

3. Introduktion till Python

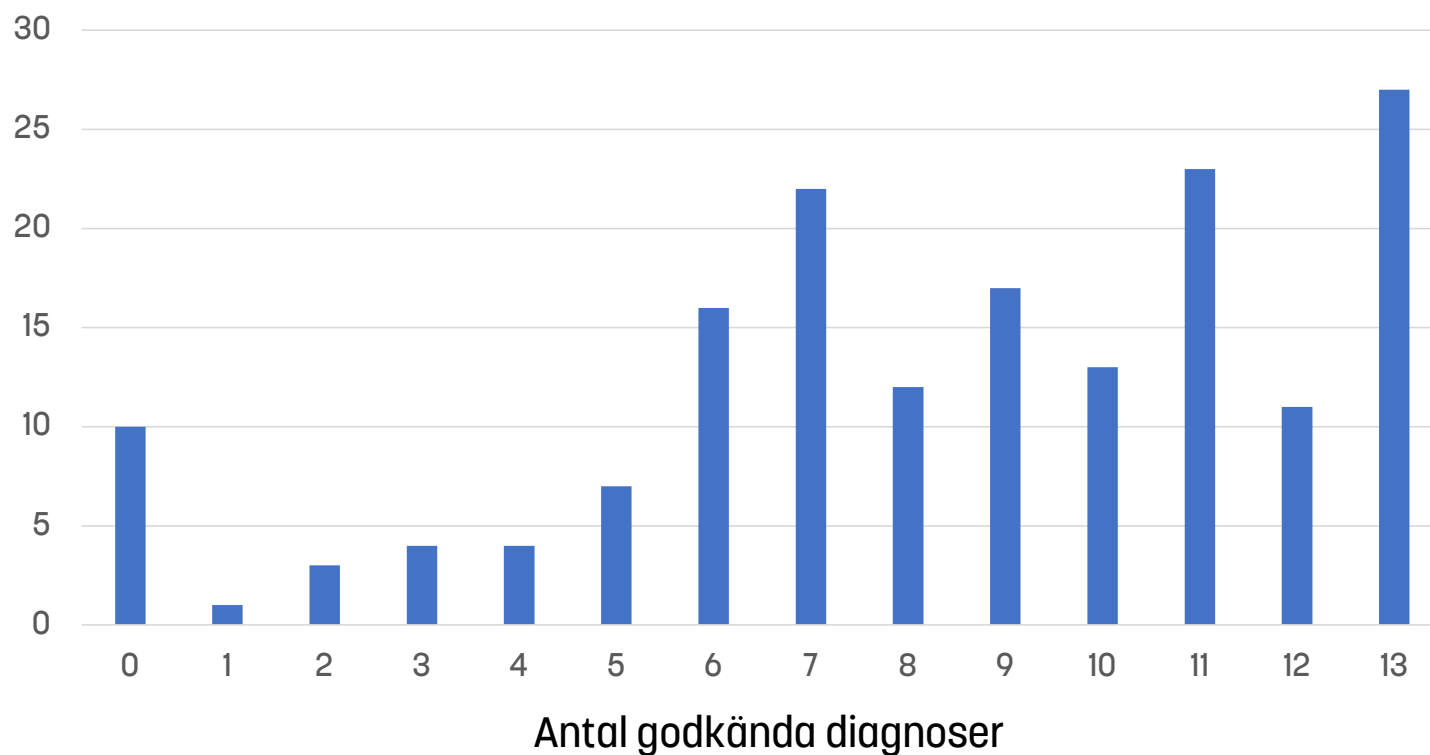
TDDE23 Funktionell och imperativ programmering, del 1

Peter Dalenius

Institutionen för datavetenskap (IDA)

2024-09-02

Resultat på datorintroduktionen



I torsdags hade 74%
klarat minst 7
diagnoser vilket är
jättebra!

Översikt



Varför har vi valt Python?



Hur kommer vi arbeta i kursen?



Vårt första Python-program

Varför har vi valt Python?

Pythons historik

- Skapades i slutet av 1980-talet av holländaren Guido van Rossum, som fortfarande spelar en stor roll för språkets utveckling.
- Det mest populära programspråket i världen just nu. [1]
- Åtta av tio toppuniversitet i USA använder Python som förstaspråk. [2]
- Version 3, som vi kommer använda, kom i december 2008. Det var en ganska stor förändring som inte är bakåtkompatibel.
- Fri programvara.

Fördelar med Python



Enklare syntax än
t.ex. Java och C++



Bred och stabil
användarbas



Kan användas
till det mesta



Stödjer flera
paradigm



Bra för att illustrera
koncept inom
programmering

Några roliga saker vi inte hinner med

- Grafiska gränssnitt
- Kommunikation över nätverk
- Databaser
- Trådar och processer
- Stora projekt
- Appar

Vi kommer fokusera på grunderna i programmering och hur man etablerar goda vanor för att bli en bra programmerare.
Allt det andra – och mycket mer – kommer längre fram.

