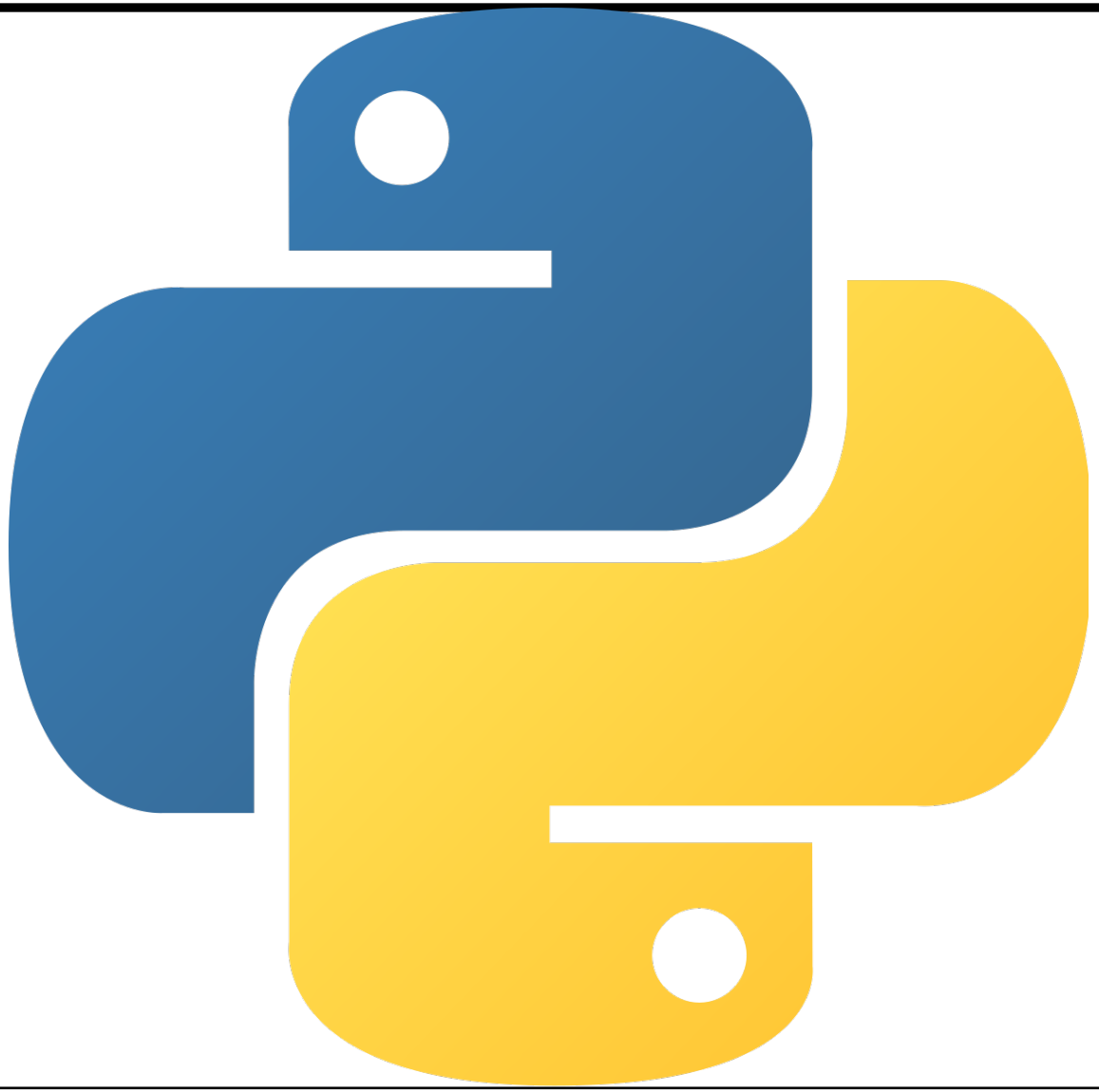
PYTHON PROGRAMMERING



Föreläsning 12

DAGENS FRÅGA

DAGENS AGENDA

- Error och exceptions: skapa egna och hantera
- Debugging
- Att testa kod
- Unit testing
- Pytest

- Moduler, pip install och köra .py genom terminalen
- Argparse

Varning för många memes

FÖRRA FÖRELÄSNING

- Exploratory data analysis:
    1. Förstå datan
    2. Data cleaning
    3. Analysera datan
- Data science: inspiration och exempel
- Plotter och numpy
- Exempel hur skapa en OOP med klass.py och main.py

ERRORS OCH EXCEPTIONS

- Syntax error: Syntaxfel som gör att koden inte kan köra för att den inte är skriven rätt
- Runtime error: Fel som upptäcks medan kod körs
- Semantisk (logiskt) error: Fel i logiken som gör att programmet inte gör rätt. Dessa kan vara svåra att upptäcka eftersom programmet inte avslutas som med de två andra typerna!

