UNIVERSIDADE FEDERAL DO PARANÁ

CURSO DE MATEMÁTICA

Dyckson Ternoski

Magnum Carvalho de Oliveira

Vinicius Alves dos Santos

SISTEMA DE GERENCIAMENTO INTERNO DE CLIENTES EM PYTHON

CURITIBA

2018

Dyckson Ternoski (GRR20185648)

Magnum Carvalho de Oliveira (GRR20185658)

Vinicius Alves dos Santos (GRR20185660)

SISTEMA DE GERENCIAMENTO INTERNO DE CLIENTES EM PYTHON

Relatório apresentado à disciplina Fundamentos de Programação de Computadores do Curso de Graduação em Matemática da Universidade Federal do Paraná.

Orientador: Prof. Jackson Antônio do Prado Lima

Curitiba, novembro de 2018

SUMÁRIO

1 INTRODUÇÃO

Esse relatório apresenta informações relativas ao trabalho realizado em Python sobre sistema de gerenciamento interno de clientes, desenvolvido pelo grupo Os Procrastinadores.

A linguagem Python foi concebida em 1989 pelo holandês Guido van Rossum, visto que ele estava desenvolvendo a linguagem ABC no CWI em Amsterdã – Holanda, e estava encontrando deficiências nessa linguagem. Tentando suprir esses problemas visto com o ABC, o holandês criou o Python com base em C.

2 OBJETIVOS

O objetivo geral do trabalho é a criação de um sistema de gerenciamento interno de clientes em Python, de forma a substituir a administração lenta e manual. Este programa, após criado, será usado em uma empresa; portanto, o código foi construído de modo que se adequasse às necessidades dela.

Os meios para chegar até este objetivo foram descritos junto ao envio do tema, e consistem em:

• Criar um arquivo que contenha o cadastro dos clientes, isto é: nome, RG, telefone e eventuais informações adicionais;

• Sistema de buscas: um "Ctrl+F" que localize determinado cadastro por meio de uma das informações dadas;

• Criação de um arquivo modificável (relatório) contendo os dados de todos os clientes que marcaram passagem para viajar em um certo dia;

• Função adicionar e remover clientes do relatório a partir do sistema de busca;

• Conversão do relatório para PDF, para ser salvo e impresso;

• Gerenciar usando telegram.

3 DESENVOLVIMENTO

O trabalho foi dividido em dois arquivos python: um chamado "principal.py" que contém e executa as funções do programa principal, e outro chamado "bot.py", sendo este o bot do telegram.

3.1 Programa Principal

O programa principal.py conta com 9 funções afim de serem executadas através de um menu, que está no programa (fora das funções). Nele são utilizadas as bibliotecas datetime, pandas e reportlab, que serão abordadas nas funções em que são utilizadas.

3.1.1 Função cadastrar

A primeira função serve para cadastrar clientes em um banco de dados. Para isso, criamos um arquivo modificável *.txt* e utilizamos a função *open()* do python para abri-lo no *Appending mode*, que permite a adição de novas informações neste arquivo. Assim, os novos clientes cadastrados são adicionados sempre ao fim do banco de dados.

```
8  def cadastrar():
9      '''
10     Esta função faz o cadastro de novos passageiros no banco de dados.
11     '''
12     cadastros = open('cadastros.txt','a')
13
14     novo_cadastro = input('Deseja adicionar novo cadastro? (s/n) >')
15     novo_cadastro = validar(novo_cadastro, 1)
16
17     # Este laço de repetição serve para adicionar quantos cadastros forem necessários.
18     while novo_cadastro == 's':
19
20         nome = input('Digite Nome: ')
21         rg = input('Digite o RG: ')
22         telefone = input('Digite o Telefone: ')
23
24         # Padroniza as informações dentro do arqquivo.
25         cadastro = nome + ';' + rg + ';' + telefone + ';\n'
26         cadastros.write(cadastro)
27
28         novo_cadastro = input('Deseja adicionar novo cadastro? (s/n) >')
29         novo_cadastro = validar(novo_cadastro, 1)
30
31     cadastros.close()
```

Figura 1 - Função para cadastrar clientes

3.1.2 Função busca

A segunda função diz respeito à procura de clientes neste banco de dados anteriormente mencionado. Agora o arquivo *.txt* é aberto no modo *read*, ou seja, apenas para leitura. Também é utilizada a função *readlines()* para colocar todas informações do arquivo em uma lista, e em seguida, procuramos o cliente desejado nesta lista, usando um *for*. Se o cliente for encontrado, há a opção de adicioná-lo à lista de embarque. A função utiliza um laço de repetição para continuar procurando clientes, e termina quando a reposta "n" é dada para a pergunta "Deseja procurar outro cadastro?". A função *validar* será tratada mais adiante, devido à necessidade do conhecimento das outras funções primeiro. Com relação ao resto das funções: assim que uma função aparecer no código, ela será explicada logo em seguida.

```
34  def busca():
35      '''
36      Esta função realiza a busca de nomes dentro do bando de dados.
37      '''
38      while True:
39          texto = input("Digite nome ou RG ou Telefone que deseja procurar: ")
40          # Esta parte lê todo o arquivo e coloca as informações em uma lista.
41          cadastros = open('cadastros.txt','r')
42          lista = cadastros.readlines()
43          lista.sort()
44          cadastros.close()
45
46          encontrados = []
47          # Laço de repetição para procurar o texto informado pelo usuário na lista.
48          for x in lista:
49              if texto in x:
50                  encontrados.append(x)
51
52          # Esta parte separa em casos de acordo com a quantidade de cadastros encontrados.
53          if len(encontrados) == 0:
54              print('Não foi encontrado cadastro para "{0}"'.format(texto))
55
56          elif len(encontrados) == 1:
57              formatar_texto_saida(encontrados)
58              relatorio = input("Deseja adicionar à lista de embarque? (s/n) >")
59              relatorio = validar(relatorio, 1)
60
61              if relatorio == 's':
62                  add_lista_embarque(encontrados)
63
64          else:
65              print('Foram encontrados os seguintes cadastros: \n')
66              formatar_texto_saida(encontrados)
67
68          busca = input('Deseja procurar outro cadastro? (s/n) >')
69          busca = validar(busca, 1)
70
71          if busca == 'n':
72              break
```

