

Texthantering

- Introduktion
- Teckenkoder
- Indexering
- Skivor
- Jämförelser av strängar
- Loopa igenom en sträng
- Operatorerna *in*, + och *
- Inbyggda funktioner
- Inbyggda metoder
- Strängar är immutable

Introduktion

För texthantering behövs variabler som innehåller text. Dessa är av typen *str* (sträng/text). En sträng består av en sekvens av tecken.

```
s = '' # En tom sträng
s1 = 'python' # eller "python" (enkelblippar el dubbelblippar)
print(s1)
print(type(s1))
```

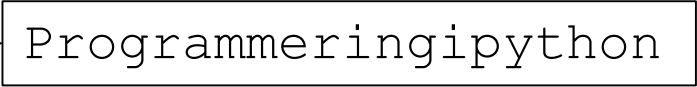


Konkatenering av strängar med +

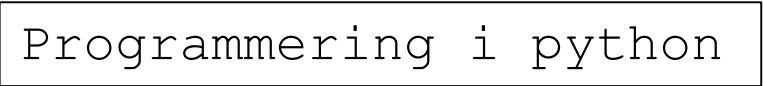
```
s2 = 'Programmering'
s3 = s2 + s1
print(s3)
```



```
s3 = s2 + 'i' + s1
print(s3)
```



```
s3 = s2 + ' i ' + s1
print(s3)
```



Specialtecken i strängar:

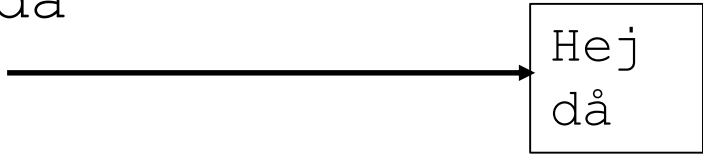
' \ ' backslash
' \n ' newline
' \t ' tabulatortecken

Skriv fem tomma rader:

```
print ( ' \n ' *5 )  
(jmf *-operatör i Fö4)
```

```
s1 = 'Hej \n då'
```

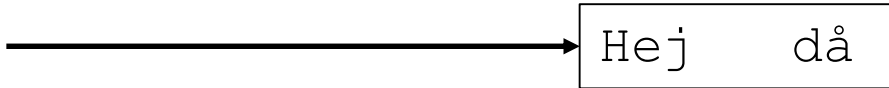
```
print(s1)
```



```
Hej  
då
```

```
s1 = 'Hej \t då'
```

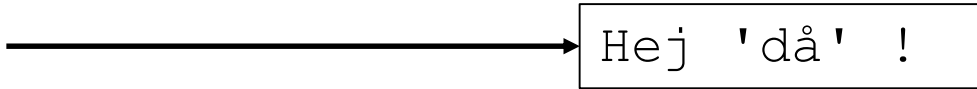
```
print(s1)
```



```
Hej      då
```

```
s1 = 'Hej \'då\' !'
```

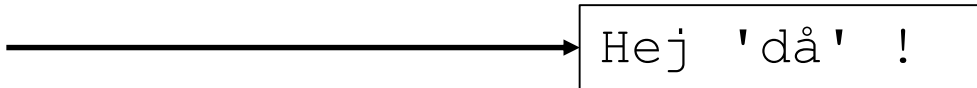
```
print(s1)
```



```
Hej 'då' !
```

```
s2 = "Hej 'då' !"    # Använd dubbelfnuttar för avgränsning
```

```
print(s2)
```



```
Hej 'då' !
```

Teckenkoder

Tecken kodas som ett heltal, enligt *Unicode* (<https://unicode-table.com/en/>). Det finns plats för drygt 1 miljon tecken. En delmängd av den koden utgörs av *ASCII-koden*. Exvis har bokstaven (tecknet) A koden 65, B har koden 66, etc.

ASCII-koden omfattar 256 olika koder/tecken, vi kikar på www.asciitabell.se. För 256 koder krävs ett binärt tal på 8 bitar. $256 = 11111111$ i kolumnen. Du ser t.ex. att tecknet A kodas som 65.

