DD1331

Grundläggande programmering för F1

Några bilder till föreläsning 4

Innehåll

Innehåll

- Funktioner

Innehåll

- Funktioner
  - Huvud, kropp

Innehåll

- Funktioner
  - Huvud, kropp
  - Parametrar, lokala variabler

Innehåll

- Funktioner
  - Huvud, kropp
  - Parametrar, lokala variabler
  - Globala variabler

Innehåll

- Funktioner
  - Huvud, kropp
  - Parametrar, lokala variabler
  - Globala variabler
- Objekt

Innehåll

- Funktioner
  - Huvud, kropp
  - Parametrar, lokala variabler
  - Globala variabler
- Objekt
  - Referenser till objekt

Innehåll

- Funktioner
  - Huvud, kropp
  - Parametrar, lokala variabler
  - Globala variabler
- Objekt
  - Referenser till objekt
  - Lika objekt $\Leftrightarrow$ Samma objekt ?

Innehåll

- Funktioner
  - Huvud, kropp
  - Parametrar, lokala variabler
  - Globala variabler
- Objekt
  - Referenser till objekt
  - Lika objekt $\Leftrightarrow$ Samma objekt ?
  - Funktioner "på" objekt

Innehåll

- Funktioner
  - Huvud, kropp
  - Parametrar, lokala variabler
  - Globala variabler
- Objekt
  - Referenser till objekt
  - Lika objekt $\Leftrightarrow$ Samma objekt ?
  - Funktioner "på" objekt
- Datastrukturer

Innehåll

- Funktioner
  - Huvud, kropp
  - Parametrar, lokala variabler
  - Globala variabler
- Objekt
  - Referenser till objekt
  - Lika objekt $\Leftrightarrow$ Samma objekt ?
  - Funktioner "på" objekt
- Datastrukturer
  - Sekvenser

Innehåll

- Funktioner
  - Huvud, kropp
  - Parametrar, lokala variabler
  - Globala variabler
- Objekt
  - Referenser till objekt
  - Lika objekt $\Leftrightarrow$ Samma objekt ?
  - Funktioner "på" objekt
- Datastrukturer
  - Sekvenser
    - Textsträngar '.....' ".....” (lite mer)

Innehåll

- Funktioner
  - Huvud, kropp
  - Parametrar, lokala variabler
  - Globala variabler
- Objekt
  - Referenser till objekt
  - Lika objekt $\Leftrightarrow$ Samma objekt ?
  - Funktioner "på" objekt
- Datastrukturer
  - Sekvenser
    - Textsträngar '.....' ".....” (lite mer)
    - Tupler (.....)

Innehåll

- Funktioner
  - Huvud, kropp
  - Parametrar, lokala variabler
  - Globala variabler
- Objekt
  - Referenser till objekt
  - Lika objekt $\Leftrightarrow$ Samma objekt ?
  - Funktioner "på" objekt
- Datastrukturer
  - Sekvenser
    - Textsträngar '.....' ".....” (lite mer)
    - Tupler (.....)
    - Listor [.....]

Innehåll

- Funktioner
  - Huvud, kropp
  - Parametrar, lokala variabler
  - Globala variabler
- Objekt
  - Referenser till objekt
  - Lika objekt $\Leftrightarrow$ Samma objekt ?
  - Funktioner "på" objekt
- Datastrukturer
  - Sekvenser
    - Textsträngar '.....' ".....” (lite mer)
    - Tupler (.....)
    - Listor [.....]
    - range(...)

Innehåll

- Funktioner
  - Huvud, kropp
  - Parametrar, lokala variabler
  - Globala variabler
- Objekt
  - Referenser till objekt
  - Lika objekt $\Leftrightarrow$ Samma objekt ?
  - Funktioner "på" objekt
- Datastrukturer
  - Sekvenser
    - Textsträngar '.....' ".....” (lite mer)
    - Tupler (.....)
    - Listor [.....]
    - range(...)
  - Dictionaries (Uppslagslistor) {.....}

Innehåll

- Funktioner
  - Huvud, kropp
  - Parametrar, lokala variabler
  - Globala variabler
- Objekt
  - Referenser till objekt
  - Lika objekt $\Leftrightarrow$ Samma objekt ?
  - Funktioner "på" objekt
- Datastrukturer
  - Sekvenser
    - Textsträngar '.....' ".....” (lite mer)
    - Tupler (.....)
    - Listor [.....]
    - range(...)
  - Dictionaries (Uppslagslistor) {.....}
  - Sets (Mängder)

Funktioner har **huvud** och **kropp**

Funktioner har **huvud** och **kropp**

```
def countdown(n):
```

huvud

Funktioner har **huvud** och **kropp**

```
def countdown(n):                    huvud
    t = n                               kropp
    while t > 0:                           :
        print(t)                           :
        time.sleep(1)                      :
        t = t-1                            :
    print("0 and GOOOOO")                kropp
```

