Flik 5: Datatyper

Hjälp, attribut och inbyggda funktioner

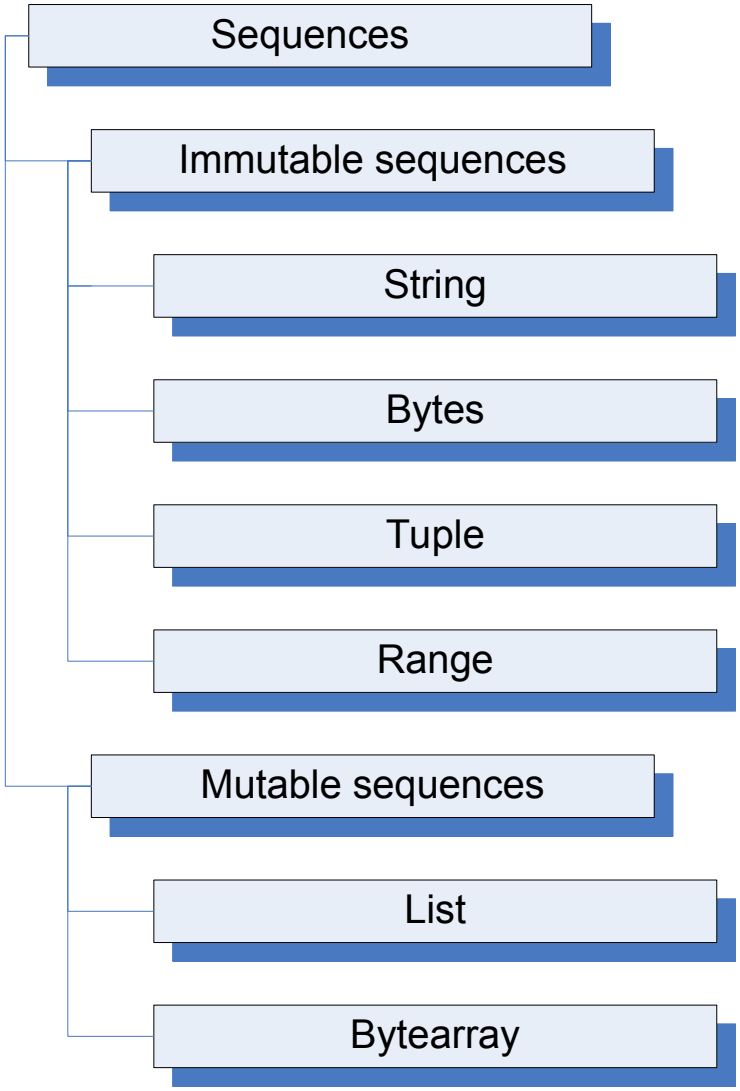
- *dir()*

```
>>> dir([])          # eller dir(list)
['__add__', '__class__', '__contains__', '__delattr__',
   '__delitem__', '__doc__', '__eq__', '__format__', '__ge__',
   '__getattribute__', '__getitem__', '__gt__', '__hash__',
   '__iadd__', '__imul__', '__init__', '__iter__', '__le__',
   '__len__', '__lt__', '__mul__', '__ne__', '__new__',
   '__reduce__', '__reduce_ex__', '__repr__', '__reversed__',
   '__rmul__', '__setattr__', '__setitem__', '__str__',
   '__subclasshook__', 'append', 'count', 'extend', 'index',
   'insert', 'pop', 'remove', 'reverse', 'sort']
```

```
>>> help([].sort)     # eller help(list.sort)
Help on built-in function sort:

sort(...)
    L.sort(key=None, reverse=False) -- stable sort *IN PLACE*
```

Sekvenser



Sekvenser - allmänt

- En sekvens går att iterera över - t.ex. med hjälp av en *for*-loop
- Strängar

```
>>> s = 'en sträng'
>>> raw_s = r'en\nsträng'    # \n tappar sin betydelse
>>> s2 = '"Run!", he said'
>>> s3 = "It's a string"
```