Tecknen i tabellen är ordnade så att versaler och gemener samlade och de är sorterade i ordning. Dock problem med ordningen på de svenska Å, Ä och Ö.

Med funktionen ***ord*** kan vi ta reda på ascii-koden

```
s = 'A'           # En sträng med ett tecken
print(ord(s))     # Ger 65
```

Med funktionen ***chr*** kan vi konvertera från en kod till motsvarande teckenvärde.

```
s = chr(98) + chr(99)  # 98->b, 99->c
print(s)               # ger bc
```

funktionen *chr* gör om en kod till ett tecken.

funktionen *str* gör om tal till en sträng.

```
s1 = '98' + '99' # konkatenera (slå ihop) de båda strängarna
```

```
s2 = 98 + 99      # addition av två heltal
```

```
s3 = chr(98) + chr(99) # slå ihop två tecken
```

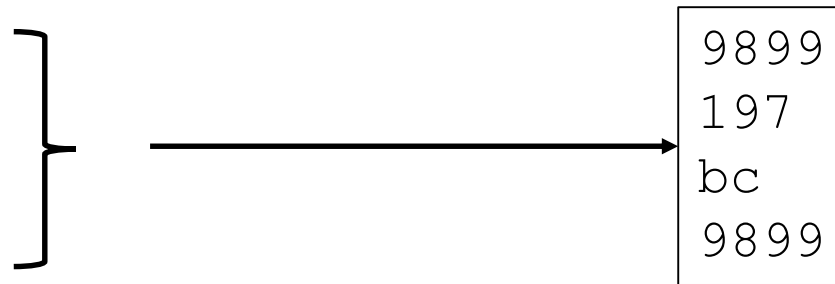
```
s4 = str(9899)
```

```
print(s1)
```

```
print(s2)
```

```
print(s3)
```

```
print(s4)
```



s1, s3, s4 är av typen str
s2 är av typen int

```
s5 = 'a7 G!'
```

Strängen *s5* är en sekvens av tecken,

kan illustreras enligt:

'a'	'7'	' '	'G'	'!'
-----	-----	-----	-----	-----

Men varje tecken är
lagrat (kodat) som:

97	55	32	71	33
----	----	----	----	----

Men detta kodas
egentligen binärt...

Göra om tecken, “kryptera”

Utgående från ascii-koden ges möjlighet att göra om tecken
Ett exempel, gör om tecknet A till B:

```
t = 'A'
```

```
t = chr( ord(t) + 1)
```

The diagram illustrates the execution of the code `t = chr(ord(t) + 1)` with three explanatory boxes:

- A bracket under `ord(t)` points to a box: `ord(t)` är ascii-koden för A, dvs 65
- A bracket under `ord(t) + 1` points to a box: `ord(t) + 1` är 65+1 = 66
- A bracket under the entire expression `chr(ord(t) + 1)` points to a box: `chr(ord(t) + 1) = chr(66)` är B

```
print(t) # Ger B
```

Skapa ett slumptecken

```
slump = chr(ord('A') + random.randint(0,25))
```

Ger ett slumptecken, en bokstav: 'A' - 'Z'

Förklaring till ovan sats:

'A' har koden 65. Om man till detta värde adderar ett slumptal som kan vara 0-25 blir resultatet 65-90. Detta motsvarar koderna för 'A' - 'Z'.

Om vi gör om dessa koder 65-90 till teckenvärden med (char) kommer det istället att bli ett teckenvärde 'A'-'Z', som lagras i variabeln slump.

Nu kan vi ana hur man kan skapa slumpade lösenord

Nu kan vi “kryptera” text:

Vi kikar på filen [encrypt.py](#)

Vad händer om texten innehåller blanktecken?

I koden *Unicode* ges möjlighet att koda mer än 1 miljon olika tecken (32-bit).
Se *Unicode*: www.unicode.coeurlumiere.com. F.n. är cirka 140 000
definierade.

(De 256 första tecknen i Unicode är ASCII-tecknen.)

