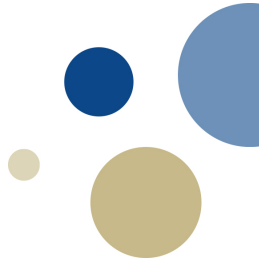
Iterables

Iterables er generelt grupper med objekter man kan *iterere* seg gjennom.

Slicing

Slicing er en måte å indeksere en bestemt del av en iterable, hovedsakelig lister, strenger og tupler.

Slicing



Slicing er en måte å indeksere en bestemt del av en iterable, hovedsakelig lister, strenger og tupler. Vi bruker firkantparenteser, siden vi aksesserer, og samme syntaks som en *range* for å bestemme hva vi vil aksessere.

```
iterable[fra_og_med_start:til_men_ikke_med_slutt:steg]
```

Slicing

Dersom vi vil starte på start, *trenger man ikke skrive noe før første kolon*.

```
iterable[:til_men_ikke_med_slutt:steg]
```

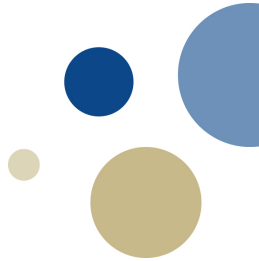
Dersom vi vil slutte på, inklusiv, slutten, *trenger man ikke skrive noe mellom første og andre kolon*.

```
iterable[fra_og_med_start::steg]
```

Dersom vi vil bruke 1 som *steg*, *trenger man ikke skrive noe etter siste kolon*, og man *trenger ikke ha med siste kolon heller*.

```
iterable[fra_og_med_start:til_men_ikke_med_slutt:]
iterable[fra_og_med_start:til_men_ikke_med_slutt]
```

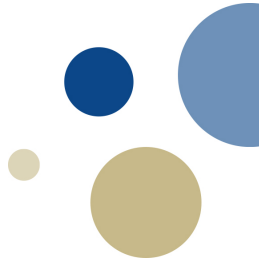
Slicing



Grunnen til at slicing går fra og med, til men ikke med, er at da kan man dele opp en iterable med samme indeks.

```
lst[:ind] + lst[ind:] == lst # -> True
```

Slicing



Dette kan f.eks. brukes for kjapp kopiering eller reversering av lister. Slicing returnerer **nye** objekter, i motsetning til f.eks. *lst.reverse()*.

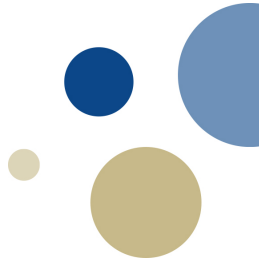
```
kopi = lst[:]
rev  = lst[::-1]
```

Slicing

Vi kan også bruke slicing til å korte ned denne koden fra tidligere.

```
def insert_string(original, index, new_string):
    # antar at index er valid
    resultat = "" # starter med tom streng
    for i in range(len(original)):
        if i == index:
            resultat += new_string
        # om vi plusser foran eller bak new_string
        # bestemmer om vi inserter foran eller bak.
        resultat += original[i]
    return resultat
```

Slicing



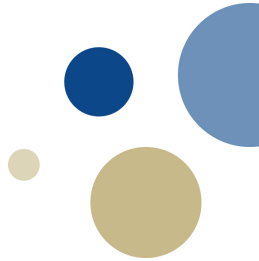
Vi kan også bruke slicing til å korte ned denne koden fra tidligere.

```
def insert_string(original, index, new_string):
    # antar at index er valid
    return original[:index] + new_string + original[index:]
```

Dictionaries

Dictionaries er kanskje den nyttigeste datastrukturen i Python, og en av de mest brukte.

Dictionaries



Dictionaries er kanskje den nyttigeste datastrukturen i Python, og en av de mest brukte. Det som er viktig å huske på er at en dictionary *kobler* sammen ulike verdier.

Dictionaries

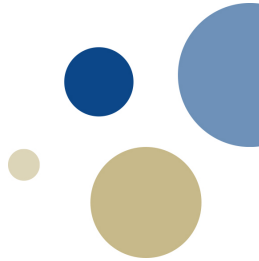
Dictionaries er kanskje den nyttigeste datastrukturen i Python, og en av de mest brukte. Det som er viktig å huske på er at en dictionary *kobler* sammen ulike verdier.
Vi kobler fra en *nøkkel* til en *verdi*.

Dictionaries

Syntaksen fungerer i prinsipp på samme måte som en liste. Vi bruker firkantparenteser til å aksessere *verdien* som ligger på en *nøkkel*.

```
dictionary = {"katt": 1, "hund": 2}
print(dictionary["katt"]) # gir 1
```

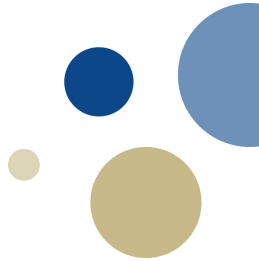
Dictionaries



Kan forenkle problemer som kan være vanskelige med lister.
Eksempel: Telle opp antall bokstaver i en streng.

```
streng = "heihei"
# alle bokstaver fra a til z
bokstaver = [chr(bokstav) for bokstav in range(ord("a"),
                                ord("z") + 1, 1)]
oppteller = [0] * len(bokstaver)
for i in range(len(bokstaver)):
    oppteller[i] = streng.count(bokstaver[i])
```

Dictionaries



Kan forenkle problemer som kan være vanskelige med lister.
Eksempel: Telle opp antall bokstaver i en streng.
Kan være vanskelig å håndtere dette, siden vi har to separate lister.
Finnes selvsagt andre løsninger, men for å få en direkte kobling mellom bokstav og antall er det enklest med dictionary.

