Telecommunication Networks

Practice 1

Contact

- Demonstrator: Mohamed Hamed
- E-mail: dg1ewf@inf.elte.hu

Requirements

- 4 absences max.
- Grade (each with weight of 1/3): 50% at each section to qualify

  - Python, socket assignmentss (4 exercises)
    . Correction on the TMS system (https://tms.inf.elte.hu/)

  Either pass or fail

  - Socket Exam
    - Python

  - Mininet Home Project
    - Routing, firewall, IP adress configurations
    - Correction on the TMS system (https://tms.inf.elte.hu/)

Assignments

Assignments:

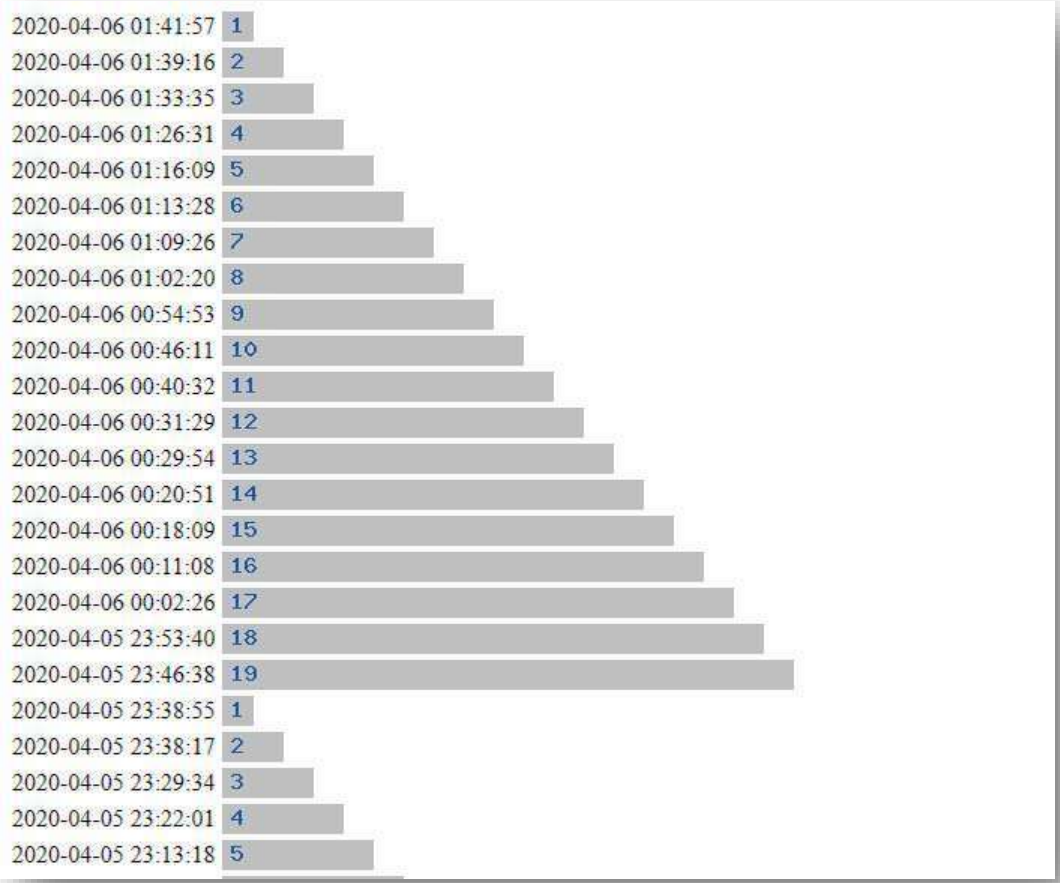
- Programming, simulation
- ~2-3 weeks to complete per homework
- TMS system corrects it – looks for copying
  - **!!! Results placed in the Comment section, evaluated periodically !!!**

