Module 07 – subprocess – Work with additional processes

The subprocess module

- The subprocess module provides a consistent interface to creating and working with additional processes.
- The subprocess module has a wide functionality for running additional processes, controlling their lO, including shell feathers like wildcards, pipes, redirections
- The subprocess module is an updated module for old and | [deprecated functionality with the same purpose like: - os.system(), os.spawn*(), os.popen*() ...

The call function

- subprocess.call(args, stdin=None, stdout=None, stderr=None, shell=False, cwd=None, timeout=None)

- Run the command described by args. Wait for command to complete, then return the returncode

The call function example

Demo

The call function example

```python
import subprocess

subprocess.call("dir *.py", shell=True)  # Remove the list and directly pass the command as a string
subprocess.call([r"C:\Python27\test.py"])  # Remove shell=True
subprocess.call("calc.exe", shell=True)  # Remove the list and directly pass the command as a string

with open(r"C:/Test/out.txt", "w") as f:  # Use 'with' statement to automatically close the file
    subprocess.call('dir "/p"', shell=True, stdout=f)  # Remove the extra quotation marks around /p

print("end")
```

The check_output function

- subprocess.check_output(args, stdin=None, stderr=None, shell =False, cwd=None, timeout=None)

- Run command with arguments and return its output.

- If the return code was non-zero it raises a CalledProcessError.

The check_output function example

```
import subprocess

def get_command_output(cmd):
    try:
        output = subprocess.check_output(cmd, cwd="C:\someDir")  # Use `cwd` instead of `cmd`
        return True, output
    except subprocess.CalledProcessError:
        return False, None

isOk, output = get_command_output("dir")
```

Lab 01

Lab

Popen class

- The **Popen** class offers a lot of flexibility to handle the common and the less common cases not covered by the convenience functions.
- To support a wide variety of use cases, the **Popen** accept a large number of optional arguments (in most cases, most of the arguments can stay with their default value)
- **Popen** doen’t block the calling function

Popen class — cont’d

- The most common Popen constructor arguments (like in call and check_output functions) are:
  - — args
  - — shell
  - — stdin, stdout, stderr:
    - Existing file descriptor
    - PIPE - indicates that a new pipe to the child will be created

Popen class

Demo

Popen example

```
import subprocess as SP
r = SP.Popen(["dir"], shell=True)
print(r.returncode)         # probably None

import subprocess as SP
r = SP.Popen(["dir"], shell=True)
print(r.returncode)         # probably None
import time
time.sleep(3)
print(r.returncode)         # probably 0
```

Popen with PIPE example

```
import subprocess as SP

r = SP.Popen(["dir"], stdout=SP.PIPE, stderr=SP.PIPE, shell=True)
output, error = r.communicate()  # Use 'communicate()' instead of 'rcommunicate()'
print(output.decode())  # Decode the output bytes and print
print(error.decode())  # Decode the error bytes and print

r = SP.Popen(["nodir"], stdout=SP.PIPE, stderr=SP.PIPE, shell=True)
output, error = r.communicate()  # Use 'communicate()' instead of 'rcommunicate()'
print(output.decode())  # Decode the output bytes and print
print(error.decode())  # Decode the error bytes and print
```

Lab 02

Lab

Questions