Hur kommer vi arbeta i kursen?

Hur ska vi jobba i kursen under HT1?

Aktivitet	Tid	Innehåll
Föreläsningar	3 x 2 h	<i>Introduktion, översikt, sammanfattning</i>
Seminarier	6 x 2 h	<i>Genomgångar, förberedelser, exempel, torrövningar</i>
Laborationer	14 x 2 h	<i>Arbete med övningar och uppgifter, diskussioner, redovisningar</i>
Eget arbete	~ 100 h	<i>Läsa, öva, experimentera, arbeta med övningar och uppgifter</i>

1 termin = 20 veckor (x 40 timmar) = 30 hp

Seminarier

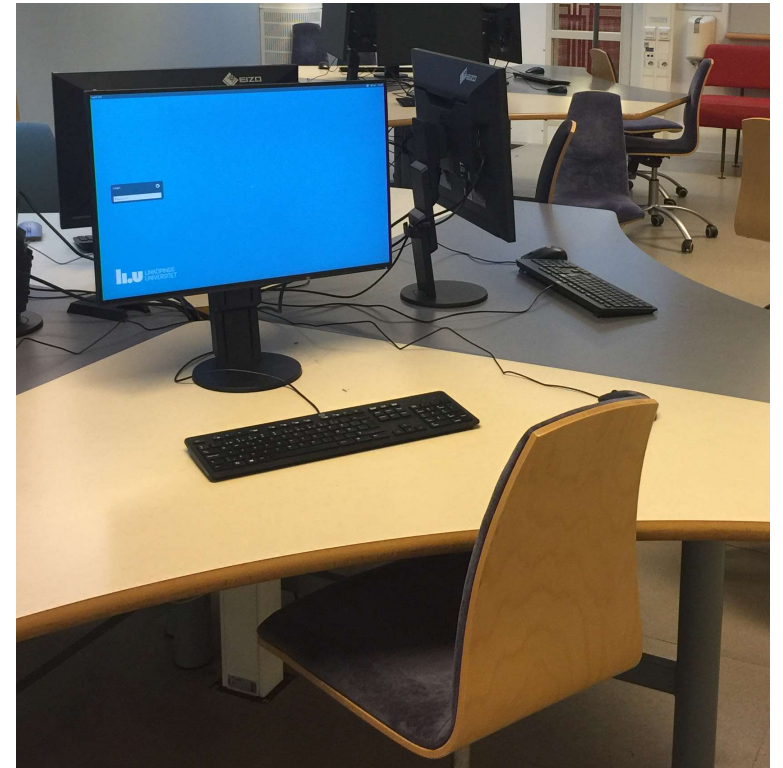
- Ett seminarium per vecka, från och med nästa vecka
- Samma innehåll i alla fem grupper, men tre olika inriktningar:
 - **En sak i taget** - För dig som vill ta det steg för steg och känna att du förstår en sak innan du går vidare till nästa.
 - **Experimentering** - För dig som vill lära dig genom att testa dig fram och gärna prövar olika lösningar på medhavd dator.
 - **Diskussion och analys** - För dig som vill diskutera och analysera utifrån det du redan kan och det du kommer att lära dig och som inte är rädd för okända begrepp eller att gå lite utanför kursens ramar.

Seminarier

- Välj själv vilken inriktning du vill gå på. Det är också okej att byta inriktning mellan gångerna, om det inte verkar passa.
- Alla seminariegrupper ligger oftast parallellt i schemat:
 - En sak i taget (S4, S5) *Linneá Lindblom, Therese Selberg*
 - Experimentering (S2, S3) *Mattias Karlsson, Christopher Claesson*
 - Diskussion och analys (S1) *Morgan Nordberg*
- Seminariematerial och mer information finns på kurswebben.

Laborationer

- En labbmängd per vecka som genomförs och redovisas individuellt.
- De tre sista veckorna har vi en större labb, uppdelad i tre delar.
- Det finns två labbpass per vecka med tillgång till assistenter. Ni har blivit indelade i grupper (se webben) där varje grupp har olika pass i schemat.
- Utöver detta förväntas man labba på egen hand, antingen i våra salar eller på egen dator.



Programmeringsstugor

- Extra tillfälle att få hjälp med programmering
- Varje vecka, från och med nästa vecka
- Kl 17-19 i någon labbsal, olika veckodagar
- Labbhandledare närvarande
- Ej redovisning av labbar, bara hjälp och stöd



Förändringar i kursen inför hösten 2024

- Lagt till en föreläsning om git
- Lagt till en föreläsning om parprogrammering

Tidigare års kursvärderingar, kommentarer till dessa samt information om åtgärder finns i kursrummet i Lisam.

