



Lektionstillfälle 5

"Enhetstest & GitHub Actions"

Utbildare: Robert Westin

NACKADEMIN

Kort summering av föregående lektion/ev. lektioner

Föregående lektion:

- Kubernetes
 - Kubectl
 - Vad är en Secret?
 - Vad är en pod?
 - Vad är en service?
- Minikube
 - Vi lärde oss grundläggande start & stop av minikube
 - Kom ihåg "minikube stop" då vi inte använder kubernetes längre

Lektionstillfällets mål och metod

Mål med lektionen:

- Enhetstester
- GitHub Action
- Övningar

Lektionens arbetsmetod/er:

- Föreläsning, Live Kod & övning

Begreppsgenomgång

- TDD – Testdriven utveckling
- RED – GREEN – REFACTOR metoden
 - Skriv ett fallerande test (RED)
 - Skriv kod så att testet passerar (GREEN)
 - Snygga till koden (re-fakturera)
- Enhetstest vs Integrationstest
 - Enhetstest
 - Snabba, inga systemberoenden
 - Isolerade, testar en liten kodbit
 - T.ex. en metod eller en klass
 - Integrationstest
 - Testar flera delar, t.ex. hur två klasser samspelar
 - Testar ett delsystem

Begreppsgenomgång forts.

YAML – Ett filformat

GitHub Organization vs GitHub User Dashboard

GitHub Repository (kan vara i user eller i orgs)

GitHub Actions

.github/workflows - Folder som innehåller YAML filer med actions

YAML Ain't Markup Language

What It Is: YAML is a human friendly data serialization standard for all programming languages.

Skapat av:
Oren Ben-Kiki
Clark Evans
Ingy döt Net

Källa: <https://yaml.org/spec/1.2/spec.html>

NACKADEMIN

YAML

The design goals for YAML are, in decreasing priority:

YAML is easily readable by humans.

YAML data is portable between programming languages.

YAML matches the native data structures of agile languages.

YAML has a consistent model to support generic tools.

YAML supports one-pass processing.

YAML is expressive and extensible.

YAML is easy to implement and use.

NACKADEMIN

Källa: <https://yaml.org/spec/1.2/spec.html>

YAML

- Scalar
 - Zero or more unicode characters
- Sequence
 - Python list, ordered, same value multiple time is possible
- Mapping
 - Python dict, unordered, *key/value, key unique*

NACKADEMIN

YAML – Scalars exempel

Example 2.19 Integers

```
canonical: 12345
decimal: +12345
octal: 0o14
hexadecimal: 0xC
```

Example 2.20 Floating Point

```
canonical: 1.23015e+3
exponential: 12.3015e+02
fixed: 1230.15
negative infinity: -.inf
not a number: .nan
```

Example 2.21 Miscellaneous

```
null:
booleans: [ true, false ]
string: '012345'
```

Example 2.22 Timestamps

```
canonical: 2001-12-15T02:59:43.1Z
iso8601: 2001-12-14t21:59:43.10-05:00
spaced: 2001-12-14 21:59:43.10 -5
date: 2002-12-14
```

YAML - Basics

Du kan starta (om du vill) ett dokument med:

```
---
```

Värden anges med key: value (mapping)

```
host: localhost
```

Detta blir ett nummer, jämför med sträng ovan

```
lucky_number: 13
```

YAML – Typer

Detta blir ett problem

```
car:
```

```
  model: 900 #number
```

```
car:
```

```
  model: xc90 #string
```

YAML – Typer

Detta blir ett problem

car:

model: !!str 900 # bestäm string eller "900"

car:

model: xc90 #string

YAML - Sequence

För att skapa en lista (obs notera också indrag)

person:

hobbies:

- Skiing
- Computers

YAML – Block Mappings

För att skapa en flera mappings i ett block
person:

info:

first_name: pelle

last_name: svensson

Vi testar med yq i terminalen:
yq eval person.yml -o json

YAML – Block Style Vs Flow Style

- Block Style

person:

info:

first_name: pelle

last_name: svensson

- Flow Style (mixed)

person:

info: {first_name: pelle, last_name: svensson}

NACKADEMIN

YAML – Block Style Vs Flow Style

- Block Style

car:

attributes:

