UNIVERSITATEA ALEXANDRU IOAN CUZA IAŞI
FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Dany – Virtual Assistant

propusă de

Andrei Sfarghiu

Sesiunea: ***iulie, 2019***

Coordonator științific

Conf. dr. Anca Vitcu

UNIVERSITATEA ALEXANDRU IOAN CUZA IAŞI
FACULTATEA DE INFORMATICĂ

Dany – Virtual Assistant

Andrei Sfarghiu

Sesiunea: ***iulie, 2019***

Coordonator științific

Conf. dr. Anca Vitcu

Contents

Introducere

În această primă secțiune voi prezenta in mod succint tema aplicației, funcționalitățile ei, precum si tehnologiile implementate in dezvoltarea acestui *virtual assistant*.

Motivație

Aplicația creată de mine, numita „Dany – Virtual Assistant”, sau doar „Dany” pe scurt, are ca rol principal ușurarea interacțiunii dintre om si PC. „Dany” face acest lucru prin implementarea de comenzi vocale; astfel, utilizatorul poate doar sa ii zică programului ce dorește sa facă, iar programul va executa comanda data de către acesta. Deși rolul principal al aplicației este de a facilita interacțiunile utilizatorului cu PC-ul, „Dany” ar putea folosit si de persoanele care din diverse motive medicale nu pot folosi *mouse*-ul si tastatura pentru a utiliza un calculator. Astfel, consider ca deși eu am creat acest *virtual assistant* pentru a-mi ușura mie viața și modul de lucru, pe viitor ar putea avea un impact pozitiv si asupra vieților altor oameni.

Un al doilea motiv pentru care am ales aceasta temă pentru lucrarea de licență este faptul ca dintotdeauna am fost pasionat de inteligența artificială; sau mai bine zis am fost dintotdeauna impresionat de modul în care inteligența artificială mi-a fost prezentată în decursul vieții. Astfel, pentru mine, ideea de a vorbi cu un PC și de a-i da comenzi vocale a reprezentat întotdeauna un lucru extraordinar. Așadar, pentru lucrarea de licență am decis sa îmbin utilul cu plăcutul, și sa fac un *virtual assistant* care să-mi ușureze interacțiunile cu propriul meu PC.

Context

Ideea de bază pentru această aplicație a pornit de la un joc de tipul *choose your own adventure.* Astfel, prima dată mi-am dorit să fac pentru proiectul de licență un *chatbot* care sa joace rolul de povestitor, utilizatorul introducând de la tastatură diverse comenzi pentru a avansa povestea. După ceva vreme m-am gândit că ar fi mai interesant dacă *chatbot*-ul i-ar

răspunde jucătorului nu doar prin intermediul unor simple fraze scrise, ci chiar cu cuvinte rostite cu ajutorul unui program de tipul *text-to-speech.* Astfel a început evoluția aplicației mele către ceea ce este in ziua de astăzi. După ce m-am gândit că ar fi mai bine ca *chatbot*-ul sa „vorbească" cu jucătorul, mi-am dat seama că și acesta la rândul lui ar trebui sa comunice cu *chatbot*-ul prin comenzi vocale, nu prin text introdus de la tastatură. Astfel a început să se contureze în mintea mea o idee: cu toate că un joc de tipul *choose your own adventure* este o idee interesantă, este însă și ceva foarte static; ce vreau să zic prin acest lucru este faptul că pe decursul unei runde de joc nu s-ar fi întâmplat nimic nou: jucătorul ar fi vorbit cu *chatbot*-ul, i-ar fi zis ce dorește să facă pentru a evada, iar *chatbot*-ul la rândul lui, i-ar fi răspuns cu o descriere a acțiunilor sale si a consecințelor acestor acțiuni. Când am realizat faptul că aceasta idee este puțin prea plictisitoare pentru gustul meu, am decis sa trec la altceva. Așadar, am decis să păstrez câteva componente, și anume: *chatbot*-ul, comenzile vocale si *text-to-speech*-ul. În acest mod, aplicația mea a evoluat dintr-un simplu *chatbot* într-un *virtual assistant*, care nu doar joacă rolul de narator, ci chiar poate ajuta utilizatorul în îndeplinirea mai multor obiective.

Bineînțeles că nu eu am venit cu ideea de *virtual assistant.* Acest tip de aplicație există pe piață de un număr buni de ani, însă a început să capete atenție odată cu lansarea lui „Siri" de către Apple. Astfel, în ziua de astăzi, aceste aplicații sunt la ordinea zilei: Google are „Google Assitant", Amazon are „Amazon Echo" (denumit si „Alexa"), Microsoft are „Cortana", Samsung are „Bixby" etc. După cum se poate observa, aproape fiecare companie din domeniul IT are câte un *virtual assistant*. Toate aceste aplicații sunt extrem de complexe, ele fiind realizate de către echipe întregi de programatori, pe decursul a ani de zile. Eu am reușit de unul singur, pe decursul câtorva luni, sa implementez 21 de comenzi care fac aproape tot ceea ce fac si aplicațiile mai sus menționate. De asemenea „Dany" poate face și câteva lucruri pe care celelalte aplicații nu le pot face: un exemplu ar putea fi că poate căuta ceva, atât pe internet, cât și în fișierele locale aflate pe *hard disk*-ul PC-ului. Țin sa menționez faptul că am încercat sa îi dau o personalitate aplicației, încercând astfel sa emulez relația dintre o persoană și asistentul ei din viața de zi cu zi. Am făcut acest lucru tocmai din pricina faptului ca nu sunt de acord cu abordare companiei Google, care dorește ca utilizatorul sa vadă aplicația „Google Assistant" doar ca pe un simplu program. Astfel, am încercat să imit felul de a vorbi al oamenilor, tocmai

pentru că atunci când utilizatorul poartă o conversație cu „Dany” îmi doresc ca acesta să uite, pe cât posibil, de faptul ca nu vorbește cu un om.