- Vanliga runtime errors:
  - *NameError:* Fel variabelnamn
  - *TypeError:* Fel typ på input
  - *ValueError:* Rätt typ men fel värde
  - *KeyError:* Key finns inte i dictionary
  - *AssertionError:* assert False
  - *ImportError:* Paket finns inte

SKAPA EGNA EXCEPTIONS

- För att skapa en exception kan vi använda nyckelordet **raise**

```
if type(x) != int:
    raise ValueError('The value of x was not an int')
```

- AssertionErrors kommer från nyckelordet **assert** som kan användas för att kolla att ett uttryck stämmer

```
assert cond == True, 'The condition was False'
```

HANTERA EXCEPTIONS

- I python hanterar man exceptions med **try** och **except**-satser
- Strukturen är:

```
try:
    # Denna kod körs alltid

except:
    # Om python stötte på errors under try
    # Så kommer denna kod att köras

finally:
    # Denna kod kommer att köras oavsett vad som hänt innan
```

- I except-satsen får vi en chans att ge användaren feedback eller rätta till felet
- finally-satsen kommer alltid att köras oavsett
- Vi kan även specificera vilket error vi vill att python ska fånga:

```
try:
    # kod

except NameError as e:
    # Här fångar vi endast NameError
    # och vi printar den utan att programmet avslutas
    print(e)
```

HANTERA FLERA EXCEPTIONS

- Vi kan också hantera flera olika sorters exceptions genom att göra en kedja av except-block
- Kan användas om vi vet att det kan uppstå en rad olika exceptions

```
try:
    # Denna kod körs alltid

except NameError as e:
    # Här kan vi hantera NameErrors

except AssertionError as e:
    # Här kan vi hantera AssertionError

except TypeError as e:
    # Här kan vi hantera TypeError
```

