TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY

Programmeerimise süvendatud algkursus ITI0140

2015

Teadaanne

On tulemas kaks programmeerimisvõistlust

Minski võistluse eelvõistlus

Esmaspäeval, **12. oktoobril 17:00 – 22:00**

IEEE Xtreme (24h)

Laupäeval (ööl reede vastu laupäeva), **24. oktoober 03:00 kuni 25. oktoober 03:00**

Lisainfo

https://courses.cs.ttu.ee/pages/ProgComp

Teemad

- Failidest lugemine
- Failidesse kirjutamine
- Funktsioonid (kirjutamine, kasutamine, Zen of Python)
- Ülesanne

Failidest lugemine - “with”

Püütonlik lähenemine failidest lugemiseks kasutab ära kavala konstruktsiooni nimega “**with**”.

With võtab kokku sellised sammud:

```
set things up
try:
    do something
finally:
    tear things down
```

Erinditöötlus (ingl *exception handling*)

Kui midagi juhtub (erind ~ viga) **või** *try*-plokk saab läbi, käivitatakse *finally*-plokk

Failidest lugemine - “with”

“**with**” võtab põhimõtteliselt kokku järgmised sammud:

```
file = open("scarlet.txt")
for line in file:
    print(line)
file.close()
```

Samaväärne on kirjutada

```
open("scarlet.txt", mode = "r")
```

“reading” mode

```
with open("scarlet.txt") as file:
    for line in file:
        print(line)
```

with paneb faili ise kinni
(ei pea *close()* tegema)

Faili kirjutamine

“writing” mode (kirjutab faili üle iga kord ja tekitab faili kui faili pole)

Faili kirjutamine on analoogne:

```
with open("output.txt", "w") as file:
    file.write("Hi there!")
```

Kui on vaja kirjutada faili juurde, siis saab kasutada “a” (“appending” mode)

Koodi (funktsioonide) “ilu” mõiste

“What Makes Code Beautiful?”
Marcel Molina – Twitter, software engineer
(RubyCon 2007)

Ilu koosneb kolmest komponendist:

- Proportsionaalsus
- Sobivus
- Arusaadavus

Funktsioonide “ilu” mõiste

<u>Proportsionaalsus</u>

Funktsioon peab täitma oma ülesande, olles ise minimaalne.

Metafoor/näide: Inimese käsi arvatavasti toimiks enamvähem sama edukalt, kui käsi oleks näiteks kolm korda suurem ja sõrmede omavaheline suuruste suhe oleks sama. Selline käsi oleks palju raskem ja ebasobiv ülejäänud kehaga võrreldes.

Funktsioonide “ilu” mõiste

Sobivus

Funktsioon peab täitma just selle ülesande, mille jaoks see funktsioon on loodud.

Metafoor/näide: Kristallhaamer – “ilus” objekt, mis küll pealtnäha sobib haamriks, aga tegelikult mitte.

Funktsioonide “ilu” mõiste

<u>Arusaadavus</u>

Funktsioon peab olema üheselt arusaadav – mis on funktsiooni eesmärk ja kuidas see saavutatakse.

Metafoor/näide: Keerukus keerukuse pärast ei ole hea asi – kui midagi saab lihtsustada ja muuta arusaavamaks, siis seda tuleb kindlasti teha. Keeruline ja arusaamatu kood ei ole “ilus”.

Van Rossumi ütlus: **“Koodi loetakse mitmeid kordi rohkem kui seda kirjutatakse.”**

Arusaadavus - näide

Mida teeb järgnev funktsioon?

```
def alternating(l):
    return [] if not l else
[max(l)] + ([min(l)] if len(l) > 1
else []) + alternating([n for i, n
in enumerate(l) if i not in
[lst.index(min(l)),
lst.index(max(l))]])
```

Arusaadavus - näide

```
def alternating(input_list):
    """
    Return a list of alternating highest and lowest values.
    (e.g., [1, 7, -1, 2] => [7, -1, 2, 1])

    Arguments:
    input_list -- unsorted list of numeric values
                  (e.g., [1, 7, -1, 2])

    Returns:
    A list of alternating highest and lowest values.
    """
    sorted_list = sorted(input_list, reverse=True)
    ret = []
    while len(sorted_list) > 0:
        ret.append(sorted_list[0])
        if len(sorted_list) > 1:
            ret.append(sorted_list[-1])
        sorted_list = sorted_list[1:-1]
    return ret
```

