



THREADS AND CLIENT/SERVER COMMUNICATION

Lab 7

EXERCISE OF MODELIZATION

- The technology current has now turned to processors multicore, that is to say machines capable of executing programs simultaneously.
- The three uses are:
 - distributed computing or computing parallel
 - after clicking a button, the GUI will start a thread that will run this process
 - the communication between computers or generally Internet communication



ASYNCHRONOUS AND SYNCHRONOUS COMMUNICATION

- The asynchronous communication is when the computer performs tasks simultaneously and listens to a port through which other computers communicate with him. The program has two threads: the main thread and the thread that just waits and processes the messages which are received via a port.
- The synchronous communication denotes the dependences between threads. In a distributed computing, the end result depends on the results returned by each thread, therefore we must wait for two execution threads to have produced the expected result.



THREAD

- Must remember the following:
 - overload the class `threading.Thread`,
 - override the constructor without forgetting to call the constructor `threading.Thread.__init__`,
 - method overloading for `run`,
 - create an instance of the new class and call the `start` method to begin the secondary thread which will form the second execution thread.



EXAMPLE

The program displays lines that come from the main thread and the secondary thread whose views differ.

```
import threading
import time

class MonThread (threading.Thread):
    def __init__(self, jusqu):      # jusqu = donnée supplémentaire
        threading.Thread.__init__(self) # ne pas oublier cette ligne
        # (appel au constructeur de la classe mère)
        self.jusqua = jusqu       # donnée supplémentaire ajoutée à la classe

    def run(self):
        for i in range(0, self.jusqua):
            print("thread ", i)
            time.sleep(0.08)      # attend 100 millisecondes sans rien faire
            # facilite la lecture de l'affichage

m = MonThread(10)                # crée le thread
m.start()                        # démarre le thread,
# l'instruction est exécutée en quelques millisecondes
# quelque soit la durée du thread

for i in range(0, 10):
    print("programme ", i)
    time.sleep(0.1)              # attend 100 millisecondes sans rien faire
    # facilite la lecture de l'affichage
```

```
thread programme 0
0
thread 1
programme 1
thread 2
programme 2
```



TWO THREADS

- The previous program was adapted to launch two secondary threads in addition to the main thread.

```
import threading
import time

class MonThread (threading.Thread):
    def __init__(self, jusquas, s):
        threading.Thread.__init__(self)
        self.jusqua = jusquas
        self.s = s

    def run(self):
        for i in range(0, self.jusqua):
            print("thread ", self.s, " : ", i)
            time.sleep(0.09)

m = MonThread(10, "A")
m.start()

m2 = MonThread(10, "B") # crée un second thread
m2.start()              # démarre le thread,

for i in range(0, 10):
    print("programme ", i)
    time.sleep(0.1)
```



SYNCHRONIZATION

- For two threads synchronization is waiting for a secondary thread by the main thread.
- In the following program, add the attribute state in the classroom MonThread which will indicate the status of thread:
 - true to start
 - false for stop
- The main thread will simply check the status of thread time to time.



EXAMPLE

```
import threading
import time

class MonThread (threading.Thread):
    def __init__(self, jusqu):
        threading.Thread.__init__(self)
        self.jusqua = jusqu
        self.etat = False      # L'état du thread est soit False (à l'arrêt)
                                # soit True (en marche)
    def run(self):
        self.etat = True       # on passe en mode marche
        for i in range(0, self.jusqua):
            print("thread itération ", i)
            time.sleep(0.1)
        self.etat = False      # on revient en mode arrêt

m = MonThread(10)             # crée un thread
m.start()                     # démarre le thread,
print("début")

while m.etat == False:
    # on attend que le thread démarre
    time.sleep(0.1) # voir remarque ci-dessous

while m.etat == True:
    # on attend que le thread s'arrête
    # il faut introduire l'instruction time.sleep pour temporiser, il n'est pas
    # nécessaire de vérifier sans cesse que le thread est toujours en marche
    # il suffit de le vérifier tous les 100 millisecondes
    # dans le cas contraire, la machine passe son temps à vérifier au lieu
    # de se consacrer à l'exécution du thread
    time.sleep(0.1)

print("fin")
```