Figura 2 - Função de busca

3.1.3 Função add_lista_embarque

A terceira função é chamada pela função anterior e adiciona o cliente encontrado na função *busca()* em uma lista de embarque. São fornecidas 5 opções de lugares para embarcar. Caso nenhuma delas seja conveniente, digita-se "o" e em seguida é aberto um *input* para digitar o nome do local. Por último, ainda na função, as informações do cliente são formatadas para ficarem padronizadas na lista.

```
74  def add_lista_embarque(cliente):
75      '''
76      Esta função recebe como argumento o cliente encontrado na busca
77      (quando apenas um é encontrado) e o adiciona na lista de embarque.
78      '''
79
80      # Esta parte solicita ao usuário uma informação importante para a lista de embarque.
81      print('''
82      Onde será o embarque?
83      (g) p/ Garagem
84      (s) p/ Santa Marta
85      (r) p/ Praça Rui Barbosa
86      (h) p/ Habbib`s
87      (t) p/ Tulio
88      (o) p/ Outro
89      ''')
90
91      embarque = input('>')
92      embarque = validar(embarque, 3)
93
94      if embarque == 'g':
95          lugar = 'Garagem'
96      elif embarque == 's':
97          lugar = 'Santa Marta'
98      elif embarque == 'r':
99          lugar = 'Rui Barbosa'
100     elif embarque == 'h':
101         lugar = "Habbib's"
102     elif embarque == 't':
103         lugar = 'Tulio'
104     elif embarque == 'o':
105         lugar = input('Digite o lugar de embarque: ')
106
107     # Esta parte formata a informação do cliente para ser adicionada à lista de embarque.
108     aux1 = cliente[0].split(';')
109     aux1[-1] = lugar
110     aux1.pop(1)
111     lista_embarque.append(aux1)
112     lista_embarque.sort()
```

Figura 3 - Função de adicionar um cliente à lista de embarque

3.1.4 Função formatar_texto_saida

A quarta função também é chamada pela função *busca*, e também será utilizada na função *editar*, que será vista mais adiante. Essa função é chamada de *formatar_texto_saida* e serve para deixar o cadastro do cliente aparecendo padronizado no *output* do programa (print).

```
275  def formatar_texto_saida(texto):
276      '''
277      Função auxiliar para formatar as informações do cadastro e imprimir na tela.
278      '''
279      for x in texto:
280          aux = []
281          aux = x.split(';')
282          print('Nome: {0}, RG: {1}, Telefone: {2}\n'.format(aux[0], aux[1], aux[2]))
283
```

Figura 4 - Função para formatar o texto das informações do cliente

3.1.5 Função visualiza_lista_embarque

A quinta função é utilizada para visualizar a lista de impressão antes de gerar o arquivo .PDF correspondente a ela.

```
114  def visualiza_lista_embarque(lista_embarque):
115      '''
116      Esta função recebe a lista de embarque como argumento e cria um Data Frame (tabela)
117      usando a biblioteca Pandas para visualização da lista antes de gerar o PDF para impressão.
118      '''
119      # Verifica se a lista não está vazia.
120      if len(lista_embarque) > 0:
121          df = pd.DataFrame(data=lista_embarque, index=range(1, len(lista_embarque)+1), columns=['Nome', 'Telefone', 'Embarque'])
122          print(df)
123
124          a = input('Deseja remover alguém da lista? (s/n) >')
125          a = validar(a, 1)
126
127          if a == 's':
128              remover()
129      else:
130          print('Ainda não foi adicionado ninguém na lista de embarque.')
```