- <u>Copying is strictly forbidden! (see: University's Code of Conduct)</u>

How to fail the course

- Send assignment in too late

New assignments on the server before midnight →



Calculating the grades

Final percentage = homework percentage * 0,33 + Mininet percentage * 0,33 + Exam percentage * 0,33

Percentage	Grade
0 - 49 %	1
50 - 59 %	2
60 - 74 %	3
75 - 84 %	4
85 – 100 %	5

Syllabus

- Python basics
- Mininet, network characteristics, basic tools: traceroute, ping
- Wireshark/tcpdump traffic analysis
- Socket programming
- CRC, coding, MD5
- Firewalls: Iptables
- MAC learning, STP, ARP, routing options
- Port forwarding, VLAN options
- Tunneling solutions, IPv4/IPv6

Python history

- Guido Van Rossum, christmas of 1989
- Van Rossum wrote in ’96:

„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*).”

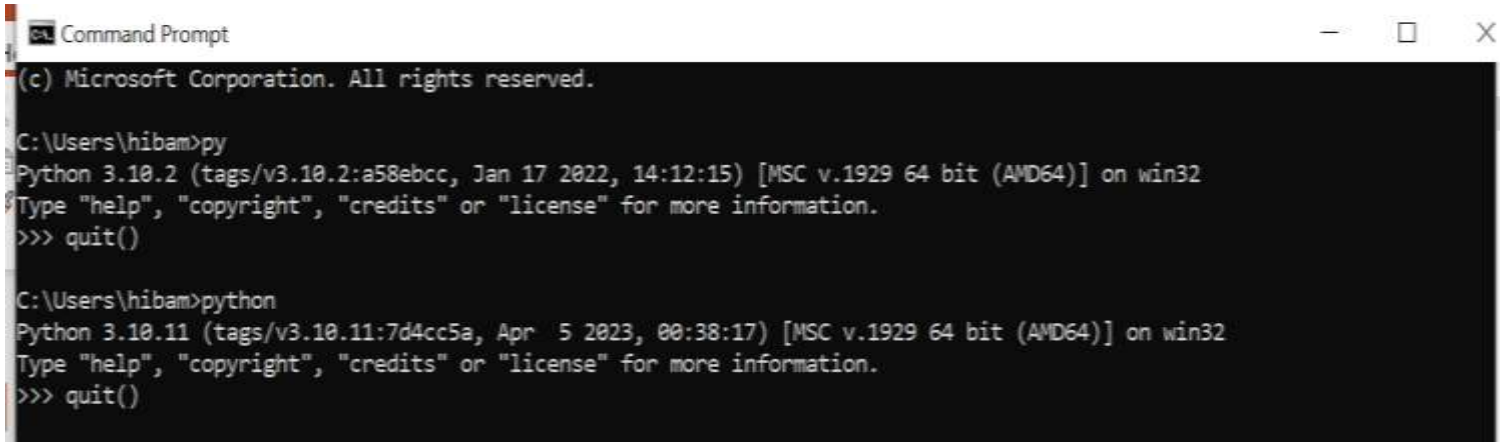
Python facts

- Design for ease of use
  - Clean, simple syntax, small amount of short keywords
- similar to human language

```
friends=['taylor','Alex','Pat','Eli']  # this is a list
for friend in friends:
    print("Hi "+ friend)
```

- Portable
  - Runs on almost everything (Linux, Windows, RasbPi, Big Data)
- Uses whitespaces for syntax
  - Because a good programmer would anyways use them, so why not?
- Variables don't need to be declared
  - They still have types, they are just guessed by the interpreter.
- Versions
  - Python 2.x, 3.x
  - DEPRECATION: Python2 not supported since 2020.
  - https://www.python.org/doc/sunset-python-2/#:~:text=We%20have%20decided%20that%20January,as%20soon%20as%20you%20can.