THE METHOD WAIT

- The method wait of the Event object expects that the object is activated. It can wait forever or wait for a period just given.

```
import threading
import time

class MonThread (threading.Thread):
    def __init__(self, jusqu'a, event):    # event = objet Event
        threading.Thread.__init__(self)  # = donnée supplémentaire
        self.jusqua = jusqu'a
        self.event = event                # on garde un accès à l'objet Event

    def run(self):
        for i in range(0, self.jusqua):
            print("thread itération ", i)
            time.sleep(0.1)
        self.event.set()                  # on indique qu'on a fini :
        # on active l'objet self.event
print("début")

event = threading.Event()                # on crée un objet de type Event
event.clear()                            # on désactive l'objet Event
m = MonThread(10, event)                 # crée un thread
m.start()                                # démarre le thread,
event.wait()                             # on attend jusqu'à ce que l'objet soit activé
# event.wait (0.1) : n'attend qu'un
print("fin")                             # seulement 1 dixième de seconde
```

```
début
thread itération 0
thread itération 1
```



PYTHON PROGRAMMING WITH SOCKETS

- A socket is one end of a bidirectional communication link between two programs running on the network.
- The client and server can communicate by writing or reading from their supports.
- Your computer communicates through ports.
- The IP address is used to identify the computer on the network and the port to identify the application in the computer (on a web server, it is Apache.).
- In this part, we will learn to make a script that simply sends an HTTP request to a web server.



COMMUNICATE BOTH APPLICATIONS (CLIENT /SERVER)

- **Server**

- Must import the Socket module, it is very important.
- Create the socket.

```
sock = socket.socket(socket.AF_INET,  
    socket.SOCK_STREAM)
```

- To be able to receive connections:

```
Host = '127.0.0.1' # 'Local computer  
Port = 234         # Choice of port
```

```
# we bind our socket:
```

```
Sock.bind((Host,Port))
```

- The server is listening and accepts a single connection.
- Must display the received data and for this, we must make an infinite loop for receiving data.

COMMUNICATE BOTH APPLICATIONS (CLIENT /SERVER)

- **Customer**
- Import the module
- Create the socket
- Definition of the information
- Connect to the server with information
- When it is connected, it is an infinite loop of inputs for sending messages



SERVER

```
import sys
import socket
import threading
import re
import time
exit_flag = True

def connect(conn):
    global flag
    while flag:
        received = conn.recv(1024)
        if not received.decode():
            break
        if received == ' ':
            pass
        elif received == "exit":
            break
        else:
            print("Client>>> " + received.decode())
    flag = False
```



SERVER

```
def sendMsg(conn):
    while flag:
        send_msg = input().encode('utf-8')
        if send_msg == ' ':
            pass
        else:
            conn.sendall(send_msg)

if __name__ == '__main__':
    flag = True
    start_time = time.monotonic()
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind(('', 11111))
    s.listen()
    (conn, addr) = s.accept()
    thread1 = threading.Thread(target=connect, args=([conn]))
    thread2 = threading.Thread(target=sendMsg, args=([conn]))
    thread1.start()
    thread2.start()
    thread1.join()
    thread2.join()
```



CLIENT

```
import sys
import socket
import threading
import time
from typing import re

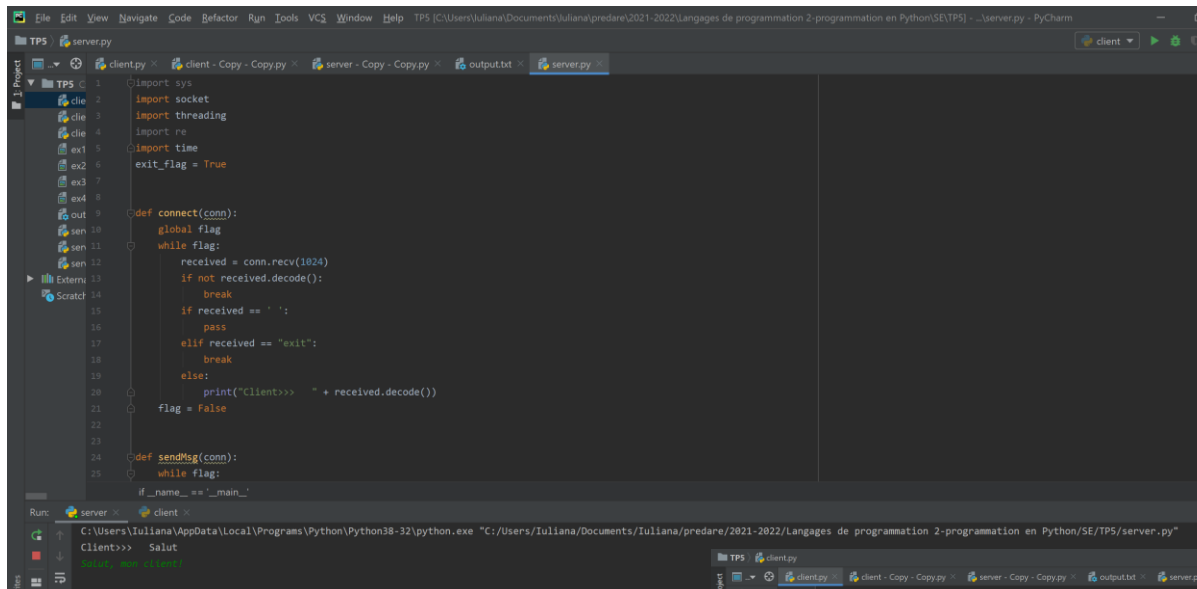
def connect(s):
    while True:
        r_msg = s.recv(1024)
        if not r_msg:
            break
        if r_msg == '':
            break
        else:
            print("Server>>> "+str(r_msg))
        if not flag:
            break
```