Funktioner har **huvud** och **kropp**

```
def countdown(n):                 huvud
    t = n                            kropp
    while t > 0:                        :
        print(t)                        :
        time.sleep(1)                   :
        t = t-1                         :
    print("0 and GOOOOO")              kropp
```

n är funktionens **formella parameter**

Funktioner har **huvud** och **kropp**

```
def countdown(n):                 huvud
    t = n                            kropp
    while t > 0:                        :
        print(t)                        :
        time.sleep(1)                   :
        t = t-1                         :
    print("0 and GOOOOO")              kropp
```

n är funktionens **formella parameter**

Namnet **n** har inget samband med namn utanför funktionen

Funktioner har **huvud** och **kropp**

```
def countdown(n):                 huvud
    t = n                            kropp
    while t > 0:                        :
        print(t)                        :
        time.sleep(1)                   :
        t = t-1                         :
    print("0 and GOOOOO")              kropp
```

n är funktionens **formella parameter**
Namnet **n** har inget samband med namn utanför funktionen

t är en **lokal variabel**, funktionens egen

Funktioner har **huvud** och **kropp**

```
def countdown(n):                 huvud
    t = n                            kropp
    while t > 0:                        :
        print(t)                        :
        time.sleep(1)                   :
        t = t-1                         :
    print("0 and GOOOOO")              kropp
```

n är funktionens **formella parameter**
Namnet **n** har inget samband med namn utanför funktionen

t är en **lokal variabel**, funktionens egen
t måste tilldelas startvärde i funktionen

Funktioner har **huvud** och **kropp**

```
def countdown(n):                 huvud
    t = n                            kropp
    while t > 0:                        :
        print(t)                        :
        time.sleep(1)                   :
        t = t-1                         :
    print("0 and GOOOOO")              kropp
```

n är funktionens **formella parameter**
Namnet **n** har inget samband med namn utanför funktionen

t är en **lokal variabel**, funktionens egen
t måste tilldelas startvärde i funktionen
Namnet **t** har inget samband med namn utanför funktionen

Funktioner **anropas**

Funktioner **anropas**

Då anges **aktuell parameter**

Funktioner **anropas**

Då anges **aktuell parameter**

`countdown(5)` **anrop** **aktuell parameter = 5**

Funktioner **anropas**

Då anges **aktuell parameter**

```
countdown(5)
```
anrop **aktuell parameter = 5**

```
x = 2
y = 3
countdown(y*y-x*x+1)
```
anrop **aktuell parameter = 6**

Funktioner **anropas**

Då anges **aktuell parameter**

```
countdown(5)
```
anrop **aktuell parameter = 5**

```
x = 2
y = 3
countdown(y*y-x*x+1)
```
anrop **aktuell parameter = 6**

```
n = 17
countdown(n)
```
anrop **aktuell parameter = 17**

Globala variabler

Globala variabler

Definieras utanför funktionerna

Globala variabler

Definieras utanför funktionerna

Kan användas i (flera) funktioner

Globala variabler

Definieras utanför funktionerna

Kan användas i (flera) funktioner

Får (bör) inte ändras

Globala variabler

Definieras utanför funktionerna

Kan användas i (flera) funktioner

Får (bör) inte ändras

```
vowels = 'aoueiy'
```

vowels är **global variabel**

Globala variabler

Definieras utanför funktionerna

Kan användas i (flera) funktioner

Får (bör) inte ändras

```
vowels = 'aoueiy'
```

vowels är **global variabel**

```
def letteranalysis(atext):
    :
    for c in atext:
        if c in vowels:
            :
```

vowels används, ändras ej

OK!

Globala variabler

Definieras utanför funktionerna

Kan användas i (flera) funktioner

Får (bör) inte ändras

```
vowels = 'aoueiy'
```

vowels är **global variabel**

```
def letteranalysis(atext):
    :
    for c in atext:
        if c in vowels:
            :
```

vowels används, ändras ej

OK!

```
def swedishlettercrunching(atext):
        :
    vowels = vowels + 'åäö'
        :
```

NEJ! Farligt!

Otillåtet att ändra vowel

Data i Python är objekt

Data i Python är objekt

- Objekt har **typ** och **värde(n)**

Data i Python är objekt

- Objekt har **typ** och **värde(n)**
- **Mutable:** **typer** som kan ändra värde

Data i Python är objekt

- Objekt har **typ** och **värde(n)**
- **Mutable:** **typer** som kan ändra värde
- **Immutable:** **typer** som inte kan ändra värde.

Data i Python är objekt

- Objekt har **typ** och **värde(n)**
- **Mutable: typer** som kan ändra värde
- **Immutable: typer** som inte kan ändra värde.
- Namn/variabler **refererar** till objekt

Data i Python är objekt

- Objekt har **typ** och **värde(n)**
- **Mutable:** **typer** som kan ändra värde
- **Immutable:** **typer** som inte kan ändra värde.
- Namn/variabler **refererar** till objekt
- Att jämföra objekt

Data i Python är objekt

- Objekt har **typ** och **värde(n)**
- **Mutable:** **typer** som kan ändra värde
- **Immutable:** **typer** som inte kan ändra värde.
- Namn/variabler **refererar** till objekt
- Att jämföra objekt
  - **x == y** **True om** likadana objekt (lika värde)