Funktsioonid

Miks on järgnev kood halb?

```
# operate with file 1
with open("scarlet.txt") as file:
    for line in file:
        # do something with the data
        # ....

# operate with file 2
with open("hound.txt") as file:
    for line in file:
        # do the same exact thing with the data
        # ....
```

Funktsioonid

- Funktsioonid aitavad koodi kordumist vähendada – DRY (Don't Repeat Yourself) põhimõte
- Koodi rühmitamine loogilistesse plokkidesse
- Järgnev arhitektuur on juba parem

```
def process_file_contents(file):
    for line in file:
        # a lot of stuff
        pass
# operate with file 1
with open("scarlet.txt") as file:
    process_file_contents(file)
# operate with file 2
with open("hound.txt") as f:
    process_file_contents(file)
```

Funktsioonid

Järgnev arhitektuur on veel parem

```
def process_file_contents(file):
    for line in file:
        # a lot of stuff
        pass

def process_file(filename):
    with open(filename) as file:
        process_file_contents(file)

# operate with file 1
process_file("scarlet.txt")

# operate with file 2
process_file("hound.txt")
```

Millal kirjutada funktsioone?

- Kui teie funktsioon võiks veel kuskil kasuks tulla
- Kui teie funktsioon muutub pikkuse tõttu arusaamatuks

Mõistlik funktsiooni pikkus on 10-20 rida.

Funktsioon peab täitma üht selget ülesannet.

Funktsiooni maksimaalne ridade arv ei ole kivisse raiutud - siin tuleb kasuks kogemus.

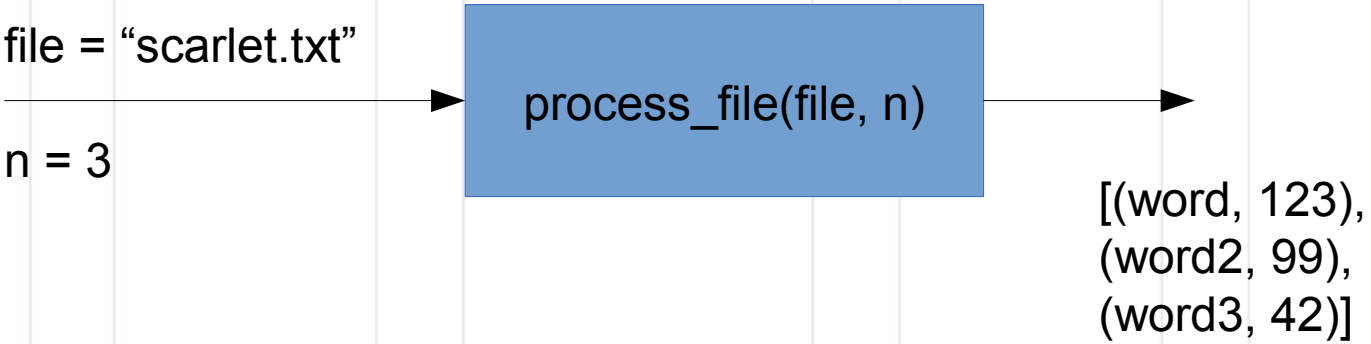
Üheks märgiks, et funktsioon on liiga keeruline on mitmetasemeline taane (tsüklid tsüklite sees).