Python terminal




```
#python
python> import this
python> print("Hello world!")
python> user_name = "Sal"
python> print("Hello " + user_name)
python> user_age = 25
python> print("You are " + str(user_age) + " years old.")
```

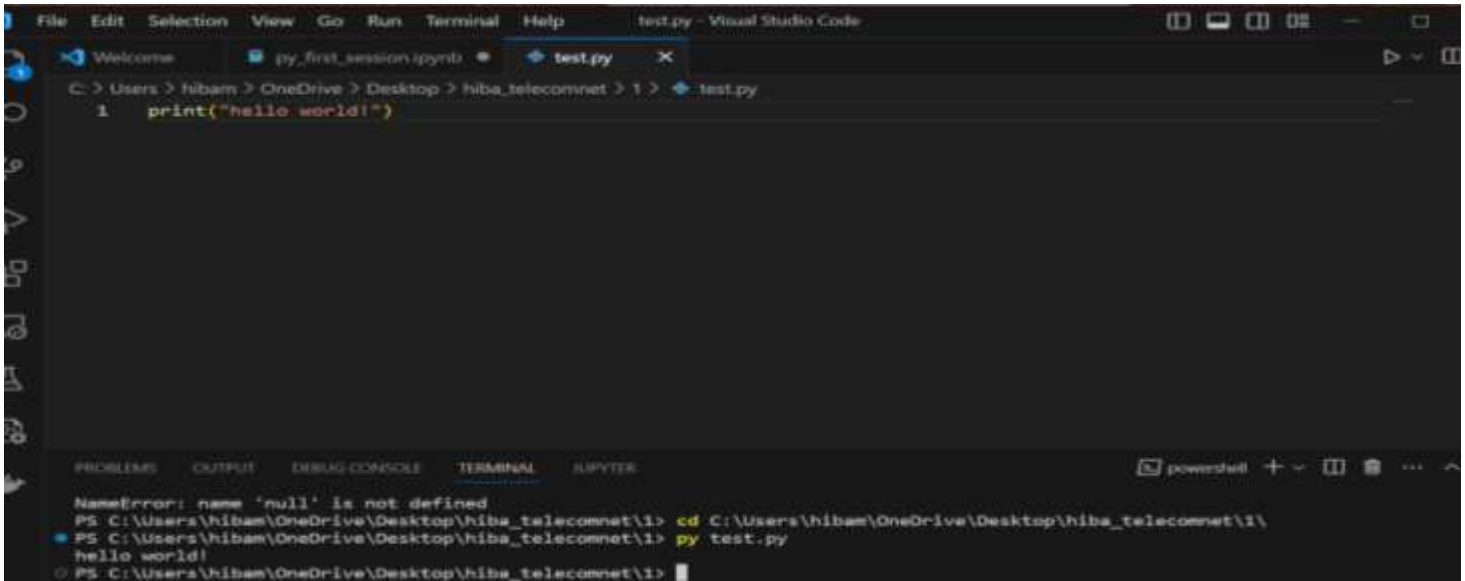
Comment: ' and " are interchangeable in Python

How to run a python script

```
#vi test.py

#!/usr/env/python
x = 1
for i in range(1,5):
    x+=i # comment: ++ oprator doesn't exist
    print(x, i, 'apple', 'x*x = %d' % (x*x))
    print(str(i) + " apple")

# "python test.py" or "py test.py"
```



Basic arithmetics

```
Python>10+2
12

Python>2*2
4

Python>3**2
9

Python>10%2
0
```

Mathematical Rounding

```
Python> import math

Python> math.floor(3.8)
3

Python> round(3.8)
4

Python> round(3.8, 1)
3.8
```

Variables

Variables & assignments , Data types & conversion:

```
Python> a = 42
Python> b = 32
Python> c = a + b
Python> print(c)
74
Python> c = 'valami'
Python> print(a+c)
ERROR
Python> print(type(a))
<class ‘int’>
Python> print(7+2.5) → Implicit conversion
```

```
base= 6
height= 3
area= (base*height)/2
print("the area of a rectangle: "+ str(area))
```