Figura 5 - Função para visualizar a lista de embarque

3.1.6 Função remover

A sexta função é a *remover()*, vista no código da função anterior. Como intuitivamente o nome sugere, ela serve para remover um cliente da lista de embarque.

```
def remover():
    '''
    Esta função remove algum cliente da lista de embarque.
    '''
    remover_linha = int(input('Digite o número da linha que deseja remover :'))
    remover_linha = validar(remover_linha, 4)

    del lista_embarque[int(remover_linha)-1]

    # Esta parte verifica se a lista não está vazia e pergunta se o usuário deseja remover outro cliente
    # e gera novamente a Tabela para visualização da lista
    if len(lista_embarque) > 0:
        df = pd.DataFrame(data=lista_embarque, index=range(1, len(lista_embarque)+1), columns=['Nome', 'Telefone', 'Embarque'])
        print(df)

        a = input('Deseja remover mais alguém? (s/n) >')
        a = validar(a, 1)

        while a == 's':
            # Esta parte é igual a parte de cima, porém dentro do laço de repetição caso o usuário deseje remover mais de um cliente.
            remover_linha = int(input('Digite o número da linha que deseja remover :'))
            remover_linha = validar(remover_linha, 4)

            del lista_embarque[int(remover_linha)-1]

            if len(lista_embarque) > 0:
                df = pd.DataFrame(data=lista_embarque, index=range(1, len(lista_embarque)+1), columns=['Nome', 'Telefone', 'Embarque'])
                print(df)

                a = input('Deseja remover mais alguém? (s/n) >')
                a = validar(a, 1)
            else:
                print('Lista de embarque vazia.')
                break
    else:
        print('Lista de embarque vazia.')
```

Figura 6 - Função para remover um cliente da lista de embarque

3.1.7 Função editar

O menu principal também oferece uma opção para editar o cadastro de um cliente já existente. Isso é um caso recorrente e necessário quando se tem um banco de dados de cadastros, já que os clientes podem mudar de número de celular ou telefone, por exemplo. Para suprir essa necessidade, criamos a função editar.

```python
def editar():
    '''
    Esta função permite editar as informações de algum cadastro no banco de dados.
    '''

    while True:
        # Abre o arquivo e copia os dados para dentro de uma lista.
        cadastros = open('cadastros.txt','r')
        lista = cadastros.readlines()
        cadastros.close()

        # Esta parte faz uma busca por texto informado pelo usuário de forma similar à função busca acima.
        texto = input("Qual cadastro deseja editar? >")
        encontrados = []

        for x in lista:
            if texto in x:
                encontrados.append(x)

        # Verifica se foi encontrado nenhum, um ou mais cadastros de acordo com o texto digitado.
        if len(encontrados) == 0:
            print('Não foi encontrado cadastro para "{0}"'.format(texto))

        elif len(encontrados) == 1:
            formatar_texto_saida(encontrados)
            # Pergunta qual informação o usuário deseja alterar.
            edit = input('Editar Nome, RG, Telefone ou buscar outro cadastro? (nome/rg/tel/outro) >')
            edit = validar(edit, 2)

            # Esta parte cria uma lista auxiliar para "segurar" as informações do cliente.
            aux = []
            aux = encontrados[0].split(';')
            aux.pop()

            if edit == 'outro':
                pass
            else:
                # Esta parte altera a informação desejada.
                if edit == 'nome':
                    aux[0] = input('Nome: ')
                elif edit == 'rg':
                    aux[1] = input('RG: ')
                elif edit == 'tel':
                    aux[2] = input('Telefone: ')

                # Esta parte abre o arquivo do banco de dados, substitui as informações do cadastro editado e reescreve o arquivo.
                cadastros = open('cadastros.txt', 'w')
                pos = lista.index(encontrados[0])
                lista.pop(pos)
                aux = aux[0] + ';' + aux[1] + ';' + aux[2] + ';\n'
                lista.append(aux)
                lista.sort()

                cadastros.writelines(lista)

                cadastros.close()
```

Figura 7 - Função para editar o cadastro de clientes

3.1.8 Função imprimir

Nessa função utilizamos a biblioteca Reportlab e DateTime, explicadas nos comentários do código.

```
240 def imprimir(lista_embarque):
241     '''
242     Esta função gera um arquivo PDF contendo a lista de embarque para impressão.
243     '''
244     # Esta parte inicia a criação da lista de embarque na forma de uma tabela em PDF usando a biblioteca Reportlab.
245     doc = SimpleDocTemplate("lista_embarque.pdf", pagesize=A4)
246     elements = []
247 
248     # Laço de repetição para inserir índices em cada linha na lista de embarque.
249     for x in range(len(lista_embarque)):
250         i = str(x+1)
251         lista_embarque[x].insert(0, i)
252 
253     # Esta parte utiliza a biblioteca DateTime para descobrir o dia que o documento está sendo gerado.
254     hoje = datetime.datetime.now()
255     hoje = hoje.strftime("%d/%m/%Y")
256     # Cria um título para a tabela contendo um texto padrão e a data.
257     titulo = 'LISTA DE EMBARQUE - VIAGEM DIA '+hoje
258 
259     # Insere na primeira linha da lista o título e na segunda linha os nomes das colunas.
260     lista_embarque.insert(0, ['     ', 'Nome','Telefone','Embarque','Polt.','Pago'])
261     lista_embarque.insert(0, [titulo])
262 
263     # Formatação da tabela pela biblioteca Reportlab.
264     t=Table(lista_embarque)
265     t.setStyle(TableStyle(
266         [('ALIGN', (0,0), (-1,-1), 'CENTER'),
267         ('SPAN', (0,0), (-1,0)),
268         ('INNERGRID', (0,0), (-1,-1), 0.25, colors.black),
269         ('BOX', (0,0), (-1,-1), 0.25, colors.black)]
270     ))
271     elements.append(t)
272     # Salva o documento criado na pasta que o programa está sendo executado.
273     doc.build(elements)
```

Figura 8 - Função para gerar um documento PDF contendo a lista de embarque

3.1.9 Função validar

Finalmente chegamos na função validar, que aparece várias vezes durante o código. A função recebe o parâmetro *n* como um inteiro para identificar uma certa categoria a ser abordada. A validação que ocorre consiste em laços de repetição, que são executados até que seja digitada uma das opções válidas.