- Panorama
- Navigation

- Flow Style (mixed)

car:

attributes: [Panorama, Navigation]

NACKADEMIN

YAML – Uppgift

Skriv YAML innehållande:

- "scalars" med format:
 - Sträng
 - Nummer
 - Datum
- En sekvens (lista)
- En mapping (dict)

Virtual Environments - Repetition

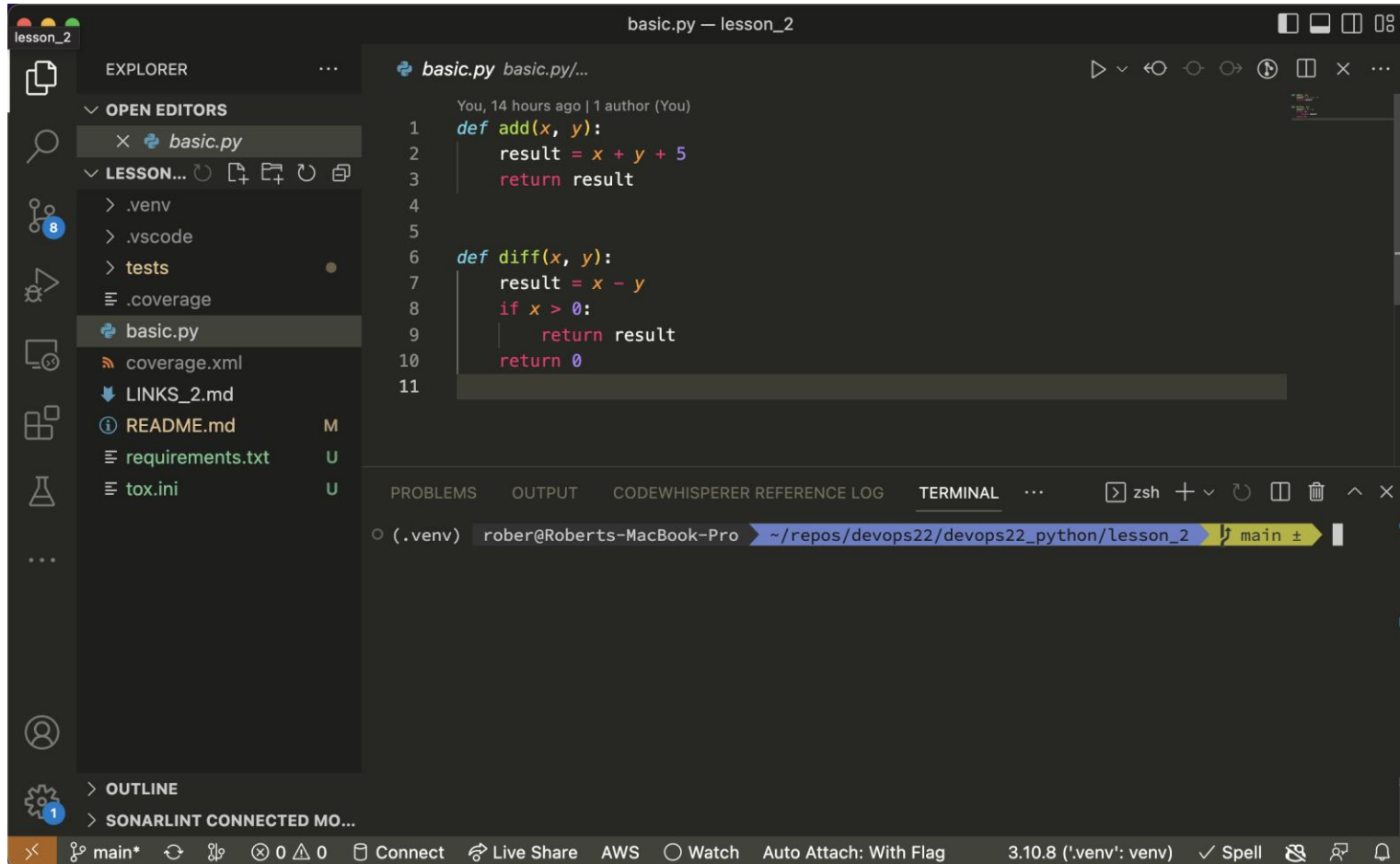
- Inbyggd modul sedan python 3.3
- Möjliggör att använda specifika versioner och paket för ett projekt
- Skapar en folder i projektet, ofta .venv
- Punkt före namnet betyder dold folder
- Checka aldrig in .venv foldern
- TIPS Lägg alltid till .venv i .gitignore
- Se README.md i kursrepot

