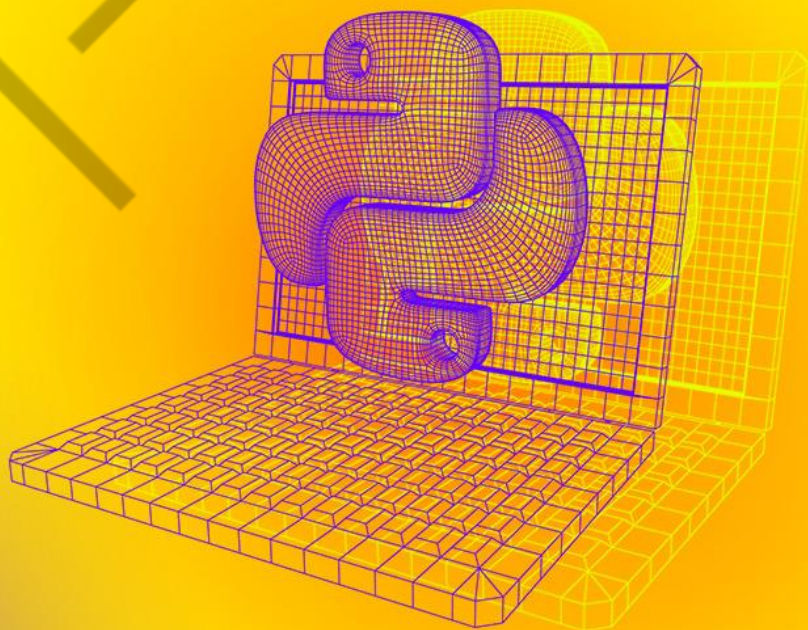


PYTHON DEVELOPMENT

TÓPICOS AVANÇADOS



6

LISTA DE FIGURAS

Figura 1 – Formato de exibição de contato único.....	12
Figura 2 – Resultado a função agenda_para_texto.....	14
Figura 3 – Na criação de um novo projeto, o PyCharm já providencia um ambiente virtual com o venv	40

EMPRE

LISTA DE QUADROS

Quadro 1 – Funções de interação com o usuário.....	20
--	----

EMSE

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Script de gravação de arquivo de texto.....	7
Código-fonte 2 – Script de gravação de arquivo de texto usando a função with	8
Código-fonte 3 – Script de leitura de arquivo de texto usando a função with	8
Código-fonte 4 – Script de gravação de arquivo json.....	9
Código-fonte 5 – Script de leitura de arquivo json.....	10
Código-fonte 6 – Criação da tupla com os contatos suportados e da agenda de exemplo.....	12
Código-fonte 7 – Criação da estrutura da função contato_para_texto	12
Código-fonte 8 – Função contato_para_texto	13
Código-fonte 9 – Criação da estrutura da função agenda_para_texto	13
Código-fonte 10 – Função agenda_para_texto	14
Código-fonte 11 – Função altera_nome_contato	15
Código-fonte 12 – Função altera_forma_contato	16
Código-fonte 13 – Teste das funções de alteração	16
Código-fonte 14 – Função excluir.....	17
Código-fonte 15 – Função inclui_contato	18
Código-fonte 16 – Teste da função inclui_contato	18
Código-fonte 17 – início da função inclui_forma_contato	18
Código-fonte 18 – início da função inclui_forma_contato	19
Código-fonte 19 – Função inclui_forma_contato	19
Código-fonte 20 – Função exibe_menu()	20
Código-fonte 21 – Função manipulador_agenda()	21
Código-fonte 22 – Função usuário_inclui_contato ()	22
Código-fonte 23 – Função usuário_inclui_forma_contato ()	22
Código-fonte 24 – Função usuário_altera_nome_contato ()	22
Código-fonte 25 – Função usuário_altera_forma_contato ()	23
Código-fonte 26 – Função usuário_contato_para_texto ()	23
Código-fonte 27 – Função usuário_exclui_contato ()	23
Código-fonte 28 – Conteúdo atual do arquivo manipulacao_agenda.py	29
Código-fonte 29 – Função agenda_para_txt	30
Código-fonte 30 – Função agenda_para_json	30
Código-fonte 31 – Função agenda_para_json	30
Código-fonte 32 – Função agenda_para_json	37

SUMÁRIO

TÓPICOS AVANÇADOS.....	6
1 UM PROJETO FINAL.....	6
1.1 O que podemos fazer com o que já aprendemos?.....	6
1.2 Trabalhando com arquivos externos	7
1.2.1 Lendo e escrevendo no formato JSON	9
1.3 A agenda virtual	10
1.3.1 Criando as funções de visualização	11
1.3.2 Criando as funções de alteração.....	15
1.3.3 Criando a função de exclusão	16
1.3.4 Criando as funções de inclusão	17
1.3.5 Pensando um pouco no usuário.....	19
1.3.6 Exportando e importando dados	30
1.4 E o futuro?.....	38
1.4.1 Boas práticas e a PEP-8	38
1.4.2 Os ambientes virtuais	39
REFERÊNCIAS.....	41

TÓPICOS AVANÇADOS

1 UM PROJETO FINAL

1.1 O que podemos fazer com o que já aprendemos?

Desde a criação do seu primeiro *hello world* em Python até agora, você já passou por diversas estruturas e técnicas que permitem resolver problemas de toda sorte.

Com os desvios condicionais, funções e loops, você poderia criar um jogo de *adventure text*, que oferecesse ao jogador uma experiência similar à dos filmes interativos da Netflix. Afinal de contas, eles nada mais são do que uma sequência de *ifs* que vão levando o jogador a avançar na história (e se você errar, o loop pode levá-lo de volta ao início).

Também já é possível criar pequenos scripts que recebam entradas de dados e construam strings que podem ser utilizadas em outros projetos, sem a necessidade de ficar manipulando os dados de forma manual.

Com certeza, todos esses projetos colocarão à prova os conhecimentos adquiridos até aqui e estimularão você a buscar novos pacotes, exemplos, bibliotecas e mergulhar em documentações de centenas de páginas. Esta etapa pode soar maçante, mas nada alegra mais um *dev* do que superar um desafio tendo aprendido algo novo.

Para o nosso “projeto final”, vamos criar um sistema que gerencia uma agenda virtual, na qual o usuário poderá inserir os contatos e as formas de contato que desejar, inclusive com a funcionalidade de salvar essa agenda em formato TXT ou JSON. Faremos isso utilizando as funções e estruturas com as quais aprendemos a trabalhar e com alguns recursos que serão apresentados neste capítulo.

Vamos lá?

1.2 Trabalhando com arquivos externos

A linguagem Python tem muitas facilidades quando comparada com outras linguagens de programação, e uma delas certamente é a manipulação de arquivos externos.

Isso acontece porque o Python tem uma função nativa chamada *open()* que pode ser utilizada para manipular arquivos de texto, tanto na leitura quanto na escrita. Veja o efeito de executar o script abaixo:

```
texto_para_gravar = "Este texto será gravado!"
#A linha abaixo cria um objeto que representará o nosso
arquivo.
#Ele está sendo aberto no modo de escrita, usando a
codificação UTF-8
arquivo = open("novo_arquivo.txt", "w", encoding="utf-8")
#A instrução write escreve um conteúdo dentro do arquivo
arquivo.write(texto_para_gravar)
#ao final da manipulação, devemos fechar o arquivo
arquivo.close()
```

Código-fonte 1 – Script de gravação de arquivo de texto
Fonte: Elaborado pelo autor (2022)

Viu como é fácil? Com três linhas de código conseguimos criar um arquivo com o conteúdo que estava dentro de uma variável. Os únicos detalhes com os quais temos que nos preocupar são:

- O modo de abertura padrão dos arquivos é o modo de leitura. Para gravar um arquivo devemos usar o modo “w”.
- A codificação utilizada é importante, pois determina como o computador deve interpretar os caracteres que estão no arquivo de texto. A codificação “utf-8” é uma das mais comuns.
- O arquivo deve ser fechado ao final da sua manipulação. Sempre!

O Python é tão “do bem”, que podemos eliminar uma das preocupações da nossa lista: o fechamento do arquivo pode ser feito automaticamente se utilizarmos a função *open* em parceria com a função *with*:

```
texto_para_gravar = "Este texto será gravado!"
#a linha abaixo cria um objeto que representará o nosso
arquivo.
#Ele está sendo aberto no modo de escrita, usando a
codificação UTF-8
with open("novo_arquivo.txt", "w", encoding="utf-8") as
arquivo:
    #A instrução write escreve um conteúdo dentro do
arquivo
    arquivo.write(texto_para_gravar)
```

Código-fonte 2 – Script de gravação de arquivo de texto usando a função with
Fonte: Elaborado pelo autor (2022)

Além de não precisarmos mais nos preocupar em fechar o arquivo de texto, ainda economizamos uma linha!