Veja o código na próxima página. Em seguida, recordaremos onde cada inteiro (n = 1, n = 2, n = 3 e n = 4) apareceu no código.

```
285 def validar(escolha, n):
286     '''
287     Esta função recebe dois argumentos, o primeiro é uma string digitada pelo usuário
288     e a segunda é um inteiro para identificar uma categoria, e então garante que
289     o usuário terá digitado corretamente a informação solicitada pelo programa.
290     '''
291 
292     if n == 1:
293         while escolha != 's' and escolha !='n':
294             escolha = input('Comando inválido, digite "s" para sim ou "n" para não >')
295 
296     elif n == 2:
297         while escolha != 'nome' and escolha != 'rg' and escolha != 'tel' and escolha != 'outro':
298             print('''
299     Comando inválido
300     Digite "nome" para editar o Nome
301     Digite "rg" para editar o RG
302     Digite "tel" para editar o Telefone
303     Digite "outro" para buscar outro cadastro
304     ''')
305             escolha = input('>')
306 
307     elif n == 3:
308         while escolha != 'g' and escolha != 's' and escolha != 'r' and escolha != 'h' and escolha != 't' and escolha != 'o':
309             print('''
310     Opção inválida, as opções disponíveis são as seguintes:
311     (g) p/ Garagem
312     (s) p/ Santa Marta
313     (r) p/ Praça Rui Barbosa
314     (h) p/ Habbib`s
315     (t) p/ Tulio
316     (o) p/ Outro)
317     ''')
318             escolha = input('>')
319 
320     elif n == 4:
321         while escolha > len(lista_embarque):
322             escolha = int(input('A lista possui apenas {0} clientes, digite novamente o número da linha :'.format(len(lista_embarque))))
323 
324     return escolha
```

Figura 9 - Função que valida as respostas dadas em relação à escolhas

3.1.9.1 Casos em que n = 1

```
68          busca = input('Deseja procurar outro cadastro? (s/n) >')
69          busca = validar(busca, 1)
```

Figura 10 - Função busca(), linhas 68 e 69.

Nesse caso, o texto é *busca*, que espera-se que seja "s" ou "n". Como podemos ver no código da função validar, o inteiro 1 serve para validar respostas de "sim" ou "não" no formato s/n. Veja abaixo mais casos em que a função validar foi usada com esse fim.

```
124          a = input('Deseja remover alguém da lista? (s/n) >')
125          a = validar(a, 1)
```

Figura 11 - Função visualiza_lista_embarque(lista_embarque), linhas 124 e 125

```
149          a = input('Deseja remover mais alguém? (s/n) >')
150          a = validar(a, 1)
```

Figura 12 - Função remover(), linhas 149 e 150

3.1.9.2 Caso em que n = 2

Nas linhas abaixo, a função é chamada para validar a resposta, que deve ser "nome", "rg", "tel" ou "outro".

```
196             # Pergunta qual informação o usuário deseja alterar.
197             edit = input('Editar Nome, RG, Telefone ou buscar outro cadastro? (nome/rg/tel/outro) >')
198             edit = validar(edit, 2)
```

Figura 13 - Função editar(), linhas 196 a 198

3.1.9.3 Caso em que n = 3

No contexto abaixo, a função validar é chamada com o intuito de legitimar a resposta do local de embarque.

```
80      # Esta parte solicita ao usuário uma informação importante para a lista de embarque.
81      print('''
82      Onde será o embarque?
83      (g) p/ Garagem
84      (s) p/ Santa Marta
85      (r) p/ Praça Rui Barbosa
86      (h) p/ Habbib`s
87      (t) p/ Tulio
88      (o) p/ Outro
89      ''')
90
91      embarque = input('>')
92      embarque = validar(embarque, 3)
```

Figura 14 - Função add_lista_embarque(cliente), linhas 80 a 92

3.1.9.4 Caso em que n = 4

Dessa vez a função é chamada para tornar válido o texto *remover_linha*, que deve ser um inteiro menor ou igual ao número de linhas, já que se há 9 clientes na lista de embarque, não existe a possibilidade de remover o cliente da linha 12.

```
134 def remover():
135     '''
136     Esta função remove algum cliente da lista de embarque.
137     '''
138     remover_linha = int(input('Digite o número da linha que deseja remover :'))
139     remover_linha = validar(remover_linha, 4)
```

