Telekommunikációs Hálózatok

1. gyakorlat

Követelmények

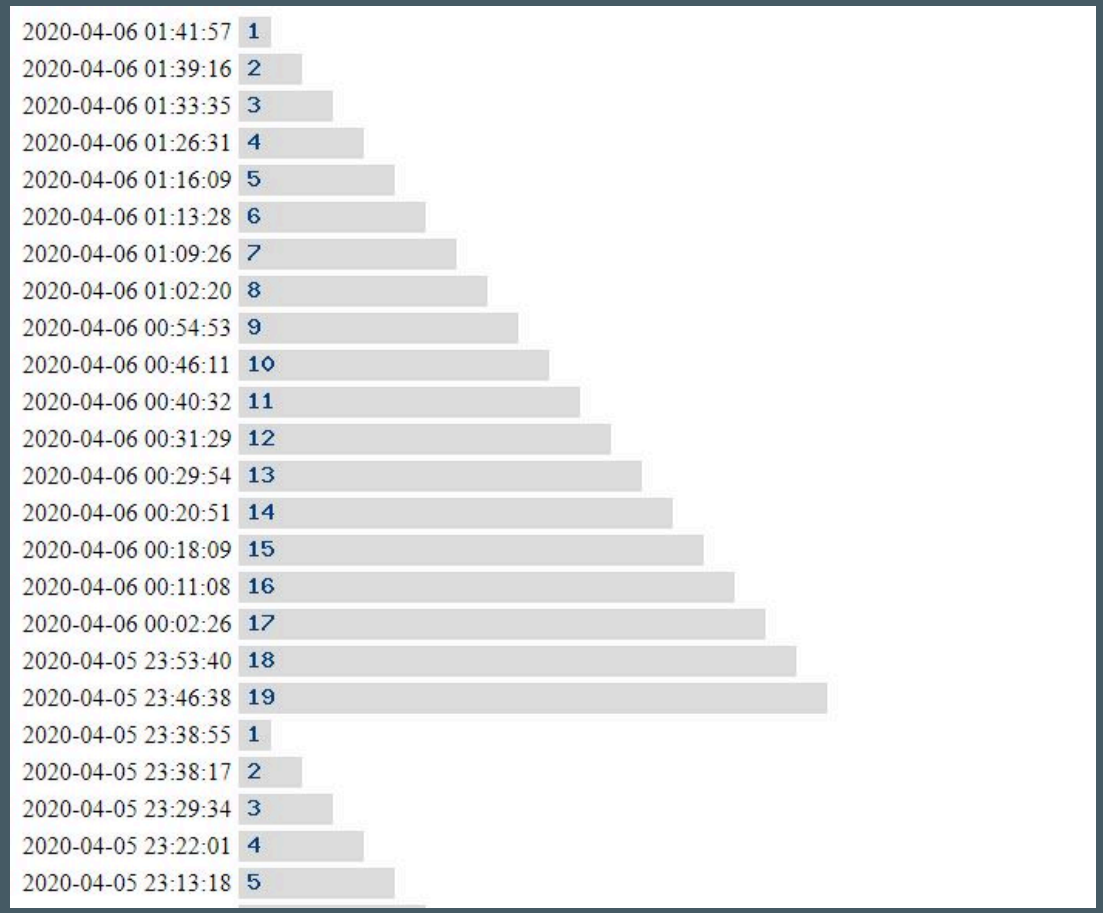
- Maximum 4 hiányzás
- Számonkérések:
  - Socket beadandók (4 db, min 50%)
  - Socket ZH (min 50%)
  - Mininet beadandó (min 50%)

Socket beadandók

- Programozási, szimulációs feladat
- 3 hét a határidő
- TMS-en keresztül kell leadni, ami értékelni fogja és figyeli a kódhasonlóságot
- Az eredményt megjegyzésbe rakja, időnként fut le a tesztelő
- Másolt kód leadása csalásnak minősül és az egyetemi szabályoknak megfelelően járunk el

Deadline Panic

Ne halogass!