Näide

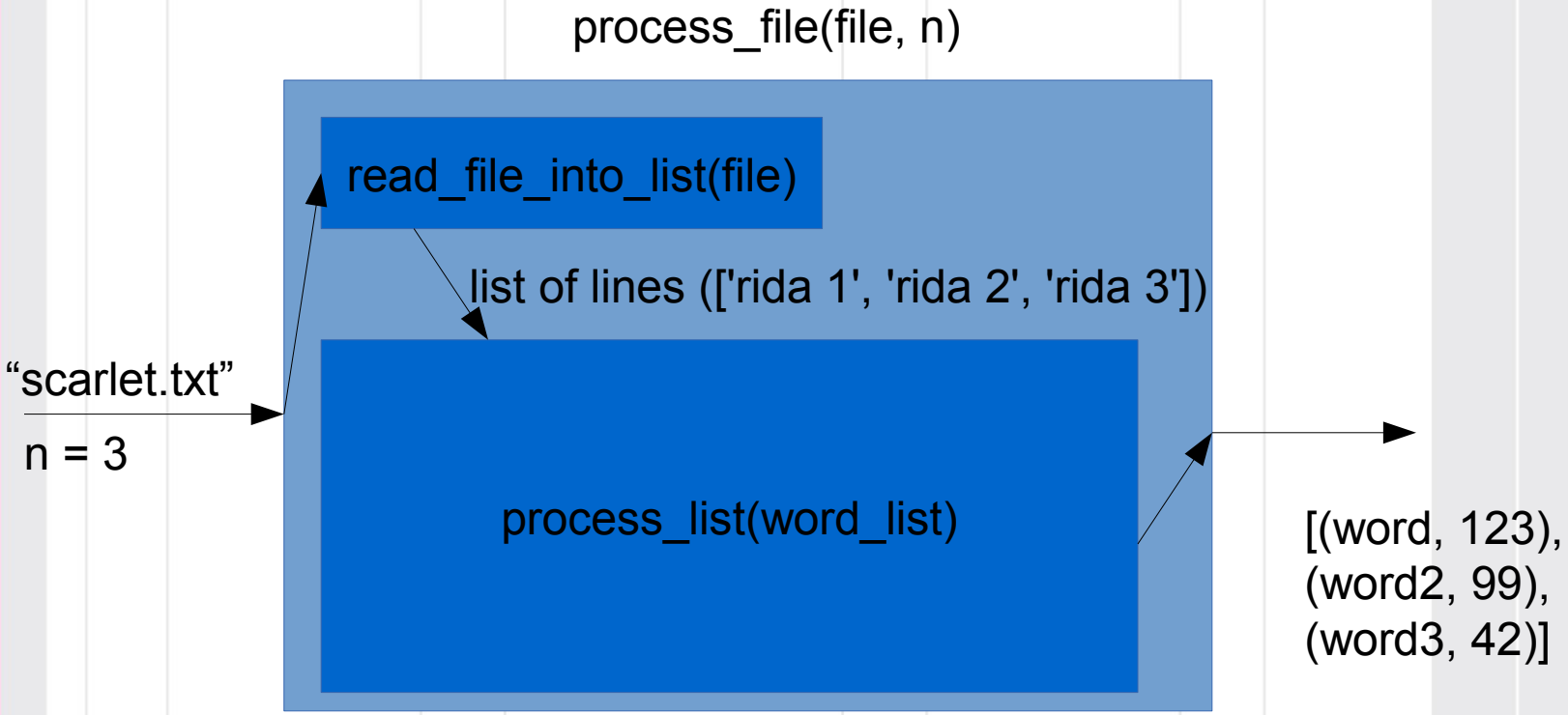
Ülesanne

Lugeda sisse **fail** ja väljastada järjend 2-korteežidest (st [(sõna, esinemisarv), (sõna2, esinemisarv2), ...]) ***n*** kõige sagedamini esinevast sõnast koos esinemiste arvuga.