Figura 15 - Função remover(), linhas 134 a 139

3.1.10 O programa

Após definir todas as funções, chegamos ao programa principal, que contém o menu e um laço de repetição para a escolha nesse menu. Cada escolha realizada chama uma função para desempenhar o que foi pedido.

```
# Início do programa.
print('''
    Digite 'c' para adicionar novo cadastro
    Digite 'b' para buscar cadastro
    Digite 'e' para editar cadastro
    Digite 'v' para visualizar documento de impressão
    Digite 'i' para imprimir a lista de embarque
    Digite 's' para sair
''')

escolha = input('>')
lista_embarque = []

while True:

    if escolha == 'c':
        cadastrar()

    elif escolha == 'b':
        busca()

    elif escolha == 'e':
        editar()

    elif escolha == 'v':
        visualiza_lista_embarque(lista_embarque)

    elif escolha == 'i':
        print('''
    Tem certeza que deseja gerar lista de embarque para impressão e fechar o programa?
    Você pode escolher "n" e em seguida selecionar "v" para verificar se os dados na
    lista de embarque estão corretos antes de imprimir.
            ''')
        certeza = input('(s/n) >')
        certeza = validar(certeza, 1)

        if certeza == 's':
            imprimir(lista_embarque)
            print('Lista de embarque gerada com sucesso!')
            print('Fim de execução!')
            break

    elif escolha == 's':
        sair = input('Tem certeza que deseja sair? A lista de embarque será perdida! (s/n) >')
        sair = validar(sair, 1)

        if sair == 's':
            break

    else:
        print('Opção inválida, tente novamente!')

    print('''
    Digite 'c' para adicionar novo cadastro
    Digite 'b' para buscar cadastro
    Digite 'e' para editar cadastro
    Digite 'v' para visualizar documento de impressão
    Digite 'i' para imprimir a lista de embarque
    Digite 's' para sair
    ''')

    escolha = input('>')
```

Figura 16 - O programa

3.2 Bot no Telegram

Nosso trabalho também conta com um bot no Telegram (TesteViagensBot) para agendamento de viagens. Enquanto o código estiver rodando no cmd, o bot estará recebendo mensagens e respondendo-as. Em algumas poucas mensagens, o cliente realiza o agendamento da viagem, que será enviado por Telegram para o dono do bot para aprovação. Nessa aprovação, são inseridas informações extras. Após a confirmação, um e-mail é enviado para o cliente com esses dados sobre a viagem.

Para a construção do código do bot, foram utilizadas as bibliotecas *telepot*, *email.mime* e *smtplib*. A biblioteca *telepot* serve para as funções básicas do bot no Telegram, como enviar e receber mensagens. Já a *email.mime* e *smtplib* servem para formatar o texto do email e enviar o e-mail respectivamente.

O código é constituído de 5 funções e algumas poucas linhas fora das definições delas. Na maior parte do código, os comentários explicam totalmente o que cada função faz.

3.2.1 Conhecendo como o telepot funciona

Na linha 7, definimos nosso bot.

```
6  #Este código abaixo é o token do bot do Telegram. É através dele que o Python comandará o bot.
7  bot = telepot.Bot("781465027:AAF61wjbH08clR3UIQCQWAoAnvb4HL_VpXM")
8  #print(bot.getUpdates())
9  '''
10 Esse print acima, quando executado, mostra todos os detalhes
11 e informações de uma mensagem enviada, do tipo da mensagem e do usuário que enviou.
12 Entretanto, aqui usaremos apenas a mensagem em si e o chatid do usuário.
13 '''
```

Figura 17 - Explicação do telepot

3.2.2 Função recebendoMsg

No Moodle, na parte de Avisos (OFF TOPIC: Bot para Telegram usando Python), há um vídeo (que está nas referências deste trabalho) ensinando como fazer para um bot no Telegram estar sempre recebendo mensagens. Nossa função recebendoMsg é uma adaptação da função do vídeo. As variáveis *mensagem* e *cid* foram colocadas como globais dentro da função, pois elas são chamadas diversas vezes em outras funções. Também, isso se faz necessário devido ao fato de que essa função recebendoMsg está sempre rodando. Isso acontece por conta da linha 140, 155 e 156, mostradas logo abaixo da imagem do código da função, e é isto que faz com que o bot esteja sempre recebendo mensagens enquanto o código estiver rodando no cmd.

```
14  mensagem = ""
15  cid = 0
16
17  #Essa função recebe as mensagens que são enviadas para o bot
18  def recebendoMsg(msg):
19      global mensagem
20      global cid
21      mensagem = msg['text'] # mensagem = mensagem enviada pelo usuário no telegram
22      cid = msg['chat']['id'] # cid = chatid do usuário, é o número pelo qual o bot
23                              #chamará ele no telegram
24      return mensagem
25      return cid
```

Figura 18 - Função que faz o bot receber as mensagens enviadas por algum usuário

```
139 #Esse comando deixa o recebimento de mensagens em loop enquanto o programa estiver rodando
140 bot.message_loop(recebendoMsg)
```

Figura 19 - Loop do recebimento de mensagens

```
150 '''
151 Esse laço faz com o que o programa fique rodando para sempre (loop infinito),
152 e assim o bot sempre receberá mensagens até que
153 o terminal seja fechado manualmente (Ctrl+C)
154 '''
155 while True:
156     pass
```

Figura 20 - Loop infinito do programa

3.2.3 Função conversa

A função conversa, como o nome diz, consiste na realização de uma conversa entre o usuário e o bot. Ela recebe dois parâmetros: *m*, que é uma mensagem enviada pela pessoa utilizando o bot, e *chatid*, o número que identifica a conta dela no Telegram.

Quando é iniciado um chat no Telegram com um bot, para começar é necessário clicar em "começar", que manda a mensagem "/start" para o bot. Logo, o comando para começar a conversa é */start* ou "Agendar", caso a pessoa já tenha feito um agendamento anteriormente em outra ocasião.