Virtual Environments – Mac OS

OBS!! Exempel för mac/linux, se docs för andra OS

```
rober@Roberts-MacBook-Pro ~/repos/devops22/devops22_python | main ± | cd lesson_2
rober@Roberts-MacBook-Pro ~/repos/devops22/devops22_python/lesson_2 | main ± | python3 -m venv .venv
rober@Roberts-MacBook-Pro ~/repos/devops22/devops22_python/lesson_2 | main ± | source .venv/bin/activate
(.venv) rober@Roberts-MacBook-Pro ~/repos/devops22/devops22_python/lesson_2 | main ± | pip install -r requirements.txt
Collecting autopep8
  Using cached autopep8-1.7.0-py2.py3-none-any.whl (45 kB)
Collecting coverage
  Using cached coverage-6.5.0-cp310-cp310-macosx_10_9_x86_64.whl (185 kB)
Collecting flake8
  Using cached flake8-5.0.4-py2.py3-none-any.whl (61 kB)
Collecting tox
  Using cached tox-3.26.0-py2.py3-none-any.whl (86 kB)
Collecting pytest
```

Virtual Environment



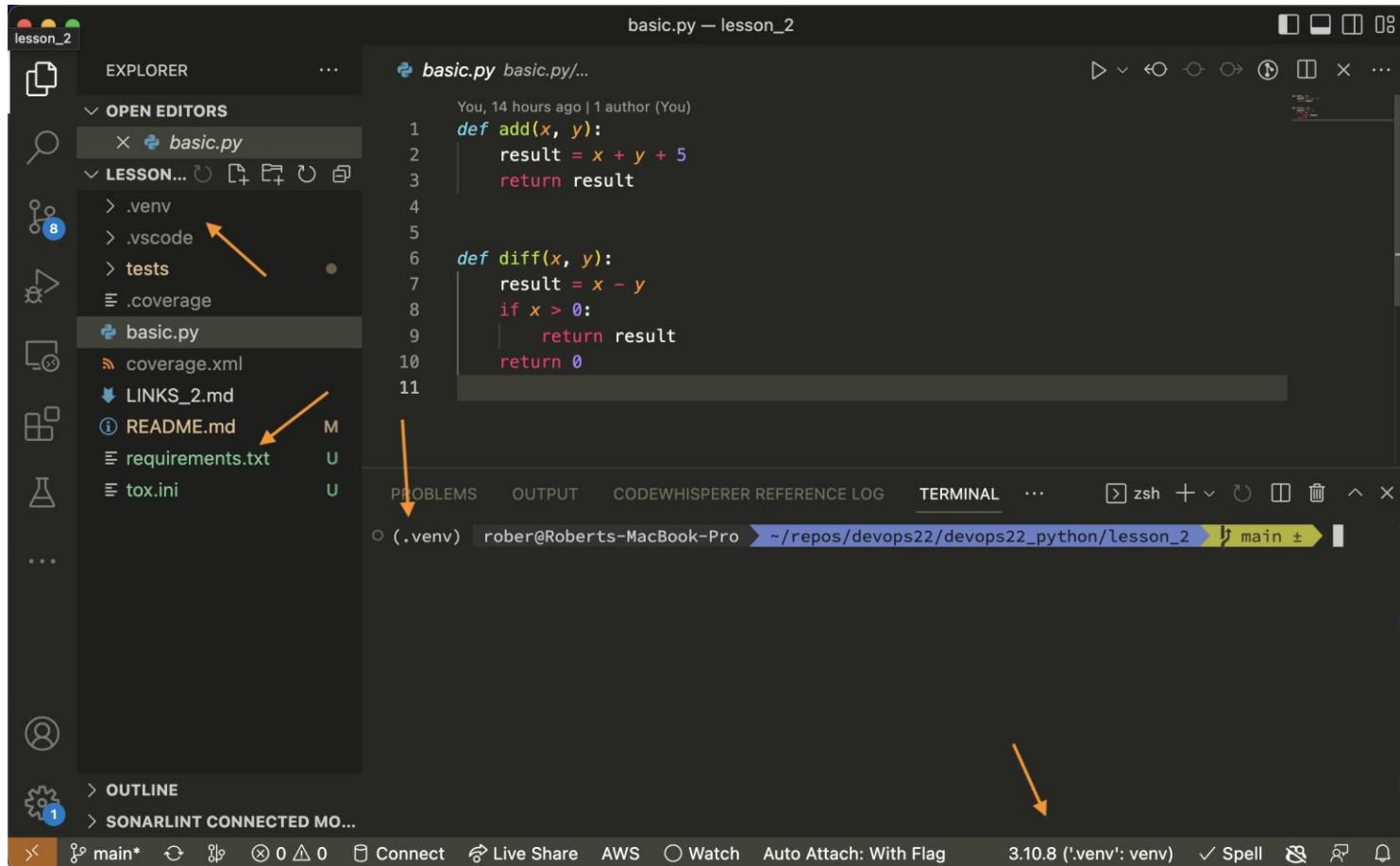
The screenshot displays the Visual Studio Code interface. The Explorer sidebar on the left shows a project named 'lesson_2' with the following files and folders: `.venv`, `.vscode`, `tests`, `.coverage`, `basic.py` (selected), `coverage.xml`, `LINKS_2.md`, `README.md`, `requirements.txt`, and `tox.ini`. The main editor window shows the file `basic.py` with the following Python code:

```
1 def add(x, y):
2     result = x + y + 5
3     return result
4
5
6 def diff(x, y):
7     result = x - y
8     if x > 0:
9         return result
10    return 0
11
```

The bottom panel shows the Terminal with a `zsh` shell prompt. The terminal output indicates the current directory is `~/repos/devops22/devops22_python/lesson_2` and the active environment is `main`.

NACKADEMIN

Virtual Environment

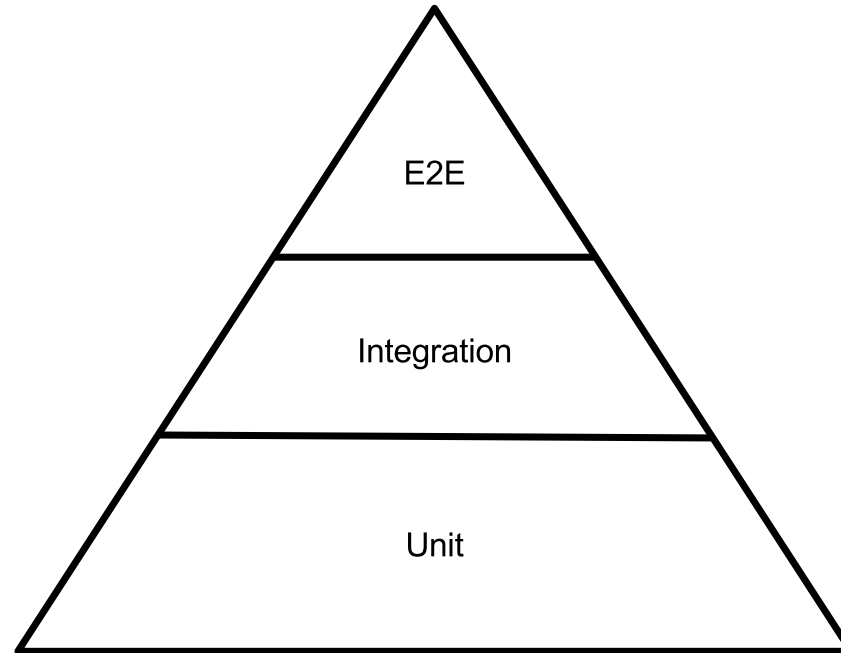


NACKADEMIN

Varför skriver vi enhetstester?

- Jag som utvecklare måste veta OM min kod fungerar
- Vi som team måste veta HUR vår kod fungerar
- Saknas tester är det svårt att ta över någons annans arbete
- Att skriva tester gör koden "automatiskt" enklare
 - Därför att det är jobbigt att skriva tester för komplicerad kod
- Enhetstester är utvecklarens ansvar
- Enhetstester är projektets stöttepelare över tid
 - Det är svårt nog att komma ihåg sin egna gamla kod.
 - Nils & Fia som kodade detta har slutat, vad gör denna kod??

Test pyramiden



NACKADEMIN

Testramverk "test frameworks"

- Tester skrivs i metoder eller klasser
- Tester är pythonkod i .py filer
- Testramverket letar testerna "Test Discovery"
- Testramverket kör testerna
- Testramverket hanterar när det går bra (pass) eller dåligt (fail, errors)
- Testramverket skriver ut vad som gick fel, för att underlätta felsökning
- Testramverket summerar körningen av alla tester
 - Hur många tester kördes?
 - Gick alla tester igenom eller var det något fel? (exit code)

Testramverk i Python

- Det finns ett inbyggt testramverk i Python som heter unittest
- <https://docs.python.org/3/library/unittest.html>
- Det är helt ok att använda, men det finns andra bättre alternativ.

Exempel på inbyggda unittest

```
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

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

Källa: <https://docs.python.org/3/library/unittest.html#basic-example>

Exempel på inbyggda unittest

```
...
```

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

```
OK
```

Källa: <https://docs.python.org/3/library/unittest.html#basic-example>

NACKADEMIN

Unittest summering

- Tester måste importera unittest
- Tester skrivs i klasser som ärver av unittest
- Tester måste anropa unittest.main()
- Tester använder metoder såsom:
 - "assertTrue"
 - "assertFalse"
 - "assertEqual"
- Exempel:
 - $X = 1, Y = 1$
 - Ett test med `assertEqual(X,Y)` kommer att passera (lyckas)

Andra vanliga testramverk i Python

- Nose
- Pytest
- Fördelar:
 - Uttökning & förbättring av inbyggda ramverket
 - Bättre stöd för fixtures (test setup av t.ex. databaser)
 - "Enklare" tester
- Nackdelar:
 - Kräver pip install av ett paket

Exempel på pytest

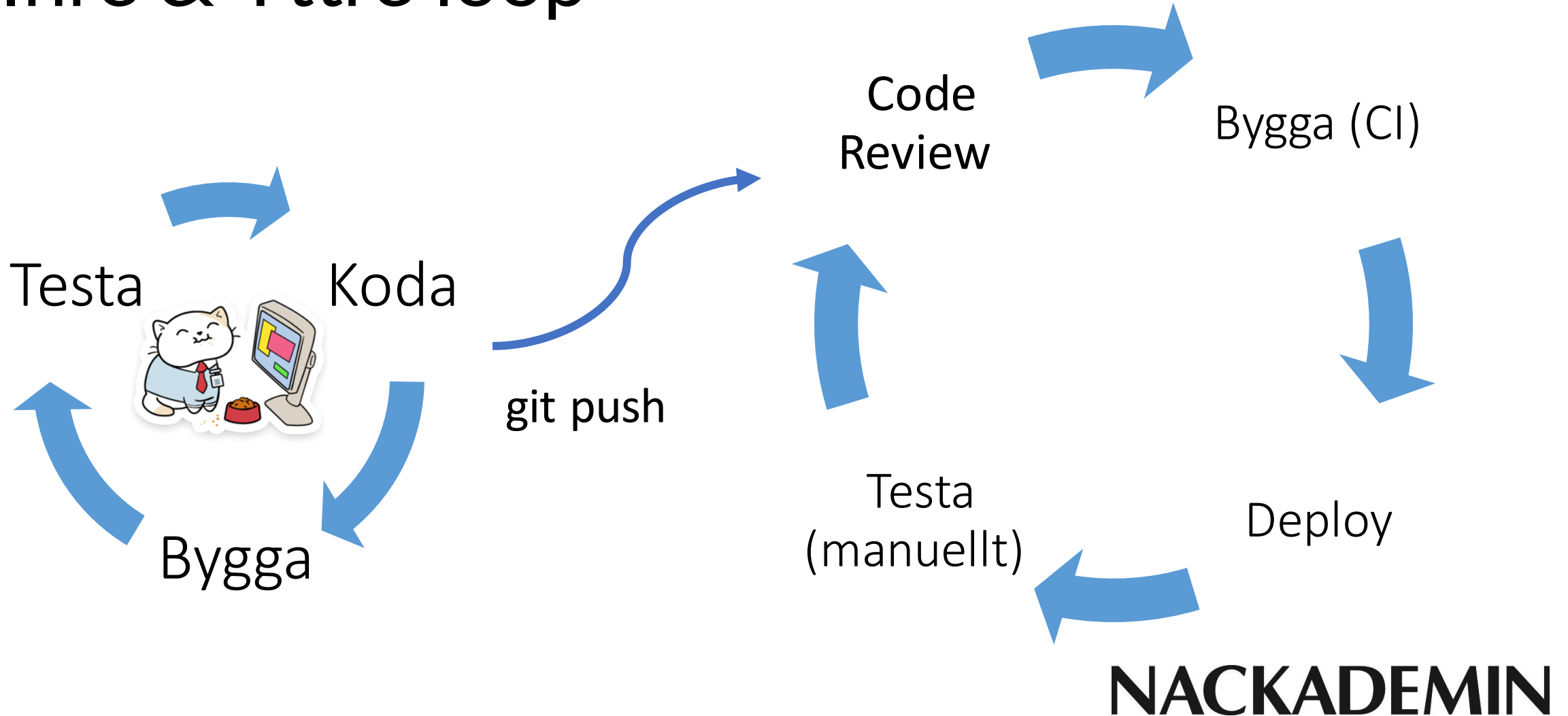
```
import pytest