Data i Python är objekt

- Objekt har **typ** och **värde(n)**
- **Mutable:** **typer** som kan ändra värde
- **Immutable:** **typer** som inte kan ändra värde.
- Namn/variabler **refererar** till objekt
- Att jämföra objekt
  - **x == y** **True om** likadana objekt (lika värde)
  - **x is y** **True om** samma objekt

Data i Python är objekt

- Objekt har **typ** och **värde(n)**
- **Mutable:** **typer** som kan ändra värde
- **Immutable:** **typer** som inte kan ändra värde.
- Namn/variabler **refererar** till objekt
- Att jämföra objekt
  - **x == y** **True om** likadana objekt (lika värde)
  - **x is y** **True om** samma objekt
- Funktioner kan anropas **på** objekt:

Data i Python är objekt

- Objekt har **typ** och **värde(n)**
- **Mutable:** **typer** som kan ändra värde
- **Immutable:** **typer** som inte kan ändra värde.
- Namn/variabler **refererar** till objekt
- Att jämföra objekt
  - **x == y** **True om** likadana objekt (lika värde)
  - **x is y** **True om** samma objekt
- Funktioner kan anropas **på** objekt:

obj.f() **x.m()**

Sekvenser

Sekvenser

Datatyper för ordnade mängder värden

Sekvenser

Datatyper för ordnade mängder värden

- Textsträngar,

  `str` `"..."` `'...'` Immutable

Sekvenser

Datatyper för ordnade mängder värden

- Textsträngar,

  `str` `"..."` `'...'` Immutable

- Tupler,

  `tuple` `(...)` Immutable

Sekvenser

Datatyper för ordnade mängder värden

- Textsträngar,

  `str` `"..."` `'...'` Immutable

- Tupler,

  `tuple` `(...)` Immutable

- Listor,

  `list` `[...]` **Mutable**

Sekvenser

Datatyper för ordnade mängder värden

- Textsträngar,

  `str` `"..."` `'...'` Immutable

- Tupler,

  `tuple` `(...)` Immutable

- Listor,

  `list` `[...]` **Mutable**

- Heltalsföljder,

  `range` `(...)` Immutable

Alla sekvenser har

Alla sekvenser har

- Längd

  `len(sekvens)` $\rightarrow$ antalet element

Alla sekvenser har

- Längd

  `len(sekvens)` $\rightarrow$ antalet element

- Repetition med for

  `for elem in sekvens:`

Alla sekvenser har

- Längd

  `len(sekvens)` $\rightarrow$ antalet element

- Repetition med for

  `for elem in sekvens:`

  går igenom alla element i sekvensen

Alla sekvenser har

- Längd

  `len(sekvens)` $\rightarrow$ antalet element

- Repetition med for

  `for elem in sekvens:`

  går igenom alla element i sekvensen

- Test av medlem,

  `elem in sekvens`

Alla sekvenser har

- Längd

  `len(sekvens)` $\rightarrow$ antalet element

- Repetition med for

  `for elem in sekvens:`

  går igenom alla element i sekvensen

- Test av medlem,

  `elem in sekvens`

  ger True om elem finns med i sekvensen

Några str – funktioner

Några str – funktioner

Exempelsträngar:

```
färg = "TurKos"
mening = "Gå och bada"
åsna = "      Ior   "
```

Några str – funktioner

Exempelsträngar:

```
färg = "TurKos"
mening = "Gå och bada"
åsna = "      Ior   "
```

upper `färg.upper()` $\rightarrow$ `'TURKOS'`

Några str – funktioner

Exempelsträngar:

```
färg = "TurKos"
mening = "Gå och bada"
åsna = "      Ior   "
```

upper `färg.upper()` → `'TURKOS'`

lower `färg.lower()` → `'turkos'`

Några str – funktioner

Exempelsträngar:

```
färg = "TurKos"
mening = "Gå och bada"
åsna = "      Ior   "
```

upper `färg.upper()` → `'TURKOS'`

lower `färg.lower()` → `'turkos'`

strip `åsna.strip()` → `'Ior'`

Några str – funktioner

Exempelsträngar:

```
färg = "TurKos"
mening = "Gå och bada"
åsna = "      Ior   "
```

upper `färg.upper()` → `'TURKOS'`

lower `färg.lower()` → `'turkos'`

strip `åsna.strip()` → `'Ior'`

split `mening.split()` → `['Gå','och','bada']`

Några str – funktioner

Exempelsträngar:

```
färg = "TurKos"
mening = "Gå och bada"
åsna = "      Ior   "
```

upper `färg.upper()` → `'TURKOS'`

lower `färg.lower()` → `'turkos'`

strip `åsna.strip()` → `'Ior'`

split `mening.split()` → `['Gå','och','bada']`

find `mening.find('h')` → `5`

Några str – funktioner

Exempelsträngar:

```
färg = "TurKos"
mening = "Gå och bada"
åsna = "      Ior   "
```

upper `färg.upper()` → `'TURKOS'`

lower `färg.lower()` → `'turkos'`

strip `åsna.strip()` → `'Ior'`

split `mening.split()` → `['Gå','och','bada']`

find `mening.find('h')` → `5`

`mening.find('R')` → `-1` **fanns ej**

Några str – funktioner

Exempelsträngar:

```
färg = "TurKos"
mening = "Gå och bada"
åsna = "      Ior   "
```