Unicode uttrycks lämpligen hexadecimal. Tecknet A är kodat till 65, dvs 41
uttryckt hexadecimalt. $41 = 4 \cdot 16^1 + 1 \cdot 16^0$

```
ch1 = '\u0041'      # 41 hexadecimalt = 65 (A)
```

```
ch2 = '\u2663'
```

```
ch3 = '\u03A8'
```

```
ch4 = '\u263A'
```

```
print(ch1, ch2, ch3, ch4)
```



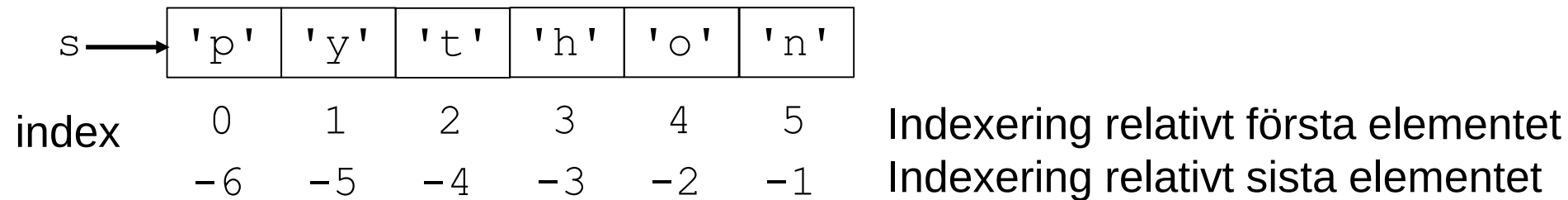
A ♣ Ψ ☺

`\u` ovan står för Unicode

Indexering och strängar

```
s = 'python'
```

s är sex tecken lång, dvs består av sex element



```
print(s[0], s[-3])
```

—————→ a h

```
s2 = s[-1] + s[2]
```

```
print(s2)
```

—————→ nt

Längden av en sträng

```
s = 'python'
```

```
print(len(s))
```

6

Skriv ut sista elementet

```
print(s[len(s)])
```

```
print(s[len(s)])  
IndexError: string index out of range
```

Två alternativ

```
print(s[len(s)-1])
```

n

```
print(s[-1])
```

n

Skivor av en sträng

När man indexerar väljer man ut ett enda element. Med *skivning* kan man välja flera element.

```
s = 'python'
    # 012345 (index)
s2 = s[3:6]      # s2 blir 'hon'
s2 = s[:4]       # s2 blir 'pyth'
s2 = s[4:]       # s2 blir 'on'
s2 = s[4:1:-1]   # s2 blir 'oht'
s2 = s[::-1]     # s2 blir 'nohtyp'
```

Allmänt gällande skivning `s[först:sist:steg]`

```
s = 'naturrutan'
if s == s[::-1]:
    print(s + ' är ett palindrom') → naturrutan är ett palindrom
```

Jämförelser av strängar

```
s1 = 'abc'
```

```
s2 = 'abcd'
```

```
s3 = 'abC'
```

```
print(s1=='abc')    # ger True
```

```
print(s1<s2)        # ger True
```

```
print(s1<s3)        # ger False
```

```
print('a' < 'Ab')   # ger False
```

Jämförelse baseras på ascii-koden, från vänster till höger

Om tecknen är lika i de båda strängarna, är en kortare sträng “mindre”

Loop genom sträng med operatoren *in*, ex räkna antal x i en text:

```
s = input('Skriv en text') # sex laxar in en laxask
n = 0
# för varje element c i s
for c in s: # c = s,e,x,...
    if c == 'x':
        n = n + 1
print('Antal tecken x:', n) # Antal tecken x: 3
```

Alt 2 (sämre)

```
# för varje index i s
for i in range(0, len(s)): # i = 0..21
    if s[i] == 'x':
        n = n + 1
print('Antal tecken x:', n) # Antal tecken x: 3
```

Alt 3:

```
n = 0
# för varje index (i) och element (c)
for i,c in enumerate(s):
    if c == 'x':
        n = n + 1
print('Antal tecken x:', n) # Antal tecken x: 3
```

