TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY

Programmeerimise süvendatud algkursus ITI0140

2014

Teemad

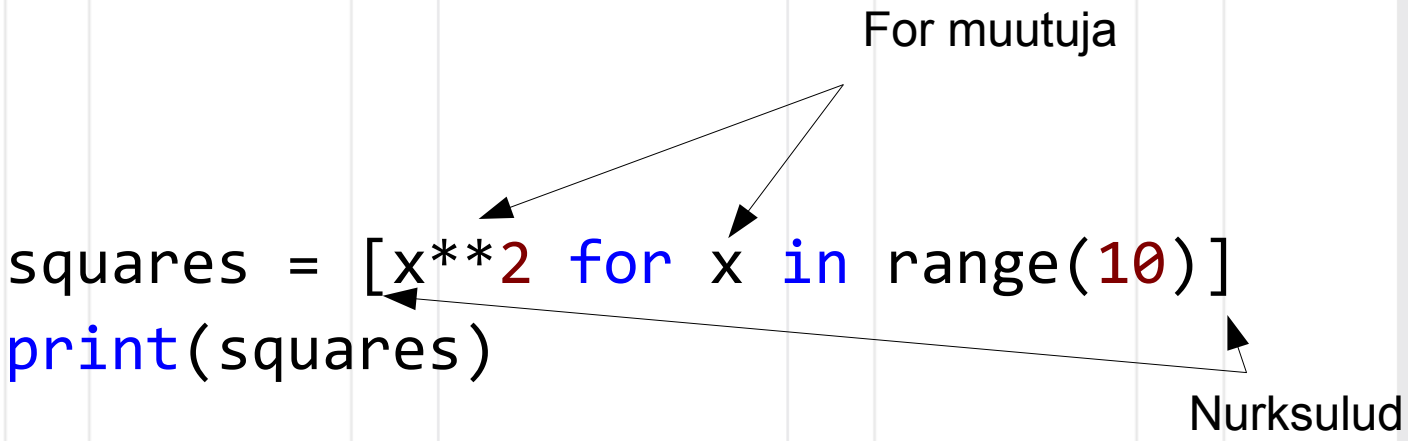
- Järjendi konstrueerimine (ingl *list comprehension*)
- Lambda funktsioon (ingl *lambda function*)
- Generaator (ingl *generator*)

Järjendi konstrueerimine

"Klassikaline" for tsükkel

```
squares = []
for x in range(10):
    squares.append(x**2)
print(squares)
>>> [0, 1, 4, 9, 16, 25, 36, 49, 64,
81]
```

Järjendi konstrueerimine




```
>>> [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Järjendi konstrueerimine

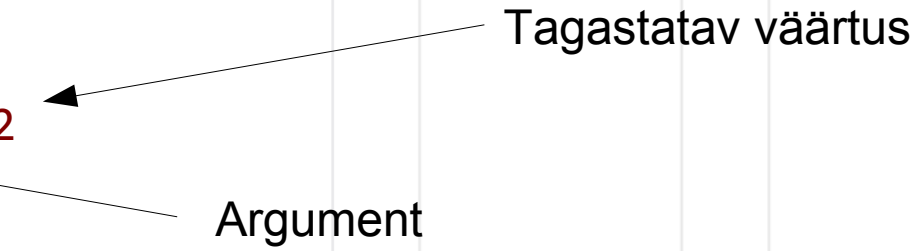
```
l = [(x, y) for x in [1,2,3] for y in
[3,1,4] if x != y]

print(l)
```

If-lause filtreerimiseks

```
 >>> [(1, 3), (1, 4), (2, 3), (2, 1), (2, 4),
(3, 1), (3, 4)]
```

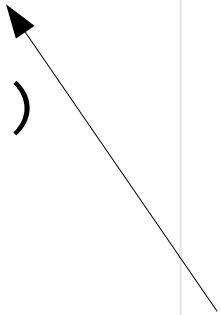
Lambda (anonüümne funktsioon)

```
def f(x):
    return x**2
print(f)
print(f(8))
<function f at 0x00000000026BF6A8>
>>> 64

g = lambda x: x**2
print(g)
print(g(8))

>>> <function <lambda> at 0x00000000026BF598>
 64
```



Lambda mitme argumendiga

```
g = lambda x, y, z: x**2 + y**3 +
z**4
print(g(2, 3, 4))

>>> 287
```



Mitu argumenti

Filter (filtreerimine)

```
l = [2, 18, 9, 22, 17, 24, 8, 12, 27]
print(filter(lambda x: x % 3 == 0, l))
```

```
>>> [18, 9, 24, 12, 27]
```

Iterable (nt järjend, hulk jne)

Filtrifunktsioon, mida rakendatakse igale elemendile

Map (operatsioonid väärtustega)

```
l = [2, 18, 9, 22, 17, 24, 8, 12, 27]
print(map(lambda x: x * 2 + 10, l))
```

Iterable

Funktsioon, mida rakendatakse igale elemendile

