Enhetstesting i Visual Studio Code

Når vi skriver kode, må vi sørge for at koden fungerer før vi begynner å bruke den på alvor. Altså, vi må teste den.
Hittil har vi kanskje laget noen kodelinjer i en __main__ funksjon for å «teste» en eller flere funksjoner. Når funksjonene «fungerer», legger vi testkoden til side og går videre til neste problem.

Det er ikke slik vi skal gjøre det. Kanskje vi gjør noen endringer / refaktorering i vår funksjon. Vil du skrive ny kode i en ny __main__ for å teste den? Kanskje du gjør det, men uansett er dette ikke slik du *bør* gjøre testing.

Når vi tester, må vi skrive tester som tester ikke bare «solskinn»-scenarier. Like viktig er grensesituasjonene, eller situasjoner der vi mater vår funksjon med *søppeldata*.
Testen bør kjøres etter hver endring av kode, de skal være raske å utføre, og de må gi klar informasjon om hva som til slutt er galt. For en bestemt funksjon som vi vil teste, skriver vi mange små enhetstester som tester noe spesifikt.

I Visual Studio Code kan vi bruke flere *Unit Testing Frameworks*. I det videre forutsettes at vi bruker det test rammeverket som installeres sammen med den Python extension som vi installerte initielt, dette heter ***unittest***.

I litteraturen / på web er begrepet *unit* forklart på ulike måter.

Vi kan holde oss til følgende definisjon: *En unit er en enkelt funksjon eller en samling funksjoner (gjerne i en klasse) som vi vil teste, poenget er at vi behandler dem som en* ***enhet****.*

Bruke unittest i VSCode

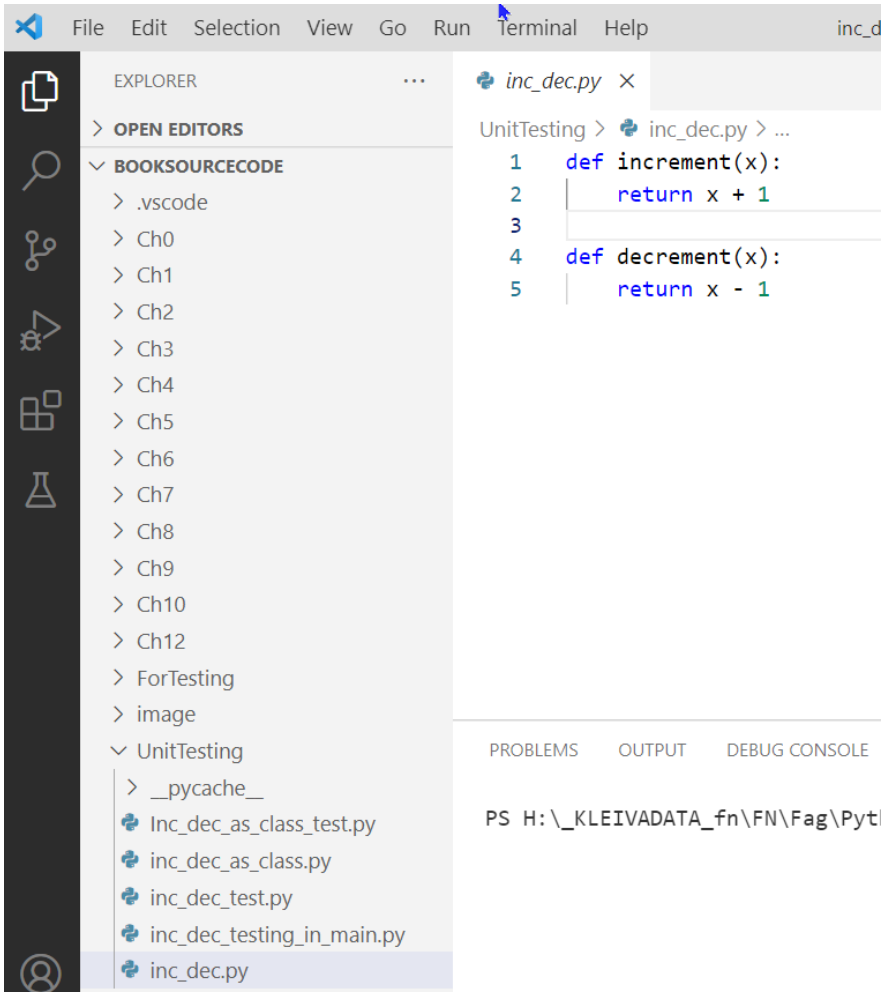
Ikonet for testing ligger til venstre i VSCode og ser ut som en kolbe (sånne fra kjemitimene):