  - Strängar med tre " eller tre ' kan löpa över mer än en rad:

```
def my_func(x):
    """Function: my_func

       Argument: Integer
       Returns: True or False depending on..."""
```

- Tupler

```
>>> tupel_1_elem = (1,)          # obs kommatecknet
>>> tupel_2_elem = (1,'sträng')
```

- Listor

```
>>> lista_exempel = [1, 's', [1, 2, 3], (4,5)]
>>> lista_array = [[1, 2], [3, 4]]
```

Sekvenser – operatorer *in* och *not in*

- Operatorn *in* respektive *not in* testar om ett element ingår i en sekvens

```
>>> a = [1, 2, 3]
>>> 1 in a
True
>>> a = 'kalle'
>>> 'al' in a
True
```

- Element i sekvens - index

```
>>> a = ['en', 'kort', 'lista']
>>> a[0]
'en'
>>> a[-1]
'lista'
>>> a[1]
'kort'
>>> a[-2]
'kort'
```

Sekvenser – operatorer *utsnitt*

- Utsnitt i sekvens (obs: numrering sker "mellan" elementen)

```
>>> a[0:2]          # a[:2] är samma sak
['en', 'kort']
>>> a[1:]
['kort', 'lista']
>>> a[1:2]
['kort']
```

- Utökade utsnitt "strides" - steg (från version 2.3)

```
>>> lista = [1, 2, 3, 4, 5]
>>> lista[::2]          # stega med faktor 2 (varannat)
[1, 3, 5]
>>> lista[::3]          # stega med faktor 3
[1,4]
>>> lista[::-1]         # stega med faktor -1, dvs baklänges
[5, 4, 3, 2, 1]
>>> lista[::-2]         # stega baklänges med faktor 2
[5, 3, 1]
>>> lista[1::2]         # stega med faktor 2, börja på pos 1
[2, 4]
```

Sekvenser – operatorer + och *

- Sammanfogning (konkatenering)

```
>>> konk_list = [1, 0, -1] + [5, 6, 's', 7]
>>> print(konk_list)
[1, 0, -1, 5, 6, 's', 7]
>>> a = [1, 2, 3]
>>> a += [4, 5]
>>> print(a)
[1, 2, 3, 4, 5]
```

  - För listor: Mer effektivt med *extend* eller *append*. Se listor längre fram

- Repetition

```
>>> s = 'abc'
>>> s * 10
'abcabcabcabcabcabcabcabcabcabc'
>>> lista = [1, 2]
>>> lista * 3          # Även: lista *= 3
[1, 2, 1, 2, 1, 2]
```

Sekvenser - typkonvertering

- *list()*, *tuple(), bytes()* används för att konvertera från en sekvens till en annan

```
>>> list('abc')
['a', 'b', 'c']
>>> tuple([1, 2, 3])
(1, 2, 3)
```

- *str()* skriver ut objektets informella strängrepresentation

```
>>> lista = list('abc')
>>> str(lista)
"['a', 'b', 'c']"
```

- *repr()* skriver ut objektets formella strängrepresentation
  - `str` ska vara läsbar, `repr` ska vara otvetydig (är ofta Pythonkod)
- Ofta returnerar str() och repr() samma sak men ibland skiljer det

```
>>> s = "Ulla"
>>> print(str(s))
Ulla
>>> print(repr(s))
'Ulla'
>>>
```

Sekvenser - len(), sum()

- *len()* - längden på en sekvens

```
>>> items = ['tidning', 'papper', 'penna']
>>> len(items)
3
>>>
```

- Observera: Ej objektorienterad syntax för len!
  - Inkonsekvent?
- *sum()* - summera elementen i en lista eller tupel

```
>>> sum([1, 2, 3, 4, 5])
15
```

Sekvenser - övriga strängmetoder

- Följande urval av strängmetoder finns

```
sträng.upper()
sträng.lower()
sträng.isalpha()
sträng.isnumeric()
sträng.lstrip()
sträng.rstrip()
sträng.capitalize()
```

  - Se *kapitel 2: Fundamental Data Types: Strings* för en komplett lista (sid 73 – 75)

Sekvenser – lägga till element i lista

- Tomma listan

```
>>> lista = []
```

- Utökning med *extend() och append()*

```
>>> lista1 = [1, 2]
>>> lista2 = [3, 4]
>>> lista1.extend(lista2)
>>> print(lista1)
[1, 2, 3, 4]
>>> lista1.append(5)
>>> print(lista1)
[1, 2, 3, 4, 5]
>>> lista1.insert(2,0)
>>> print(lista1)
[1, 2, 0, 3, 4, 5]
```

Sekvenser – ta bort element i lista

- Ta bort element

```
>>> lista1 = [5, 4, 3, 1, 2]
>>> del lista1[1]
>>> print(lista1)
[5, 3, 1, 2]
>>> lista1.pop()
2
>>> print(lista1)
[5, 3, 1]
>>> lista1.pop(1)
3
>>> print(lista1)
[5, 1]
```

  - Använd *del lista[m:n]* för att ta bort en 'slice' av listan

Sekvenser – kopiera lista

- Ny referens

```
>>> lista1 = [5, 4, 3, 1, 2]
>>> lista1_ref = lista1          # samma utrymme (id() är samma)
>>> lista1.append(9)
>>> print(lista1_ref)
[5, 4, 3, 1, 2, 9]
```

- Kopiering ("shallow copy – bara första dimensionen")

```
>>> lista1 = [5, 4, 3, 1, 2]
>>> lista1_kopia = list(lista1) # nytt utrymme - olika id()
>>> lista1.append(9)
>>> print(lista1_kopia)
[5, 4, 3, 1, 2]
```

- Kopiering – *deepcopy()* (kopierar godtyckligt antal dimensioner)

```
>>> import copy
>>> r1 = [1, 2]
>>> l1 = [1, 2, r1]          # kopiera endast referensers
>>> l2 = copy.deepcopy(l1) # kopiera rekursivt allting
```

Sekvenser – söka, sortera, ...

- Söka efter element

```
>>> lista1 = [5, 4, 3, 1, 2]
>>> list1.index(4)          # Lägsta index där 4 förekommer
1
```

- Listor av listor (arrayer/matriser)

```
>>> lista1=[[1,2], [3,4]]
>>> print(lista1[0][1])
2
```

- Sortera element

```
>>> sorted(list1)
[1, 2, 3, 4, 5]
>>> lista1.sort()
>>> print(lista1)
[1, 2, 3, 4, 5]
>>> sorted([(3,2), (4,1), (1,4)], key=operator.itemgetter(1))
[(4, 1), (3, 2), (1, 4)]
```

- Övriga; *.remove()*, *.reverse()* m.fl.
  - Se *kapitel 3: Fundamental Collection Data Types: Lists*
- Listor är idealiska för att skapa andra datastrukturer som köer, stackar

Sekvenser - tupler

- Indexering, utsnitt, längd etc på samma sätt som för listor
- Tupler är "immutable"
- Kan dock innehålla element, t.ex. listor, som är mutable...
- Fördel med tupler - att skicka med som argument till okänd funktion. Ingen kan manipulera innehållet i bifogad tupel (förutsatt att elementen är immutable). Jämför C call-by-value, call-by-reference

Sekvenser – sequence unpacking

- Tilldelning

```
>>> a = (1, 2, 3)
>>> a = 1, 2, 3      # Parenteser kan utelämnas
>>> x, y, z = a      # Sekvens med 3 element
>>> x, *y = a
>>> x
1
>>> y                # *y blir alltid en lista
[2, 3]
```

- Packa upp lista:

```
>>> L = [ 'Linus', 'Ken' ]
>>> print(L)
['Linus', 'Ken']
>>> print(*L)
Linus Ken
>>>
```

Sekvenser – loop unpacking

- Loopa över sekvens med sammansatta element:

```
>>> tlist = [(1,2),(3,4)]
>>> for x, y in tlist:
...     print(x, y)
...
1 2
3 4
```

Övning

- **ovn0501/genomsnitt.py** Skriv ett program som läser in ett antal heltal (ett per rad) från en fil vars namn ska ges som en kommandoradsparameter. Beräkna och skriv ut medelvärdet av de tal som är kvar om man stryker de tre minsta och de tre största talen.

```
$ cat tal.txt
1000
10
-1001
5
1002
-1000
-1002
1001
15
$ python3 genomsnitt.py tal.txt
10.0
$
```

Övning

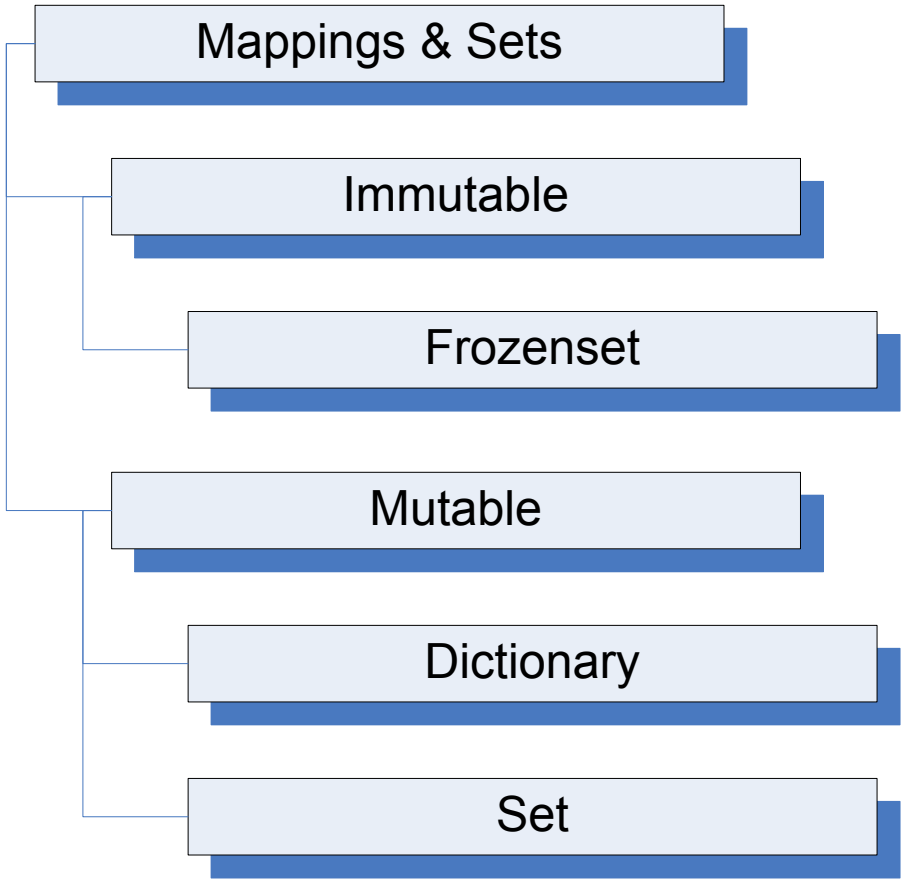
- **ovn0502/manader.py*** Skriv ett program som tar två månadsnummer (1-12) som kommandoradsparametrar. Programmet ska beräkna antalet dagar från och med den första månaden till och med den andra månaden. T.ex. ska parametrarna 3 5 ge resultatet 92 (31+30+31).

```
$ python3 manader.py 1 12
365
$
```

- **ovn0503/datum.py**** Som ovan, fast nu ska man istället ange två datum på formatet d/m, som exempelvis 28/8 24/12 (vilket ger resultatet 119, 4+30+31+30+24).

```
$ python3 datum.py 28/8 24/12
119
$ python3 datum.py 1/1 31/12
365
$
```

Mappings och sets



Mappings och sets - dictionaries

- Använder en nyckel "key" för att slå upp ett värde
  Jämför telefonkatalog namn->telefonnummer
- Motsvarighet till Perl "hashes"
- En sekvens kan bara indexera med numeriska nycklar *a[0], a[1]* osv
- Ett dictionary kan indexera med andra nycklar, t.ex. strängar *a['str']*
- Ingen inbördes ordning (i motsats till sekvenser).
  - Bara relation nyckel->värde
  - Går inte att t.ex. leta reda på femte elementet
  - Ett nytt element hamnar inte nödvändigtvis sist

Mappings och sets - dictionaries

- Tilldelning

```
>>> score = {'Bill':12, 'Steve':19, "Linus":33, "Ken":27}
>>> score
{'Bill': 12, 'Steve': 19, 'Linus': 33, 'Ken': 27}
```

- Element

```
>>> score['Bill']
12
>>> score['Andrew'] = 20
>>> score['Ken'] = 31
>>> score
{'Bill': 12, 'Steve': 19, 'Linus': 33, 'Ken': 31,
  'Andrew':20}
```

Komma åt alla elementen

- Åtkomst

```
>>> list(score.keys())
['Bill', 'Steve', 'Linus', 'Ken', 'Andrew']
>>> list(score.values())
[12, 19, 33, 31, 20]
>>> list(score.items())
[('Bill', 12), ('Steve', 19), ('Linus', 33), ('Ken',
  31), ('Andrew', 20)]

>>> for name in score:
...     print(name, "fick", score[name], "poäng.")
Bill fick 12 poäng.
Steve fick 19 poäng.
Linus fick 33 poäng.
Ken fick 31 poäng.
Andrew fick 20 poäng.
>>> for name, val in score.items():
...     print(name, "fick", val, "poäng.")
```

Mappings och sets - dictionaries

- Uppdatering

```
>>> score['Ken'] = 27
```

- Medlemskap

```
>>>  'Steve' in score
True
```

- Ta bort

```
>>> del score['Andrew']      # ta bort enskilt element
>>> score.clear()            # en tom dictionary {}
>>> del score                # ta bort hela
```

- Antal element

```
>>> score = {"Bill":12,"Steve":17,"Linus":33,"Ken":27}
>>> len(score)
4
>>> len(score.items())
4
>>> len(score.keys())
4
```

Mer om dictionaries

- Tilldelning med *dict()*

```
>>> score = dict(Bill=12, Steve=19, Linus=33, Ken=27)
```

- Tilldelning från en lista av par

```
>>> L = [('Bill', 12), ('Steve', 19), ('Linus', 33),
         ('Ken', 27)]
>>> score = dict(L)
>>> score
{'Bill': 12, 'Steve': 19, 'Linus': 33, 'Ken': 27}
```

- Tom dictionary

```
>>> score = {}
>>> score = dict()
```

- Kopiering med *dict()* - "shallow copy"

```
>>> d2 = dict(d1)          # alternativ: d2 = d1.copy()
```

- Använd copy.deepcopy() för "fullständig" kopiering av referenser

Åtkomst med get

- Man får KeyError om nyckel nas

```
>>> score = dict(Bill=12, Steve=19, Linus=33, Ken=27)
>>> score['Ken']
27
>>> score['Andrew']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Andrew'
```

- Funktionen get returerar None eller defaultvärde om nyckel saknas

```
>>> score.get('Ken')
27
>>> score.get('Andrew')
>>> score.get('Andrew', 0)
0
>>> sorted(score.keys(), key=score.get, reverse=True)
['Linus', 'Ken', 'Steve', 'Bill']
>>>
```

Exempel

- Sortera med avseende på värde i en dict

```
import operator

score = {"Bill":12, "Steve":17, "Linus":33, "Ken":27}

par = list(score.items())
# [('Steve',17),('Ken',27),('Bill',12),('Linus',33)]

par.sort(key=operator.itemgetter(1), reverse=True)
# [('Linus',33),('Ken',27),('Steve',17),('Bill',12)]

print(par[0][0], "vann med", par[0][1], "poäng")
# Linus vann med 33 poäng
```

Mappings och sets - dictionaries

- Kollision vid tilldelning - ingen check görs! Sista elementet gäller

```
>>> d = {'x':1, 'x':2}
>>> d
{'x': 2}
```

- Enbart "immutable" objekt tillåtna som hash (key)
  (även tupler med enbart "immutable" element)