STACK TRACE

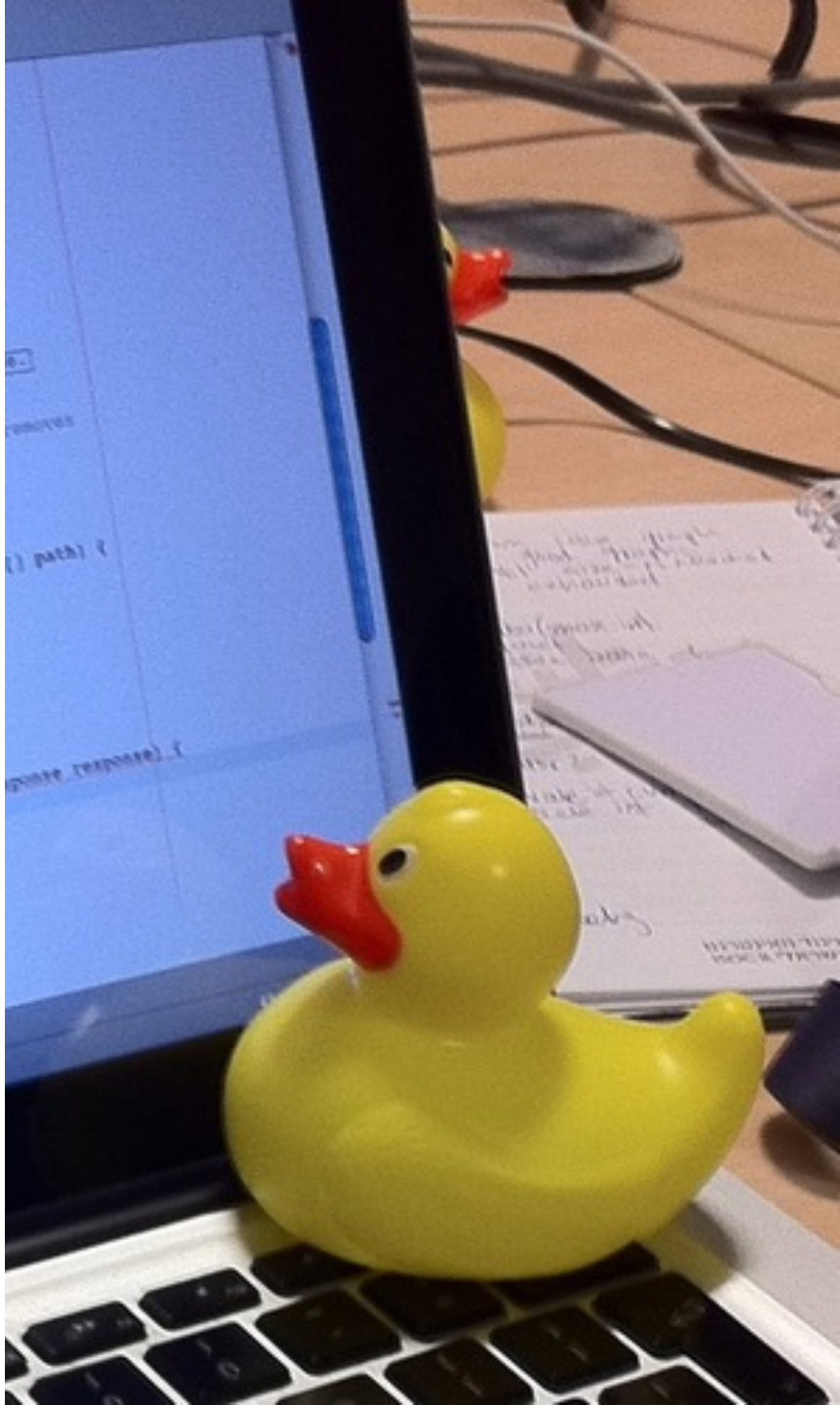
- En stack trace är det meddelande vi får då vi får en error
- I början kan den verka skrämmande och svårtolkad men det är värt att lära sig läsa den!
- Stack tracen läses från botten upp i python – skiljer sig från andra språk
- Den sista raden är det faktiska felet
- Stack tracen listar alla funktionsanrop som gjorts innan felet

```
~/projects/py/Environments/my_env 35s
my_env ❯ python example.py
Traceback (most recent call last):
  File "example.py", line 5, in <module>
    say('Michael')
  File "example.py", line 3, in say
    print('Hello, ' + nam)
NameError: name 'nam' is not defined
```

DEBUGGING

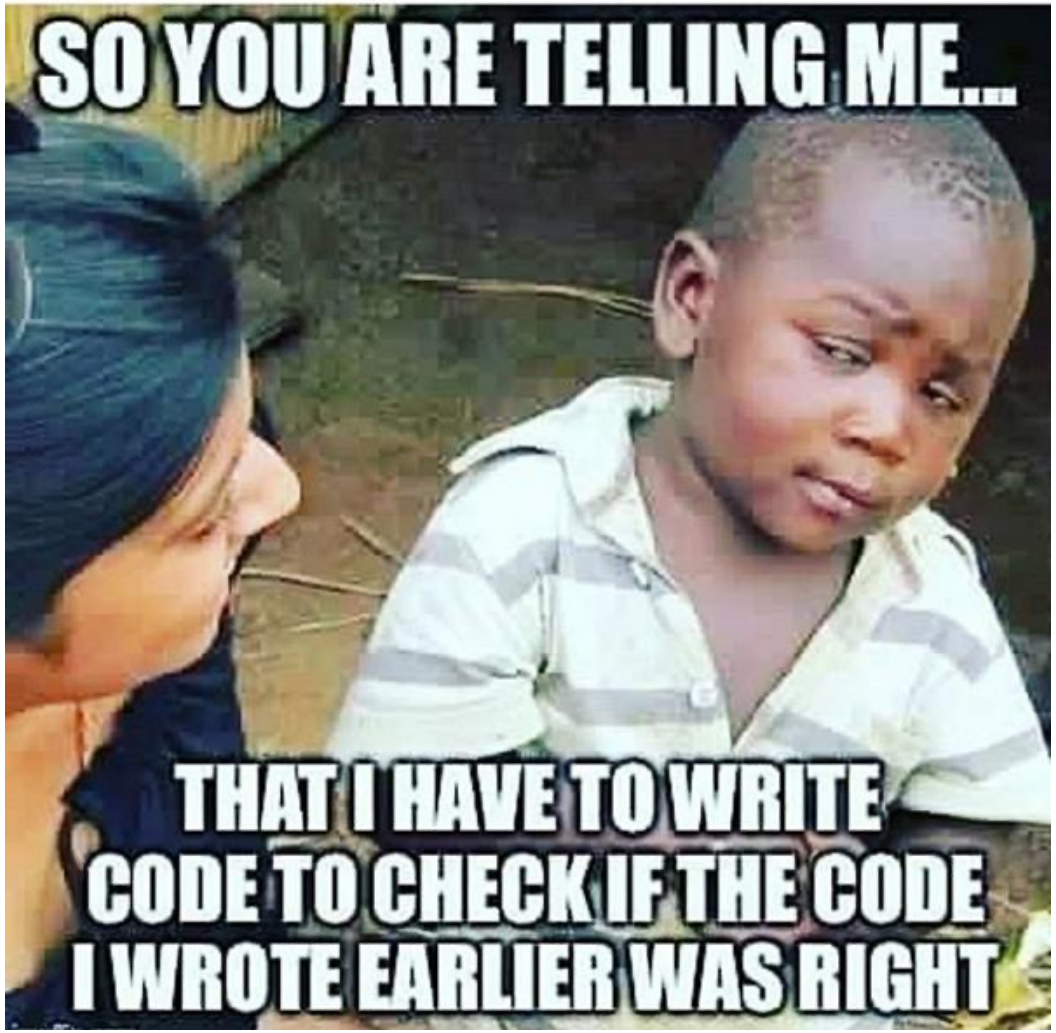
- Debugging är en process där man hittar fel och buggar i ett program. Det finns många olika sätt att debugga