Første gang vil du bli spurt om testrammeverk (velg *unittest*) og katalogen der testene skal ligge. Du kan velge å legge testene til samme katalog som koden du skal teste ligger. Du vil kanskje også bli spurt om hvilket navngivingssystem testrammeverket skal se etter for å identifisere .py filer som inneholder tester. Velg en av de foreslåtte og navngi dine tester etter denne standarden. Jeg har

valgt å navngi mine .py test filer slik:

Dersom fila jeg skal teste heter IncDec.py, så kaller jeg fila med tester for IncDec_test.py.

Dersom begge disse filene ligger i samme katalog kan jeg starte testen ved å trykke på det vanlige «kjør» symbolet som vi er vant til. Under vist min organisering av en mappe som jeg har kalt UnitTesting. Her ligger mine første funksjoner jeg ønsker å teste i fila inc_dec.py



Selve testen av funksjonene decrement og increment ligger i fila inc_dec_test.py:

Koden i inc_dec_test.py:

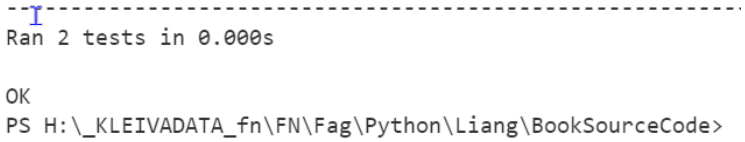
```
from inc_dec import *    # The code to test
import unittest   # The test framework

class Test_TestIncrementDecrement(unittest.TestCase):
    def test_increment(self):
        self.assertEqual(increment(3), 4)

    def test_decrement(self):
        self.assertEqual(decrement(3), 2)

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

Dersom du kjører denne koden («Run Python File») vil testmotoren svare med:



Hvis testene feilet får du beskjed om det.

Dersom vi pakker disse funkjonene inn i en klasse, vil hhv klassen og testene se slik ut:

Klassen (også laget for å kunne kjøre standalone), i fila `Inc_dec_as_class.py`

```
class IncDec:
    def __init__(self, value=0):
        self.__value = value

    def increment(self):
        self.__value += 1
        return self.__value

    def decrement(self):
        self.__value -= 1
        return self.__value

def main():
    inc_dec = IncDec(4)
    print(inc_dec.increment())
    print(inc_dec.decrement())

if __name__ == "__main__":
    main()
```

Test-koden, som ligger i fila `Inc_dec_as_class_test.py`:

```
from inc_dec_as_class import IncDec   # The code to test
import unittest   # The test framework

class Test_TestIncrementDecrement(unittest.TestCase):
    def setUp(self):
        self.__inc_dec = IncDec(4)

    def test_increment(self):
        self.assertEqual(self.__inc_dec.increment(), 5)

    def test_decrement(self):
        self.assertEqual(self.__inc_dec.decrement(), 3)

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

assert statementene er selve testene…

Du kan kalle test klassen hva du vil, men den MÅ arve `unittest.TestCase`, og du kan kalle test metodene hva du vil. Det er likevel en god regel at navnet på metoden forteller hva du tester.

Metoden setUp() er en metode som vil bli kjørt *foran hver test*, den brukes for å skape det objektet vi skal teste.

Inne i selve testen bruker du såkalte asserts: dette er kort og godt et kall til en metode, og resultatet sammenlignes med den forventede verdien. Se på koden, og sammenlign med det du kaller og forventet verdi, så skjønner du tegninga.

Her er en oversikt over de asserts som du kan bruke i testene:
https://docs.python.org/3/library/unittest.html#assert-methods

Dersom du velger kjemikolbe symbolet skal unittest runneren finne de kataloger og filer og tester du kan kjøre, og du får mulighet til å kjøre testene fra dette view'et: