Basis apps

Testen en Verbeteren

hoofdstuk

3

Python unittesten

Algemene informatie

Onderwerp	Python unittests
Leerdoel(en)	1. De student kan unittests in Python maken. 2. De student kan unittests in Python interpreteren. 3. De student kan code verbeteren n.a.v. Python unittests.
Vereiste voorkennis	Alle opgedane kennis van Thema's 1-4, 7 en 8.
Kwalificatiedossier	☐ B1-K1-W1: Plant werkzaamheden en bewaakt de voortgang ☐ B1-K1-W2: Ontwerpt software ☐ B1-K1-W3: Realiseert (onderdelen van) software ☒ B1-K1-W4: Test software ☒ B1-K1-W5: Doet verbetervoorstellen voor de software ☒ B1-K2-W1: Voert overleg ☐ B1-K2-W2: Presenteert het opgeleverde werk ☒ B1-K2-W3: Reflecteert op het werk

Inhoudsopgave

Algemene informatie 2
Inhoudsopgave 3
Introductie 4
Inhoud 4
- Online voorbeelden 4
- Interpreteren unittest-meldingen 5
- Voorbeeldcode 7

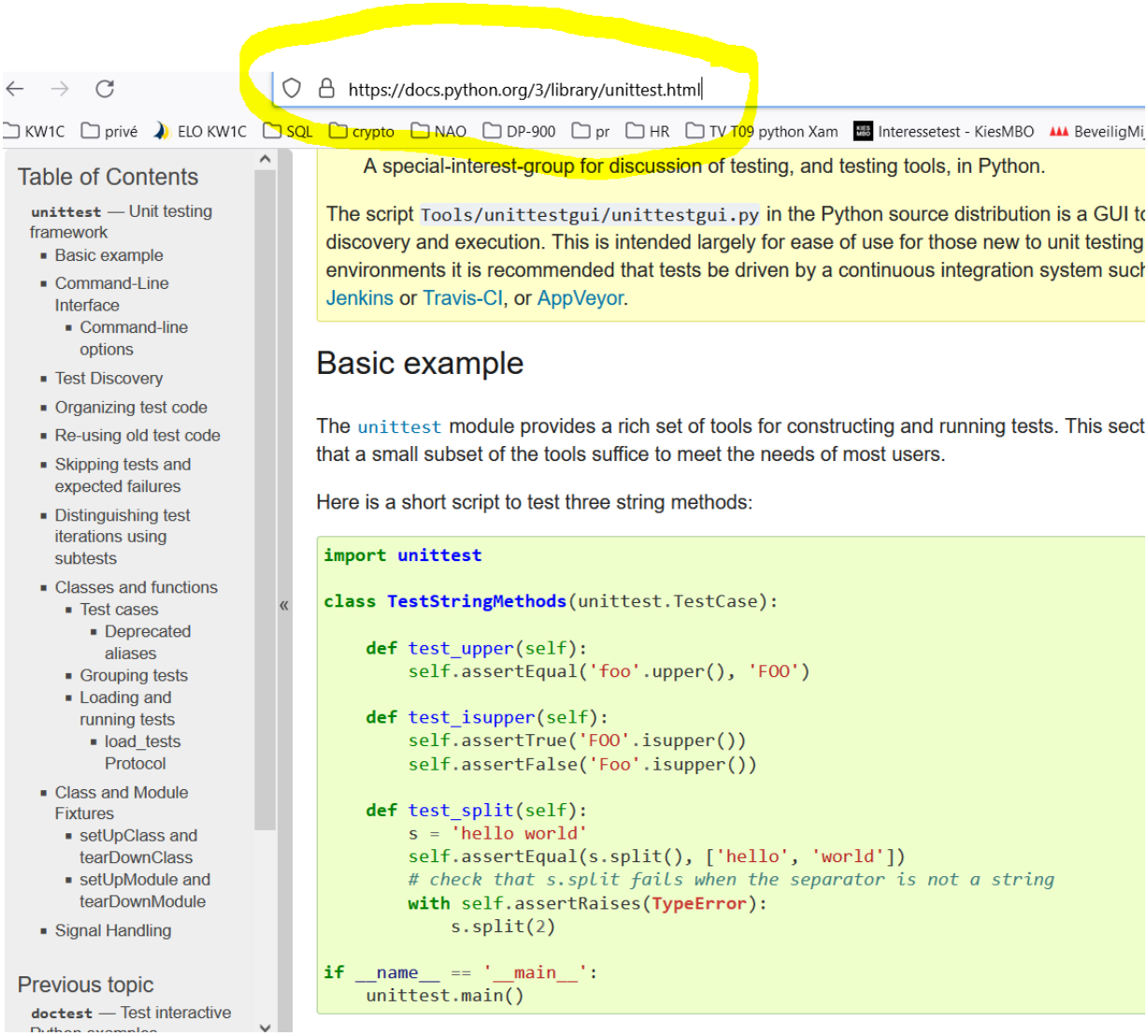
Introductie

In thema 8 heb je kennis gemaakt met unittest. Dit werd toen toegepast op de standalone applicatie in C# en Visual Studio.
Een unittest is een universeel begrip. Dat betekent dat diverse frameworks en syntaxis beschikken over unittest-mogelijkheden. Je gaat nu kennismaken met Python unittests.

Inhoud

Online voorbeelden

Uiteraard zijn er veel voorbeelden online te vinden. Een mooi compleet voorbeeld met diverse mogelijkheden vind je op: https://docs.python.org/3/library/unittest.html



Wanneer je een Pythonbestand uitvoerbaar wil maken, heb je onderstaande if-statement in je bestand nodig:

```
57
58 # nodig om dit bestand uitvoerbaar te maken
59 if __name__ == '__main__':
60     unittest.main()
61
```

Je vindt hier meer uitleg over dit if-block.

Interpreteren unittest-meldingen

Wanneer een test goed verloopt krijg je een pass. Wanneer je als voorbeeld in een testclass drie functies zou hebben staan die allemaal goed verlopen, krijg je onderstaande melding:

```
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK

Process ended with exit code 0.
```

Echter…. je test uiteraard met het idee dat er ook wat mis kan gaan. Hieronder zie je een voorbeeld van een unittest waar in de class weer drie testen staan. Alleen de tweede test krijgt een PASS. Dit zie je aan de punt → deze staat gelijk aan een PASS. Je ziet dus dat de eerste en de derde test een FAIL krijgen.
Je krijgt dus géén OK te zien voor de test met een PASS.
In dit voorbeeld zie je ook bij de AssertionError staat dat de twee strings niet gelijk zijn.

```
F.F
======================================================================
FAIL: test_myFirstUnittest (__main__.testStringDevs)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "D:\ROB_T0910\docs\Python_raspPi\unittestPy2_instr\unittest2.py", line 29, in test_myFirs
t
    self.assertEqual(veranderdwoord, controlewoord)
AssertionError: 'AMSTERDAM' != 'AMSTeRDAM'
- AMSTERDAM
?     ^
```

Het kan ook voorkomen dat je een error hebt in je code, of in je unittest. De melding ziet er dan zo uit:
(in dit geval: Een test met PASS, een test/code met een error, een test met een FAIL)

```
.EF
======================================================================
ERROR: test_mySecondUnittest (__main__.testStringDevs)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "D:\ROB_T0910\docs\Python_raspPi\unittestPy2_instr\unittest2.py",
```

Hoe het ook zij, er komt altijd een exit code uit. Dat ziet er dan bijv. zo uit:

```
Process ended with exit code 1.
```

Wanneer exit code 1 is, dan betekent dat dat er iets **fout** is gegaan. Dat kan zijn: een FAIL, maar ook met een Error krijg je code 1.

Alleen wanneer **alle** unittests een PASS hebben krijg je:

```
Process ended with exit code 0.
```

Je kunt nu **T9_TV_Oefening_T_6** maken.

Je kunt nu **T9_TV_Oefening_V_6** maken.

Voorbeeldcode

Een Python unittest kan er zó uitzien (class met 1 unittest-functie):

```
"""
    Author:     Rob JM Wessels
    Date:       Oct 6, 2022

    Subject:    unittest2 - test
"""

# import libraries en inhoud functions
import unittest
from functions import *

# testclass aanmaken
class testStringDevs(unittest.TestCase):
    # testfunctie aanmaken
    def test_myFirstUnittest(self):

        # Arrange
        invoerwoord = "Amsterdam"
        controlewoord = "AMSTERDAM"

        # Act
        veranderdwoord = changeStringUp(invoerwoord)

        # Assert
        self.assertEqual(veranderdwoord, controlewoord)
```

 Je kunt nu **T9_TV_Oefening_TV_7** maken.