12
Testning och Automatisering

Styra interaktiva program

- pexpect - inspiration från Expect (som bygger på språket TCL)
- Kräver inte Expect eller TCL - 100% pure python
- Används för att starta och styra interaktiva applikationer för att efterlikna en människa som interaktivt matar in data till applikationen, t.ex. en ftp-inloggning
- Används flitigt för automatiserad testning
- Se vidare https://pexpect.readthedocs.io/

Exempel: Hämta fil med ftp

```
import pexpect
child = pexpect.spawn('ftp ftp.sunet.se')
child.expect('Name .*: ')
child.sendline('anonymous')
child.expect('Password:')
child.sendline('initgoran@gmail.com')
child.expect('ftp> ')
child.sendline('cd pub')
child.expect('ftp> ')
child.sendline('get HEADER.html')
child.expect('ftp> ')
child.sendline('bye')
child.expect(pexpect.EOF)
```

Exempel 2: pexpect

- Med loggning och felhantering

```
import pexpect
import sys

child = pexpect.spawn('ftp ftp.sunet.se',
                      timeout=10,
                      encoding='utf-8',
                      logfile=sys.stdout)
```

Exempel 2: pexpect (forts.)

```
while True:
    ret = child.expect(['Name .*: ', 'Password:', 'ftp> ',
                        'refused', pexpect.EOF,
                        pexpect.TIMEOUT])
    if ret == 0:
        child.sendline('anonymous')
    elif ret == 1:
        child.sendline('initgoran@gmail.com')
    elif ret == 2:
        break
    else:
        print("Login error", file=sys.stderr)
        sys.exit(1)
```

Exempel 2: pexpect (forts.)

```
cmds = [ 'cd /mirror/ubuntu', 'cd dists/eoan',
         'get Release.gpg', 'bye' ]
for cmd in cmds:
    child.sendline(cmd)
    ret = child.expect(['ftp> ', pexpect.EOF,
                        pexpect.TIMEOUT,
                        'Invalid', '550 Failed'])
    if ret:
        break
if cmd == cmds[-1]:
    print("All OK")
```

Test - doctest

```
"""dc.py - Konvertera till versaler

>>> conv_to_big('abc')
'ABC'
>>> conv_to_big('abc12')
'ABC12'
>>> conv_to_big([1,2])
'[1, 2]'
>>> conv_to_big([1,2,'a'])
"[1, 2, 'A']"
"""

def conv_to_big(st):
    return str(st).upper()
```

Test – doctest (forts.)

```
def _test():
    import doctest
    doctest.testmod()

if __name__== '__main__':
    _test()
```

Test - doctest

- Kör testerna genom att köra själva modulen; *python3 dc.py*
- Gick allt bra visas ingen utskrift
- "Debug-mod"; *python3 dc.py -v*

Test - unittest (PyUnit)

- Pythons version av JUnit
- Ett testfall består av ett antal individuella tester och det skapas genom att subklassa *unittest.TestCase*
- Individuella tester skapas genom att döpa metoder till ett namn som börjar på "test", t.ex. *testequal*, *testconnect* etc
- Två extra metoder; setUp() och tearDown() kan skapas för att initiera data respektive städa upp efter varje individuellt test
- Ett antal ärvda metoder kan anropas i varje individuellt test;
  - *assertEqual(x, y)* - för att verifiera att x == y
  - *assertTrue()* - för att verifiera ett villkor (något som returnerar True/False)
  - *assertRaises()* - för att verifiera att ett undantag kastas
  - motsv failEqual osv finns också
- Se vidare pythondokumentationen

Testa modulen random med PyUnit

```
import random, unittest

class TestSequenceFunctions(unittest.TestCase):
    def setUp(self):
        self.seq = list(range(10))
    def testshuffle(self):
        random.shuffle(self.seq)
        self.seq.sort()
        self.assertEqual(self.seq, list(range(10)))
    def testchoice(self):
        element = random.choice(self.seq)
        self.assertTrue(element in self.seq)
    def testsample(self):
        self.assertRaises(ValueError, random.sample, self.seq, 20)
        for element in random.sample(self.seq, 5):
            self.assertTrue(element in self.seq)
```

Testa modulen random (forts.)