✓ def test_upper():
    |     assert 'foo'.upper() == 'F00'

✓ def test_isupper():
    |     assert 'F00'.isupper() == True
    |     assert 'Foo'.isupper() == False

✓ def test_split():
    |     s = 'hello world'
    |     assert s.split() == ['hello', 'world']
    |     # check that s.split fails when the separator is not a string
    |     with pytest.raises(TypeError):
    |         s.split(2)
```

Inre & Yttre loop



Vad är viktig för den inre utvecklarloopen?

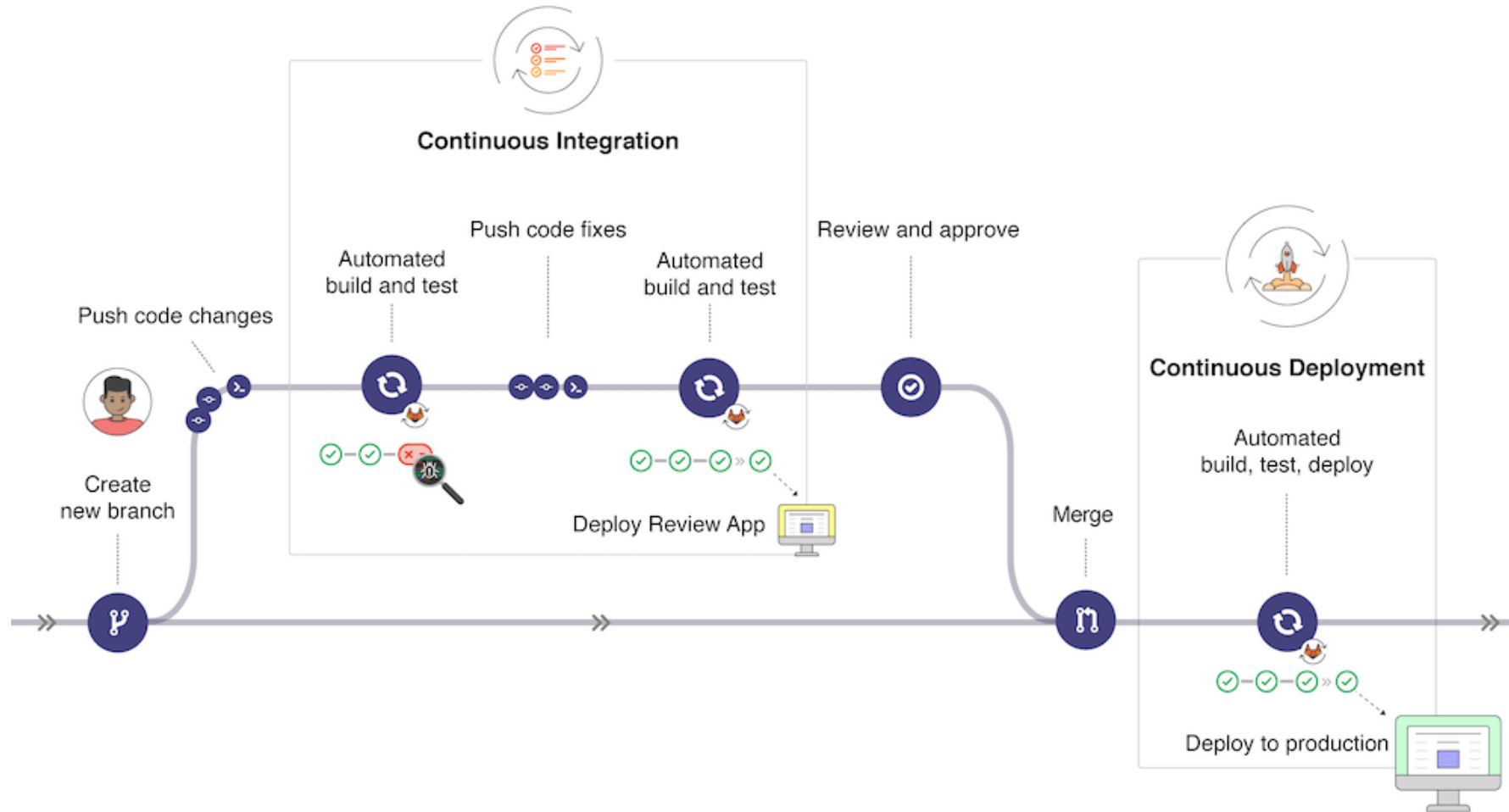
- Snabbt
 - Smidigt att arbeta
 - Hitta de enklaste och vanligaste felen
 - Enhetstester
 - Linting
 - Kodstil
-
- Stöd utvecklaren i sitt arbete!

Vad är viktig för den yttre CI-loopen?

- Säkerställa projektets kvalitet
- Går det att bygga?
- Testa att köra på riktigt?
- Granska ändring - Är ändringen bra?
- Konfiguration, deploy till test miljö med extra debug info?
- Icke funktionella krav, säkerhet, prestanda
- Skydda systemet mot ändringar som inte håller måttet!

NACKADEMIN

CI & CD



NACKADEMIN

Källa: <https://docs.gitlab.com/ee/ci/introduction/>

Pytest

- Vi kommer använda pytest
- Varför?
 - <https://pypistats.org/packages/nose>
 - <https://pypistats.org/packages/pytest>
- Mest använda alternativ
- Fungerar bra till olika typer av testning (unit, integration, system)
- Beprövat och stabilt

GitHub Actions

```
1  name: Python package
2
3  on: [push]
4
5  jobs:
6    build:
