Useful Modules

Scientific Programming with Python

Nicola Chiapolini

July 14, 2023

Argparse

```python
#!/usr/bin/env python3
import argparse

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="Process some integers.")
    parser.add_argument("integers", metavar="N", type=int, nargs="+",
                        help="an integer for the accumulator")
    parser.add_argument("--sum", dest="accumulate", action="store_const",
                        const=sum, default=max,
                        help="sum the integers (default: find the max)")
    args = parser.parse_args()

    print(args.accumulate(args.integers))
```

```
$ ./argparsedemo.py --sum 1 2 3 4
10
```

Argparse: Summary

- argparse allows parsing of commandline options.
- part of Python's standard library
- useful to quickly change how program gets executed

Click

Makes writing CLI (Command Line Interfaces) even easier.

```python
#!/usr/bin/env python3
""" Process some integers. """
import click

@click.command()
@click.argument("integers", type=int, nargs=-1, required=True)
@click.option("--sum/--max", "-s/-m", "sum_", default=False, help="use sum or max (default: max)")
def process(sum_, integers):
    accumulator = max
    if sum_:
        accumulator = sum
    print(accumulator(integers))

if __name__ == "__main__":
    # pylint: disable=no-value-for-parameter
    process()
```

```
$ ./clickdemo.py --sum 1 2 3 4
10
```

Configparser

Load configuration from files.

```python
import configparser

conf = configparser.ConfigParser()
conf.read(["config.ini"])

accumulator = max
if conf["setup"].getboolean("sum", False):
    accumulator = sum

integers = [
    int(v) for v in (conf["values"]["ints"]).split(",")
    ]

print(accumulator(integers))
```

```
[setup]
sum = True

[values]
ints = 1,2,3
```

Requests

`requests` perform web-requests, both GET and POST (and more) to interact with anything reachable over the internet.

`BeautifulSoup` parses XML/HTML documents.

```
import requests
from bs4 import BeautifulSoup

# Re-use the connection to the server
session = requests.Session()
# Get the webpage
url = "https://www.physik.uzh.ch/~python"
response = session.get(url)
# Fail early if unexpected response
response.raise_for_status()
# Read it into a datastructure that is easy to query
soup = BeautifulSoup(response.text, "lxml")
links = [a['href'] for a in soup.select("a.internal")]
```

Selenium

`selenium` allows to remote controle a real browser. I.e. all JavaScript and other dynamic content will really be rendered.

```
# coding: utf-8
from selenium import webdriver
driver = webdriver.Chrome()
driver.get("https://www.uzh.ch")
driver.find_elements("tag name", "a")
driver.find_elements("tag name", "a")[-1].click()
```

Subprocess

Sometimes you need to run external commands, for which no Python module exists. This can be done with the `subprocess` module.
It has recently (Python 3.7) been simplified a lot:

```
import subprocess

# show 'du -h *' and 'du -h .' on the commandline
result = subprocess.run(["du", "-h", "."])
print(result)
print()

result = subprocess.run(["du", "-h", "."], capture_output=True)
print(result)
print("output: ", result.stdout.decode())
print("error:  ", result.stderr.decode())
print()

result = subprocess.run(["du", "-h", "*"], capture_output=True)
print("output: ", result.stdout.decode())
print("error:  ", result.stderr.decode())
print()

result = subprocess.run(["du -h *"], capture_output=True, shell=True)
print("output: ", result.stdout.decode())
print("error:  ", result.stderr.decode())
print()

result2 = subprocess.run(["cat"], capture_output=True, input=b"test")
print("output:  ", result2.stdout.decode())
```