År	Helhetsbetyg	Svarsfrekvens
2023	3,97	21%
2022	3,70	33%
2021	3,76	34%
2020	4,05	44%
2019	3,94	29%
2018	4,44	34%

Hur vet jag vilken grupp jag är med i?

1. Logga in i Webreg på <https://www.ida.liu.se/webreg> eller följ länkar från kurswebben

2. Klicka på kursens namn

Student

2022

[TDDE23 - Functional and Imperative Programming, Part 1](#)

3. Klicka på aktiviteten *LAB1*

Course staff

LAB1

4. Klicka på fliken *Signup*

Course staff

LAB1

LAB1 - Laboration 1-5

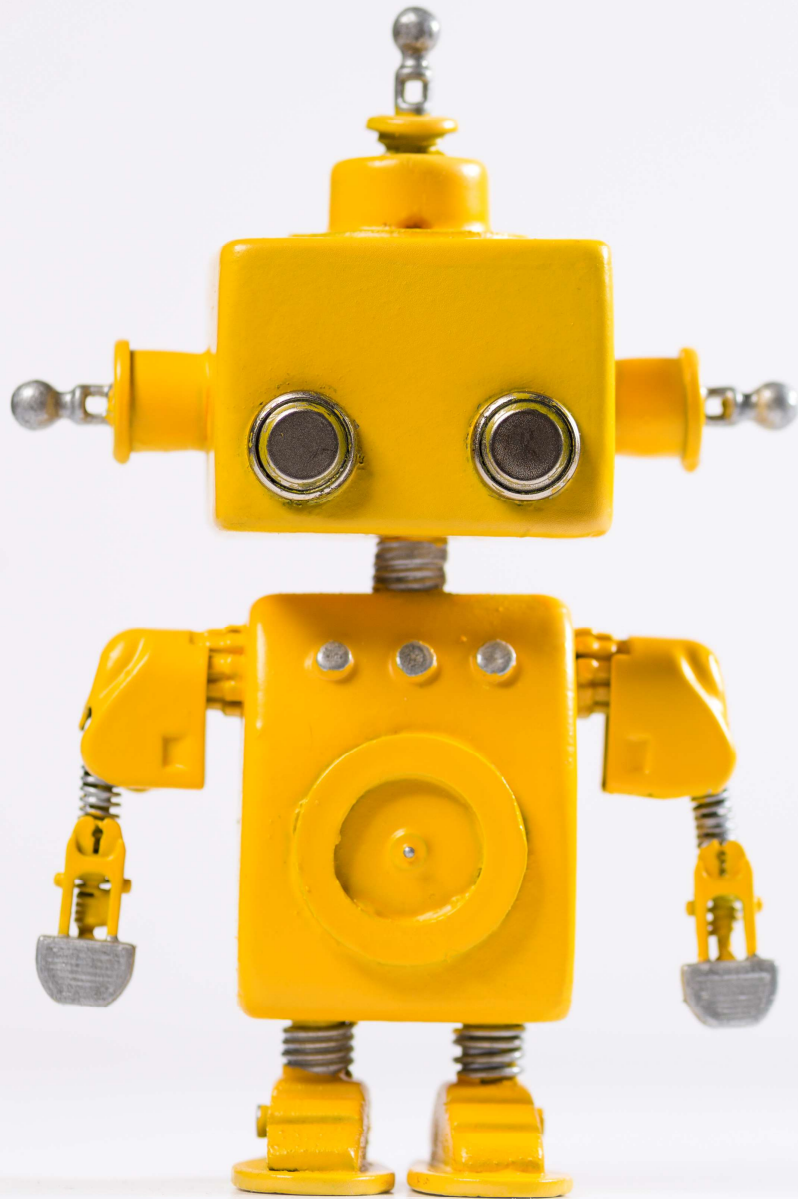
Sök på sidan efter ditt
namn (visas med orange
bakgrund)

Sign up

Results

IDA:s regler för laborationer

- **Det är inte tillåtet** att lämna in lösningar som har kopierats från andra studenter, eller från annat håll, även om modifieringar har gjorts. Om otillåten kopiering eller annan form av fusk misstänks, är läraren skyldig att göra en anmälan till universitetets disciplinnämnd.
- **Du ska kunna** redogöra för detaljer i koden för ett program. Det kan också tänkas att du får förklara varför du har valt en viss lösning. Detta gäller alla i en grupp.
- **Om du förutser** att du inte hinner redovisa i tid, ska du kontakta din lärare. Då kan du få stöd och hjälp och eventuellt kan tidpunkten för redovisningen senareläggas. Det är alltid bättre att diskutera problem än att, t.ex., fuska.
- **Om du inte följer** universitetets och en kurs examinationsregler, utan försöker fuska, t.ex. plagiera eller använda otillåtna hjälpmedel, kan detta resultera i en anmälan till universitetets disciplinnämnd. Konsekvenserna av ett beslut om fusk kan bli varning eller avstängning från studierna.