Mappings och sets - set

- Motsavarar den matematiska konstruktionen ”mängd”
  - operationer som union, intersection, subset etc
- Skapas med *variabel = {sekvens av element}*

```
>>> fruit = {'apple', 'orange', 'apple', 'pear'}
>>> fruit
{'apple', 'orange', 'pear'}
>>> 'orange' in fruit
True
>>> 'crabgrass' in fruit
False
>>> fruit.add('lemon')
>>> fruit.remove('orange')
>>> fruit
{'apple', 'lemon', 'pear'}
```

Mappings och sets - set

- Som dict, fast inga värden utan bara nycklar!
- Skillnad mot lista:
  - elementen kan bara förekomma en gång
  - ingen inbördes ordning

```
>>> dudes = [ 'Bill', 'Steve', 'Linus', 'Ken' ]
>>> set(dudes)
{'Bill', 'Ken', 'Steve', 'Linus'}
>>> s = set(dudes)
>>> ' '.join(s)
'Bill Ken Steve Linus'
>>> set()
set()                  # Varför skrivs tom set inte {} ?
>>>
```

Mappings och sets - set

- Exempel 2

```
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                           # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                       # in a but not in b
{'r', 'd', 'b'}
>>> a | b                       # in either a or b
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                       # in both a and b
{'a', 'c'}
>>> a ^ b                       # in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

Mappings och sets - set och frozenset

- Operationer

```
in     not in     ==     !=     <     <=     >     >=
       |    &           -      ^
|=     &=           -=     ^=     (ej frozenset())
```

- Metoder

```
s.union(t)     s.intersection(t) m.fl
```

- Antal element - *len()*

```
>>> s = {1, 2, 3, 4, 3, 2, 1}
>>> len(s)
4
>>> s
{1, 2, 3, 4}
```

- Immutable set: skapa med *frozenset()* istället för *set()*

Dict/Set comprehensions

- Dict Comprehensions
  - {*K(x):V(x)* for *iter_var* in *iterable*}
  - {*K(x):V(x)* for *iter_var* in *iterable* if *cond_expr*}
- Exempel

```
>>> pris = dict(banan=15, apelsin=18, kiwi=24)
>>> med_moms = { k:v*1.25 for k,v in pris.items() }
>>> med_moms
{'kiwi': 30.0, 'apelsin': 22.5, 'banan': 18.75}
```

- Set Comprehensions
  - {*expr* for *iter_var* in *iterable*}
  - {*expr* for *iter_var* in *iterable* if *cond_expr*}

```
>>> { x*2 for x in (1, 1, 2, 2, 3) if x % 2 }
{2, 6}
```

Övning

- **ovn0504/frukter.py** Skapa en dict med fruktnamn och kilopris enligt nedan. Skriv ut frukter och kilopris, sorterat i bokstavsordning på fruktnamnet.

```
$ python3 frukter.py
Banan 21
Plommon 18
Päron 25
Äpple 15
$
```

- **ovn0505/fruktpris.py** Som ovan, fast sortera efter kilopris.

```
$ python3 fruktpris.py
Äpple 15
Plommon 18
Banan 21
Päron 25
$
```

Övning

- **ovn0506/ordfrekvens.py** Skriv ett program som läser in en textfil och beräknar antalet förekomster av varje ”ord” i filen. Utskriften ska vara sorterad så att de mest frekventa orden anges först. Namnet på filen ska tas som en kommandoradsparameter. Versaler och gemener ska betraktas som olika. Förutsätt att filen använder utf-8 som teckenkodning.

```
$ cat fil.txt
Ett och två, tre
och tre och fyra.
$ python3 ordfrekvens.py fil.txt
och 3
tre 2
Ett 1
två, 1
fyra. 1
$
```

Övning

- **ovn0507/ordsnitt.py*** Skapa funktionen `gemensamma_ord` som tar namnen på två textfiler som argument och returnerar en mängd (set) av alla ord som förekommer i båda filerna. Versaler och gemener ska betraktas som olika. Använd utf-8 som teckenkodning.
- **ovn0508/vokaler.py*** Skapa funktionen `antal_vokaler` som tar namnet på en textfil som argument och som returnerar det totala antalet vokaler i filen. Gör inte skillnad mellan vokaler och gemener. Bokstäverna AEIOUYÅÄÖ räknas som vokaler. Använd utf-8 som teckenkodning.

Bokstavsordning

- För *riktig* bokstavsordning, använd metoden `locale.strxfrm`