Cerințe funcționale ale aplicației

Prima regulă pe care mi-am impus-o când m-am hotărât asupra acestei teme de licența a fost de a oferi utilizatorului o experiența *seamless.* Ce vreau să spun prin acest lucru este faptul că odată pornit, *virtual assistant*-ul va asculta și auzi tot ceea ce i se spune. Acest lucru nu este făcut sub nicio formă pentru a încălca confidențialitatea utilizatorului; comenzile rostite de către acesta nu sunt salvate nicăieri, iar în cazul în care utilizatorul încearcă să folosească o comanda pe care „Dany” nu o știe, i se va spune faptul că acea comandă nu este recunoscută. *Virtual assistant*-ul ascultă și aude tot, tocmai datorită faptului ca îmi doresc să ofer o experiență cât mai *seamless.*

O a doua regulă pe care am dorit să o urmez este de a oferi o metodă ușoară de a interacționa cu „Dany”. Odată ce *virtual assistant*-ul este pornit, folosirea acestuia de către utilizator este cât se poate de simplă: Conversația va începe întotdeauna cu „Dany” care îl va saluta pe utilizator și îl va întreba cu ce poate să îl ajute. Acum utilizatorul poate folosi comanda de „*help*” pentru a vedea o listă cu exemple pentru toate comenzile disponibile. Acum că utilizatorul știe toate comenzile disponibile, el poate profita la maxim de toate *feature*-urile lui „Dany”. Astfel utilizatorul poate folosi orice comandă în orice ordine. După cum am spus și mai sus, îmi doresc să ofer o experiență simplă de utilizare, astfel ca indiferent cine va folosi aplicația în viitor, să nu fie descurajat de o metodă complicată.

Abordare tehnică a aplicației

Pentru construirea acestei aplicații am folosit limbajul de programare Python 2.7. Am făcut acest lucru din două motive: primul motiv este acela că Python dispune de o mulțime de biblioteci deja create de alți utilizatori ai limbajului, cu care se poate face aproape orice. Al doilea motiv este simplicitatea limbajului și deci rapiditatea care vine odată cu această

simplicitate. Astfel nu am întâmpinat nicio problemă în optimizarea aplicației. De asemenea aș dori sa menționez și să motivez utilizarea câtorva tehnologii instrumentale în construirea acestui *virtual assistant:*

- Biblioteca Tkinter: aceasta bibliotecă este utilizată pentru crearea interfeței grafice. Am utilizat aceasta biblioteca din mai multe motive: vine deja integrată în Python 2.7; este relativ simplu de utilizat; mi-a oferit o mai mare flexibilitate în ceea ce privește modul în care îmi afișez conversația dintre utilizator și „Dany".

- Bibliotecile pynput.mouse și pynput.keyboard: am folosit aceste doua biblioteci pentru a face *onclick event* și pentru a simula scrisul de la tastatură. *Onclick event* mi-a fost de folos la redarea unei melodii de pe YouTube: este foarte ușor de căutat o melodie pe YouTube, adică este simplu de ajuns pe pagina pe care sunt afișate căutările; este însă mult mai dificil de a intra pe pagina unei melodii sau a unui videoclip. Astfel m-am gândit să fac un *onclick event* pentru a trece peste *parsing*-ul de cod HTML de pe YouTube. Biblioteca pynput.keyboard a fost utilizată pentru simularea unor comenzi de la tastatură (Windows Key + D, Ctrl +Shift + Esc etc.).

- Biblioteca tempfile: Această bibliotecă a fost utilizată pentru a salva fișierul de tipul .mp3 în care este stocat răspunsul lui „Dany". Prima data am încercat să salvez fișierul fără a utiliza biblioteca tempfile, Însă *virtual assistant*-ul îmi răspundea doar la prima comandă, dupa care indiferent câte comenzi i-aș fi dat, nu mai răspundea. Astfel am aflat că atunci când Python salvează ceva într-un fișier, nu mai poate salva altceva cu același nume; nu poate da *overwrite* la fișier. Pentru a remedia acest lucru am utilizat biblioteca tempfile. Așadar, acum îmi salvez răspunsul lui „Dany" într-un *temporary file*, iar dupa ce *voice assistant*-ul termina ce are de făcut cu el fișierul este șters, lăsând astfel loc pentru următorul răspuns al lui „Dany".

- Biblioteca bs4: Această bibliotecă (denumita BeautifulSoup4) este utilizată pentru a face *parsing.* Am intenționat original să o folosesc pentru a compune URL-ul complet al unei melodii specifice de pe YouTube, însă am descoperit o metoda mult mai simplă pentru a face acest lucru. Cu toate acesta am folosit biblioteca bs4 pentru a face *parsing* pe fișierul de tipul .xml în care este salvat *RSS feed*-ul de la Google News. Am făcut acest lucru pentru a îl face pe *virtual assistant* să poată citi cu voce tare titlurile știrilor.

- Biblioteca wikipedia: Am folosit biblioteca wikipedia pentru accesa informații asupra majorității domeniilor. Astfel este suficient ca utilizatorul să întrebe despre ceva sau cineva, iar „Dany" va știi în general câteva informații despre acel lucru sau persoană. În cazul în care nu poate găsi nimic despre entitatea căutată, „Dany" va căuta informații pe Google.

- Biblioteca smtplib: A fost folosită pentru *log in*-ul și trimiterea de *e-mail*-uri de pe gMail. Am folosit această biblioteca pentru că, din câte știu eu, este singura metodă de a trimite *e-mail*-uri de pe un cont de gMail prin Python.

Structură documentație

Am hotărât să structurez această lucrare în două capitole. Astfel primul în primul capitol va fi vorba despre structura și modul de operare al *virtual assistant*-ului, iar cel de-al doilea capitol va fi dedicat interacțiunii utilizatorului cu „Dany":

- Capitolul 1: Modul de funcționare al *virtual assistant*-ului

- Capitolul 2: Interacțiunile utilizatorului cu *virtual assistant*-ul prin intermediul comenzilor

Un ultim lucru pe care aș dori sa îl menționez în introducere este faptul că am întâmpinat și câteva dificultăți în construirea acestui *virtual assistant.* Câteva dintre acestea ar fi:

- Interfața grafică întrerupea funcționarea programului; am rezolvat acest lucru făcând fereastra interfeței *inactive*(`text_box.config(state = Tkinter.DISABLED)`). Astfel interfața devine *normal*(`text_box.config(state = Tkinter.NORMAL)`) doar în momentul când am nevoie să îmi afișez ceva nou. Din acest motiv interfața grafică a aplicației va fi mereu *not responding,* mai puțin atunci când este actualizată. Deși acest lucru poate părea un *bug,* această metodă este cea pe care am găsit-o în documentația bibliotecii Tkinter.

- Am întâmpinat dificultăți în *parsing*-ul de cod HTML. Original am dorit să-mi compun eu singur URL-ul pentru orice melodie de pe YouTube. Am reușit să ajung până pe pagina cu toate rezultatele unei căutări, însă nu am mai putut sa compun URL-ul unei pagini cu o melodie specifică. După ceva timp în care m-am ocupat de alte părți ale proiectului m-am întors aspura acestei probleme cu o idee nouă de rezolvare: de pe pagina cu căutările am făcut un *onclick event* cu ajutorul bibliotecii pynput.mouse pe *link*-ul către melodia dorita.

1 Modul de funcționare al *virtual assistant*-ului

În acest capitol voi prezenta arhitectura aplicației, funcțiile importante, modul în care au fost modelate datele, interacțiunile dintre date și interfața grafică etc.

1.1 Arhitectura din spatele *virtual assistant*-ului

Virtual assistant-ul are la baza un *loop* de tipul `while True` și trei funcții importante:

- danySays(audio): aceasta este funcția care va fi mereu apelată atunci când „Dany” zice câte ceva.

- myCommand(): această funcție va memora comanda pe care o va da utilizatorul *virtual assistant*-ului.

- assistant(command): această funcție va verifica dacă utilizatorul a rostit o comandă pe care sa o recunoască „Dany”, comanda fiind imediat executată de către *virtual assistant,* în cazul în care îi este familiară.

- În *loop* avem:
```
while True:
assistant(myCommand())
```

  Așadar, *virtual assistant*-ul va asculta după o comanda de la utilizator la infinit. Astfel se realizează dialogul dintre cei doi.

1.2 Funcțiile de bază ale lui *virtual assistant*-ului

În acest subcapitol voi discuta mai în detaliu cele trei funcții de baza ale lui „Dany” și despre interfața grafică. Astfel sper să pot oferi o imagine în ansamblu asupra aplicației și asupra modului în care a fost gândită și proiectată.

1.2.1 Funcția danySays(audio)

```
58  def danySays(audio):
59      aux = ('Dany: ' + audio)
60      print aux
61      write(aux)
62      root.update_idletasks()
63      root.update()
64      mixer.init()
65      language = 'en'
66      sf = TemporaryFile()
67      myobj = gTTS(text = audio, lang = language, slow = False)
68      myobj.write_to_fp(sf)
69      audio = MP3(sf)
70      audio_length = audio.info.length
71      sf.seek(0)
72      mixer.music.load(sf)
73      mixer.music.play()
74      time.sleep((audio_length) + 0.1)
```

Această funcție este cât se poate de simplă: primește ca parametru un *string,* pe care îl va rosti cu voce tare. Apelul de funcție `write(aux)`,de pe linia 61, are ca rol afișarea *string*-ului aux în interfața grafică. Apelez astfel pe linia 67 funcția gTTS() care îmi transformă textul într-un fișier .mp3, care va fi salvat într-un *temporary file.* După aceea, pe linia 73 apelez funcția `mixer.music.play()`pentru a da *play* la fișierul .mp3 cu răspunsul lui „Dany”.

Pe linia următoare(linia 74) avem `time.sleep((audio_length) + 0.1).` Prin intermediul acesteia „Dany” va vorbi o linie de dialog până la final, va aștepta o zecime de secundă, după care va începe să rostească cea de-a doua linie de dialog. Am introdus această linie de cod, tocmai pentru că se mai întâmpla ca „Dany” sa nu termine de zis o frază, să se oprească, și să înceapă sa zică altceva.

Ultimele două linii de cod din funcția de mai sus sunt folosite pentru interfața grafică. Ce doresc să spun prin acest lucru este că, având în vedere faptul că interfața este gândită să fie *idle by default,* am nevoie să îi dau *update* de fiecare dată când doresc să îmi afișez ceva nou. Astfel, aceste ultime două linii apar tocmai pentru că doresc să afișez tot ceea ce spune *virtual assistant*-ul imediat după ce a terminat de vorbit.

1.2.2 Funcția myCommand()

```
78  #https://towardsdatascience.com/build-your-first-voice-assistant-85a5a49f6cc1
79  def myCommand():
80      r=sr.Recognizer()
81      with sr.Microphone() as source:
82          r.adjust_for_ambient_noise(source, duration = 1)
83          audio = r.listen(source)
84      try:
85          command = r.recognize_google(audio).lower()
86          aux2 = ('User: ' + command + '\n')
87          write(aux2)
88          root.update_idletasks()
89          root.update()
90          print aux2
91      except sr.UnknownValueError:
92          #print('...')
93          command = myCommand()
94      return command
```

Această funcție am găsit-o pe deja implementată pe internet, așadar am atașat mai sus link-ul către site-ul de unde am găsit-o. *Link*-ul este de asemenea atașat în secțiunea de „Bibliografie” a acestui document. Funcția `myCommand()` are ca rol preluarea unei comenzi vocale dată de către utilizator. Acest lucru se realizează pe linia 83:

```
audio = r.listen(source)
```

Linia de mai sus de aceasta îl face pe *virtual assistant* să asculte nivelul zgomotului din jur timp de o secundă și să facă diverse ajustări pentru a îl auzi mai bine pe utilizator.

Urmează o secvența de tipul `try-except.` Aceasta are rolul de a nu întrerupe execuția programului în cazul unei pauze puțin mai lungi în conversație. Astfel fără această structură, dacă *virtual assistant*-ul ar fi pornit, acesta s-ar aștepta să primească o comanda, însă dacă nu ar primi nicio comandă ar returna o eroare, și astfel ar întrerupe execuția programului.

În interiorul lui `try` avem linia de cod:

```
command = r.recognize_google(audio).lower()
```

Aceasta are scopul de a face string-ul în care este salvată comanda dată de către utilizator să fie scris tot cu litere mici. Rolul este de a simplifica comparația dintre comenzi și șabloanele pe care le știe „Dany”.

1.2.3 Funcția assistant(command)

```
108  #function for executing commands
109  def assistant(command):
110      root.update_idletasks()
111      root.update()
112
113      #time
114      if 'time' in command:
115          try:
116              currentTime = datetime.datetime.now()
117              danySays('Current time is ' + currentTime.strftime("%X"))
118          except Exception:
119              danySays('I am sorry, I cannot tell you the time')
120
121
122      #date
123      elif 'date' in command:
124          try:
125              currentDate = datetime.datetime.now()
126              danySays('Today is ' + currentDate.strftime("%x"))
127          except Exception:
128              danySays('I am sorry, I do not know what day is today')
```

```
534      #any command dany doesn't know
535      else:
536          danySays("Sorry, I didn't quite catch that...")
```

Această funcție este de departe cea mai mare ca lungime, având peste 400 de linii de cod. Aici sunt implementate toate comenzile care se afla la baza *virtual assistant*-ului. Cu toate acestea funcția începe prin a face un update la interfața grafica. Acest lucru este necesar pentru afișarea corectă și la timp a conversației în interfață.