Explicit conversion

Functions/methods

```
#!/usr/bin/env python

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

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

print("End")
```

Functions/methods

1- Built-in functions: print, round, type, str() are functions provided by the language
2- Define our own functions using def

```
def exponents(x):
       return x**2, x**3, x**4   #return a tuple

print(exponents(2))
# (4,8,16)

a, b, c = exponents(2)  #tuple unpack
print(a,b,c)
# 4 8 16

_, rv, _ = exponents(2)
print(rv)
# 8
```

Lambda Functions (anonymous functions)

3- Define our own functions using lambda

```
#!/usr/bin/env python

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("Number:" + str(i))
print("End")
```

Conditionals & logical operators

- Comparing things:
  - `print(10>1)` → `True`
  - `print ("cat"=="dog")` → `False`
  - `print (1!=2)` → `True`
  - `# results are Booleans`
  - `print( 1=="1")` → `False`
  - `print( 1<"1")` → `Error`
- Logical Operators:
  - And, Or, Not:
  - `print(("yellow">"cyan") & ("brown"> "magneta"))` → `False`
  - `print ((25>50) | (1!=2))` → `True`
  - `print( not 42=="answer")` → `True`

Branching

With if statements, else statements, elif statements

```
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')
```

Branching: the ability of a program to alter its execution sequence

Loop

Loops: perform repeatative tasks

- While loop: instructs computer to continuously execute code based on the value of a condition

While loop:

```
x=0
while x<5:
    print("not there yet, x= ", x)
    x= x+1
print("x= ",x)
```

- For loop: iterates over a sequence of values

```
mylist = [3, 65, 2, 77, 9, 33]

for i in mylist:
    print('Element:', i)

for i in range(2,10,2): # every 2nd number from [2,8)
    print (i)
```

Loop

```
Nested loops: loop inside a loop
You run local basketball teams league that will play
against each other but no team plays against itself
List of teams:
teams=[‘dragons’,’wolves’,’pandas’,’unicorn]
Write a script that outputs all possible team pairings:
for home_team in teams:
    for away_team in teams:
        if home_team != away_team:
           print( home_team ,”vs”, away_team)
```

String methods

```
Python>print('apple'.upper())
APPLE

Python>print("LO" in "Hello".upper())
True

Python>print("Decimal Number: %d, Float: %f, String: %s" %
(12,33.4,"appletree"))
Decimal Number: 12, Float: 33.400000, String: appletree
```

Strings: text between quotes

Lists

[0]	[1]	[2]	[3]	[4]
2	39	42	5	10

```
Python> players = [12,31,27,'48',54]

Python> print(players)
[12, 31, 27, '48', 54]

Python> players[0]
12

Python> players[-1]
54

Python> players + [22, 67]
[12, 31, 27, '48', 54, 22, 67]

Python> print(len(players))
5
```

List: collection of items

Lists

```
Lists are mutable: can add, remove, modify
Python> players = [12,31,27,'48',54]
Modify contents of lists: 1-append method
Python> players.append(89)
2- insert method:
py> players.insert(0,12)
3-remove
Py> players.remove(31)
4- modify by assign different value
py> players[2]=99

Check length: how many elements in list
Python> print(len(players))
6
```

Use indexes to create a slice of the list:

```
Python> players[2:]
[27, 48, 54, 89]
```

Tuple – non-modifiable list

```
Py> fullname=(‘Grace’,’M’,’Hopper’)
Python> players = (12, 31, 27, '48', 54)

Python> players[2] = ’alma’
ERROR

Python> del players[2]
ERROR

Python> players[2:]
(27, '48', 54)
```

```
def convert_seconds(seconds):
    hours= seconds//3600
    minutes= (seconds-hours * 3600)//60
    remaining_seconds= seconds-hours*360 - minutes*60
    return hours, minutes, remaining_seconds
result= convert_seconds(5000) #tuple
hours, minutes, seconds= result
print(hours, minutes, seconds)
```

