Föreläsning 3
Programmeringsteknik och C
DD1316

- uppdateringsoperatorer
- listor
- tupler
- strängar
- for-slingor
- importera moduler
- random

Listor

- En lista är en *muterbar* (ändringsbar), ordnad samling objekt.
- Listor skapas med hakparenteser:

```
[12, 13, 14, 15]
```


Uppdateringsoperatorer

- För många binära operatorer, tex `*` `/` + och –, finns motsvarande uppdateringsoperatorer

  `*=` `/=` `+=` och `-=`
- Exempel:

```
x += 1
```

  betyder

```
x = x + 1
```

- Generellt:

```
VAR OP= ARG ⇔ VAR = VAR OP ARG
```


Innehåll i listor

- En lista kan innehålla element av olika typer:
  `["hej" , 151 , 10.59]`
- En lista kan innehålla en eller flera listor:
  `[ ["hej", 151], 10.59]`
- Man kan åstadkomma godtycklig komplicerade strukturer m.h.a listor:

```
list1 = ["hej"]
list1 += ["hejdå", list1]
```

+,* och listor

- Listor kan konkateneras:

```
lista = [1,2,3] + [4,5,6]
```

  Vad innehåller variabeln lista?
- Listor kan upprepas:

```
lista = [1,2]
lista *= 3
```

  Vad innehåller variabeln lista?

Indexering

- Ett enskilt element på ett givet index kan väljas ut:

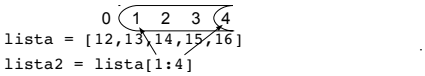
```
           0  1  2  3
lista = [12,13,14,15]

print(lista[2])
```

Vad skrivs ut?

Dellista

- Ett startindex och ett övre begränsningsindex ger dellistan.




```
lista2 = lista[1:4]
```

Vad innehåller `lista2`?

Dellista

- Ett startindex, ett övre begränsningsindex och ett intervall ger en annan typ av dellista.

```
lista  = [6,7,1,8,5,2,4,9]
lista2 = lista[1:8:3]
```

Vad innehåller lista2?

Att välja element (sammanfattning)

Låt `ls` vara en lista

- `ls[i]` ger elementet med index `i`
- `ls[i:j]` ger en ny lista med alla element från och med `i` till (men ej med) `j`
- `i` eller `j` kan utelämnas
  - `ls[i:]` ger alla element från och med `i`
  - `ls[:j]` ger alla element till (men ej med `j`)
- `ls[i:j:s]` ger vart `s`:e element från och med `i` till `j`
- Negativa index räknar bakifrån (med `0` efter sista elementet):
  - `ls[-1]` ger sista elementet
  - `ls[:-1]` ger alla element utom det sista elementet

Ändra

- Identifierade element / dellistor kan ändras:

```
lista = [1,2,3,4,5,6]
lista[1:3] = ["a"]
lista[4] = "b"
print(lista)
```

Vad skrivs ut?

Borttagning

- Element/dellista kan tas bort:

```
lista = [1,2,3,4,5,6]
del lista[1:3]
del lista[3]
print(lista)
```

Vad skrivs ut?

Metoder för listor

- Omvänd ordning på elementen

```
lista = [2,3,1,4,5,7,6]
lista.reverse()
print(lista)
```

Vad skrivs ut?

- Sortera elementen

```
lista.sort()
print(lista)
```

Vad skrivs ut?

Metoder för listor

- Lägg till element på slutet
  ```
  lista.append(13)
  ```
- Lägg till lista på slutet
  ```
  lista.extend([1,2,3])
  ```


Tupler

- En tupel är en *omuterbar* (ej ändringsbar) lista.
- En tupel skapas med vanliga parenteser:
  ```
  t = (1,2,3,4)
  ```


+, * och tupler

- Tupler kan konkateneras:
  ```
  t = (1,2,3) + (4,5,6)
  print(t)
  ```
  Vad skrivs ut?
- Tupler kan upprepas:
  ```
  t = (1,2,3)
  print(t * 3)
  ```
  Vad skrivs ut?