După aceea începe o structura de tipul `if-else-elif.` Aici se verifică daca în interiorul comenzii date de utilizator se află vreun cuvânt cheie pe care *virtual assistant*-ul să îl

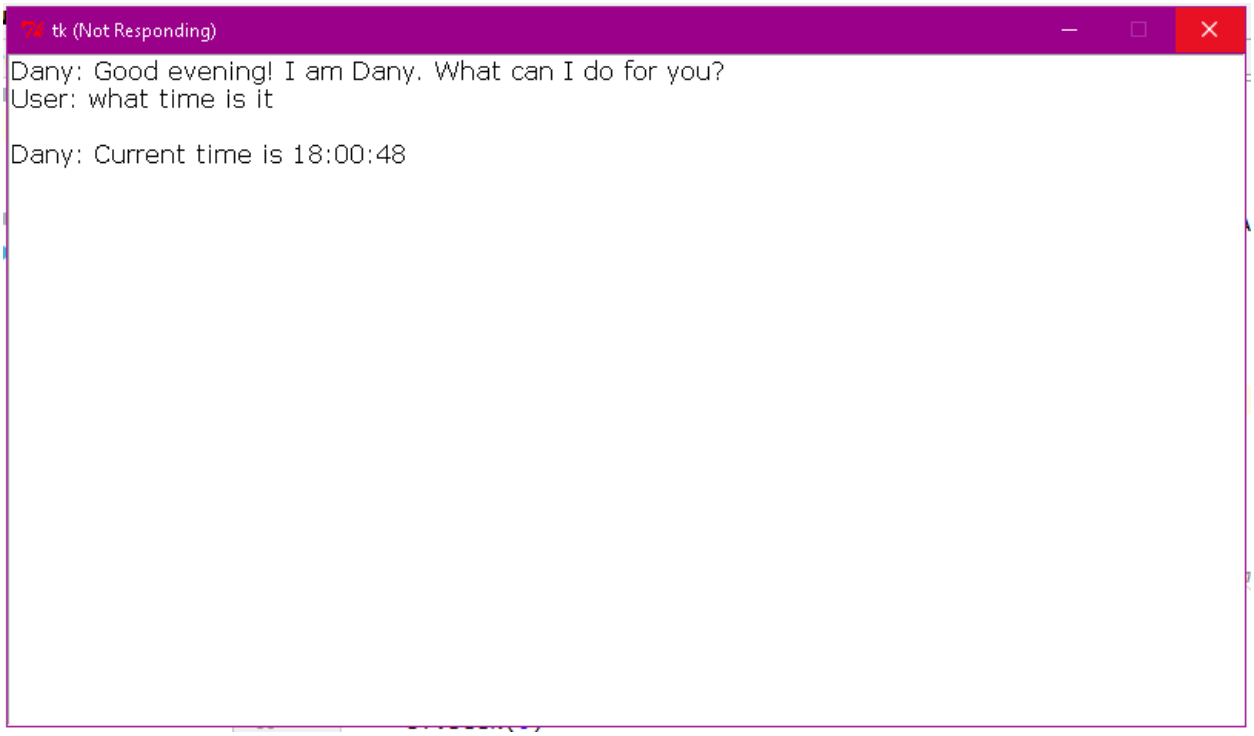
asocieze cu vreo comandă. Dacă utilizatorul a rostit o comandă care nu conține nici un cuvânt cheie „Dany” va zice că nu înțelege ce i s-a spus.

Dacă utilizatorul rostește o frază care să conțină unul dintre cuvintele cheie asociate de către „Dany” cu una dintre cele 21 de comenzi, atunci *virtual assistant*-ul va începe executarea respectivei comenzi. Astfel dacă utilizatorul întreabă: „*what **time** is it*?”, „Dany” va verifica dacă vreunul dintre cuvinte este cuvânt cheie. Așadar, *virtual assistant*-ul va descoperi cuvântul „time” pe care îl știe, și deci va începe execuția secvenței de cod de sub linia 114:

```
if 'time' in command:.
```

Fiecare secvență de cod ce aparține unei comenzi este pusă într-o structură de tipul `try-except.` Acest lucru este făcut cu scopul de a asigura bună rulare a codului în cazul în care una dintre comenzi va da *return* la o eroare. Motivul este asigurarea oferirii unei experiențe *seamless* utilizatorului. Așadar, în cazul în care apare vreo eroare, „Dany” îi va zice utilizatorului că acea comandă este indisponibilă, însă *virtual assistant*-ul va asculta în continuare și după alte comenzi; programul nu se va opri.

1.2.4 Interfața grafică




```
43  #these are for the GUI
44  #https://www.reddit.com/r/Learnprogramming/comments/3vq0dm/python_how_can_i_print_text_out_in_the_gui_rather/
45  def write(string):
46      text_box.config(state = Tkinter.NORMAL)
47      text_box.insert("end", string + "\n")
48      text_box.see("end")
49      text_box.config(state = Tkinter.DISABLED)
50
51  root = Tkinter.Tk()
52  text_box = Tkinter.Text(root, wrap = "word", state = Tkinter.DISABLED, font = 'verdana')
53  text_box.grid(row = 0, column = 0, columnspan = 4)
54  root.update_idletasks()
55  root.update()
```

Codul pentru interfața grafică a fost găsit pe site-ul menționat mai sus. Am menționat de asemenea site-ul și în secțiunea de „Bibliografie” a acestui document. Varianta de cod pe care am găsit-o pe internet a necesitat însă unele modificări pentru a putea fi integrată în cadrul aplicației mele.

În prima poză se poate observa interfața grafică a aplicației. După cum am menționat și mai sus, ferastra va fi mereu *not responding,* până când se va scrie ceva în ea.