Az értékelő feldolgozási sora, határidő környékén

Ponthatárok

$$\% = \frac{beadando\%}{3} + \frac{mininet\%}{3} + \frac{ZH\%}{3}$$

Százalék	Érdemjegy
0 - 49%	Elégtelen(1)
50 - 59%	Elégséges(2)
60 - 74%	Közepes(3)
75 - 84%	Jó(4)
85 - 100%	Jeles(5)

Témakörök

- Python alapok
- Socket programozás
- CRC, kódolások, MD5
- Tűzfalak: Iptables
- Wireshark/tcpdump forgalom elemzés.
- Mininet, hálózati karakterisztikák, alapvető eszközök: traceroute, ping
- MAC learning, STP, ARP, routing beállítások
- Port forwarding, VLAN beállítások
- Tunneling megoldások IPv4/IPv6

Python történelem

Guido Van Rossum, ’96-ban:

“Over six years ago, in December 1989, I was looking for a”hobby” programming project that would keep me occupied during the week around Christmas. My office … would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python’s Flying Circus).”

Python

- Könnyű tanulásra lett tervezve
  - Tiszta, egyszerű szintaxis, kevés, rövid kulcsszó
- Hordozható:
  - Majdnem mindenen elfut (linux, windows, mac, Raspberry Pi,)
- Whitespace-t használ a blokkok elkülönítéséhez
  - Az olvashatóság miatt amúgy is így kéne
- A változókat nem szükséges deklarálni
  - Ettől még nem típus-független nyelv
- Verziók
  - python 3.x (laborgépeken: python, py)
  - python2 DEPRECATED

Zen of python

```
# import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Python REPL(Read-Eval-Print Loop)

Indítás

```
$ python
$ py # laborgépeken
```

Hello World:

```
print("Hello world!")
user_name = "Foo"
print("Hello " + user_name)
user_age = 25
print("You are " + str(user_age) + " years old.")
```

```
Hello world!
Hello Foo
You are 25 years old.
```

Egyszerű számítások

```
print(10+2)
print(2*2)
print(3**2)
print(10%2)
```

```
12
4
9
0
```

Matematikai kerekítések

```
import math
print(math.floor(3.8))
print(round(3.8))
print(round(3.8,1))
```

```
3
4
3.8
```

Változók

```
a = 42
b = 32
c = a + b
print(c)
c = 'valami'
print(a+c)
```

74

TypeError: unsupported operand type(s) for +: 'int' and 'str'

String műveletek

Mindegy, hogy '-t vagy "-t használsz, csak azt konzisztensen

```
print("alma".upper())
print("LO" in "Hello".upper())
print("Decimal Number: %d, Float: %f, String: %s" % (12,33.4,"almafa"))
x = 42
print(f"x == {x}")
```

```
ALMA
True
Decimal Number: 12, Float: 33.400000, String: almafa
x == 42
```

List

```
players = [12, 31, 27, '48', 54]
print(players)
print(players[0])
print(players[-1])
print(players + [22, 67])
print(len(players))
```

```
[12, 31, 27, '48', 54]
12
54
[12, 31, 27, '48', 54, 22, 67]
5
```

List

```
players = [12, 31, 27, '48', 54]
players.append(89)
print(len(players))
print(players[2:])
```

```
6
[27, '48', 54, 89]
```

Tuple – immutable List

Nem módosítható

```
players = (12, 31, 27, '48', 54)
print(players[2:])
players[2] = 'alma'
del players[2]
```

(27, '48', 54)

TypeError: 'tuple' object does not support item assignment

Set

```
mylist = [8,3,2,3,2,4,6,8,2]
myset = set(mylist)
print(mylist)
print(myset)
mysortedlist = sorted(mylist)
print(mysortedlist)
```

```
[8, 3, 2, 3, 2, 4, 6, 8, 2]
{2, 3, 4, 6, 8}
[2, 2, 2, 3, 3, 4, 6, 8, 8]
```

Dictionary

```
team = {
  91: "Ayers, Robert",
  13: "Beckham Jr,",
  3:  "Brown, Josh",
  54: "Casillas, Jonathan",
  21: "Collins, Landon"
}
print(len(team))
team[3] = "Chihiro"
print(91 in team)
print("alma" in team)
```

```
5
True
False
```

Dictionary

```
team = {
  91: "Ayers, Robert",
  13: "Beckham Jr,",
  3:  "Brown, Josh",
  54: "Casillas, Jonathan",
}
print(team.keys())
print(team.values())
```