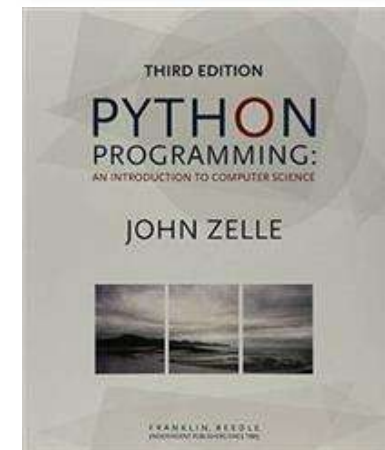
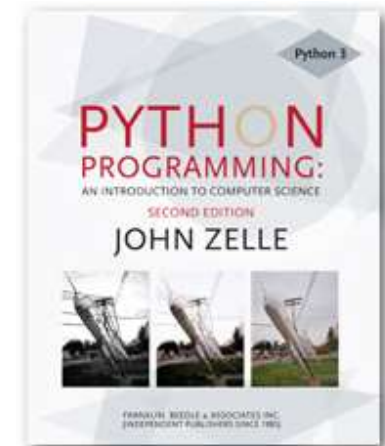


Om generativa AI-tjänster

- Det är inte tillåtet att låta en annan person eller AI-tjänst göra stora delar av labben åt dig.
- Om du vill, använd gärna generativa AI-tjänster för att ställa frågor om programmering, eller som stöd för delar av övningar. Men ännu hellre, fråga en klasskamrat eller din labbassistent.
- Du examineras bara delvis på den inlämnade koden. Viktigast är att du kan svara på assistentens frågor vid den muntliga redovisningen.

Kurslitteratur (absolut inte obligatorisk)

- John M. Zelle (2010) *Python Programming: An Introduction to Computer Science. Second Edition*. ISBN 978-1-59028-241-0
- John M. Zelle (2016) *Python Programming: An Introduction to Computer Science. Third Edition*. ISBN 978-1-59028-275-5
- Boken har ett bra pedagogiskt upplägg som ansluter väl till det vi vill göra i kursen.
- Den täcker dock inte allt som vi vill gå igenom, så en del saker kommer enbart att tas upp på seminarier.
- Om man skaffar en annan Python-bok, se till att den täcker Python 3!



Personal

- Examiner: Peter Dalenius
- Kursadministratör: Anna Grabska Eklund
- Laborationsassistenter:
 - D1A: Kevin Magron, Noah Ljungberg
 - D1B: Linnéa Lindblom, Hampuz Togerez
 - D1C: Christopher Claesson, Younes Slibi
 - U1A: Bo Forsberg, Mattias Karlsson
 - U1B: Morgan Nordberg, Alice Lindholm

Information och kommunikation

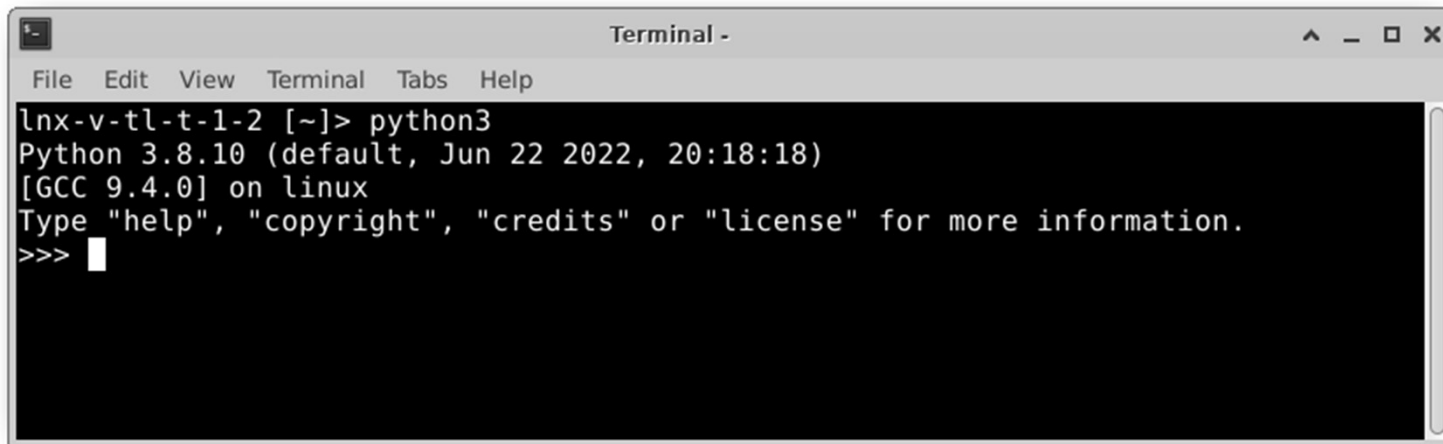
- På kurswebben finns all information om kursen man behöver – *studiematerial, material från föreläsningar och seminarier, instruktioner för laborationer, planering, deadlines, schema, regler för redovisning*