- Huvudprogrammet

```
if __name__ == '__main__':
    unittest.main()
```

- Exekvera

```
$ python3 examples/12_pyuint.py -v
testchoice (__main__.TestSequenceFunctions) ... ok
testsample (__main__.TestSequenceFunctions) ... ok
testshuffle (__main__.TestSequenceFunctions) ... ok

----------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Modulen pytest

- `pytest` är en populär extern modul för testning
- Funktioner vars namn börjar med `test` är testfall
- Pythonfiler vars namn inleds med `test_` innehåller testfall
- Exekvering:
  - `python3 -m pytest`
  - `python3 -m pytest myprog.py`
  - `python3 -m pytest myprog.py -v`
  - `pytest-3 myprog.py`

Testa modulen random med pytest

```
import random
seq = list(range(10))

def test_shuffle():
    L = list(seq)
    random.shuffle(L)
    assert sorted(L) == seq

def test_choice():
    for i in range(1000):
        assert random.choice(seq) in seq

def test_sample():
    for x in random.sample(seq, len(seq)//2):
        assert x in seq
```

Testa random med pytest (forts.)

```
$ python3 -m pytest examples/test_random.py -v
========= test session starts ==========
platform linux -- Python 3.7.3, pytest-3.10.1, py-1.7.0, ...
cachedir: .pytest_cache
rootdir: /home/goran/GIT/pythonkurs, inifile:
collected 3 items

examples/test_random.py::test_shuffle PASSED          [ 33%]
examples/test_random.py::test_choice PASSED           [ 66%]
examples/test_random.py::test_sample PASSED           [100%]

========= 3 passed in 0.02 seconds =========
$
```

Testa random med pytest (forts.)

```
import pytest

def test_sample_err():
    with pytest.raises(ValueError):
        random.sample(seq, len(seq) + 1)

def test_choice_dist():
    found = set()
    res = set(seq)
    while found != res:
        found.add(random.choice(seq))
```

Test - pytest

- Göra testerna ”körbara”:

```
if __name__ == '__main__':
    import pytest
    pytest.main([__file__, "-vx"])
```

Test - beroenden

- Kod som läser från `sys.stdin` och skriver till `sys.stdout`

```
def guess(a, b):
    res = a + b
    while True:
        try:
            g = input("What is {:d}+{:d}: ".format(a, b))
            if int(g) == res:
                break
            print("Wrong!", end=" ")
        except ValueError:
            print("Bad input!", end=" ")
    print("Correct!")
```

Testa stdin/stdout

```
from contextlib import redirect_stdout
import io
import sys

def test_guess():
    f = io.StringIO()
    with redirect_stdout(f):
       sys.stdin = io.StringIO("4\r\n")
       guess(2, 2)
    assert f.getvalue().find("Correct") >= 0
```

Test – beroenden

- Komponenter man vill testa kan vara beroende av andra komponenter
  - ökar komplexiteten
  - kan ha sidoeffekter
  - svårare att automatisera
- Ett vanligt knep är att ersätta externa komponenter med fejkade motsvarigheter
  - ”mock objects”
- Kod bör skrivas så att den blir ”testbar”
  - minimera beroenden mellan komponenter
  - underlätta ”mock” av externa komponenter
- Modulen `unittest.mock` är ofta smidig vid mockning
  - fast Python är i sig sensationellt lätt att mocka.

Mock med inbyggd logik

```
class MockUser:
    def __init__(self, L):
        self.lines = L
    def readline(self):
        if self.lines:
            return self.lines.pop(0)
        else:
            raise EOFError


def test_guess():
    f = io.StringIO()
    with redirect_stdout(f):
       sys.stdin = MockUser(["5\n", "kalle\n", "4\n"])
       guess(2, 2)
    assert f.getvalue().find("Correct") >= 0
```

Övning

- **ovn1201/test_maxtal .py** Skriv ett antal testfall med pytest för funktionen från **ovn0308/funktion2.py**.
- **ovn1202/funktion4.py** Som **ovn0308/funktion2.py** men lägg till doctest.
- **ovn1203/autoguess.py** Skriv ett program som automatiserar körningen av programmet **ovn0602/gissatal2.py**.