Indexering

- Ett enskilt element på ett givet index kan väljas ut:

  ```
  tupel = (12,13,14,15)
  print(tupel[2])
  ```

  Vad skrivs ut?

Deltupel

- Ett startindex och ett övre begränsningsindex ger deltupeln.

  ```
  tupel = (12,13,14,15,16)
  print (tupel[2:4])
  ```

  Vad skrivs ut?

Omuterbarhet

- En tupel kan **inte** ändras.

  ```
  t1 = (1, [2, 3])
  t1[0] = 33        ger fel
  t1[1] = [2,3,33]  ger fel
  ```

- Följande ger **inte** fel:

  ```
  t1[1].append(33)
  ```

  Varför inte?

Strängar

- En sträng kan bektraktas som en tupel där alla elementen är tecken.

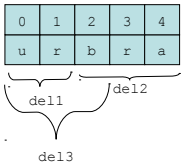
```
nose = ”En lång näsa”
```

0	1	2	3	4	5	6	7	8	9	10	11
E	n		l	å	n	g		n	ä	s	a

Skapa delsträng

- Precis som för tupler kan vi välja ut delar:

```
state = ”urbra”
del1 = state[0:2]
del2 = state[2:]
del3 = state[:3]
```



`len()` och `in`

- Antalet element i listor och tupler:

```
print(len([1,2,5]))
```

Vad skrivs ut?

- `in` testar medlemskap i lista eller tupel:

```
if 3 in [1,2,3]:
      print(”3 finns i listan”)
else:
      print(”3 finns inte i listan”)
```

Vad skrivs ut?

`for`-slinga

- `for` används för att *iterera* över (gå igenom) alla element i en lista, tupel, eller sträng.

```
lista = [12,13,14,15,16]
for x in lista:
      print(x, end=” ”)
```

Skriver ut:
12 13 14 15 16

Syntax

for **variabel** in **lista**:

Block som exekveras för varje element.

`range()`

- Ofta vill man göra något för varje heltal i ett intervall.
- `range()` är praktisk i detta fall:

```
for x in range(5):
      print(x, end=” ”)
```

Skriver ut:
0 1 2 3 4

Mer om `range()`

- Kan användas på följande 3 sätt:

```
1.  list(range(3))
[0,1,2]
2.  list(range(3,10))
[3,4,5,6,7,8,9]
3.  list(range(3,15,2))
[3,5,7,9,11,13]
```


Mer om `range()`

- Avtagande heltalslistor

```
list(range(6,1,-1))
   [6,5,4,3,2]

list(range(10,0,-2))
   [10,8,6,4,2]
```


Importering av moduler

Det finns en hel del funktioner man kan använda genom att importera dem från pythons standard bibliotek. För att importera de så använder man reserverade ordet `import`.
Ett bibliotek av tillgängliga moduler finns beskrivet här:
`http://docs.python.org/3/library/index.html`

Exempel:

```
import random
random.random()

import math
math.sin(3.14)
```


`randrange(low, high)` och `random()`

- Slumpmässigt heltal i ett givet intervall:

```
import random
t = random.randrange(3,6)
```

Ett av talen 3, 4 eller 5 väljs slumpmässigt och tilldelas variabeln t.

- Slumpmässigt flyttal:

```
r = random.random()
```

Variabeln r kommer att tilldelas ett slumpmässigt decimaltal i intervallet [0, 1).

Sammanfattning

- Listor, tupler och strängar är strukturmässigt lika varandra
- Tupler och strängar är omuterbara
- Tupler är snabbare än listor
- Tupler ska användas när man har ett antal värden som inte kommer att ändras under körningen av programmet.
- `for`-slingor i kombination med listor, tupler och strängar är en kraftfull konstruktion.
- `range()` underlättar att skapa långa listor av heltal