Cea de-a doua poză începe cu funcția `write(string).` Aceasta este o funcție care îmi face îmi scrie un *string* în interfață. Funcția începe prin a face interfața să nu mai fie *disabled,* cu ajutorul liniei de cod 46:

```
text_box.config(state = Tkinter.NORMAL)
```

Acum că interfața nu mai este *not responding,* îmi voi scrie *string*-ul pe care doresc să îl afișez, după care voi da din nou *disable* la interfață prin intermediul liniei de cod:

```
text_box.config(state = Tkinter.DISABLED)
```

Fac acest lucru pentru a asigura buna execuție a programului, întrucât fără a da *disable* la interfață, *loop*-ul ce stă la baza întregii aplicații nu mai funcționează cum trebuie.

Liniile de cod ce urmează(51 - 55) au rolul de a crea interfața grafică; astfel pe linia 52 aleg fontul cu care se va scrie în interfață, selectez modul *default* al interfeței ca fiind *disabled,* și îi zic interfeței că în cazul în care un cuvânt nu încape pe linia pe care îl scrie, să treacă pe linia următoare, nu să despartă cuvântul. Pe următoarea linie este setată dimensiunea interfeței. Ultimele doua linii au rolul de a afișa interfața grafică încă de la începutul execuției programului (fără aceste doua linii de cod interfața ar apărea doar după ce „Dany” rostește prima replică, nu de la început).

1.3 Explicație asupra ultimului *import*

Ca o ultimă precizare în acest capitol, aș dori să vorbesc puțin despre ultimul *import* făcut în cadrul aplicației:

```
36  with open(os.devnull, 'w') as f:
37      oldstdout = sys.stdout
38      sys.stdout = f
39      from pygame import mixer
40      sys.stdout = oldstdout
```

Această secvență de cod are ca rol întreruperea *stdout*-ului până când este importată biblioteca pygame. După importarea bibliotecii *stdout*-ul este pornit înapoi la loc. Acest lucru

este făcut deoarece atunci când importăm în mod normal biblioteca pygame, ne este afișat un mesaj în consolă. Am considerat că acest mesaj poate lăsa o impresie neplăcută utilizatorului, așa că am decis să îl scot. În poza de mai jos se poate observa acest mesaj:

```
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
Dany: Good evening! I am Dany. What can I do for you?
```

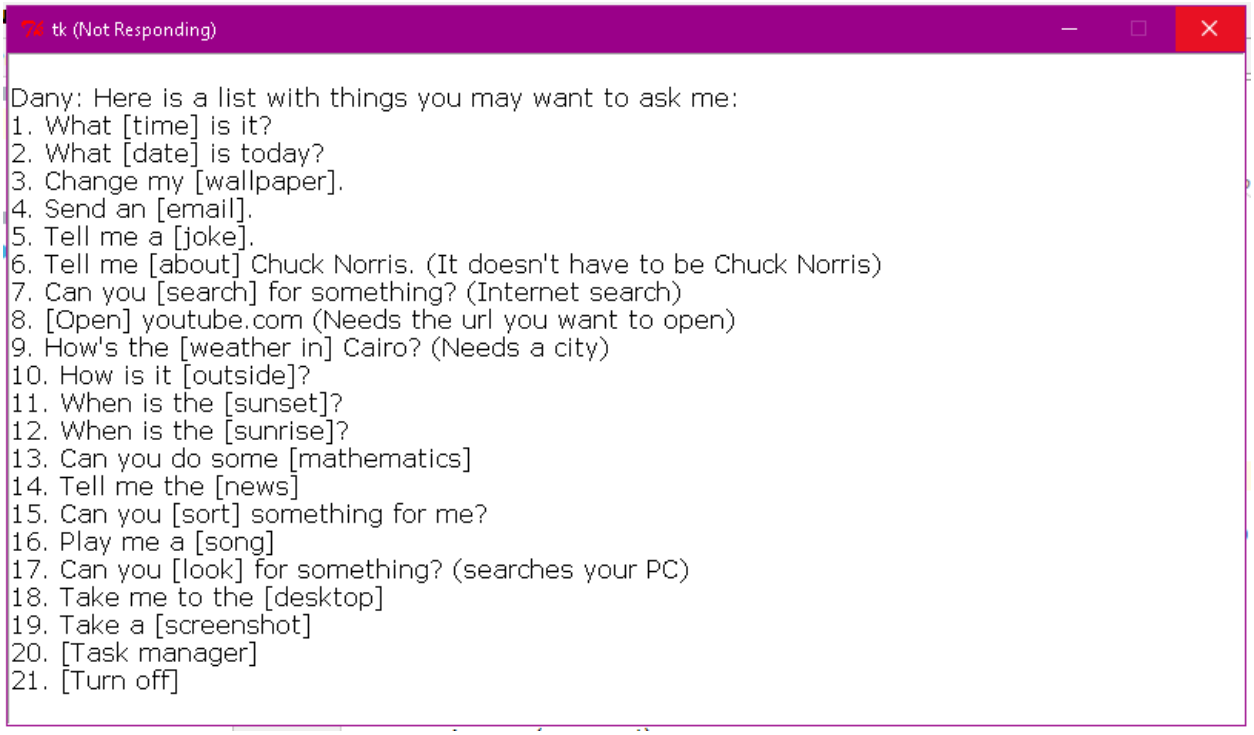
2 Interacțiunile utilizatorului cu *virtual assistant*-ul prin intermediul comenzilor

Pe decursul acestui capitol voi prezenta modul în care utilizatorul va interacționa cu *virtual assistant*-ul. Așadar voi prezenta modul în care funcționează fiecare comandă și felul în care ar trebui să decurgă o conversație între „Dany” și utilizator.

De asemenea voi preciza și dificultățile pe care le-am întâmpinat în realizarea proiectului, precum si soluțiile pe care le-am găsit pentru depășirea acestora.

2.1 Comanda „help”

Aceasta este prima comandă pe care ar trebui să o folosească un utilizator nou. Comanda „help” nu face decât sa listeze toate cele 21 de comenzi. Astfel utilizatorul își poate face o idee despre capabilitățile lui „Dany” și știe de unde să înceapă.



Acesta este *output*-ul comenzii „help”. După cum se poate observa „Dany” va oferi câte un exemplu de utilizare pentru fiecare comandă. Țin să menționez faptul că fiecare cuvânt cheie (cuvânt pe care „Dany” îl asociază cu o comandă) este trecut între paranteze pătrate.

2.2 Comenzile *virtual assistant*-ului

În acest subcapitol voi prezenta modul în care funcționează comenzile lui „Dany”. De asemenea am să încerc să grupez comenzile similare într-un singur capitol.