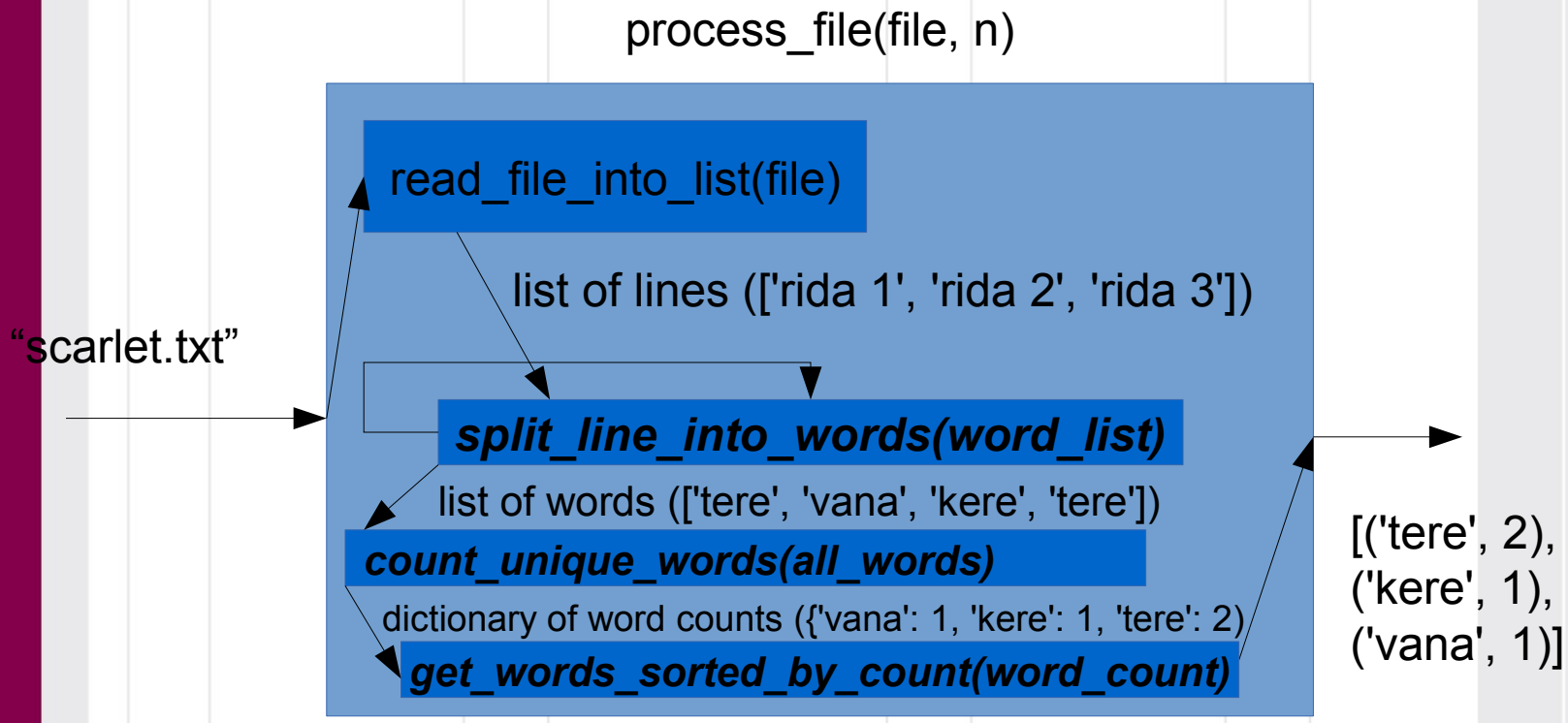
Näide - arhitektuur



Näide - arhitektuur



Näide - arhitektuur



Näide funktsioonide kasutamisest

```
def process_file(file, n):
    """
    Read words from file and return the n most frequent words.

    Arguments:
    file -- file name (e.g., "scarlet.txt")
    n -- number of top words to return (e.g., 3)

    Returns:
    A list of 2-tuples with most frequent words (e.g., [('hello',
123), ('world', 42), ... ]
    """
    lines = read_file_into_list(file)
    all_words = []
    for line in lines:
        all_words += split_line_into_words(line)
    word_count = count_unique_words_in_list(all_words)
    return get_words_sorted_by_count(word_count, n)

print(process_file("scarlet.txt", 3))
```

```
[('the', 2294), ('and', 1264), ('of', 1220)]
```

Easter egg

Tehke lahti Pythoni shell (python.exe) ja kirjutage sinna

>>> **import this**

```
   import this
   """The Zen of Python, by Tim Peters. (poster by Joachim Jablon)"""

 1 Beautiful is better than ugly.
 2 Explicit is better than impl..
 3 Simple is better than cσмplεx.
 4 Cσмplεx is better than cOmp1|c@ted.
 5 Flat is better than nested.
 6 S p a r s e  is better than dense.
 7 Readability counts.
 8 Special cases aren't special enough to break the rules.
 9 Although practicality beats purity.
10 raise PythonicError("Errors should never pass silently.")
11 # Unless explicitly silenced.
12 In the face of ambiguity, refuse the temptation to guess.
13 There should be one-- and preferably only one --obvious way to do it.
14 # Although that way may not be obvious at first unless you're Dutch.
15 Now is better than ...                    never.
16 Although never is often better thanrightnow.
17 If the implementation is hard to explain, it's a bad idea.
18 If the implementation is easy to explain, it may be a good idea.
19 Namespaces are one honking great idea -- let's do more of those!
```

Ülesanne

Ülesanne on nähtaval

- **https://ained.ttu.ee**
- **https://courses.cs.ttu.ee/pages/ITI0140**