7      runs-on: ubuntu-latest
8      strategy:
9        matrix:
10         python-version: ["3.9", "3.10"]
11      steps:
12        - uses: actions/checkout@v3
13        - name: Set up Python ${ matrix.python-version }
14          uses: actions/setup-python@v4
15          with:
16            python-version: ${ matrix.python-version }
17        - name: Install dependencies
18          run: |
19            python -m pip install --upgrade pip
20            pip install pytest
21            if [ -f requirements.txt ]; then pip install -r requirements.txt; fi
22        - name: Test with pytest
23          run: |
24            pytest
25
```

NACKADEMIN

GitHub Actions

Vi kodar och lär oss läsa GitHubs guider:

<https://docs.github.com/en/actions/quickstart>

NACKADEMIN

GitHub Actions

Bra källa för exempel:

<https://github.com/actions/starter-workflows>

NACKADEMIN

GitHub Actions

Bra källa för grundläggande förståelse:

<https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions>

NACKADEMIN

Dagens övning

- Se Studentportalen

Vad vi lärt oss idag

- Vad är TDD
 - En programmerings filosofi eller metodik
- Vad är ett enhetstest
 - Definition på en slags test
- Vad är ett testramverk
 - Unittest, nose, Pytest
- Vi lärde oss grundläggande hantering av pytest
- Vi lärde oss skriva enkla testfall

Nästa lektion

- Vi fortsätter med enhetstester & kvalitetsverktyg
 - Mock
 - Coverage
 - Linting