Useful Modules

Scientific Programming with Python

Nicola Chiapolini & Christian Elsasser

University of Zurich
Faculty of Science

June 27, 2025

General

difflib

- find differences between two sequences or strings

```
import difflib

start = ("bacon", "egg", "ham", "cheese")
end = ["bacon", "egg", "spam", "cheese"]
diff = difflib.ndiff(start, end)
for line in diff:
    print(line)
```

```
  bacon
  egg
- ham
+ spam
  cheese
```

- calculate how close two sequences or strings are

```
options=["hannah", "anna", "hanna", "andrea", "maria"]
close = difflib.get_close_matches("hana", options)
print(close)

options=[(1,2,3),(2,3,4),(3,4,5),(3,2,1)]
close = difflib.get_close_matches([2,3], options)
print(close)
```

```
['hanna', 'hannah', 'anna']
[(2, 3, 4), (1, 2, 3)]
```

glob

- find files on file system

```python
import glob

matches = glob.glob("tree/*")
print("\n".join(matches))

print("")
matches = glob.glob("tree/**/*.py", recursive=True)
print("\n".join(matches))
```

```
tree/subdir
tree/fileA
tree/fileB.py

tree/fileB.py
tree/subdir/file1.py
```

- you might be interested in `os.walk` as well

```python
import os

tree = os.walk("tree")
for entry in tree:
    print(entry)
```

```
('tree', ['subdir'], ['fileA', 'fileB.py'])
('tree/subdir', [], ['file2', 'file1.py'])
```

Data Types

datetime

- proper representation of dates and times

```
import datetime as dt

t1 = dt.datetime(2017,9,11,13,15)
print(t1)
```

```
2017-09-11 13:15:00
```

- reading and formatting of strings

```
t2 = dt.datetime.strptime("2017-09-11","%Y-%m-%d")
print(t2)

t3 = dt.datetime.now()
print(t3.strftime("%a, %d %B %Y %H:%M"))
```

```
2017-09-11 00:00:00
Sat, 13 July 2024 17:50
```

`datetime` (cont.)

- including timezones

```
tz_cest = dt.timezone(dt.timedelta(hours=+2))
t2 = dt.datetime(2017,9,11,13,15,tzinfo=tz_cest)
print(t2)

# Hour in UTC
print("no tzinfo:  ", t1.utctimetuple().tm_hour)
print("with tzinfo:", t2.utctimetuple().tm_hour)
```

```
2017-09-11 13:15:00+02:00
no tzinfo:   13
with tzinfo: 11
```

- proper representation of dates and times

```
import pytz

zurich = pytz.timezone('Europe/Zurich')
sydney = pytz.timezone('Australia/Sydney')

zurich_dt = zurich.localize(dt.datetime(2018, 3, 11, 19, 0))
print(zurich_dt)
sydney_dt = zurich_dt.astimezone(sydney)
print(sydney_dt)

print(zurich_dt==sydney_dt)
```

```
2018-03-11 19:00:00+01:00
2018-03-12 05:00:00+11:00
True
```

`datetime` (cont.)

- `timedelta` allows to do calculations

```
feb24 = dt.date(2024, 2, 28)
feb25 = dt.date(2025, 2, 28)
tdelta = dt.timedelta(days=1)
print(feb24+tdelta)
print(feb25+tdelta)
```

```
2024-02-29
2025-03-01
```

- a lot of useful functions: e.g. random date in interval

```
import random

start = dt.date(1990, 1,1,).toordinal()
end = dt.date(2000, 1,1,).toordinal()
rand = dt.date.fromordinal(random.randint(start, end))
print(rand)
```

```
1990-12-05
```

enum

- represent a set of possible values

```
from enum import StrEnum

class Compass(StrEnum):
    n = "north"
    ne = "northeast"
    e = "east"
    se = "southeast"
    s = "south"
    sw = "southwest"
    w = "west"
    nw = "northwest"

print(Compass.sw)
print(Compass("north") == Compass.e)
#print(Compass("osten")) # ValueError
```

```
southwest
False
```

dataclasses

- represent structured data

```python
from dataclasses import dataclass
import datetime

@dataclass
class Article:
    title: str
    authors: list[str]
    date: datetime.date
    url: str

dna = Article(
    """Molecular Structure of Nucleic Acids:
A Structure for Deoxyribose Nucleic Acid""",
    ["Watson, J. D.", "Crick, F. H. C."],
    datetime.date(1953, 4, 25),
    "https://doi.org/10.1038/171737a0",
)
print(dna.title)
```

```
Molecular Structure of Nucleic Acids:
A Structure for Deoxyribose Nucleic Acid
```

Functional Tools

itertools

- lots of tools to work with sequences
- chain sequences

```
from itertools import chain, cycle, permutations
```

```
['A', 'B', 'C', 'D', 'E', 'F']
```

```
res = [v for v in chain('ABC', 'DEF')]
print(res)
```

- infinitly cycle sequences

```
gen = cycle('ABC')
print([next(gen) for _ in range(5)])
```

```
['A', 'B', 'C', 'A', 'B']
```

- generate all permutations

```
res = ["".join(v) for v in permutations('ABC')]
print(res)
```

```
['ABC', 'ACB', 'BAC', 'BCA', 'CAB', 'CBA']
```

Settings and User Input

argparse

- Parse commandline arguments to adjust programm execution easily

```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 1 2 3 4
4

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

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
```

readline

- improve user experience for interactive input

```
import readline

while True:
    name = input("Your Name: ")
    print("Hello", name)
```

- uses GNU readline, only available on Linux and Mac
- only incomplete windows alternatives seem to exist

Demo

Output

logging

- control output verbosity

```python
import logging

logging.basicConfig(
        format='%(levelname)s - %(name)s - %(message)s',
        level=logging.WARNING # this is the default
    )

# recommended setup, works well with imports
logger = logging.getLogger(__name__)

logger.debug("This output is only for debugging")
logger.error("There was an error.")
```

```
ERROR - __main__ - There was an error.
```

logging (cont.)