Figura 21 - Exemplo de início de conversa com um bot

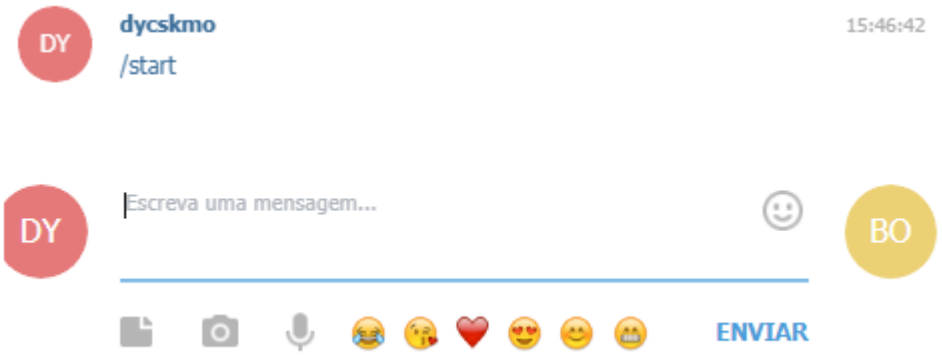

Figura 22 - Após clicar em "Começar", a mensagem /start é automaticamente enviada ao bot

Após isso, a conversa segue solicitando informações e termina chamando a função *permitir* , que logo será explicada.

```
def conversa(m,chatid):
    if m == "/start" or m == "Agendar":
        bot.sendMessage(chatid,"Olá. Seja bem vindo ao TesteViagensBot.\nPreencha as informações abaixo para agendar uma viagem.")
        bot.sendMessage(chatid,"Nome:")
        '''
        A função sendMessage pertence à biblioteca do telepot,
        e serve para que o bot mande uma mensagem. Nesse caso,
        ele está mandando uma mensagem para a pessoa com o id identificado
        na função recebendoMsg e o corpo da mensagem é "Nome:"

        O laço abaixo serve para que o bot só continue os comandos
        quando o usuário digitar a resposta da pergunta anterior
        '''
        while mensagem == "/start" or mensagem == "Agendar":
            pass
        nome = mensagem
        bot.sendMessage(chatid,"RG:")
        while mensagem == nome:
            pass
        rg = mensagem
        bot.sendMessage(chatid,"Telefone:")
        while mensagem == rg:
            pass
        telefone = mensagem
        bot.sendMessage(chatid,"E-mail:")
        while mensagem == telefone:
            pass
        email = mensagem
        bot.sendMessage(chatid,"Local de embarque: (Opções padrão: Garagem, Santa Maria, Praça Rui Barbosa, Habib's, Tulio)")
        while mensagem == email:
            pass
        local = mensagem
        bot.sendMessage(chatid,'''Agendamento realizado.
        Após a confirmação da viagem, você receberá um e-mail com todas as informações necessárias.
        Para agendar outra viagem, digite "Agendar"''')
        permitir(nome,rg,telefone,email,local)
```

Figura 23 - Função que realiza a conversa entre o bot e o cliente que quer agendar a viagem

Essa função conversa também está sendo chamada em loop (veja abaixo), mas perceba que ela só gera efetivamente a conversa de agendamento quando a mensagem for "/start" ou "Agendar". Caso a pessoa mande outras mensagens, o bot não responderá nada. Isso evita uma sobrecarga do bot. Caso houvesse uma resposta "Digite /start ou Agendar" para cada mensagem diferente, alguém poderia "floodar" o bot mandando várias mensagens "a" seguidas, por exemplo, e então o bot ficaria sempre emitindo essa mensagem, deixando o programa mais lento para quem realmente quer utilizá-lo.

```
142 '''
143 Esse laço faz com que a função conversa sempre seja chamada
144 para qualquer mensagem enviada. Assim que a pessoa digitar /start,
145 a função faz seu papel
146 '''
147 while True:
148     conversa(mensagem,cid)
```

Figura 24 - Loop infinito que sempre está chamando a função conversa.

3.2.4 Função cadastrar

O papel desempenhado por essa função é o mesmo da função cadastrar do programa principal. Quando a função é chamada, os dados do cliente são jogados para o mesmo arquivo em que estão os clientes adicionados pelo programa principal. O cadastro só é permitido caso haja a aprovação do dono do bot. Veremos isso agora na função *permitir*.

```
68 def cadastrar(nome,rg,telefone):
69     '''
70     Essa função será utilizada para cadastrar os dados que o usuário forneceu
71     na função conversa, e eles serão guardados no arquivo .txt,
72     que contém as informações sobre os clientes
73     '''
74     cadastros = open('cadastros.txt','a')
75     cadastro = nome + ';' + rg + ';' + telefone + ';\n'
76     cadastros.write(cadastro)
77     cadastros.close()
```

Figura 25 - Função para cadastro de clientes

3.2.5 Função permitir

Quando essa função é chamada, o bot envia mensagens para o dono do bot para a confirmação do cadastro do cliente e envio do e-mail com as informações adicionais sobre a viagem. Essas informações são fornecidas neste mesmo momento pelo dono do bot.