<http://www.ida.liu.se/~TDDE23>

- Vi kommer skicka ut kompletterande information till studentmejl, som ni förväntas läsa varje dag

Vårt första Python-program

Hur man startar Python-interpretatorn

A screenshot of a terminal window titled "Terminal -". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal content shows the command `python3` being executed in a shell with the prompt `lnx-v-tl-t-1-2 [~]>`. The output displays the Python version `Python 3.8.10 (default, Jun 22 2022, 20:18:18)`, the compiler `[GCC 9.4.0] on linux`, and instructions to type `"help"`, `"copyright"`, `"credits"`, or `"license"` for more information. The prompt `>>>` is shown with a cursor, indicating the Python interpreter is ready for input.

```
lnx-v-tl-t-1-2 [~]> python3
Python 3.8.10 (default, Jun 22 2022, 20:18:18)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

Öppna en terminal och kör kommandot **python3**.
När du ser prompten `>>>` är Python redo att ta emot kommandon.

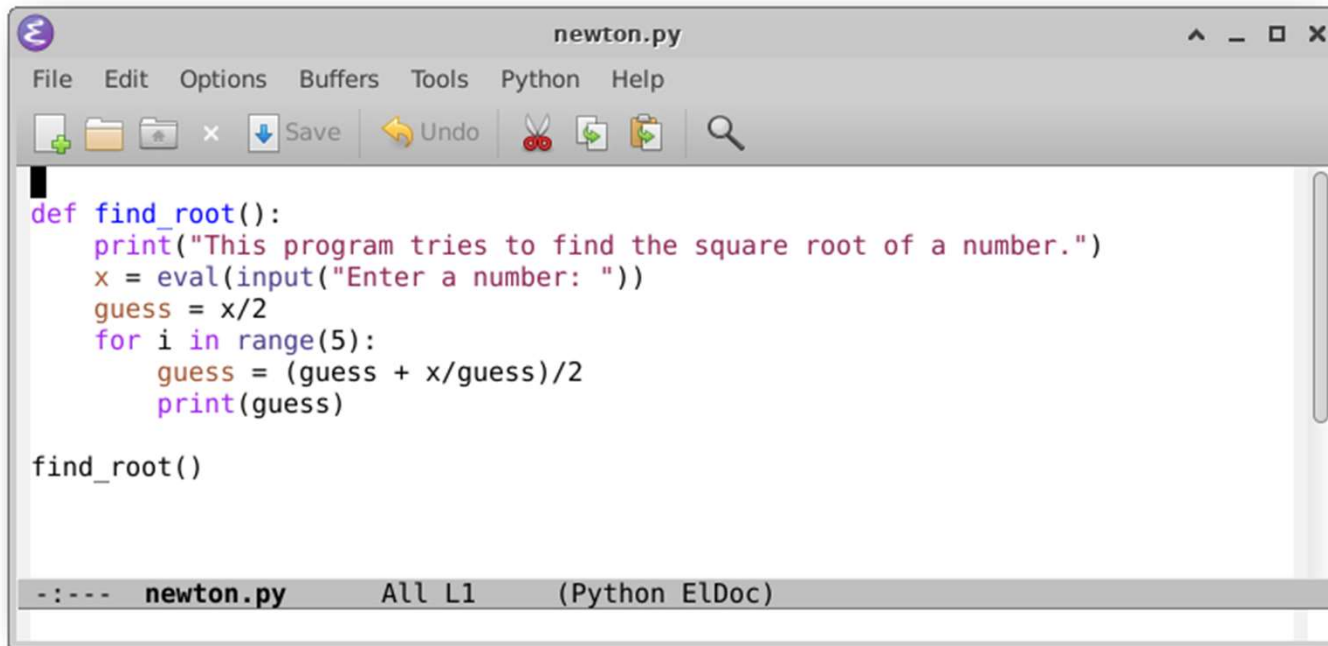
Hur man använder Python-interpretatorn