Sets

```
Python> mylist = [8, 3, 2, 3, 2, 4, 6, 8, 2]

Python> myset = set(mylist)

Python> print(mylist)
[8, 3, 2, 3, 2, 4, 6, 8, 2]

Python> print(myset)
set([8, 2, 3, 4, 6])

Python> mysortedlist = sorted(mylist)

Python> print(mysortedlist)
[2, 2, 2, 3, 3, 4, 6, 8, 8]
```

no duplicates of unordered items

```
set_a={1,2,3,4,5,5}
>{1, 2, 3, 4, 5}
print(set_a)
print(set_a[0])→ Error: 'set' object is not subscriptable
```

Dictionary

```
Python> team = {
    91: "Ayers, Robert",
    13: "Beckham Jr,",
    3:  "Brown, Josh",
    54: "Casillas, Jonathan",
    21: "Collins, Landon"}

Python> len(team)
5

Python> team[3] = "Chihiro"

Python> print(91 in team)
True

Python> print ('apple' in team)
False
```

- key ,value pairs
- mutable

Dictionary

```
Python> team = {
    91: "Ayers, Robert",
    13: "Beckham Jr,",
    3:  "Brown, Josh",
    54: "Casillas, Jonathan",
    21: "Collins, Landon"}

Python> print (team.keys())
dict_keys([91, 13, 3, 54, 21])

Python> print (team.values())
dict_values(['Ayers, Robert', 'Beckham Jr,', 'Brown,
Josh', 'Casillas, Jonathan'
, 'Collins, Landon'])
```

Iterate over dictionary

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

• Player name: Brown, Josh; #: 3
• Player name: Nassib, Ryan; #: 12
• …
```

List, Dict, Tuple generation (list comprehensions)

```
mylist = [x*x for x in range(10)]
# [0, 1, 4, 9, 16, 25, 36, 64, 81]

mydict = {x:x*x for x in range(5)}
# {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

mydict2 = {x:x*x for x in range(5) if x!=2}
# {0: 0, 1: 1, 3: 9, 4: 16}

mytuple = tuple(x*x for x in range(3))
# (0, 1, 4)
```

map

map and filter: functions to generate list from list, to process a list
map: returns every item in an iterable
filter: returns values if True
format: takes two arguments: 1st: defined function, 2nd:the list

```
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)
# [97.7, 98.60000000000001, 99.5, 100.4, 102.2]

print(temperatures_in_C)
#[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 handling & operations

File handling functions: open and close

Open function:

used to read, write, and create files

modes: "r": to open and read txt format

"rb": opens and read binary fromat

"r+": opens for read and write

"w": opens for write

"a": opens and edit/append data

```
f = open("demofile.txt", "r")
```

Readline: returns the first line in the file

read: returns the entire contents as a string

readlines: returns an array with multiple lines

```
print(f.read())
print(f.readline())
for x in f:
  print(x)

f.close()
```

File operations

```
with open('apple.txt', 'r') as f: # r-read
   for line in f:
      print( line.strip().split(',') )
```

strip() returns a new string with the whitespace characters removed.
split() method Returns a list of strings after breaking the given string by the specified separator

```
Creating files:

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

f.write("Bla Bla")
```

```
file.write("this is a new file created.")
    file.writelines(["\nthis is a nes file created" ,"\nthis is another line to be added"])
```

We are replacing the file with 'w'
To add to the file use mode: append 'a'

Exception handling

Exceptions and errors:

two types of errors

Syntax errors: caused by human, They have minimal impact

Exception errors: Happen during execution and can easily go unnoticed

errors need to be handled by developer(need to deal with potential issues)

Throwing exception:

```
Py> a=5/0
Py> print(a)
#ZeroDivisionError: division by zero
```