1. Print debugging: använd print() för att kolla värden
   - Lätt men ofta ineffektivt
   - Bra när man programmerar OOP för att se att man har gått in i rätt metod. Tex skriva print('a')
2. Vscode debugger
   - Breakpoints – koden kör fram hit
   - Du kan stega igenom koden själv
   - Du kan köra pythonkod under tiden, t.ex. för att inspektera en variabel
   - Går att använda i jupyter notebooks!
   - Du kan se all kod i sitt sammanhang
   - Variabler direkt i en lista



ATT TESTA KOD MANUELLT

- När man utvecklar kodbas och lägger till funktionalitet är det viktigt att säkerställa att inga nya buggar tillförts
- Testa manuellt: stoppa in data i ett system/funktion och verifiera att resultatet är rimligt
- Detta kan fungera, men är det effektivt? Oftast inte!
- Att testa kod manuellt fungerar men är sällan effektivt när man utvecklar en applikation

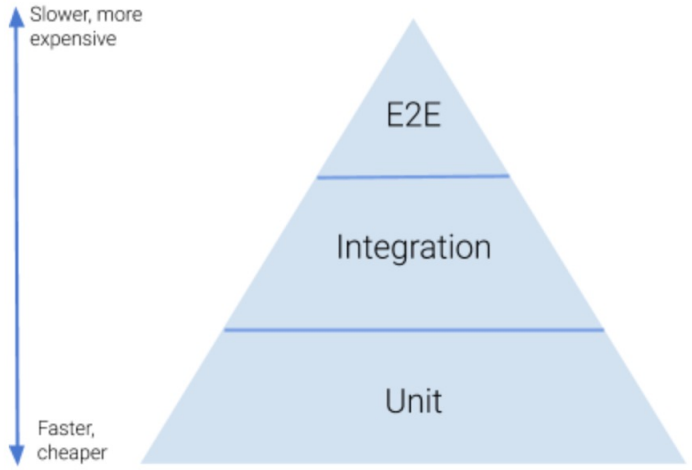


ATT TESTA KOD AUTOMATISKT

- För att testa kod automatiskt måste vi skriva tester som kan köras – alltså mer kod
- Fördelar:
  - Många tester kan köras snabbt och effektivt
  - Vi kan köra tester automatiskt
- Nackdelar:
  - Vi måste definiera tester i kod
  - Dåliga tester (eller buggar) kan ge ett falskt sken av att systemet funkar

UNIT TESTING

- Unit testing är den enklaste och mest använda sortens testning inom mjukvaruutveckling
- Som namnet antyder handlar det om att testa små enheter av koden – enstaka funktioner eller moduler – istället för att testa ett helt system
- För att kunna göra unit tests måste koden vara modulär
  - *Anledningen är att vi vill isolera all funktionalitet*
- Med bra unit testing kan vi automatiskt testa units varje gång vi gör en commit till koden och se om alla komponenter fungerar som det ska