```
>>> 2+3
5
>>> 2+4*5-6
16
>>> print("Hello world!")
Hello world!
>>> def greeting():
...     print("Nobody expects the Spanish Inquisition.")
...     print("Our chief weapon is surprise... and fear.")
...
>>> greeting()
Nobody expects the Spanish Inquisition.
Our chief weapon is surprise... and fear.
>>>
```

Hur man använder Python-interpretatorn

```
>>> def skryt(namn):  
...     print("Det är bra att kunna Python.")  
...     print(namn, "är jättebra på Python.")  
...  
>>> skryt("Peter")  
Det är bra att kunna Python.  
Peter är jättebra på Python.  
>>> skryt("Jenny")  
Det är bra att kunna Python.  
Jenny är jättebra på Python.  
>>> skryt  
<function skryt at 0x218390>  
>>> print  
<built-in function print>  
>>>
```

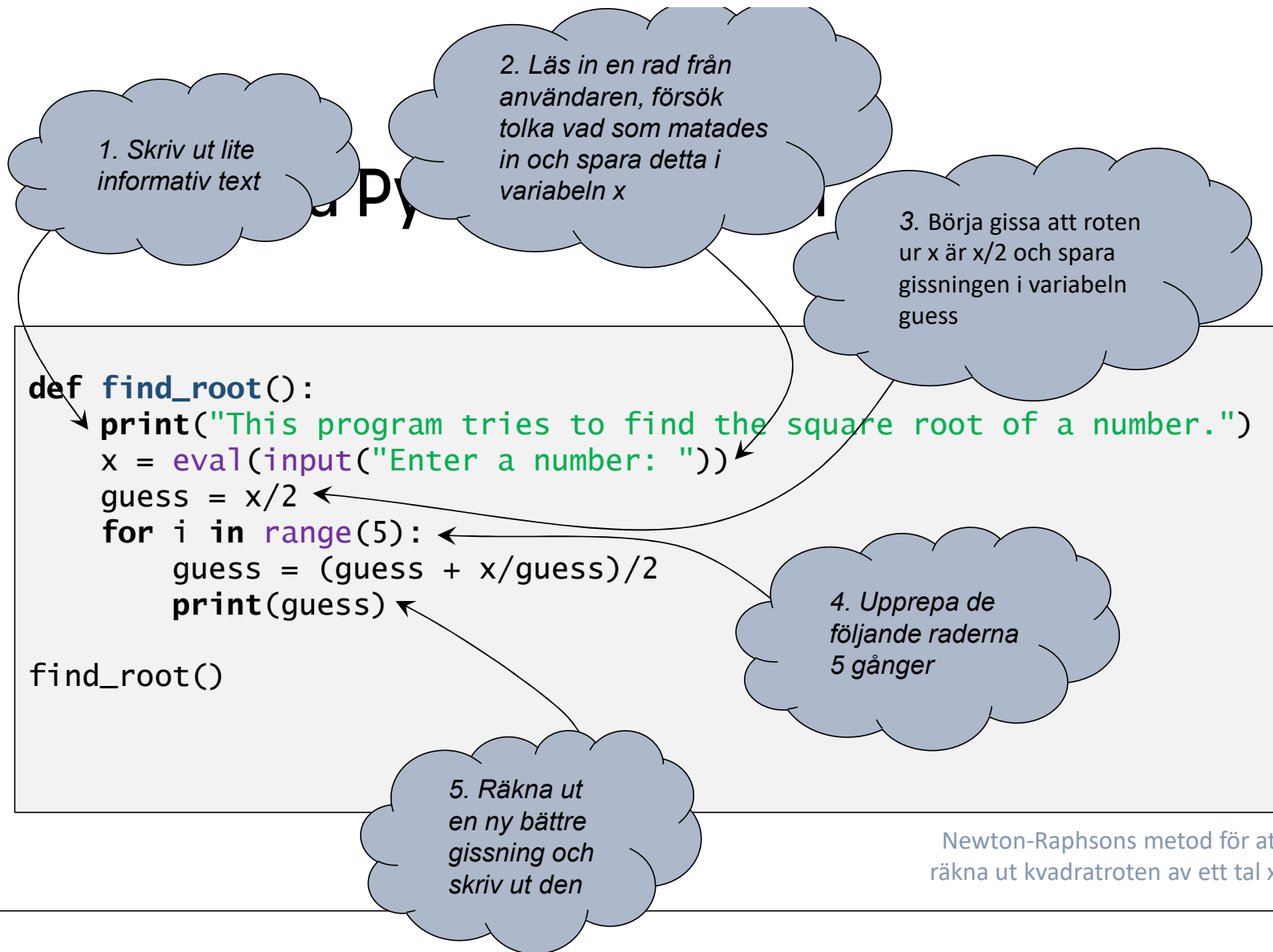
Vårt första Python-program