upper `färg.upper()` → `'TURKOS'`

lower `färg.lower()` → `'turkos'`

strip `åsna.strip()` → `'Ior'`

split `mening.split()` → `['Gå','och','bada']`

find `mening.find('h')` → `5`

`mening.find('R')` → `-1` **fanns ej**

`mening.find('bada')` → `7`

Några str – funktioner

Exempelsträngar:

```
färg = "TurKos"
mening = "Gå och bada"
åsna = "      Ior   "
```

upper `färg.upper()` → `'TURKOS'`

lower `färg.lower()` → `'turkos'`

strip `åsna.strip()` → `'Ior'`

split `mening.split()` → `['Gå','och','bada']`

find `mening.find('h')` → `5`

`mening.find('R')` → `-1` **fanns ej**

`mening.find('bada')` → `7`

OBS! Strängarna ändras ej, de är immutable!

Tupler

Tupler

- En följd av data omgiven av paranteser

Tupler

- En följd av data omgiven av paranteser

```
finatal = (1644, 1729, 47)
träningsdagar = ('mån','tis','tor','lör')
```

Tupler

- En följd av data omgiven av paranteser

```
finatal = (1644, 1729, 47)
träningsdagar = ('mån','tis','tor','lör')
```

- Kan innehålla data av valfria typer

Tupler

- En följd av data omgiven av paranteser

```
finatal = (1644, 1729, 47)
träningsdagar = ('mån','tis','tor','lör')
```

- Kan innehålla data av valfria typer

```
("Anna E", 257342, True, 1.61)
```

Tupler

- En följd av data omgiven av paranteser

```
finatal = (1644, 1729, 47)
träningsdagar = ('mån','tis','tor','lör')
```

- Kan innehålla data av valfria typer

```
("Anna E", 257342, True, 1.61)
```

- Indexeras som strängar

Tupler

- En följd av data omgiven av paranteser

```
finatal = (1644, 1729, 47)
träningsdagar = ('mån','tis','tor','lör')
```

- Kan innehålla data av valfria typer

```
("Anna E", 257342, True, 1.61)
```

- Indexeras som strängar

```
träningsdagar[1:3] → ('tis','tor')
```

Tupler

- En följd av data omgiven av paranteser

```
finatal = (1644, 1729, 47)
träningsdagar = ('mån','tis','tor','lör')
```

- Kan innehålla data av valfria typer

```
("Anna E", 257342, True, 1.61)
```

- Indexeras som strängar

```
träningsdagar[1:3] → ('tis','tor')
```

- Kan inte ändras, är **immutable**

Tupler

- En följd av data omgiven av paranteser

  `finatal = (1644, 1729, 47)`
  `träningsdagar = ('mån','tis','tor','lör')`

- Kan innehålla data av valfria typer

  `("Anna E", 257342, True, 1.61)`

- Indexeras som strängar

  `träningsdagar[1:3]` $\rightarrow$ `('tis','tor')`

- Kan inte ändras, är **immutable**

  ~~`träningsdagar[1] = 'ons'`~~ inte tillåtet!

Listor

Listor

- Som tupler men ändringsbara (mutable)!

Listor

- Som tupler men ändringsbara (mutable)!
- Skrivs med hakparanteser

Listor

- Som tupler men ändringsbara (mutable)!
- Skrivs med hakparanteser

```
nummer = [3, 54, 28, 10]
frukt = ["äpple","kiwi","päron"]
```

Listor

- Som tupler men ändringsbara (mutable)!
- Skrivs med hakparanteser

```
nummer = [3, 54, 28, 10]
frukt = ["äpple","kiwi","päron"]
```

- Indexeras som strängar och tupler

Listor

- Som tupler men ändringsbara (mutable)!
- Skrivs med hakparanteser

```
nummer = [3, 54, 28, 10]
frukt = ["äpple","kiwi","päron"]
```

- Indexeras som strängar och tupler

`frukt[2]` $\rightarrow$ `"päron"`

`nummer[1:]` $\rightarrow$ `[54, 28, 10]`

Listor

- Som tupler men ändringsbara (mutable)!
- Skrivs med hakparanteser

```
nummer = [3, 54, 28, 10]
frukt = ["äpple","kiwi","päron"]
```

- Indexeras som strängar och tupler

`frukt[2]` $\rightarrow$ `"päron"`

`nummer[1:]` $\rightarrow$ `[54, 28, 10]`

- Ändra innehållet:

Listor

- Som tupler men ändringsbara (mutable)!
- Skrivs med hakparanteser

```
nummer = [3, 54, 28, 10]
frukt = ["äpple","kiwi","päron"]
```

- Indexeras som strängar och tupler

`frukt[2]` → `"päron"`

`nummer[1:]` → `[54, 28, 10]`

- Ändra innehållet:

`frukt[1] = "plommon"` →

`frukt` → `["äpple","plommon","päron"]`

Listor

- Som tupler men ändringsbara (mutable)!
- Skrivs med hakparanteser

```
nummer = [3, 54, 28, 10]
frukt = ["äpple","kiwi","päron"]
```

- Indexeras som strängar och tupler

`frukt[2]` → `"päron"`

`nummer[1:]` → `[54, 28, 10]`

- Ändra innehållet:

`frukt[1] = "plommon"` →

`frukt` → `["äpple","plommon","päron"]`

- Har funktioner ...