2.2.1 Comenzile „date” și „time”

Aceste două comenzi îmi afișează data și respectiv ora la momentul de față. Comenzile sunt cât se poate de simple; nu fac decât să apelez funcția:

```
datetime.datetime.now()
```

După care îmi prelucrez puțin datele și le afișez.

2.2.2 Comanda „wallpaper”

Această comandă are ca rol schimbarea *wallpaper*-ului PC-ului. Astfel *virtual assistant*-ul alege o poză la întâmplare de pe PC-ul utilizatorului și o setează pe post de *wallpaper.*
Schimbarea de *wallpaper* se face cu ajutorul liniei de cod următoare:

```
ctypes.windll.user32.SystemParametersInfoA(SPI_SETDESKWALLPAPER, 0,
"C:\Users\Andrei Sfarghiu\Pictures\Saved Pictures\Wallpaper_1.jpg", 0)
```

Astfel folosim biblioteca ctypes pentru a da *load* la user32.dll, după care apelăm funcția SystemParametersInfoA() pentru a schimba *wallpaper*-ul.

2.2.3 Comanda „email”

Această comandă este utilizată pentru trimiterea de *email*-uri de către utilizator. Astfel am utilizat biblioteca smtplib pentru a crea o conexiune către *server*-ul de *email* Gmail, la portul 587. După aceea am apelat funcțiile `.ehlo()` și `.starttls()` pentru a mă identifica și respectiv pentru a securiza conexiunea.

Utilizatorul va fi întrebat cui dorește să îi trimită *email*-ul. Acesta are de ales dintre mai mulți destinatari dintr-o listă pe care i-o va prezenta „Dany”. Mi-am dorit ca utilizatorul să îi dicteze *virtual assistant*-ului adresa la care să trimită *email*-ul, însă după ce am implementat acest *feature* am realizat că „Dany” nu înțelege foarte bine numele românești. Așadar, în favoarea unei experiențe cât mai *smooth* am decis că ar fi cel mai bine ca utilizatorul să aibă de ales adresa destinatarului dintr-o lista predefinită.

După ce utilizatorul alege destinatarul, acesta îi va dicta mesajul *email*-ului lui „Dany”. *Email*-ul va fi apoi trimis prin intermediul funcției sendmail().

2.2.4 Comanda „joke”

Această comandă va face *virtual assistant*-ul să îi spună o glumă utilizatorului. Acest lucru este posibil prin utilizarea bibliotecii requests. Astfel apelăm funcția `requests.get` și facem un *GET request* pe o pagină ce conține de fiecare dată o glumă la întâmplare cu Chuck Norris.

2.2.5 Comanzile „about”, „search” și „open”

Comanda „about” are rolul de a-i oferi utilizatorului informații despre aproape orice și oricine. Astfel este necesar ca utilizatorul să spună „*Tell me about Freddie Mercury*” iar *virtual assistant*-ul va oferi informații despre Freddie Mercury. Este important să apară cuvântul „*about*” înaintea subiectului unei căutări, întrucât aplicația folosește

```
re.search('about (.*)', command)
```

pentru a căuta informații despre ceea ce apare dupa cuvântul „*about*”. După aceea este utlizată funcția summary() din biblioteca wikipedia pentru a căuta pe wikipedia informații despre ceea ce dorește utilizatorul să știe. În cazul în care căutarea s-a efectuat cu succes „Dany” va citi prima propoziție de pe wikipedia. Dacă utilizatorul își dorește să afle mai multe, „Dany” va deschide pagina, pentru ca acesta să poată să o citească. Dacă în schimb căutarea nu s-a efectuat cu succes (daca nu există pagină pe wikipedia despre ceea ce dorește utilizatorul să caute), atunci „Dany” va face o căutare pe Google și va deschide această pagină pentru utilizator.

Comanda „search” este foarte similară cu cazul în care „Dany” nu găsește informația căutata prin comanda „about” pe wikipedia. Așadar, prin această comandă, utilizatorul face o căutare pe Google. Comanda concatenează un URL incomplet al unei căutări pe Google cu ceea ce zice utilizatorul, după care folosește open() din biblioteca webbrowser pentru a deschide pagina web aferentă căutării.

Comanda „open” este folosită pentru deschiderea unui *site* prin intermediul URL-ului. Astfel dacă utilizatorul zice „Open youtube.com” *virtual assistant*-ul îi va deschide pagina cerută. Acest lucru se face asemănător cu modul de căutare al comenzii „search”; se face tot prin concatenare de URL.

Codul pentru comanda „about” poate fi văzut la „Anexa 1”.

2.2.6 Comenzile „weather in” și „outside”

Aceste două comenzi folosesc API-ul de la OWM (Opwn Weather Map) pentru a îl informa pe utilizator în legătură cu vremea. Prima comandă este utilizată pentru a afla vremea în orice oraș de pe Pământ, iar a doua este folosită pentru a afla vremea de afară.

Cunoașterea vremii este posibilă prin intermediul utilizarea bibliotecii PyOWM și prin folosirea unui *API-key* oferit de Open Weather Map. Astfel dacă utilizatorul zice „How is the

weather in Cairo”, *virtual assistant*-ul știe că trebuie să facă o căutare a vremii după numele orașului, nume ce apare după cuvintele cheie „weather in”.

Comanda „outside” este ceva mai simpla; va returna vremea de la un anumit set de coordonate. Aș dori să menționez faptul că aceste coordonate au fost fixate pe poziția orașului Iași.

2.2.7 Comenzile „sunset” și „sunrise”

Aceste două comenzi au rolul de a îl informa pe utilizator de ora la care apune și respectiv răsare soarele. În acest scop este utilizată biblioteca astral.

Aș dori să menționez faptul că deși aceste două comenzi nu par foarte importante, au fost implementate pentru a servi un scop personal. Am proiectat acest *virtual assistant* pentru a fi în primul rând asistentul meu personal. Așadar, cum am început destul de recent un *hobby* unde ora de răsărit a soarelui este destul de importantă, am decis să implementez și o metodă care să îmi spună această oră.

2.2.8 Comanda „mathematics”

Această comandă vine în completarea comenzii „about” (cea care caută pe wikipedia). Ce vreau să zic prin acest lucru este faptul că aceste două comenzi, luate împreună, ar trebui să poată oferii utilizatorului informații despre aproape orice.