```
dict_keys([91, 13, 3, 54])
dict_values(['Ayers, Robert', 'Beckham Jr,', 'Brown, Josh', 'Casillas, Jonathan'])
```

If

```
if 100 in team:
  print("Yes, 100 is in the team")
elif 76 in team:
  print("100 is not in the team, but 76 is in it...")
else:
  print("Both 100 and 76 are not in the team")
```

Both 100 and 76 are not in the team

For ciklus

```
mylist = [3,65,2,77]
for i in mylist:
  print("Element:", i)

# Írassuk ki a számokat növekvő sorrendben kettesével!
for i in range(2,10,2): # 2-től 9-ig 2-esével
  print(i)

for (k,v) in team.items():
  print("Player name: %s; #: %d" % (v,k))
# ...
```

```
Element: 3
Element: 65
Element: 2
Element: 77
2
4
6
8
Player name: Ayers, Robert; #: 91
Player name: Beckham Jr,; #: 13
Player name: Brown, Josh; #: 3
Player name: Casillas, Jonathan; #: 54
```

While ciklus

```
i=1
while i<10:
  print(i)
  i+=1
```

```
1
2
3
4
5
6
7
8
9
```

Python script futtatása

```
# vim test.py
#!/usr/bin/env python3
x = 1
for i in range(1, 5):
  x+=i # nincs ++ operátor
  print(x, i, 'alma', 'x*x = %d' % (x*x))
  print(str(i) + " alma")
```

```
2 1 alma x*x = 4
1 alma
4 2 alma x*x = 16
2 alma
7 3 alma x*x = 49
3 alma
11 4 alma x*x = 121
4 alma
```

Futtatás:

```
$ ./test.py # chmod +x test.py
$ python test.py
$ py test.py # laborgépeken
```

Függvények

```
def is_even(num):
  if (num % 2) == 0:
    return True
  else:
    return False

for i in range(1,10):
  if (is_even(i)):
    print("Num:"+str(i))
print("Done")
```

```
Num:2
Num:4
Num:6
Num:8
Done
```

Függvények

```
def complex(x):
  return x**2, x**3, x**4

print(complex(2))
a, b, c = complex(2)
print(a,b,c)
_, rv, _ = complex(2)
print(rv)
```

```
(4, 8, 16)
4 8 16
8
```

Lambda Függvények

```
is_even = lambda num: (num % 2) == 0
is_even_2 = lambda num: True if (num % 2) == 0 else False
for i in range(1,10):
  if (is_even(i)):
    print("Num:"+str(i))
print("Done")
```

```
Num:2
Num:4
Num:6
Num:8
Done
```

List, Dict, Tuple generálás

```
print([ x*x for x in range(10) ])
print({ x:x*x for x in range(5) })
print({ x:x*x for x in range(5) if x!=2 })
print(tuple( x*x for x in range(3) ))
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
{0: 0, 1: 1, 3: 9, 4: 16}
(0, 1, 4)
```

Map

```
def fahrenheit(T):
  return ((float(9)/5)*T + 32)

def celsius(T):
  return (float(5)/9)*(T-32)

temperatures = (36.5, 37, 37.5, 38, 39)
F = map(fahrenheit, temperatures)
C = map(celsius, F)
temperatures_in_F = list(map(fahrenheit, temperatures))
temperatures_in_C = list(map(celsius, temperatures_in_F))
print(temperatures_in_F)
print(temperatures_in_C)
```

```
[97.7, 98.60000000000001, 99.5, 100.4, 102.2]
[36.5, 37.00000000000001, 37.5, 38.00000000000001, 39.0]
```

Filter

```
fibonacci = [0,1,1,2,3,5,8,13,21,34,55]
odd_numbers = list(filter(lambda x: x % 2, fibonacci))
print(odd_numbers)
```

```
[1, 1, 3, 5, 13, 21, 55]
```

File műveletek

```
f = open("demofile.txt", "r")
print(f.read())
print(f.readline())
for x in f:
  print(x)
f.close()
```

```
with open("alma.txt", "r") as f:
   for line in f:
      print(line.strip().split(","))
```