CLIENT

```
def receive(s):
    global flag
    while True:
        s_msg = input().encode('utf-8')
        if s_msg == '':
            pass
        if s_msg.decode() == 'exit':
            print("wan exit")
            break
        else:
            s.sendall(s_msg)
    flag = False

if __name__ == '__main__':
    start_time = time.monotonic()
    if 3 != len(sys.argv):
        print("usage: %s [ip address][port] " % sys.argv[0])
        sys.exit(0)
    flag = True
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.connect((sys.argv[1], int(sys.argv[2])))
    thread1 = threading.Thread(target=connect, args=([s]))
    thread2 = threading.Thread(target=receive, args=([s]))
    thread1.start()
    thread2.start()
    thread1.join()
    thread2.join()
```


THE RESULT - SERVER AND CLIENT SIDE



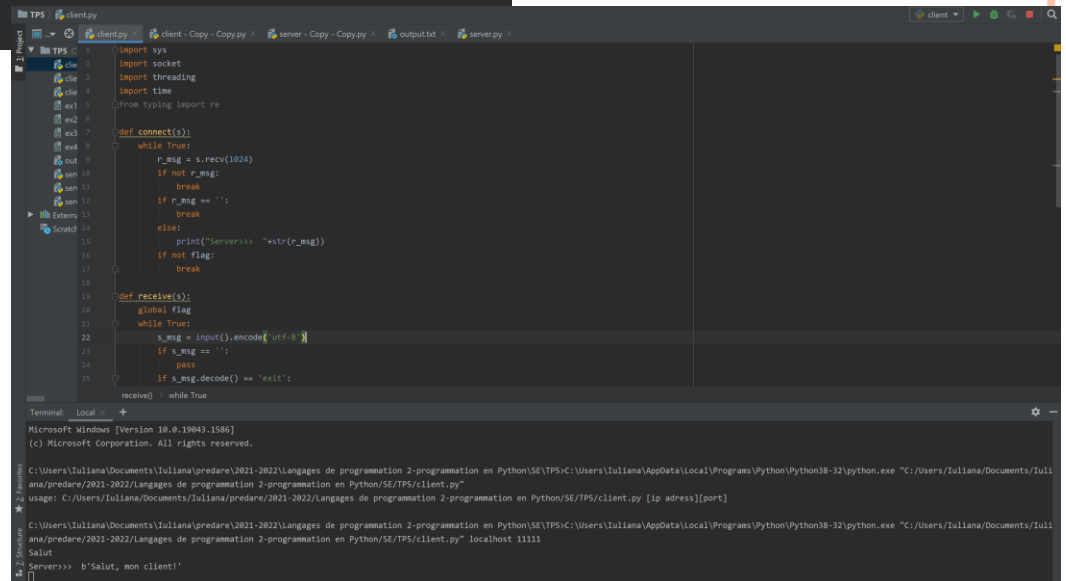
```
1 import sys
2 import socket
3 import threading
4 import re
5 import time
6 exit_flag = True
7
8
9 def connect(conn):
10     global flag
11     while flag:
12         received = conn.recv(1024)
13         if not received.decode():
14             break
15         if received == ' ':
16             pass
17         elif received == "exit":
18             break
19         else:
20             print("Client>>> " + received.decode())
21     flag = False
22
23
24 def sendMsg(conn):
25     while flag:
26         if __name__ == "__main__":
27             pass
```