Några listfunktioner

`append(x)`	Lägger till x sist i listan.

Några listfunktioner

`append(x)`	Lägger till x sist i listan.
`sort()`	Sorterar i stigande ordning,

Några listfunktioner

`append(x)`	Lägger till x sist i listan.
`sort()`	Sorterar i stigande ordning, om elementen kan jämföras!

Några listfunktioner

`append(x)`	Lägger till x sist i listan.
`sort()`	Sorterar i stigande ordning, om elementen kan jämföras!
`reverse()`	Vänder listan.

Några listfunktioner

`append(x)`	Lägger till x sist i listan.
`sort()`	Sorterar i stigande ordning, om elementen kan jämföras!
`reverse()`	Vänder listan.
`insert(i,x)`	Stoppar in x på plats i.

Några listfunktioner

`append(x)`	Lägger till x sist i listan.
`sort()`	Sorterar i stigande ordning, om elementen kan jämföras!
`reverse()`	Vänder listan.
`insert(i,x)`	Stoppar in x på plats i.
`pop(i)`	Plockar ut elementet på plats i, värdet returneras

Några listfunktioner

`append(x)`	Lägger till x sist i listan.
`sort()`	Sorterar i stigande ordning, om elementen kan jämföras!
`reverse()`	Vänder listan.
`insert(i,x)`	Stoppar in x på plats i.
`pop(i)`	Plockar ut elementet på plats i, värdet returneras
`remove(x)`	Tar bort första förekomsten av x,

Några listfunktioner

`append(x)`	Lägger till x sist i listan.
`sort()`	Sorterar i stigande ordning, om elementen kan jämföras!
`reverse()`	Vänder listan.
`insert(i,x)`	Stoppar in x på plats i.
`pop(i)`	Plockar ut elementet på plats i, värdet returneras
`remove(x)`	Tar bort första förekomsten av x, om x inte finns blir det felavbrott!

Några listfunktioner

`append(x)`	Lägger till x sist i listan.
`sort()`	Sorterar i stigande ordning,
	om elementen kan jämföras!
`reverse()`	Vänder listan.
`insert(i,x)`	Stoppar in x på plats i.
`pop(i)`	Plockar ut elementet på plats i,
	värdet returneras
`remove(x)`	Tar bort första förekomsten av x,
	om x inte finns blir det felavbrott!

Kan ändra listans innehåll!

Listor är **mutable**

Gå igenom en lista

Gå igenom en lista

```
for element in alist:
    do something with element
```

Gå igenom en lista

```
for element in alist:
    do something with element
```

T.ex multiplicera ihop alla element:

Gå igenom en lista

```
for element in alist:
    do something with element
```

T.ex multiplicera ihop alla element:

```
alist = [2,5,7,3,4,2]
prod = 1
for t in alist:
    prod = prod * t

print("Produkten =", prod)
```

Repetition över listans **värden**

Ibland behöver man indexen i listan

Ibland behöver man indexen i listan

Sätt alla negativa element i en tallista till 0

Ibland behöver man indexen i listan

Sätt alla negativa element i en tallista till 0

$$\mathtt{alist} = [3,\ -2.9,\ 5.5,\ 3.7,\ -1,\ 2.6,\ 6.8,\ -0.1]$$

Ibland behöver man indexen i listan

Sätt alla negativa element i en tallista till 0

```
alist = [3, -2.9, 5.5, 3.7, -1, 2.6, 6.8, -0.1]
```

Repetera över indexen istället

Ibland behöver man indexen i listan

Sätt alla negativa element i en tallista till 0

```
alist = [3, -2.9, 5.5, 3.7, -1, 2.6, 6.8, -0.1]
```

Repetera över indexen istället

```
for i in range(len(alist)):
    if alist[i] < 0:
        alist[i] = 0

print(alist)
```

Ibland behöver man indexen i listan

Sätt alla negativa element i en tallista till 0

```
alist = [3, -2.9, 5.5, 3.7, -1, 2.6, 6.8, -0.1]
```

Repetera över indexen istället

```
for i in range(len(alist)):
    if alist[i] < 0:
        alist[i] = 0

print(alist)
```

Resultat:

```
[3, -2.9, 5.5, 3.7, -1, 2.6, 6.8, -0.1]   # före
```

Ibland behöver man indexen i listan

Sätt alla negativa element i en tallista till 0

```
alist = [3, -2.9, 5.5, 3.7, -1, 2.6, 6.8, -0.1]
```

Repetera över indexen istället

```
for i in range(len(alist)):
    if alist[i] < 0:
        alist[i] = 0

print(alist)
```

Resultat:

```
[3, -2.9, 5.5, 3.7, -1, 2.6, 6.8, -0.1]    # före
[3,    0, 5.5, 3.7,  0, 2.6, 6.8,  0]      # efter
```

Alternativt sätt: enumerate

Alternativt sätt: enumerate

Repetera över indexen och värdena

Alternativt sätt: enumerate

Repetera över indexen och värdena

```
alist = [3, -2.9, 5.5, 3.7, -1, 2.6, 6.8, -0.1]
```

