If a positive number, only allow the process to run for `timeout` seconds. If it exceeds this, terminate it and raise an exception.

EXIT STATUS

When a process ends, it can provide an integer code to whatever process started it.

This **exit status** (also known as **exit code** or **return code**) is sometimes used to report errors.

- 0 indicates success
- Non-zero values indicate error; in some cases the specific number gives info about the error

The object returned by `run()` has a `.returncode` attribute.

SIDE NOTE: EXIT()

Python's `exit()` function accepts an integer argument, which sets the exit status.

If not given, the default is 0.

- `exit(0)` or `exit()` — exit normally
- `exit(1)` — exit indicating an error

IO REDIRECTION

The `capture_output` argument of `run()` is a boolean indicating whether the output of the process should be captured and returned.

If output is captured, it is placed in the `.stdout` and `.stderr` attributes of the return object.

The `input` argument of `run()` specifies input (as bytes) to be sent to the process, instead of allowing it to read from the terminal. This can be used to simulate keyboard input.

WHEN TO USE SUBPROCESS

Your program needs to do something, and an external program exists that can handle it.

This is more common on Linux and MacOS where there are lots of command line utilities.

Common applications:

- Opening a web browser or file editor
- Calling converters or compressors
- Running software development utilities like `git`

WHEN TO AVOID SUBPROCESS

If you want to simulate a full keyboard interaction with a program (input, wait, review output, decide, more input, ...), use a module like `pexpect` instead.

If the target program is written in Python, the functionality you need may be available as an associated module. If so, it is usually better to use the module directly.

SECURITY

Never pass untrusted input data as part of a call to `subprocess.run()`. Doing so will make your program a stepping stone to breaking computer security.

Using `shell=True` should also be avoided if possible. A single shell command can start multiple processes, and attackers can exploit this.

POPEN

We covered the *function-oriented* interface using `subprocess.run()`. There is also an object-oriented interface.

`subprocess.Popen(...)` builds and returns an object representing an external process.

This constructor returns immediately, allowing your program to proceed concurrently with the external process.

POPEN OBJECTS

Methods of the `subprocess.Popen` object:

- `poll()` — returns return code if process is done, or returns None if it is still running
- `wait()` — waits until the process exits, similar to `threading.Thread.join()`
- `terminate()` — end the process

REFERENCES

- Python module documentation:
  - subprocess
- Subprocess module tutorial by David Muller at DigitalOcean

REVISION HISTORY

- 2021-11-29 Initial publication