As restrições nos laços de repetição "and cid == 680275748" são necessárias para que apenas o dono do bot possa confirmar e mandar as informações. Caso a mensagem seja "Sim", mas o chatid que enviou ela não seja do dono do bot, então o comando não será executado. Desse modo, a permissão é concedida apenas pelo dono. O *chatid* utilizado no código (680275748) é da nossa conta no Telegram. Não foi colocado *else* nos *ifs* justamente por conta dessa restrição obrigatória do chatid.

```python
def permitir(nome,rg,telefone,email,local):
    '''
    Para impedir flood e bagunça do arquivo .txt com a informação dos clientes,
    sempre que dados forem fornecidos para um agendamento, eles precisarão passar por
    aprovação do dono do bot. Assim, só serão cadastrados os clientes que são realmente reais,
    e não clientes fakes jogados no chat do bot para "inundar" o arquivo.
    É um anti-spam, porém seletivo manualmente.
    '''
    bot.sendMessage(680275748, '''Um cliente realizou um agendamento. Confira os dados da viagem:
        Nome: {0}
        RG: {1}
        Telefone: {2}
        E-mail: {3}
        '''.format(nome,rg,telefone,email))
    bot.sendMessage(680275748, "Permitir o cadastro no arquivo de clientes? (Sim/Nao)")
    while mensagem == local:
        pass
    while mensagem != "Sim" and mensagem != "Nao" and cid == 680275748:
        nova = mensagem
        bot.sendMessage(680275748,"Digite Sim ou Nao para responder")
        while mensagem == nova:
            pass
    if mensagem == "Sim" and cid == 680275748:
        cadastrar(nome,rg,telefone)
        bot.sendMessage(680275748,"Cadastro adicionado com sucesso")
        bot.sendMessage(680275748,"Adicione as informações referentes à viagem:")
        bot.sendMessage(680275748,"Horário de embarque:")
        while mensagem == "Sim" and cid == 680275748:
            pass
        horario = mensagem
        bot.sendMessage(680275748,"Preço:")
        while mensagem == horario and cid == 680275748:
            pass
        preco = mensagem
        bot.sendMessage(680275748,"Feito. Viagem confirmada e um e-mail com as informações foi enviado para o cliente.")
        mandar_email(nome,rg,telefone,email,local,horario,preco)
    elif mensagem == "Nao" and cid == 680275748:
        bot.sendMessage(680275748, "Cadastro descartado")
```

Figura 26 - Função para permitir e confirmar o agendamento da viagem

3.2.6 Função mandar_email

Nessa função, um e-mail é enviado automaticamente para o cliente, confirmando então o seu agendamento. Para a realização dessa função, foi criado o email *paristurismocwb@gmail.com*, com senha *telegram* para que o Python fizesse login nele e enviasse os e-mails; isso foi feito utilizando a biblioteca smtp. O servidor usado para tal fim foi smtp.gmail.com, com autenticação StartTLS e porta 587.

A função recebe 7 parâmetros, sendo 6 deles usados na construção do texto do e-mail, e o parâmetro *email* sendo justamente o endereço de e-mail que o cliente forneceu no cadastro com o bot.

Após essa função, o que resta no código são os comandos de loop, já mostrados anteriormente. Portanto, o código está terminado e o bot pronto para ser usado. Basta rodar o código no cmd e conversar com ele no Telegram: TesteViagensBot. Caso queira receber as mensagens da função *permitir*, basta substituir o chatid (cid) na função pelo chatid da sua conta.

```python
def mandar_email(nome,rg,telefone,email,local,horario,preco):
    '''
    Nessa função, serão utilizados comandos das bibliotecas MIMEMultipart, MIMEText e smtp
    para enviar um e-mail ao cliente que agendou a viagem.
    As bibliotecas MIME cuidam da formatação do texto, e a smtp
    é responsável pela criação do servidor para envio automático do e-mail.
    '''
    msg = MIMEMultipart()

    message = '''Olá. Sua viagem foi confirmada. Seguem abaixo as informações:
    Nome: {0}
    RG: {1}
    Telefone: {2}
    Local de embarque: {3}
    Horário de embarque: {4}
    Preço: {5}
    '''.format(nome,rg,telefone,local,horario,preco)

    password = "telegram"
    msg['From'] = "paristurismocwb@gmail.com"
    msg['To'] = email
    msg['Subject'] = "Confirmação da viagem - Paris Turismo"

    msg.attach(MIMEText(message, 'plain'))

    server = smtplib.SMTP('smtp.gmail.com: 587')
    server.starttls() #Este comando é importante para tornar a conexão segura
    server.login(msg['From'], password)
    server.sendmail(msg['From'], msg['To'], msg.as_string())
    server.quit()
```

Figura 27 - Função responsável por mandar o e-mail de confirmação para o cliente

```python

#Esse comando deixa o recebimento de mensagens em loop enquanto o programa estiver rodando
bot.message_loop(recebendoMsg)

'''
Esse laço faz com que a função conversa sempre seja chamada
para qualquer mensagem enviada. Assim que a pessoa digitar /start,
a função faz seu papel
'''
while True:
    conversa(mensagem,cid)

'''
Esse laço faz com o que o programa fique rodando para sempre (loop infinito),
e assim o bot sempre receberá mensagens até que
o terminal seja fechado manualmente (Ctrl+C)
'''
while True:
    pass
```

Figura 28 - Após a função, temos os loops das funções recebendoMsg e conversa, já abordadas

4 CONCLUSÃO

Por fim, após concluir os códigos e ter ambos os programas (principal e bot no Telegram) funcionando, podemos afirmar que cumprimos todos nossos objetivos.