Alternativt sätt: enumerate

Repetera över indexen och värdena

```
alist = [3, -2.9, 5.5, 3.7, -1, 2.6, 6.8, -0.1]
```

```
enumerate(alist)
```

Alternativt sätt: enumerate

Repetera över indexen och värdena

```
alist = [3, -2.9, 5.5, 3.7, -1, 2.6, 6.8, -0.1]
```

`enumerate(alist)` $\rightarrow$ `[(0,3),(1,-2.9),(2,5.5)...]`

Alternativt sätt: enumerate

Repetera över indexen och värdena

```
alist = [3, -2.9, 5.5, 3.7, -1, 2.6, 6.8, -0.1]
```

`enumerate(alist)` $\rightarrow$ `[(0,3),(1,-2.9),(2,5.5)...]`

Sätt alla negativa element i alist till 0

Alternativt sätt: enumerate

Repetera över indexen och värdena

```
alist = [3, -2.9, 5.5, 3.7, -1, 2.6, 6.8, -0.1]
```

`enumerate(alist)` → `[(0,3),(1,-2.9),(2,5.5)...]`

Sätt alla negativa element i alist till 0

```
for i,v in enumerate(alist):
    if v < 0:
        alist[i] = 0

print(alist)
```

Alternativt sätt: enumerate

Repetera över indexen och värdena

```
alist = [3, -2.9, 5.5, 3.7, -1, 2.6, 6.8, -0.1]
```

`enumerate(alist)` → `[(0,3),(1,-2.9),(2,5.5)...]`

Sätt alla negativa element i alist till 0

```
for i,v in enumerate(alist):
    if v < 0:
        alist[i] = 0

print(alist)
```

Samma resultat:

```
[3, 0, 5.5, 3.7, 0, 2.6, 6.8, 0]
```

Nästlade listor = listor i listor i listor ...

Nästlade listor = listor i listor i listor ...

```
kläder = [['hatt', 'keps', 'turban', 'mössa'],
          ['T-shirt','skjorta','blus'],
          ['jeans'],
          ['sandaler','boots']]
```

Nästlade listor = listor i listor i listor ...

```
kläder = [['hatt', 'keps', 'turban', 'mössa'],
          ['T-shirt','skjorta','blus'],
          ['jeans'],
          ['sandaler','boots']]
```

Ett index ger en lista

Nästlade listor = listor i listor i listor ...

```
kläder = [['hatt', 'keps', 'turban', 'mössa'],
          ['T-shirt','skjorta','blus'],
          ['jeans'],
          ['sandaler','boots']]
```

Ett index ger en lista

`kläder[1]` $\rightarrow$ `['T-shirt','skjorta','blus']`

Nästlade listor = listor i listor i listor ...

```
kläder = [['hatt', 'keps', 'turban', 'mössa'],
          ['T-shirt','skjorta','blus'],
          ['jeans'],
          ['sandaler','boots']]
```

Ett index ger en lista

`kläder[1]` $\rightarrow$ `['T-shirt','skjorta','blus']`
`kläder[3]` $\rightarrow$ `['sandaler','boots']`

Nästlade listor = listor i listor i listor ...

```
kläder = [['hatt', 'keps', 'turban', 'mössa'],
          ['T-shirt','skjorta','blus'],
          ['jeans'],
          ['sandaler','boots']]
```

Ett index ger en lista

`kläder[1]` $\rightarrow$ `['T-shirt','skjorta','blus']`
`kläder[3]` $\rightarrow$ `['sandaler','boots']`

Två index ger element från inre lista

`kläder[0][2]` $\rightarrow$ `'turban'`
`kläder[3][0]` $\rightarrow$ `'sandaler'`

En nästlad lista kan representera en matris

```
mat = [[3,  5,  0],
       [7, -1,  1],
       [2,  6,  8]]
```

En nästlad lista kan representera en matris

```
mat = [[3,  5,  0],
       [7, -1,  1],
       [2,  6,  8]]
```

Ett index ger en rad

två index ger ett element

En nästlad lista kan representera en matris

```
mat = [[3,  5,  0],
       [7, -1,  1],
       [2,  6,  8]]
```

Ett index ger en rad

`mat[0]` $\rightarrow$ `[3, 5, 0]`

två index ger ett element

`mat[0][1]` $\rightarrow$ `5`

En nästlad lista kan representera en matris

```
mat = [[3,  5,  0],
       [7, -1,  1],
       [2,  6,  8]]
```

Ett index ger en rad

`mat[0]` → `[3, 5, 0]`
`mat[1]` → `[7,-1, 1]`

två index ger ett element

`mat[0][1]` → `5`
`mat[1][0]` → `7`

En nästlad lista kan representera en matris

```
mat = [[3,  5,  0],
       [7, -1,  1],
       [2,  6,  8]]
```

Ett index ger en rad

`mat[0]` → `[3, 5, 0]`
`mat[1]` → `[7,-1, 1]`
`mat[2]` → `[2, 6, 8]`

två index ger ett element

`mat[0][1]` → `5`
`mat[1][0]` → `7`
`mat[2][2]` → `8`

En nästlad lista kan representera en matris

```
mat = [[3,  5,  0],
       [7, -1,  1],
       [2,  6,  8]]
```