```
def find_root():
    print("This program tries to find the square root of a number.")
    x = eval(input("Enter a number: "))
    guess = x/2
    for i in range(5):
        guess = (guess + x/guess)/2
        print(guess)

find_root()
```

The screenshot shows a Python IDE window titled "newton.py". The menu bar includes File, Edit, Options, Buffers, Tools, Python, and Help. The toolbar contains icons for file operations (new, open, save, save all), editing (undo, redo, cut, copy, paste), and search. The code editor displays a Python function `find_root()` that prompts the user for a number, calculates its square root using the Newton-Raphson method for five iterations, and prints the result. The status bar at the bottom shows the file name "newton.py", the encoding "All L1", and the interpreter "Python ElDoc".



Varför så mycket mellanrum

```
def find_root():  
    print("This program tries to find the square root of a number.")  
    x = eval(input("Enter a number: "))  
    guess = x/2  
    for i in range(5):  
        guess = (guess + x/guess)/2  
        print(guess)
```

find_root()

*Underordnade
rader måste skjutas
in en bit på raden,
indenteras.*

Newton-Raphsons metod för att
räkna ut kvadratroten av ett tal x.

Olika sätt att köra vårt program

```
petjo96@chipolata2:python$ python3
Python 3.4.3 (default, Nov 17 2016, 01:08:31)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import newton
This program tries to find the square root of a number.
Enter a number: 2
1.5
1.4166666666666665
1.4142156862745097
1.4142135623746899
1.414213562373095
>>>
```

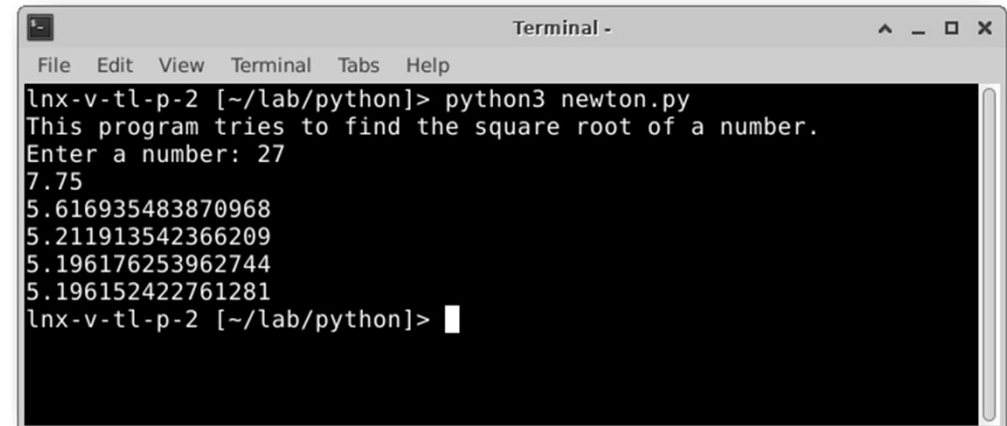
Olika sätt att köra vårt program

```
>>> newton.find_root()
This program tries to find the square root of a number.
Enter a number: 10
3.5
3.178571428571429
3.162319422150883
3.1622776604441363
3.162277660168379
>>> from newton import *
>>> find_root()
This program tries to find the square root of a number.
Enter a number: 3
1.75
1.7321428571428572
1.7320508100147274
1.7320508075688772
1.7320508075688772
>>> exit()
petjo96@chipolata2:python$
```

Olika alternativ för att testköra program

Kör hela programmet från terminalen

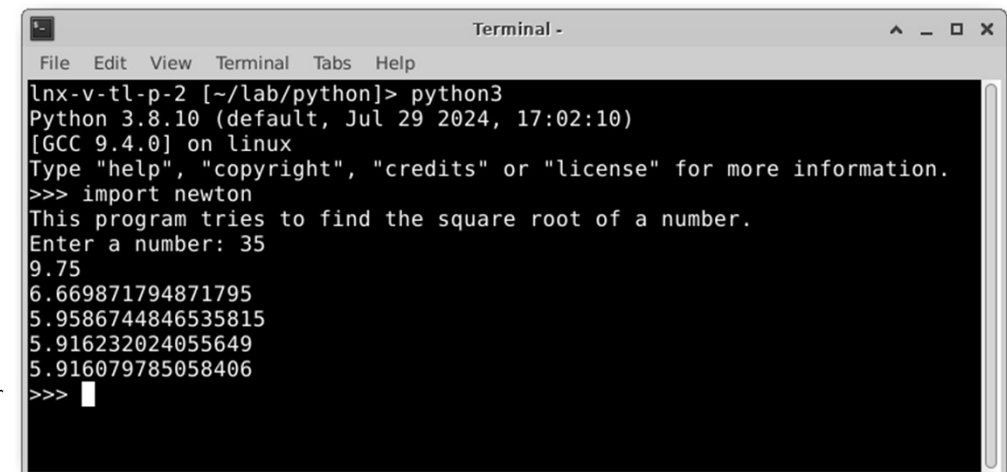
- Måste ha med anrop till huvudfunktion
- Svårare att felsöka vidare