```
>>> import locale
>>> locale.setlocale(locale.LC_COLLATE, "sv_SE.utf-8")
'sv_SE.utf-8'

>>> L = "En ål och en öl. Och äpple!".split()

>>> L.sort()
>>> print(L)
['En', 'Och', 'en', 'och', 'äpple!', 'ål', 'öl.']

>>> L.sort(key=locale.strxfrm)
>>> print(L)
['en', 'En', 'och', 'Och', 'ål', 'äpple!', 'öl.']
>>>
```

Strängformatering

- Metoden `str.format`

```
>>> '{0} and {1}'.format('spam', 'eggs')
spam and eggs
>>> '{1} and {0}'.format('spam', 'eggs')
eggs and spam

>>> 'This {food} is {desc}'.format(food='spam',
...                                desc='horrible')
This spam is horrible

>>> 'value: {0:.2f}'.format(1/3)
value: 0.33
>>> 'value: {0:6.2f}'.format(5)
value:   5.00

>>> '#{0:<5d}#{1:^5d}#{2:>5d}#'.format(10, 100, 1000)
'#10   # 100 # 1000#'
```

- :d motsvarar heltal, :f motsvarar flyttal, :s motsvarar strängar

Strängformatering av objekt

- Det går att definiera `format` för egendefinierade datatyper/klasser.

```
>>> import datetime
>>> julafton = datetime.date(2019, 12, 24)
>>> idag = datetime.date.today()
>>> print(julafton-idag)
49 days, 0:00:00
>>> '{0:%Y-%m-%d}'.format(idag)
'2019-11-05'
>>> s = 'Idag är {0:%Y-%m-%d}, julafton {1:%d/%m}'
>>> s.format(idag, julafton)
'Idag är 2019-11-05, julafton 24/12'
>>>
```

Strängformatering (andra metoder)

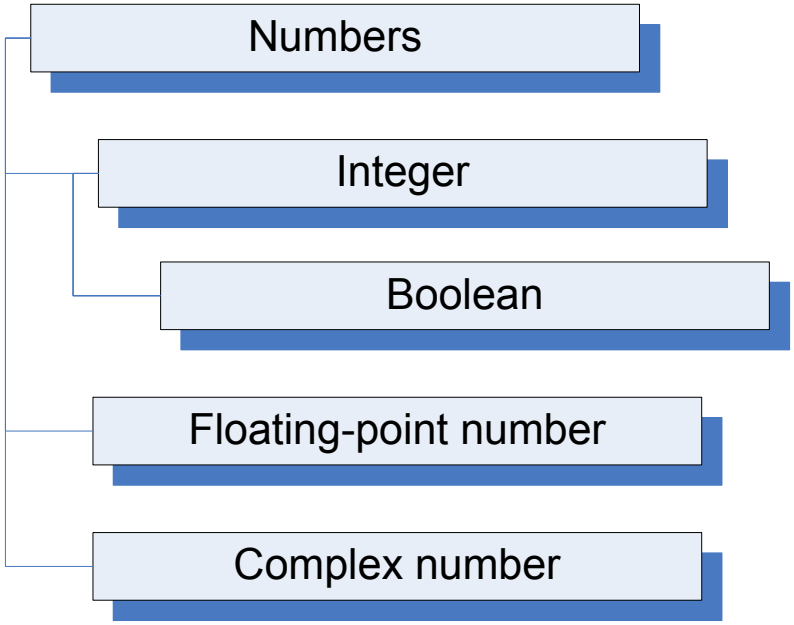
- Från Python 3.6 finns *f-strängar*