Exception handling:

try & except

```
def divide_by(a,b):
    return a/b
try:
   ans=divide_by(40,0)
   print(ans)
except ZeroDivisionError as e:
    print(e,": we can't divide by zero")
```

Reading from standard input

```
x = input("Give me a number")

# type(x) always str !!!

print("Given number: ", x)
```

System arguments

```
import sys

print sys.argv[0]   #← name of the script

print sys.argv[1]   #← first argument
print sys.argv[2]   #← second argument
…
```

Classes

```
class Student:
          name= ''
          exam_score= 0

         def __init__(self,_name,_score): # constructor
                  self.name =  _name
                  self.exam_score = _score

         def __str__(self): # human readable
                  return self.name+"("+str(self.exam_score)+")"
         def __repr__(self): # machine readable
                  return "in list "+self.name+"("+str(self.exam_score)+")"

p = Student("Ford",20)

print(p)
# Ford(20)

print([p])
# [in list Ford(20)]
```

Import vs main()

```
def main():
        print("This is main.")

if __name__ == "__main__":
        print ("This will run only if running as a script.")
        main()
```

```
import practice1test

practice1test.main()
```

```
from practice1test import main

main()
```

```
$ python3 practice1test.py
This will run only if running as a script.
This is main.

$ python3 practice1import.py
This is main.
```

JSON - **J**ava**S**cript **O**bject **N**otation

A data representation format, lightweight, widely used, every major language has some form of library to parse JSON.
JSON can represent different data types: strings, numbers, Booleans… the most widely used are:
Arrays (similar to python list)
Objects: key, value pair (similar to python dictionary)
The real strength of JSON is the ability to represent nested data (objects inside objects inside arrays …
Representing hierarchy instead of plain flat data formats

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

Help: https://realpython.com/python-json/

JSON & Python –Parsing

Parsing: converting between python data types and JSON data types

```
import json
#by using triple quoted string ‘’’ this is a python string that happens to
be a valid 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"] ) )
```

Loads():convert from a JSON string into a python object

JSON & Python – Type conversion during deserialization

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

JSON & Python – Type conversion during serialization

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

JSON & Python – import json

Retrieve JSON object

Dumps(): converts from python object into a JSON string

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

Read JSON object from JSON file

```
import json

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

Load(): to load JSON file into a python object
Loads: to load a string

JSON & Python – JSON files

Save JSON object into JSON file

```
import json

data = {
    "president": {
        "name": "Zaphod Beeblebrox",
        "species": "Betelgeusian"
    }
}

with open("data_file.json", "w") as write_file:
    json.dump(data, write_file)
```

dump: to dump to a JSON file
dumps: to dump to a JSON string

Task 1

Write a function that tells us if the given input is a leap-year or not.

Read the years from a file.

Definition of a leap-year: A year is a leap year if it's divisible by 4, unless it is also divisible by 100. The only exception is if it's divisible by 400. Then it's a leap year again :)

Examples:

- Leap-year: 1992, 1996, 2000, 2400
- Not leap-year: 1993, 1900

Task 2

- Write a script that computes how many points you have to score for each grade. The input is a json file that contains the maximal,minimum, and received scores for the mininet and homeworks, also, the maximal score attainable for the exam as well as the minimal percentage needed for passing the exam.

```
{  xx „socketScore": {"max": 20, "received":10 },
   "zhPont": {"max": 20, "min":0.5 },
   "mininetScore": {"max": 20, "received":20, "min":0.5 },
}
```

```
python exam_calculator.py
2 :  10.0
3 :  10.0
4 :  16.0
5 :  Impossible
```

Task 2

- Another example 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
    }
}
```

If done & bored

Write a function that gives us the nth fibonacci number.

fibonacci(0) -> 0

fibonacci(1) -> 1

fibonacci(2) -> 1

fibonacci(3) -> 2

…

fibonacci(n) -> fibonacci(n-2) + fibonacci(n-1)

THE END