Run: server client

C:\Users\Iuliana\AppData\Local\Programs\Python\Python38-32\python.exe "C:/Users/Iuliana/Documents/Iuliana/predare/2021-2022/Langages de programmation 2-programmation en Python/SE/TPS/server.py"

Client>>> Salut

Server>>> b'Salut, mon client!'



```
1 import sys
2 import socket
3 import threading
4 import time
5 from typing import re
6
7 def connect(s):
8     while True:
9         r_msg = s.recv(1024)
10         if not r_msg:
11             break
12         if r_msg == ' ':
13             break
14         else:
15             print("Server>>> " + str(r_msg))
16         if not flag:
17             break
18
19
20 def receive(s):
21     global flag
22     while True:
23         s_msg = input().encode('utf-8')
24         if s_msg == ' ':
25             pass
26         elif s_msg.decode() == "exit":
27             break
28     receive()
29
30
31 if __name__ == "__main__":
32     pass
```

Terminal: Local

Microsoft Windows [Version 10.0.19043.1586]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Iuliana\Documents\Iuliana\predare\2021-2022\Langages de programmation 2-programmation en Python\SE\TPS>C:\Users\Iuliana\AppData\Local\Programs\Python\Python38-32\python.exe "C:/Users/Iuliana/Documents/Iuliana/predare/2021-2022/Langages de programmation 2-programmation en Python/SE/TPS/client.py"

usage: C:/Users/Iuliana/Documents/Iuliana/predare/2021-2022/Langages de programmation 2-programmation en Python/SE/TPS/client.py [ip address][port]

C:\Users\Iuliana\Documents\Iuliana\predare\2021-2022\Langages de programmation 2-programmation en Python\SE\TPS>C:\Users\Iuliana\AppData\Local\Programs\Python\Python38-32\python.exe "C:/Users/Iuliana/Documents/Iuliana/predare/2021-2022/Langages de programmation 2-programmation en Python/SE/TPS/client.py" localhost 11111

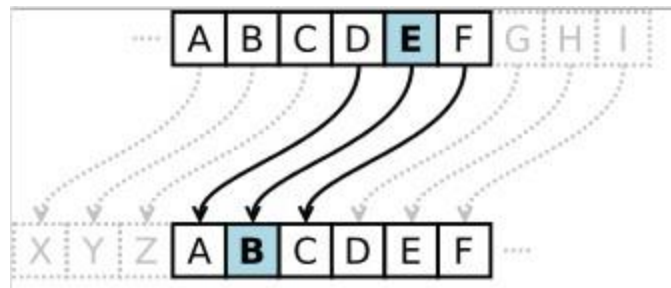
Salut

Server>>> b'Salut, mon client!'

EXERCISE - PYTHON AND THE CRYPTOGRAPHY

○ Encrypt and decrypt the Julius Caesar

Caesar cipher replaces each letter with a different letter placed at a fixed number of spaces in the alphabet. The figure shown here uses a left shift of three positions, so that (for example) each occurrence of E in the plaintext becomes B in the ciphertext.



- Encryption can also be represented using modular arithmetic by firstly converting the letters into numbers, according to $A = 0, B = 1, \dots, Z = 25$. Encryption of a letter x by a shift n can be described mathematically as, $E_n(x) = (x + n) \mod 26$.
- Decryption is performed similarly, $D_n(x) = (x - n) \mod 26$.
- View encryption at server side after the customer has sent a message.

HOMEWORK - THEAT SCYTALÉ OF THE GREEKS



The scytale was a wooden stick used for reading or writing a coded dispatch.

After winding the belt on the scytale. The message was written by placing a letter on each convolution. To decrypt, the recipient must have a stick with a diameter identical to that used for encoding. He was enough to wind the scytale around the stick to get the message clear.