```
lnx-v-tl-p-2 [~/lab/python]> python3 newton.py
This program tries to find the square root of a number.
Enter a number: 27
7.75
5.616935483870968
5.211913542366209
5.196176253962744
5.196152422761281
lnx-v-tl-p-2 [~/lab/python]> 
```

Importera programmet till Python-interpretatorn

- Enkelt att testa på flera olika sätt
- Svårt att ladda om ändringar i programmet



```
lnx-v-tl-p-2 [~/lab/python]> python3
Python 3.8.10 (default, Jul 29 2024, 17:02:10)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import newton
This program tries to find the square root of a number.
Enter a number: 35
9.75
6.669871794871795
5.9586744846535815
5.916232024055649
5.916079785058406
>>> 
```

Ordning och reda på programkoden

Programkod organiseras i *moduler*, och om man inte gör något speciellt kommer varje fil att bli en modul.

import newton

Ladda in modulen (filen) newton som finns i filen newton.py. Funktioner i modulen går att anropa med newton.funktionsnamn

from newton import *

Ladda in modulen (filen) newton så att allt som finns i den går att anropa direkt. Detta är bra för våra små program, men blir rörigt när vi kommer lite längre fram.

Satser – ryggraden i ett program

Ett Python-program består av ett antal *satser* (*eng. statements*) som utförs i tur och ordning. Vi ska titta på de fem viktigaste, där vi redan sett de fyra första:

- Enkla satser
 - *Tilldelning* (*eng. assignment*)
 - *Funktionsanrop*
- Sammansatta satser
 - *Iteration* (*upprepning*) med *for*
 - *Funktionsdefinition*
 - *Selektion* (*val*) med *if*

Olika typer av satser: Tilldelning

Hur ser det ut? namn = uttryck

Vad händer? Uttrycket beräknas och resultatet sparas i den namngivna variabeln.

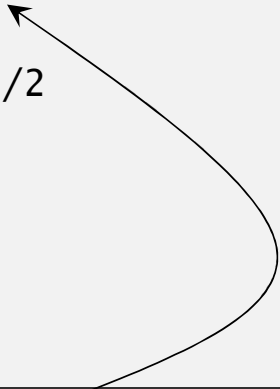
```
def find_root():  
    print("This program tries to find the square root of a number.")  
    x = eval(input("Enter a number: "))  
    guess = x/2  
    for i in range(5):  
        guess = (guess + x/guess)/2  
        print(guess)  
  
find_root()
```

Olika typer av satser: Funktionsanrop

Hur ser det ut? funktionsnamn(frivilliga argument)

Vad händer? Argumenten skickas till funktionen, underprogrammet, för behandling.

```
def find_root():  
    print("This program tries to find the square root of a number.")  
    x = eval(input("Enter a number: "))  
    guess = x/2  
    for i in range(5):  
        guess = (guess + x/guess)/2  
        print(guess)  
  
find_root()
```



Funktionsanrop som inte är självständiga satser

Olika typer av satser: Iteration med for

Hur ser det ut? **for** namn **in** område: undersatser

Vad händer? Undersatserna körs så många gånger som området specificerar.

```
def find_root():  
    print("This program tries to find the square root of a number.")  
    x = eval(input("Enter a number: "))  
    guess = x/2  
    for i in range(5):  
        guess = (guess + x/guess)/2  
        print(guess)  
  
find_root()
```

Olika typer av satser: Funktionsdefinition

Hur ser det ut? **def** funktionsamn(frivilliga argument): undersatser
Vad händer? Undersatserna sparas för att kunna anropas senare.

```
def find_root():  
    print("This program tries to find the square root of a number.")  
    x = eval(input("Enter a number: "))  
    guess = x/2  
    for i in range(5):  
        guess = (guess + x/guess)/2  
        print(guess)
```

```
find_root()
```

Olika typer av satser: Selektion med if

Enkelt val

```
if a > 0:  
    print("positivt")
```

Två alternativ

```
if a > 0:  
    print("positivt")  
else:  
    print("negativt")
```

Tre eller fler alternativ

```
if 0 <= a < 10:  
    print("ental")  
elif a < 0:  
    print("negativt")  
else:  
    print("tio eller mer")
```

Lycka till!