Ett index ger en rad

`mat[0]` → `[3, 5, 0]`
`mat[1]` → `[7,-1, 1]`
`mat[2]` → `[2, 6, 8]`

två index ger ett element

`mat[0][1]` → `5`
`mat[1][0]` → `7`
`mat[2][2]` → `8`

Kolumer är lite svårare att komma åt

En nästlad lista kan representera en matris

```
mat = [[3,  5,  0],
       [7, -1,  1],
       [2,  6,  8]]
```

Ett index ger en rad

`mat[0]` → `[3, 5, 0]`
`mat[1]` → `[7,-1, 1]`
`mat[2]` → `[2, 6, 8]`

två index ger ett element

`mat[0][1]` → `5`
`mat[1][0]` → `7`
`mat[2][2]` → `8`

Kolumer är lite svårare att komma åt

Om det finns tid så skriver vi en kolumnfunktion.
Funktionen finns på föreläsningens webbsida

Objekt och referenser

```
a = [8, 13, 16]
b = a
```

Objekt och referenser

```
a = [8, 13, 16]
b = a

a[1] = 77
```

Objekt och referenser

```
a = [8, 13, 16]
b = a

a[1] = 77
```

Nu är listan a ändrad till `[8, 77, 16]`

Objekt och referenser

```
a = [8, 13, 16]
b = a

a[1] = 77
```

Nu är listan a ändrad till `[8, 77, 16]`
Har något hänt med b ?

Objekt och referenser

```
a = [8, 13, 16]
b = a

a[1] = 77
```

Nu är listan a ändrad till `[8, 77, 16]`
Har något hänt med b ?

JA! a och b refererar till samma lista.
b är också `[8, 77, 16]`

Objekt och referenser

```
a = [8, 13, 16]
b = a

a[1] = 77
```

Nu är listan a ändrad till `[8, 77, 16]`
Har något hänt med b ?

JA! a och b refererar till samma lista.
b är också `[8, 77, 16]`

`b = a[:]` ger kopiering

Gör en nxn–matris med nollor

Gör en nxn–matris med nollor

```
nollrad = [0]*n
matris = [nollrad]*n
```

Gör en nxn–matris med nollor

```
nollrad = [0]*n
matris = [nollrad]*n
```

Antag n = 4, titta på matrisen

Gör en nxn–matris med nollor

```
nollrad = [0]*n
matris = [nollrad]*n
```

Antag n = 4, titta på matrisen

```
[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
```

Gör en nxn–matris med nollor

```
nollrad = [0]*n
matris = [nollrad]*n
```

Antag n = 4, titta på matrisen

```
[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
```

Ser bra ut!

Gör en nxn–matris med nollor

```
nollrad = [0]*n
matris = [nollrad]*n
```

Antag n = 4, titta på matrisen

```
[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
```

Ser bra ut!

Ändra ett element

```
matris[0][0] = 9
```

Gör en nxn–matris med nollor

```
nollrad = [0]*n
matris = [nollrad]*n
```

Antag n = 4, titta på matrisen

```
[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
```

Ser bra ut!

Ändra ett element

```
matris[0][0] = 9
```

titta på matrisen

Gör en nxn–matris med nollor

```
nollrad = [0]*n
matris = [nollrad]*n
```

Antag n = 4, titta på matrisen

```
[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
```

Ser bra ut!

Ändra ett element

```
matris[0][0] = 9
```

titta på matrisen

```
[[9,0,0,0],[9,0,0,0],[9,0,0,0],[9,0,0,0]]
```

Ser inte bra ut! Samma nollrad på 4 ställen.

Gör en nxn–matris med nollor, alla rader olika listor

```
nollrad = [0]*n
matris = []
for k in range(n):
    matris.append(nollrad[:]) #KOPIA!
```

Gör en nxn–matris med nollor, alla rader olika listor

```
nollrad = [0]*n
matris = []
for k in range(n):
    matris.append(nollrad[:]) #KOPIA!
```

Antag n = 4, titta på matrisen

Gör en nxn–matris med nollor, alla rader olika listor

```
nollrad = [0]*n
matris = []
for k in range(n):
    matris.append(nollrad[:]) #KOPIA!
```

Antag n = 4, titta på matrisen

```
[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
```

Gör en nxn–matris med nollor, alla rader olika listor

```
nollrad = [0]*n
matris = []
for k in range(n):
    matris.append(nollrad[:]) #KOPIA!
```

Antag n = 4, titta på matrisen

```
[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
```

Ser bra ut!

Ändra nu ett element

```
matris[0][0] = 9
```

Gör en nxn–matris med nollor, alla rader olika listor

```
nollrad = [0]*n
matris = []
for k in range(n):
    matris.append(nollrad[:]) #KOPIA!
```

Antag n = 4, titta på matrisen

```
[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
```

Ser bra ut!