E a leitura de arquivos de texto é tão fácil quanto a escrita. A diferença é que o modo de manipulação mudará de “w” para “r”, e não usaremos o método .write() e sim o método .read():

```
#a linha abaixo cria um objeto que representará o nosso
arquivo.
#Ele está sendo aberto no modo de leitura, usando a
codificação UTF-8
with open("novo_arquivo.txt", "r", encoding="utf-8") as
arquivo:
    #A instrução .read() lê o conteúdo em formato de
string
    conteudo = arquivo.read()
    #A linha abaixo exibe o conteúdo que foi lido
    print(conteudo)
```

Código-fonte 3 – Script de leitura de arquivo de texto usando a função with
Fonte: Elaborado pelo autor (2022)

O mais legal de manipular arquivos de texto é descobrir que boa parte dos arquivos que usamos no dia a dia são texto puro formatado de acordo com algum padrão. Você pode experimentar armazenar tags HTML dentro de uma string e gravar em um arquivo de texto, mudando a extensão para html.

A nossa experiência será com outro formato: o json!

1.2.1 Lendo e escrevendo no formato JSON

JSON é um padrão de escrita de informações para facilitar a troca de dados de maneira simples e rápida entre sistemas. Um JSON é um texto escrito no padrão chave-valor (lembra do dicionário?) e é tão eficaz que se tornou o padrão para o tráfego de dados em REST API's.

Na sua jornada no mundo dev, você vai lidar com o formato JSON muitas vezes e agora vamos aprender a gerar e ler arquivos nesse formato com a ajuda de um módulo que fará quase tudo para você: o módulo json.

O módulo json é nativo do Python e apresenta uma série de recursos para lidar com o formato json, mas nos interessam duas funcionalidades em especial: uma que converte estruturas Python como dicionário e listas para uma string e formato json e outra que faz o processo oposto.

Observe o script abaixo:

```
#primeiro precisamos importar o módulo json
import json

#vamos criar um dicionário como exemplo
dicionario = {
    "nome": "Python",
    "missão": "Ser incrível!"
}

#Agora vamos utilizar a função dumps do módulo json para
converter nosso dicionário
#O resultado será uma string com estrutura do JSON
texto = json.dumps(dicionario, indent=4,
ensure_ascii=False)

print(f"O dicionário foi convertido para a str texto, e
seu conteúdo é: {texto}")

#Já que o nosso JSON é só um texto, podemos gravá-lo
normalmente
with open("arquivo.json", "w", encoding="utf-8") as
arquivo:
    arquivo.write(texto)
    print("Pronto! O texto no formato json foi salvo
dentro de um arquivo!")
```

Código-fonte 4 – Script de gravação de arquivo json

Fonte: Elaborado pelo autor (2022)

Mais fácil impossível! A função `dumps()` recebeu como argumentos o dicionário, a indentação que queríamos para cada nível do JSON e o parâmetro `ensure_ascii=False` para garantir que não teríamos problemas com acentos e caracteres latinos. Depois disso, bastou pegar a string que foi gerada e salvar em um arquivo de texto.

E o processo para carregar esse arquivo dentro de uma estrutura Python é mais simples ainda! Basta lermos o texto de um arquivo e depois utilizar o método `loads()`:

```
#primeiro precisamos importar o módulo json
import json

#Vamos ler o conteúdo do arquivo json como um texto normal
with open("arquivo.json", "r", encoding="utf-8") as
arquivo:
    conteudo = arquivo.read()

#Agora vamos colocar a nossa string na função loads para
que o Python interprete e gere
#a estrutura adequada:
dicionario = json.loads(conteudo)

print(f"O conteúdo do arquivo foi carregado e convertido:
{dicionario}")
```

Código-fonte 5 – Script de leitura de arquivo json

Fonte: Elaborado pelo autor (2022)

Com o uso da função `open()`, você ganhou uma arma poderosa, e com o módulo `json()` essa arma se tornou mais precisa que um sabre de luz!

Agora já estamos prontos para criar o nosso projeto final!

1.3 A agenda virtual

Para construirmos o nosso projeto, precisamos, primeiramente, definir quais vão ser as suas funcionalidades e os formatos que serão utilizados para manipular os dados. Vamos começar pelas funcionalidades:

- A agenda será um dicionário, cujas chaves serão os nomes das pessoas que serão contatos e os valores serão dicionários.
- Os dicionários que conterão os valores, ou seja, as formas de contato utilizadas para falar com alguém, poderão ter até 3 chaves (telefone, e-mail e endereço), e os valores dessas chaves serão listas contendo os contatos em si.
- O usuário poderá incluir um novo contato informando o nome e ao menos uma forma de contato; incluir uma nova forma de contato para um contato já existente; alterar um contato ou uma forma de contato; excluir um contato; exibir todo o dicionário em forma de texto; exibir apenas um contato em forma de texto; exportar todos os contatos para um arquivo de texto; exportar todos os contatos para um arquivo json; carregar os contatos a partir de um arquivo json.