- detailed control of output

```python
log = logging.getLogger(__name__)
# set lowest log-level, handlers can only
# increase threshold/reduce verbosity
log.setLevel(logging.DEBUG)

# add an output-channel for stderr
log.addHandler(logging.StreamHandler())
# reduce ouput for last handler
log.handlers[-1].setLevel(logging.ERROR)

# create and configure output to a file
# (use "a" instead of "w" to append)
logfile = logging.FileHandler("demo.log", "w")
# define format ouf log-lines
# - see `pydoc3 logging.Formatter` for variables
# - extra variables (e.g. `%(var)s`) must be passed
#   via keyword argument `extra`
logfile.setFormatter(logging.Formatter(
    "%(asctime)s -- %(levelname)s, %(var)s: %(message)s"
    ))
log.addHandler(logfile)

log.error("This is bad", extra={"var": 42})
log.info("Info: %s", "busy", extra={"var": 0})
```

console:

```
This is bad
```

logfile

```
2024-07-13 17:50:00,152 -- ERROR, 42: This is bad
2024-07-13 17:50:00,152 -- INFO, 0: Info: busy
```

textwrap

- wrapping and filling text

```
import textwrap

text = """This is an example
text that should be rewrapped
into longer lines, ignoring any
newlines in the original.
"""

res = textwrap.fill(text, width=30)
print(res)

print()

res = textwrap.indent(res, prefix=' # ')
print(res)
```

```
This is an example text that
should be rewrapped into
longer lines, ignoring any
newlines in the original.

 # This is an example text that
 # should be rewrapped into
 # longer lines, ignoring any
 # newlines in the original.
```

texttable

- output tables to terminal

```
from texttable import Texttable

output = Texttable(max_width=30)
output.set_cols_align(["l", "r", "r"])
output.add_row(["n", "2*n", "n**2"])
for n in range(5):
    output.add_row([n, 2*n, n**2])
print(output.draw())
```

```
+---+-----+------+
| n | 2*n | n**2 |
+---+-----+------+
| 0 |   0 |    0 |
+---+-----+------+
| 1 |   2 |    1 |
+---+-----+------+
| 2 |   4 |    4 |
+---+-----+------+
| 3 |   6 |    9 |
+---+-----+------+
| 4 |   8 |   16 |
+---+-----+------+
```

Interacting with the Web

requests

- interact with things reachable over the internet
- http `get` requests: e.g. fetching a website

```python
import requests

url = "https://www.uzh.ch"
response = requests.get(url)
response.raise_for_status()

lines = response.text.split("\n")
lines = [l for l in lines if l.strip() != ""]
for line in lines[:5]:
    print(line)
```

```
<!DOCTYPE html>
<html lang="de" data-template="ct01">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
```

- http `post` requests: e.g. filling a form

```python
res = requests.post(
        "https://httpbin.org/post",
        data={"user": "me", "pass": "secrete"}
        )
print("Status:", res.status_code)
```

```
Status: 200
```

bs4 / lxml

- extract elements from XML/HTML documents: e.g. links

```
from bs4 import BeautifulSoup

content = BeautifulSoup(response.text,"lxml")
links = content.find_all(name="a") # by tag name
#links = content.select("a.Link")   # by css selector
for link in links[:5]:
    print(link["href"])
```

```
https://www.uzh.ch
/cmsssl/de/search.html
https://www.students.uzh.ch
https://www.staff.uzh.ch
/cmsssl/de.html
```

- alternative option `lxml`

```
import lxml.html

content = lxml.html.fromstring(response.text)
links = content.findall(".//a") # by xpath
for link in links[:5]:
    print(link.get("href"))
```

```
https://www.uzh.ch
/cmsssl/de/search.html
https://www.students.uzh.ch
https://www.staff.uzh.ch
/cmsssl/de.html
```

selenium

- remote control a real browser (for JavaScript and other dynamic content)

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service

driver = webdriver.Chrome(
    service=Service("/usr/bin/chromedriver")
    )
driver.get("https://www.uzh.ch")
driver.find_elements("tag name", "a")
driver.find_elements("tag name", "a")[-1].click()
```

Demo

Running External Commands

subprocess

- run external command

```
import subprocess

result = subprocess.run(["uname", "-om"])
print("res:", result)
```

```
x86_64 GNU/Linux
res: CompletedProcess(args=['uname', '-om'], returncode
```

- capture output to access it from python

```
result = subprocess.run(
    ["uname", "-om"],
    capture_output=True
    )
print("---")
print(result.stdout.decode())
print("---")
```

```
---
x86_64 GNU/Linux

---
```

subprocess (cont.)

- metacharacters are not interpreted unless `shell=True`

```
ext = "py"
res = subprocess.run(
        [f"ls -1 *.{ext}"], # single element
        shell=True # RISKY!
    )

# using:
#   ext = input("ext: ")
# creates a shell injection vulnerability
```

```
capture.py
run.py
shell.py
stdin.py
```

Demo

- pass data to standard input with `input=...`

```
data = """# A title
And a simple paragraph
with some text.
"""
res = subprocess.run(["pandoc", "-t", "html"],
        input=data.encode()
    )
```

```
<h1 id="a-title">A title</h1>
<p>And a simple paragraph with some text.</p>
```