```
f = open("demofile.txt", "w") # w-write, a-append
f.write("lorem ipsum")
```

Standard inputtról olvasás

```
x = input("Please enter a number:")
# x tipusa mindig str!
print("Entered number:", x)
```

Parancssori argumentumok

```
import sys
print(sys.argv[0]) # a script neve
print(sys.argv[1]) # első paraméter
print(sys.argv[2]) # masodik paraméter
# ...
```

Osztályok

```
class Student:
  name = ''
  zhpoint = 0
  def __init__(self,_name,_point):
    self.name =  _name
    self.zhpoint = _point

  def __str__(self): # human readable
    return  f"{self.name}: {self.zhpoint} point"

  def __repr__(self): # machine readable
    return self.name+"("+str(self.zhpoint)+")"

p = Student("Ford",20)
print(p)
print([p])
```

```
Ford: 20 point
[Ford(20)]
```

Import vs main()

importtest.py

```
def main():
    print("Inside main")
if __name__ == "__main__":
    print("Executed as script")
    main()
```

testimport.py:

```
import importtest
importtest.main()
```

vagy:

```
from importtest import main
main()
```

Output:

```
$ python3 importtest.py
Executed as script
Inside main
$ python3 testimport.py
Inside main
```

JSON - JavaScript Object Notation

Lásd: https://www.rfc-editor.org/rfc/rfc8259

```
{
   "firstName": "Jane",
   "lastName": "Doe",
   "hobbies": ["running", "sky diving", "singing"],
   "age": 35,
   "children": [
      {
         "firstName": "Alice",
         "age": 6
      },
      {
         "firstName": "Bob",
         "age": 8
      }
   ]
}
```

JSON & Python

Dictionary mentése JSON file-ba

```
import json
data = {
    "president": {
        "name": "Zaphod Beeblebrox",
        "species": "Betelgeusian"
    }
}
with open("data_file.json", "w") as write_file:
    json.dump(data, write_file)
```

JSON formátumú string előállítása dictionary-ből

```
json_string = json.dumps(data)
```

Szerializáció →

Python	JSON
dict	object
list, tuple	array
str	string
int, float	number
True	true
False	false
None	null

Deszerializáció ←

Python	JSON
dict	object
list	array
str	string
int	number(int)
float	number(real)
True	true
False	false
None	null

JSON & Python

JSON objektum beolvasása JSON file-ból

```
import json
with open("data_file.json", "r") as read_file:
    data = json.load(read_file)
    print(data["president"]["name"])
```

JSON & Python

JSON objektum beolvasása JSON formátumú stringből

```
import json
json_string = """
{
   "researcher": {
      "name": "Ford Prefect",
      "species": "Betelgeusian",
      "relatives": [
         {
            "name": "Zaphod Beeblebrox",
            "species": "Betelgeusian"
         }
      ]
   }
}
"""
data = json.loads(json_string)
for rel in data["researcher"]["relatives"]:
      print("Name: %s (%s)" % ( rel["name"], rel["species"] ) )
```

Gyakorlás I.

Írjunk függvényt ami megadja a paraméterben kapott évszámról, hogy szökőév-e. Az évszámokat egy file-ból olvassuk be! Egy év szökőév ha osztható néggyel, de akkor nem ha osztható százzal, hacsak nem osztható négyszázzal. Példák:

- szökőév: 1992, 1996, 2000, 2400
- nem szökőév: 1993, 1900

Gyakorlás II.

Írjunk scriptet, ami kiszámolja, hogy hány pont szükséges a zh-n az egyes jegyek eléréséhez. A bemenet egy json-t tartalmazó file legyen, amely tartalmazza a mininet, a beadandók és a ZH-n elért és elérhető maximális pontot. A kimenet pedig az egyes érdemjegyekhez szükséges minimális pont. (Részpont nincs!)

Input:

```
{
  "homework": {
    "max": 4,
    "point": 2,
    "min_percent": 0.5
  },
  "zh": {
    "max": 10,
    "min_percent": 0.5
  },
  "mininet": {
    "max": 100,
    "point": 76,
    "min_percent": 0.5
  }
}
```

Output:

```
$ python zh_grade.py
2: 5
3: 6
4: 10
5: Nope
```