Astfel prin intermediul comenzii „mathematics” utilizatorul va introduce o întrebare pe *site*-ul wolframalpha.com. Acest lucru este posibil prin intermediul API-ului oferit de WolframAlpha. Așadar „Dany” folosește un *API-key* pentru a face o căutare pe *site*-ul *wolframalpha.com,* returnând utilizatorului răspunsul.

2.2.9 Comanda „news”

Această comandă are rolul de a îl informa pe utilizator în legătură cu știrile. Astfel utilizatorul va fi întrebat dacă dorește să audă știrile locale, sau știrile globale. Dacă va fi aleasă opțiunea locală „Dany” va deschide o pagină către news.google.ro. Dacă în schimb utilizatorul va dori să afle știrile globale, atunci „Dany” va deschide o pagină către news.google.com, dar în același timp va începe să facă *parsing* asupra *RSS feed*-ului de la Google News și va citi în timp real titlurile primelor trei știri.

Parsing-ul este posibil prin intermediul bibliotecii BeautifulSoup4. Astfel virtual assistant-ul va găsi prin fișierul .xml al paginii tot ceea ce face parte din grupul „*item*” (grup asociat titlurilor de știri).

Codul pentru comanda „news” poate fi văzut la „Anexa 2”.

2.2.10 Comanda „sort”

Această comandă utilizează algoritmul „BubbleSort” pentru a sorta un *string*. Acest *string* poate fi compus din numere, caz în care va fi sortat crescător, sau poate fi format din cuvinte, caz în care va fi sortat alfabetic.

2.2.11 Comanda „song”

Această comandă are rolul de a pune o melodie (sau un videoclip care nu este o melodie) pe Youtube. *Virtual assistant*-ul va concatena URL-ul „https://www.youtube.com/results?search_query=” împreună cu titlul melodiei oferit de către utilizator. Astfel am reușit să ajungem pe pagina cu toate rezultatele căutării. Urmează ca „Dany” sa facă un *onclick event* și astfel să simuleze o apăsare a *mouse*-ului asupra melodiei dorite. Acest lucru este posibil datorită utilizării bibliotecii pynput.mouse.

Aș dori să menționez că original am dorit să fac *parsing* asupra codului HTML al paginii de YoutTube, însă am avut dificultăți în această privință. După ceva timp însă mi-a venit ideea de a fac un *onclick event,* și am reușit astfel să simplific întregul mod de execuție al comenzii.

2.2.12 Comenzile „look”, „desktop”, „screenshot” și „task manager”

Toate aceste patru comenzi folosesc biblioteca pynput.keyboard pentru a simula apăsarea de taste de pe tastatură.

Astfel comanda „look” va căuta în fișierele locale, de pe *hard disk*-ul PC-ului, folosind tasta „Windows key” pentru a porni funcția de căutare integrată deja pe sistemul de operare Windows 10.

Comanda „desktop” are rolul de a minimiza toate aplicațile deschise de utilizator și de a îl duce pe acesta pe *desktop*. În cazul în care utilizatorul se află deja pe *desktop,* comanda va maximiza toate aplicațiile anterior minimizate. Acest lucru este posibil simulând combinația de taste „Windows key” și „D”.

Comanda „screenshot” se folosește de combinația de taste „windows key” și „print screen” pentru a face un *screenshot* și pentru a îl salva într-un anumit fișier. Dacă utilizatorul dorește să vadă poza imediat după ce este făcută, acesta poate menționa acest lucru, iar „Dany” va deschide fișierul cu pricina.

Comanda „task manager” este utilizata pentru a deschide *task manager*-ul fără a mai deschide meniul denumit „Ctrl+Alt+Del Options”. Acest lucru este posibil prin simularea apăsării tastelor „Ctrl”, „Shift” și „Esc”. Aș dori să menționez faptul că pe parcursul dezvoltării acestei aplicații, am sesizat faptul că am folosit această comandă cel mai mult; Mi-a fost mult mai la îndemâna sa folosesc comanda vocala „task manager” pentru a monitoriza buna execuție a programului. De asemenea am folosit comanda și pentru a observa și închide procese costisitoare din punct de vedere al memoriei consumate.

2.2.13 Comanda „turn off”

Comanda „turn off” este utilizată pentru a opri execuția virtual assistant-ului. Astfel dacă utilizatorul dorește ca „Dany” să nu mai audă ce zice acesta, trebuie apelată această comandă.

Comanda funcționează apelând funcția exit() din biblioteca sys.

Concluzii

Această lucrare a avut ca scop dezvoltarea unui *virtual assistant* care să înlăture pe cât posibil interacțiunea utilizatorului cu PC-ul prin intermediul *mouse*-ului și a tastaturii. Așadar consider că din acest punct de vedere „Dany" este un adevărat succes. Este adevărat că „Dany" nu poate să facă totul; am încercat să folosesc PC-ul timp de o zi întreagă doar cu ajutorul acestui *virtual assistant,* și deși mi-a luat ceva timp să mă obișnuiesc, am ajuns să apreciez modul organic în care se îmbină comenzile vocale cu *mouse*-ul și tastatura. Astfel, până la finalul zilei am reușit să folosesc deodată ambele metode de a interacționa cu PC-ul.

Cu toate acestea nu consider ca *virtual assistant*-ul creat de mine este perfect. Există nenumărate îmbunătățiri pe care i le-aș putea aduce și tot nu aș fi mulțumit întru totul. Doresc să continui dezvoltarea lui „Dany" și după finalizarea examenlui de licența; doresc să aduc îmbunătățiri în vederea folosirii lui „Dany" precum un Amazon Echo. Astfel voi putea beneficia de un *virtual assistant* personalizat, și despre care știu cu siguranță că nu colectează datele utilizatorului în cine știe ce scop nejustificat.

Pe parcursul dezvoltării acestei aplicații am fost nevoit să recurg la căi inventive pentru depășirea unor obstacole. Mi-a făcut însă plăcere să depun efortul de a căuta diferite metode de implementare și să o aleg pe ceas mai potrivită. Astfel consider că fiecare pas din dezvoltarea lui „Dany" a fost bine gândit. Sper ca pe viitor, dacă va mai continua și altcineva dezvoltarea acestui *virtual assistant,* să înțeleagă de ce am implementat totul în acest mod. După părerea codul aplicației este ușor de înțeles, funcțiile, metodele și variabilele având fiecare câte un nume sugestiv.