It can be easily broken because the cipher is a transposition of the letters of the message and not in substitution of letters. Once unwound, the tape on which the message is located in this form:
MSETSEUEARR_SISG_EMQN



HOMEWORK - THE SCYTALE OF THE GREEKS

- Such encryption is a variation of those based on the grids is filled line by line with the plaintext: rectangular transpositions. Ciphertext corresponds to the columns of the grid. The stick size represents the width of the transposition of grid.
- With this vision of encryption, it may therefore be addressed with the following steps:
 - Count the number of letters on the tape (here 21)
 - Create grids of different sizes so you can put all the letters. With 21, we have 2×11 , 3×7 , 4×6 , 5×5 , 4×6 , etc.
 - For each grid, filling the boxes line by line with the ciphertext. The same is done with a copy of this grid but filling column at this time. That is to have such a grid of 4×6 and 6×4 .
 - Check if a plain text appears perpendicular to the filling of the grid (if it was filled row by row, is read column by column).



HOMEWORK - THE SCYTALE OF THE GREEKS

- In our case, with a grid of 3×7 :
- The message does not appear in the columns, attempting with the reverse grid, or 7×3 :

M	S	E
T	S	E
U	E	A
R	R	_
S	I	S
G	_	E
M	Q	N

M	S	E	T	S	E	U
E	A	R	R	_	S	I
S	G	_	E	M	Q	N

- You can read column by column:
"MESSENGER_TRES_MESQUIN".



Include a speech to text and text to speech functionality for the program of the previous laboratory.

Installation required:

Python Speech Recognition module:

```
pip install speechrecognition
```

PyAudio: Use the following command for linux users

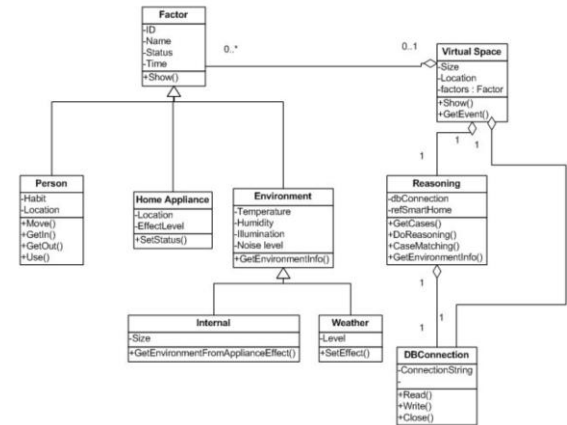
```
sudo apt-get install python3-pyaudio
```

Windows users can install pyaudio by executing the following command in a terminal

```
pip install pyaudio
```

Python pytsx3 module:

```
pip install pyttvx3
```



Homework

Translation of Speech to Text:

First, we need to import the library and then initialize it using `init()` function. This function may take 2 arguments.

`init(driverName string, debug bool)`

drivername: [Name of available driver] sapi5 on Windows | nsss on MacOS

debug: to enable or disable debug output

After initialization, we will make the program speak the text using `say()` function.

This method may also take 2 arguments.

`say(text unicode, name string)`

text: Any text you wish to hear.

name: To set a name for this speech. (optional)

Finally, to run the speech we use `runAndWait()` All the `say()` texts won't be said unless the interpreter encounters `runAndWait()`.



Homework

Code sample:

```
import speech_recognition as sr
import pyttsx3
```

```
# Initialize the recognizer
```

```
r = sr.Recognizer()
```

```
# Function to convert text to speech
```

```
def SpeakText(command):
```

```
    # Initialize the engine
```

```
    engine = pyttsx3.init()
```

```
    engine.say(command)
```

```
    engine.runAndWait()
```

```
# Loop infinitely for user to speak
```

```
while(1):
```

```
    # Exception handling to handle exceptions at the runtime
```

```
    try:
```

```
        # use the microphone as source for input.
```

```
        with sr.Microphone() as source2:
```

```
            # wait for a second to let the recognizer adjust the energy threshold based on the surrounding noise level
```

```
            r.adjust_for_ambient_noise(source2, duration=0.2)
```

```
            #listens for the user's input
```

```
            audio2 = r.listen(source2)
```

```
            # Using google to recognize audio
```

```
            MyText = r.recognize_google(audio2)
```

```
            MyText = MyText.lower()
```

```
            print("Did you say "+MyText)
```

```
            SpeakText(MyText)
```

```
except sr.RequestError as e:
```

```
    print("Could not request results; {0}".format(e))
```

```
except sr.UnknownValueError:
```

```
    print("unknown error occured")
```

Input: voice speech (Hi buddy how are you)

Output: Did you say hi buddy how are you