A princípio esse projeto parecia fácil de ser realizado e seria feito muito rápido, porém no processo de desenvolvimento nos deparamos com inúmeras dificuldades. A ideia original era fazer o programa seguindo uma ordem de início ao fim separando em casos de acordo com as funcionalidades do programa, mas isso deixaria o código desorganizado e de difícil compreensão e leitura, por isso optamos pela criação de funções.

A nova estrutura do código ficou muito mais organizada e permitiu uma fácil implementação de novas funcionalidades não previstas na ideia original para o programa, além de que a parte principal ficou curta e intuitiva.

No geral, construir o código foi difícil e encontramos vários problemas pelo caminho. Um dos desafios foi aprender a usar as bibliotecas: cada necessidade do programa era um novo "tour" pelo Google lendo documentações e passando por vários fóruns até achar a biblioteca adequada.

Também mudamos nosso planejamento diversas vezes: em uma delas, após criar o código da lista de embarque utilizando a biblioteca PrettyTable, quando chegou na função de gerar o PDF percebemos que ela não era boa para isso, e a tabela ficaria muito feia, longe do que gostaríamos. Acabamos mudando para o Pandas e refizemos a função da lista de embarque utilizando esta nova biblioteca. Entretanto, tivemos outros problemas.

O novo plano se tornou utilizar a biblioteca Pandas para a construção de uma tabela referente à lista de embarque para então criar um documento PDF para impressão, mas isto se mostrou como uma tarefa extremamente difícil de executar pois, dentre outros motivos, precisava da utilização da linguagem de programação HTML, e por isso foi decidido que esta tabela seria usada apenas dentro do programa para fácil manipulação de dados (inserção e exclusão de clientes da lista de embarque) e que seria utilizada a biblioteca Reportlab para gerar o arquivo PDF, e com isso o resultado obtido foi satisfatório. Além disso, construir tabelas com o DataFrame do Pandas foi bem melhor do que com o PrettyTable. Deve-se mencionar também que foi utilizada a biblioteca DateTime para inserir no arquivo PDF final a data correspondente.

Outra ocasião de mudança de planos diz respeito ao módulo/biblioteca Tkinter. As telas estavam deixando o programa muito lento, cerca de 15-30 segundos para

rodar o programa e mais uns 10 segundos entre cada click (decisão, correspondente ao input) na tela. Tentamos otimizar, mas sem muito sucesso. Como as telas eram algo apenas para embelezar o programa e não uma necessidade urgente, acabamos por desistir da biblioteca e fizemos o programa com prints no cmd mesmo.

Com relação ao bot do Telegram, foi algo novo e também divertido. Como esperado, foi um tempo até entender a biblioteca do telepot, mas o vídeo tutorial no Moodle ajudou bastante para começar. É incrível como após entender os comandos e funções da biblioteca, a construção do código flui perfeitamente, e a realização após terminá-lo e vê-lo funcionando é maravilhosa.

Este trabalho foi muito enriquecedor pois tivemos que nos adaptar à lógica de programação, aprender a pesquisar e compreender as diversas bibliotecas e funcionalidades existentes, além de suas documentações. Fixamos os conceitos mais importantes na programação e, por fim, ganhamos perspectiva de um mundo de possibilidades que a programação de computadores permite fazer.

REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (ABNT). Informação documentação - Trabalhos acadêmicos - Apresentação. **NBR14724**. Rio de Janeiro, 2011.

AMADEU, M. S. U. et al. **Manual de normalização de documentos científicos de acordo com as normas da ABNT**. Curitiba, 2015.

WILL. **Python Telegram Bot - Como criar um bot, receber e enviar mensagens.** 2017. Disponível em: <https://www.youtube.com/watch?v=2TCkaJdcicQ>.

LEE, Nick. **Introduction - telepot 12.7 documentation.** 2018. Disponível em: <https://telepot.readthedocs.io/en/latest/>.

VICENZI, Alexandre. **Enviando emails com o Python.** 2014. Disponível em: <https://butecoopensource.github.io/enviando-emails-com-o-python/>

SEMPREUPDATE. **Usando Python para enviar e-mails (Parte 2 - módulo email).** 2016. Disponível em: <https://sempreupdate.com.br/python-enviando-emails-modulo-email-programar/>

DRISCOLL, Michael. **Report Tables - Creating Tables in PDFs with Python.** 2010. Disponível em: <https://www.blog.pythonlibrary.org/2010/09/21/reportlab-tables-creating-tables-in-pdfs-with-python/>

PANDAS-DEV. **Pandas Documentation - PyData.** 2018. Disponível em: <https://pandas.pydata.org/pandas-docs/stable/>

REPORTLAB. **ReportLab Documentation.** 2018. Disponível em: <https://www.reportlab.com/documentation/>

IGNORÂNCIA ZERO. **Aulas Python - 051 - Arquivos I: Modos de Abertura e Métodos.** 2014. Disponível em: <https://www.youtube.com/watch?v=JJb9DlNWQqQ>