```
>>> [14, 46, 28, 54, 44, 58, 26, 34, 64]
```

Generaator

```
def gen():
    yield 3
    yield 5
    yield 9

def main():
    x = gen()
    for i in range(4):
        print(next(x))
```

NB! Mitte return

Küsime generaatorilt järgmise väärtuse

```
>>> 3
>>> 5
>>> 9
Traceback (most recent call last):
    print(next(x))
StopIteration
```

NB! next() ei kasuta i väärtust

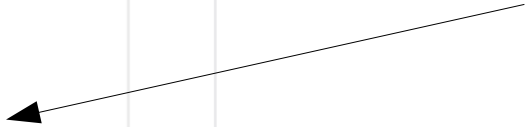
Generaator

```
def reverse(data):
    for index in range(len(data)-1, -1, -1):
        yield data[index]

for char in reverse("golf"):
    print(char, end=".")
```

```
>>> f.l.o.g.
```

Generaator

```
def counter (maximum):
    i = 0
    while i < maximum:
        yield i
        i += 1
```



```
x = list(counter(4))
print(x)
```

```
>>> [0, 1, 2, 3]
```

NB! Kui generaator genereerib palju väärtusi, siis võib järjend väga suureks minna, sest kõik väärtused laetakse mällu (generaatori eelised kaovad)

Generaator

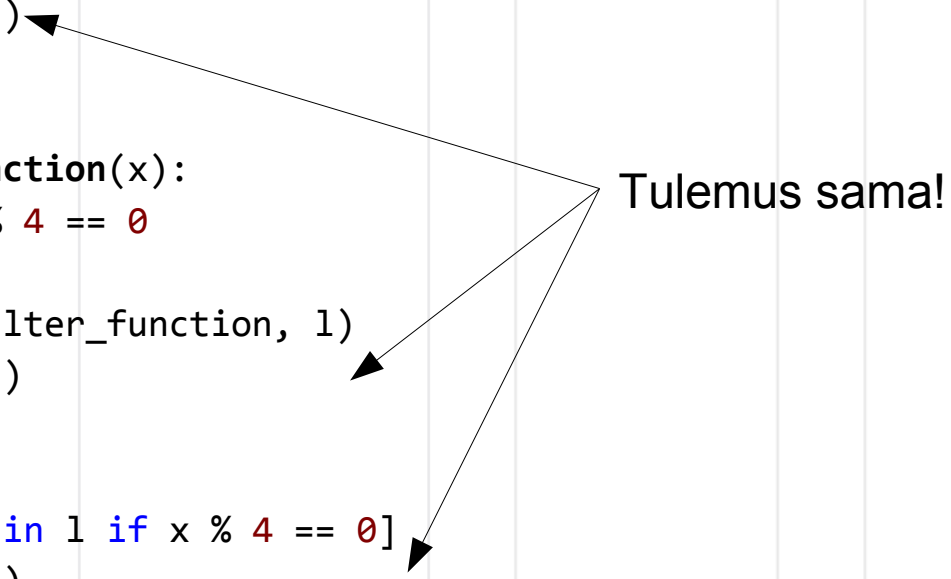
```
def counter (maximum):
    i = 0
    while i < maximum:
        yield i
        i += 1


x = counter(4)
for i in x:
    print(i)
```

```
>>> 0
1
2
3
```

Küsime generaatorilt ühe väärtuse, teeme sellega midagi, siis küsime järgmise väärtuse

Millal siis mida kasutada?

```
l = [1, 2, 3, 4, 5, 6, 7, 8, 9]

m4 = filter(lambda x: x % 4 == 0, l)
print(list(m4))


def filter_function(x):
    return x % 4 == 0

m4 = filter(filter_function, l)
print(list(m4))


m4 = [x for x in l if x % 4 == 0]
print(list(m4))

>>>> [4, 8]
[4, 8]
[4, 8]
```



Lambda tagastamisel

```
def multiply(n):
    print("n =", n)
    return lambda x: x * n

f = multiply(3)
print(f)
print(f(4))
print(f(5))
```

Mõnes kohas on lambda kasutamine lühem, kavalam, mugavam, loetavam ja/või kiirem!

Sama kehtib generaatori ja listi konstrueerimise kohta.

```
>>>> n = 3
<function multiply.<locals>.<lambda> at
0x000000000260FEA0>
12
15
```

Ülesanne: lambda

The Hound of the Baskervilles:
http://www.gutenberg.org/cache/epub/3070/pg3070.txt

Kirjutada generaator, mis tagastaks uue **sõna** iga järjestikuse väljakutse peale (generaator loeb failist vajaliku osa sümbolhaaval).

Kasutades tehtud generaatorfunktsiooni, konstrueerida **järjend** (list comprehensioni abil), mis sisaldab igat **neljandat** sõna (esimesed **100** tk).

Rakendada sellele järjendile **lambda** funktsiooniga **filtrit**, mis väljastaks sellest järjendist sõnad, mille pikkus on **suurem** kui antud järjendi **keskmine sõnapikkus**. Väljastada sõnad **suure** algustähega.