Alt 4, enklast med pythons inbyggda metod count:

```
print(s.count('x')) # 3
```

Operatorerna *in*, + och *

Förutom att användas i loop kan operatoren *in* användas för att undersöka om ett element finns i en sträng:

```
s = 'sex laxar i en laxask'
if 'x' in s:
    print('x finns i', s) -> x finns i sex laxar i en laxask
```

Operatoren + för att bygga en ny sträng:

```
s1 = 'nej'
s1 = s1 + '!' # -> 'nej!' (en ny sträng s skapas)
```

Kan ej ändra elementen i en sträng:

```
s1[1] = 'Q' #
```

→

```
s1[1] = 'Q'
TypeError: 'str' object does not
support item assignment
```

Operatoren * för att bygga en ny sträng:

```
s2 = 3 * s1 # el s1 * 3
print(s2) # -> 'nej!nej!nej!'
```

Exempel på inbyggd funktion lämplig för strängar:

```
s = 'python'  
print(len(s)) # ger 6
```

Finns andra inbyggda funktioner (ex min,max), men ej så användbara på strängar

```
String ramsa = "Sex laxar i en laxask";
```

```
ramsa = ramsa.replaceAll("lax", "mört");
```

Vad blir värdet av ramsa?

Några exempel på metoder för strängar (metoder anropas med punktnotation)

```
s = 'python'
print(s.find('y'))    # ger 1, metoden ger ett index
print(s.upper())      # 'PYTHON'

print(s.isalpha())    # ger True, alla tecken är bokstäver
print(s.isdecimal()) # ger False, alla tecken är ej siffror

ramsa = 'Sex laxar i en laxask'
print(ramsa.replace('lax', 'mört'))

# ger Sex mörtar i en mörtask
```

fortsättning →

Några exempel på metoder för strängar (metoder anropas med punktnotation)

```
ramsa = 'Sex laxar i en laxask'
```

```
lst = ramsa.split() # Splittar map på blanktecken
```

```
# ger ['Sex', 'laxar', 'i', 'en', 'laxask']
```

```
n = ramsa.count('ax') # Ger 2
```

```
s = '  abc '
```

```
s2 = s.strip() # ger 'abc'
```

```
s3 = s2.upper() # ger 'ABC'
```

```
s4 = s3.lower() # ger 'abc'
```

```
m = s4.index('b') # ger 1, dvs 2:a tecknet, börjar på 0
```

```
s = '8abcKL'
```

```
print(s.isalpha()) # Ger False, kollar om bara bokstäver
```

```
print(s[2:].isalpha()) # Ger True
```

Alla metoder, se https://www.w3schools.com/python/python_ref_string.asp

Operatorer på strängar:

in, **+**, *****, **>**, **<**, **==** (+ och * känner vi till sedan tidigare)

```
if s1 in s2: # Om strängen s1 finns i strängen s2
```

```
...
```

```
if (s1 == s2): # Om strängarna är exakt lika
```

```
...
```

```
if (s1 < s2): # 'bus' < 'busa' ger True
```

```
...           # 'buss' < 'busar' ger False
```

Python jämför tecken för tecken tills strängarna är slut eller det skiljer

Typomvandling av strängar

int(s) och **float(s)** fungerar om *s* är sträng som går att omvandla.

Strängar är *immutable*, dvs det går ej att ändra dess element, till skillnad mot exvis listor.

```
s = 'abcd'
```

```
s[1] = 'X'      # Vill byta ut c mot X
```

```
TypeError: 'str' object does not support item assignment
```

Kan ej ändra element i en sträng

```
s = 'abcd'
```

```
s[2:3] = 'X'   # Önskar byta c mot X mha skivning
```

```
TypeError: 'str' object does not support item assignment
```

Kan komma runt det med exvis:

```
s = s[:2] + 'X' + s[3:]
```

Men `s` refererar nu till en annan sträng

Alla sträng-metoder returnerar nya värden, de ändrar ej original-strängen.