PYTEST

- Pytest är ett välanvänt bibliotek för att göra unit testing i python
- Lätt att använda- för att göra unit tests med pytest skriver vi bara vanliga pythonfunktioner
  - Det lättaste sättet är att använda *assertions*
- Pytest letar automatiskt upp testfunktioner som följer ett standardformat:
  - Filer: *test_*.py* eller **_test.py*
  - Funktioner: vars namn börjar med *test*
- För att gruppera tester kan man även skriva testklasser
- Då man kör pytest kommer det att rapportera alla tester som inte passerar

DATA CAMP

- **Unit Testing for Data Science in Python**
- https://app.datacamp.com/learn/courses/unit-testing-for-data-science-in-python

PIP OCH IMPORTER

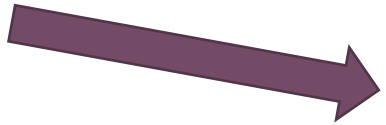
- Mycket i Python bygger på att vara smart och inte uppfinna hjulet igen. Om jag har skrivit en bit kod, till exempel en funktion, vill jag göra allt jag kan för att slippa upprepa den koden på massa fler ställen. Jag vill vara så effektiv som möjligt!

- I alla Python-filer kommer det finnas en massa importer högst upp i filen. Högst upp i filen lägger man importer
- `import argparse`

PIP OCH REQUIREMENTS.TXT

- Pip är ett sätt att lätt installera externa paket/moduler som vi kan använda när vi sedan kodar – vi använder detta mycket! För att slippa skriva pip install X, pip install Y, pip install Z, osv så kan vi definiera en requirements.txt fil och sedan kalla på den genom

  `pip install –r requirements.txt`



numpy

matplotlib

…

- Är likvärdigt med att skriva

  `pip install numpy`

  `pip install matplotlib`

  …

- Vi kan också skapa våra egna moduler!

VAD ÄR EN MODUL?

- När jag har skrivit min funktion som till exempel modifierar 2 strängar (förnamn, efternamn), så vill jag kunna göra så att jag kan använda denna funktion i fler skript. Jag kanske även har flera olika funktioner som alla ändrar representationen av mitt namn på olika sätt.
- För att på ett lätt sätt komma åt dessa funktioner från andra filer får jag skapa en modul – helt enkelt en fil med alla funktioner i. Då slipper jag ha samma kod på flera olika ställen – smart!

  main.py

  string_manipulators.py
- I main.py kan vi skriva
- `import string_manipulators`
- Eller
- `from string_manipulators import change_name_representation`

ATT KOMMENTERA KOD

- Ofta säger vi att vi vill skriva ren kod och att vi vill att kod ska vara väldokumenterad och kommenterad
- Egentligen ska ren kod vara self-explanatory – vi borde inte behöva ha kommentarer om vår kod är solklar
- Ofta blir kommentarer utdaterade – vi skriver kod och kommentarer, någon går in och ändrar i koden men inte i kommentaren och nästa läsare blir förvirrad då inget stämmer överens
- Hitta ett gyllene mellanläge:
  - Använd alltid docstrings (i VSCode finns paket som kan användas). Skriva tre """ och enter
  - När du själv ser att din kod kan vara svår att förstå av t ex en klasskamrat – lägg till en kortfattad kommentar
  - Se till att uppdatera kommentarerna när du ändrar i koden!

```
def main(param1, param2):
    """[summary]

    Args:
        param1 ([type]): [description]
        param2 ([type]): [description]
    """
```

KÖRA .PY FILER GENOM TERMINAL

- För att köra kod från kommandotolken öppnar man först en ny terminal i VSC
- Sedan ställer man sig i samma mapp som python-filen man ska köra (använd cd och ls för att navigera och se till man är på korrekt ställe)
- Man skriver,
  python main.py
- Detta kör python scriptet
- OBS!! En del datorer kan ha python version 2.x kopplat till kommandot python i terminalen och vi måste skriva python3 main.py
- Kolla genom att skriva python --version och python3 --version för att se om versionen är samma som den ni har i ert environment.
- Lättaste sättet att kolla detta är i VSC nedre vänster hörn se vilken version som används