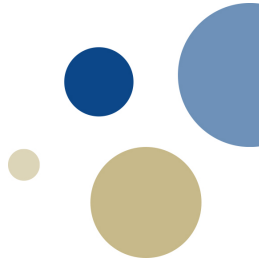
```
streng = "heihei"
bokstaver_til_antall = {}
for bokstav in streng:
    antall = bokstaver_til_antall.setdefault(bokstav, 0)
    bokstaver_til_antall[bokstav] = antall + 1
```

Dictionaries

Metoder

Metode	Eksempel
clear	dictionary.clear()
copy	kopi = dictionary.copy()
fromkeys	dict.fromkeys(keys, values)
get	value = dictionary.get(key)
items	t_list = dictionary.items()
values	verdier = dictionary.values()

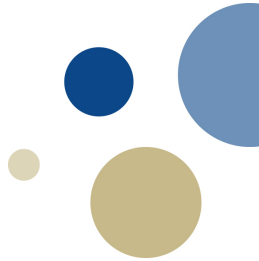
Dictionaries



Metoder (Cont.)

Metode	Eksempel
keys	nøkler = dictionary.keys()
pop	dictionary.pop(key)
popitem	dictionary.popitem()
setdefault	value = dictionary.setdefault(key, default)
update	dictionary.update(iterable)
update	dictionary.update(other_dict)

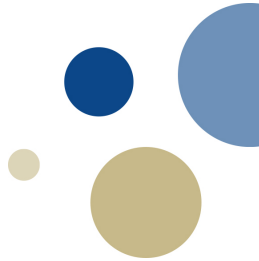
Dictionaries



Kombineres godt med for-løkker. Avhengig av hva du trenger, har vi flere metoder for å iterere oss gjennom en dictionary.

```
dictionary = {"hello": "world", 12: 9}
```

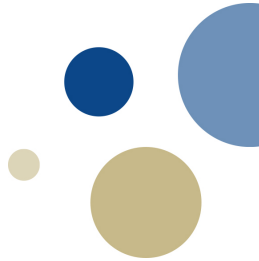
Dictionaries



Kombineres godt med for-løkker. Avhengig av hva du trenger, har vi flere metoder for å iterere oss gjennom en dictionary.
Dersom vi kun er interessert i verdiene, og ikke nøklene (ofte hvis noe kun skal printes ut):

```
dictionary = {"hello": "world", 12: 9}
for value in dictionary.values():
    print(value)
```

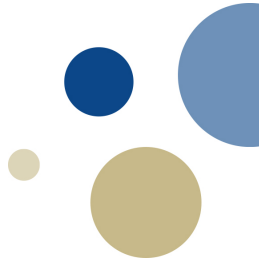
Dictionaries



Kombineres godt med for-løkker. Avhengig av hva du trenger, har vi flere metoder for å iterere oss gjennom en dictionary.
Dersom vi kun er interessert i nøklene, og ikke verdiene, eller ønsker å aksessere verdiene med firkantparenteser:

```
dictionary = {"hello": "world", 12: 9}
for key in dictionary.keys():
    print(key, dictionary[key])
```

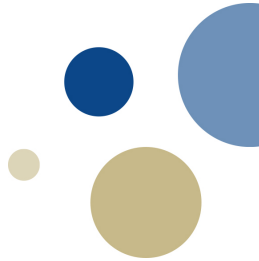
Dictionaries



Kombineres godt med for-løkker. Avhengig av hva du trenger, har vi flere metoder for å iterere oss gjennom en dictionary.
Dersom vi er interessert i både verdi og nøkler, som vi kanskje oftest er:

```
dictionary = {"hello": "world", 12: 9}
for key, value in dictionary.items():
    print(key, value)
```

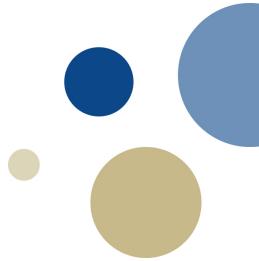
Dictionaries



Husk: det er lett å gå fra nøkkel til verdi, men vanskelig å gå fra verdi til nøkkel:

```
# value to key
dictionary = {"hello": "world", 12: 9}
key = "hello"
value = dictionary[key]
```

Dictionaries



Husk: det er lett å gå fra nøkkel til verdi, men vanskelig å gå fra verdi til nøkkel:

```
# value to key
dictionary = {"hello": "world", 12: 9}
key = None # ingen key enda, setter den til None
value = "hello"
for current_key, current_value in dictionary.items():
    if current_value == value:
        key = current_key
        break # bare en key pr. value
```

Sets

Det finnes også set, som ligner veldig på lister. Disse kan kun ha unike verdier, som er egentlig eneste grunn til å bruke disse. Problemer man kan løse med set kan også løses med lister, men man kan få kortere (men muligens mindre leslige) løsninger.