```
>>> fr = 'Stockholm'
>>> to = 'Haparanda'
>>> dist = 103.786
>>> f"{fr} till {to} är {dist:.1f} mil"
'Stockholm till Haparanda är 103.8 mil'
>>> f'Från {idag:%Y-%m-%d} till {julafton:%d/%m}.'
'Från 2019-11-05 till 24/12.'
```

- I äldre Python: *format_sträng % (argument_att_konvertera)*

```
>>> "%s till %s är %.1f mil" % (fr, to, dist)
'Stockholm till Haparanda är 103.8 mil'
```

Tal


Numbers
Integer
Boolean
Floating-point number
Complex number

Tal - heltal

- Heltal kan skrivas som decimala, hexadecimala, oktala eller binära

```
>>> 23
23
>>> 0x23
35
>>> 0o23
19
>>> 0b1111
15
```

  - *bin()* konverterar till binär representation
  - *hex()* konverterar till hexadecimal representation
  - *oct()* konverterar till oktal representation
  - *int()* konverterar till heltal, även strängar med decimal notation
- Heltal i Python har obegränsad storlek (tills minnet tar slut)

```
>>> 2**9999

99753155844037919244187108134179254191174841594309622742600
44749264719415110973315959980842018097298949665564711604562
13577824567470689055879689296604816197892786502339689726338
26232756330299477602750434590966557712543042303090523427545
37433044812444045244947419004626970816628925310784154736951
27845619403261254832193722052337993581349272661143426908084
71578878148203814184403803661142675458207380919781907294847
31949705420480268133910532310713666697018262782824765301571
34011748470016796715832572964888663983288780308629...
```

Tal - boolean

- Boolean - *True* eller *False*
  - Från version 2.3 separat typ
  - En subklass av Integer
  - Alla objekt har ett 'inbyggt' True eller False - *bool()* konverterar
    - 0, 0.0, {}, (), “” ger False
    - Även objektet *None* ger False
    - *any(lista)* returnerar True om ett element i listan/tupeln är True
    - *all(lista)* returnerar True om alla element i listan/tupeln är True

```
>>> bool(1)
True
>>> bool(0)
False
>>> bool(2)
True
>>> print('%d' % True)
1
```

Tal - flyttal

- Flyttal - motsvarar C double
  - IEEE754 8byte (64bit) representation
  - Punkt "." markerar att det är ett flyttal
  - "e" eller "E" markerar "scientific notation", dvs 23e2=2300.0, 23e-2=0.23

```
0.0     -12.     4.44     23e2     2300.0     23e-2     .23
```

  - *float()* konverterar till flyttal

Tal - komplexa tal

- Komplexa tal
  - j = √-1
  - Två flyttal; en real + en imag; syntax: *real* + *imag*j (eller J)

```
0+1j     -2.3-4.5j     23.23e2-12e-34j
```

  - *complex()* konverterar till komplext tal
  - Attribut:

```
>>> a = 23+12j
>>> a.real
23
>>> a.imag
12
>>> a.conjugate()
(23-12j)
```

Tal - operatorer och funktioner

- Standardoperatorer för matematik:

  `+, -, *, /, //, %, **`

- Heltalsdivision `//` avrundar alltid nedåt:

  `1/2 == 0.5,  1//2 == 0,  -1//2 == -1`

- Bit-operatorer (enbart heltal):

  `~  - invertering`
  `<< - vänstershift`
  `>> - högershift`
  `&  - bitvis AND`
  `|  - bitvis OR`
  `^  - bitvis XOR`

- Inbyggda funktioner
  - `abs` `# abs(-17.2) == 17.2`
  - `divmod` `# divmod(17, 5) == (3, 2)`
  - `pow` `# pow(2, 5) == 32`
  - `round` `# round(3.7) == 4`

Tal - övrigt

- Returnera unicode-tecken för heltal (och tvärtom)
  - `chr(97)` => `'a'`
  - `ord('a')` => `97`
- `int('123',4)` returnerar decimala representationen för 123 i basen 4
- Modulen `math` för `sqrt()`, `pi`, `sin()`, `cos()` etc.
- Modulen `random` för slumptal
- Modulen `decimal` för exakt decimal representation
- Modulen `fractions` för exakt representation av rationella tal, t.ex. 2/3