Ändra nu ett element

```
matris[0][0] = 9
```

titta på matrisen

Gör en nxn–matris med nollor, alla rader olika listor

```
nollrad = [0]*n
matris = []
for k in range(n):
    matris.append(nollrad[:]) #KOPIA!
```

Antag n = 4, titta på matrisen

```
[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
```

Ser bra ut!
Ändra nu ett element

```
matris[0][0] = 9
```

titta på matrisen

```
[[9,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
```

Ser bra ut! Varje matrisrad är en ny kopia av nollrad.

Dictionaries – uppslagslistor

Dictionaries – uppslagslistor

- Liknar listor men alla immutables kan vara ”index”

Dictionaries – uppslagslistor

- Liknar listor men alla immutables kan vara ”index”
- Skrivs med {...}

```
engelska = {}
```

Dictionaries – uppslagslistor

- Liknar listor men alla immutables kan vara ”index”
- Skrivs med {...}

```
engelska = {}
```

- Lagrar datapar som nyckel och värde

Dictionaries – uppslagslistor

- Liknar listor men alla immutables kan vara "index"
- Skrivs med {...}

```
engelska = {}
```

- Lagrar datapar som nyckel och värde

```
engelska['fågel'] = 'bird'
engelska['fisk'] = 'fish'
engelska['sköldpadda'] = ['turtle','tortoise']
engelska['bra'] =
    ['good','fine','decent','satisfactory']
engelska[4] = 'four'
```

Dictionaries – uppslagslistor

- Liknar listor men alla immutables kan vara ”index”
- Skrivs med {...}

```
engelska = {}
```

- Lagrar datapar som nyckel och värde

```
engelska['fågel'] = 'bird'
engelska['fisk'] = 'fish'
engelska['sköldpadda'] = ['turtle','tortoise']
engelska['bra'] =
    ['good','fine','decent','satisfactory']
engelska[4] = 'four'
```

- Värden kan vara vilken typ som helst

Dictionaries – uppslagslistor

- Liknar listor men alla immutables kan vara "index"
- Skrivs med {...}

```
engelska = {}
```

- Lagrar datapar som nyckel och värde

```
engelska['fågel'] = 'bird'
engelska['fisk'] = 'fish'
engelska['sköldpadda'] = ['turtle','tortoise']
engelska['bra'] =
    ['good','fine','decent','satisfactory']
engelska[4] = 'four'
```

- Värden kan vara vilken typ som helst
- Nycklar kan vara tal, text eller tupler (immutables)

Dictionaries – uppslagslistor

- Liknar listor men alla immutables kan vara ”index”
- Skrivs med {...}

```
engelska = {}
```

- Lagrar datapar som nyckel och värde

```
engelska['fågel'] = 'bird'
engelska['fisk'] = 'fish'
engelska['sköldpadda'] = ['turtle','tortoise']
engelska['bra'] =
    ['good','fine','decent','satisfactory']
engelska[4] = 'four'
```

- Värden kan vara vilken typ som helst
- Nycklar kan vara tal, text eller tupler (immutables)
- Har ingen inbördes ordning, kan inte sorteras

Några funktioner för dictionaries

`keys()`	Ger lista med alla nycklar.
`values()`	Ger lista med alla värden.
`get(k)`	Returnerar värdet för nyckel k. Listan oförändrad.
`pop(k)`	Hämtar värdet för nyckel k, tar ut paret (k,v) ur listan.

Operationer

`k in dict`	Finns k som nyckel i dict?
`for k in dict:`	Gå igenom nycklarna, värdena är då `dict[k]`

Dictionaryexempel

Håll reda på vad du ska träna olika dagar:

Dictionaryexempel

Håll reda på vad du ska träna olika dagar:

```
traning = {'måndag':'Skivstång', 'tisdag':'Spinning',
           'torsdag':'Step', 'lördag':'Skivstång'}
```

Dictionaryexempel

Håll reda på vad du ska träna olika dagar:

```
traning = {'måndag':'Skivstång', 'tisdag':'Spinning',
           'torsdag':'Step', 'lördag':'Skivstång'}

while True:
    dag = input('Vilken dag är det? ')
    if dag in traning:
        print('Idag ska du träna', traning[dag])
    else:
        print('Idag får du vila')
```

Dictionaryexempel

Romerska siffrors värde

Dictionaryexempel

Romerska siffrors värde

```
romerska = {'I':1, 'V':5, 'X':10, 'C': 100, 'M':1000}
```

Dictionaryexempel

Romerska siffrors värde

```
romerska = {'I':1, 'V':5, 'X':10, 'C': 100, 'M':1000}

romerska['L'] = 50
romerska['D'] = 500
```

Dictionaryexempel

Romerska siffrors värde

```
romerska = {'I':1, 'V':5, 'X':10, 'C': 100, 'M':1000}

romerska['L'] = 50
romerska['D'] = 500

undantag = {'IX': 9, 'XC': 90, 'CM': 900}
```

Dictionaryexempel

Romerska siffrors värde

```
romerska = {'I':1, 'V':5, 'X':10, 'C': 100, 'M':1000}

romerska['L'] = 50
romerska['D'] = 500

undantag = {'IX': 9, 'XC': 90, 'CM': 900}
```

Översättning av romerska tal till arabiska och tvärtom är en lätt P-uppgift.