„Dany " pentru mine a reprezentat un pas în față; acesta a fost de departe proiectul cel mai intensiv din punctul de vedere al timpului necesar dezvoltării sale. Așadar pot spune cu adevărat că sunt mândru de ceea ce am realizat și că sper ca și alții să fie la fel de impresionați ca și mine de capabilitățile lui „Dany".

Un ultim lucru pe care aș dori să îl menționez este că lucrul la acest proiect m-a învățat cum să îmi gestionez corect timpul. Astfel am învățat ca fiecare pas din proiectarea unei aplicații este

important. Așadar pentru mine munca la acest proiect a fost una plăcută, ale cărei roade am avut plăcerea să le văd abia la sfârșit.

Bibliografie

Am decis să împart bibliografia în trei secțiuni diferite: cărți pe care le-am citit și care mi-au fost de folos, *link*-uri documentații ale diverselor biblioteci din python și *link*-uri folositoare de pe internet.

Cărți

[1] „How To Think Like A Computer Scientist: Learning Python” scrisă de Allen Downey, Jeff Elkner și Chris Meyers

[2] „Python Cookbook” de Alex Martelli, Anna Martelli Ravenscroft și David Ascher

Documentații ale bibliotecilor utilizate

[1] Documnetația bibliotecii SpeechRecognition: https://pypi.org/project/SpeechRecognition/

[2] Documentația bibliotecii gTTS: https://pypi.org/project/gTTS/

[3] Documentația bibliotecii requests: https://pypi.org/project/requests/

[4] Documentația bibliotecii wikipedia: https://pypi.org/project/wikipedia/

[5] Documentația bibliotecii pyowm: https://pypi.org/project/pyowm/

[6] Documentația bibliotecii astral: https://pypi.org/project/astral/

[7] Documentația bibliotecii mutagen: https://pypi.org/project/mutagen/

[8] Documentația bibliotecii pynput: https://pypi.org/project/pynput/

[9] Documentația bibliotecii wolframalpha: https://pypi.org/project/wolframalpha/

Link-uri folositoare

În această secțiune am decis să menționez toate *link*-urile către paginile pe care am găsit metode de a implementa diverse funcționalități. Aș dori să menționez că majoritatea acestor metode le-am implementat și eu în proiectul meu; bineînțeles, am modificat aceste implementări pentru a se potrivii cu modul în care funcționează „Dany”.

[1] https://towardsdatascience.com/build-your-first-voice-assistant-85a5a49f6cc1

[2] https://gtts.readthedocs.io/en/latest/module.html#examples

[3] https://rogulski.it/blog/alexa-skill-in-python/index.html

[4] https://stackoverflow.com/questions/33186255/how-to-extract-values-from-a-python-request

[5] https://wikipedia.readthedocs.io/en/latest/quickstart.html

[6] https://www.w3schools.com/python/python_regex.asp

[7] https://openweathermap.org/guide

[8] https://github.com/csparpa/pyowm

[9] https://pyowm.readthedocs.io/en/latest/usage-examples-v2/weather-api-usage-examples.html#owm-weather-api-version-2-5-usage-examples

[10] https://github.com/KhanradCoder/PyDa-Course-Code

[11] https://www.geeksforgeeks.org/python-program-for-bubble-sort/

[12] https://astral.readthedocs.io/en/latest/module.html

[13] https://mutagen.readthedocs.io/en/latest/user/gettingstarted.html

[14] https://codereview.stackexchange.com/questions/164275/python-text-to-speech-program

[15] https://nitratine.net/blog/post/simulate-keypresses-in-python/

[16] https://stackoverflow.com/questions/51464455/why-when-import-pygame-it-prints-the-version-and-welcome-message-how-delete-it

[17] https://www.reddit.com/r/learnprogramming/comments/3vq0dm/python_how_can_i_print_text_out_in_the_gui_rather/

[18] https://buildmedia.readthedocs.org/media/pdf/wolframalpha/latest/wolframalpha.pdf

Anexe

Am decis să atașez aici câteva secvențe de cod la care am făcut trimitere de-a lungul acestei lucrări.

Anexa 1

```
    #searches wikipedia; if it finds a disambiguation page, it searches google instead
    elif 'about' in command:
        search_var = re.search('about (.*)', command)
        try:
            topic = search_var.group(1)
            danySays(wikipedia.summary(topic, sentences = 1))
            danySays('Would you like to know more?')
            response = myCommand()
            if 'yes' in response or 'yeah' in response:
                danySays('Okay! I will open the wikipedia page on ' + topic + ' for you to read')
                topic2 = topic.replace(" ", "_")
                site = 'https://en.wikipedia.org/wiki/' + topic2
                webbrowser.open(site)
            else:
                danySays('Okay. Anything else I can do for you?')
        #the exception will happen when the page I try to acces doesn't exist or is a disambiguation page
        except Exception:
            topic = search_var.group(1)
            topic2 = topic.replace(" ", "+")
            danySays('I am sorry, I cannot tell you about that. I will instead search the web for ' + topic)
            site = 'https://www.google.com/search?client=firefox-b-d&q=' + topic2
            webbrowser.open(site)
```

Aici se poate observa codul pentru comanda „about” a *virtual assistant*-ului.

Anexa 2

```
358     #news from google's rss feed
359     #https://towardsdatascience.com/build-your-first-voice-assistant-85a5a49f6cc1
360     elif 'news' in command:
361         try:
362             danySays('Would you like the local news or the world news?')
363             response = myCommand()
364             if 'local' in response:
365                 local_news = "https://news.google.com/?hl=ro&gl=RO&ceid=RO:ro"
366                 webbrowser.open(local_news)
367                 danySays('Okay! Opening local news...')
368             elif 'world' in response:
369                 world_news = "https://news.google.com/?hl=en-US&gl=US&ceid=US:en"
370                 world_news_rss = "https://news.google.com/rss?hl=en-US&gl=US&ceid=US:en"
371                 reader = urlopen(world_news_rss)
372                 xml = reader.read()
373                 reader.close()
374                 bSoup = soup(xml, "lxml")
375                 news_titles = bSoup.findAll("item")
376                 webbrowser.open(world_news)
377                 for news in news_titles[:3]:
378                     danySays(news.title.text)
379         except Exception:
380             danySays('I am sorry, I cannot find any news')
381
```

Aici se poate observa codul pentru comanda „news” a *virtual assistant*-ului.