No PyCharm, vamos criar um projeto com o nome *agendaContatos*. Dentro dele, vamos criar um arquivo chamado *manipulacao_agenda.py*.

1.3.1 Criando as funções de visualização

Ao longo da construção deste projeto, vamos criar algumas variáveis e objetos de forma pontual e depois modificá-los ou excluí-los. Para começar, criaremos uma tupla contendo as formas de contato que o nosso programa suporta e um dicionário com alguns dados falsos para nos ajudar no processo de desenvolvimento das funções:

```
contatos_suportados = ("telefone", "email", "endereco")

agenda = {
    "Pessoa 1":{
        "telefone":["11 1234-5678"],
        "email":["pessoa@email.com",
"email@profissional.com"],
        "endereco":["Rua 123"]
    },
    "Pessoa 2":{
        "telefone":["11 9874-5678"],
        "email":["pessoa2@email.com",
"pessoa2@profissional.com"],
```

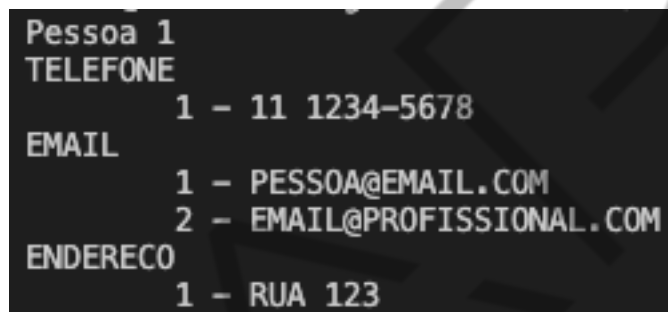
```
        "endereço": ["Rua 345"]
    }
}
```

Código-fonte 6 – Criação da tupla com os contatos suportados e da agenda de exemplo

Fonte: Elaborado pelo autor (2022)

Se verificarmos novamente a lista de tarefas que nosso programa deve realizar, uma delas é a de exibir um único contato e outra é de exibir a agenda completa. Faz sentido imaginar que se escrevermos uma boa função para exibir um único contato, podemos chamá-la quando quisermos exibir a agenda completa.

Vamos supor a seguinte exibição para um único contato:



```
Pessoa 1
TELEFONE
    1 - 11 1234-5678
EMAIL
    1 - PESSOA@EMAIL.COM
    2 - EMAIL@PROFISSIONAL.COM
ENDERECO
    1 - RUA 123
```

Figura 1 – Formato de exibição de contato único

Fonte: Elaborado pelo autor (2022)

Seria possível criarmos uma função que recebesse o *nome* do contato (que é a chave do nosso dicionário geral) e um dicionário com as formas de contato (que é o valor de cada chave do nosso dicionário geral). Podemos propor a seguinte estrutura para essa função:

```
def contato_para_texto(nome_contato:str,
**formas_contato):
    """Recebe um nome de contato com string e um
    dicionário
        com as formas de contato.
        Retorna uma string com os dados recebidos"""
```

Código-fonte 7 – Criação da estrutura da função contato_para_texto

Fonte: Elaborado pelo autor (2022)

Para desenvolvermos essa função, precisamos lembrar da estrutura do dicionário que será recebido com as formas de contato: ele tem as formas de contato como chaves e os valores são listas.

Será possível, então, percorrer esse dicionário com um loop for e percorrer cada valor utilizando um segundo loop.

Para construirmos a nossa saída em formato de texto, também faz sentido criarmos uma variável string que vai recebendo os textos extraídos do dicionário. Assim, temos esta função:

```
def contato_para_texto(nome_contato:str,
**formas_contato):
    """Recebe um nome de contato com string e um
    dicionário
    com as formas de contato.
    Retorna uma string com os dados recebidos"""
    formato_texto = f"{nome_contato}"
    for meio_contato, contato in formas_contato.items():
        formato_texto =
f"{formato_texto}\n{meio_contato.upper()}"
        contador_formas = 1
        for valor in contato:
            formato_texto =
f"{formato_texto}\n\t{contador_formas} - {valor.upper()}"
            contador_formas = contador_formas + 1

    return formato_texto
```

Código-fonte 8 – Função `contato_para_texto`
Fonte: Elaborado pelo autor (2022)

Essa função será chamada no momento adequado, mas para testá-la podemos utilizar a linha `print(contato_para_texto("Pessoa 2", **agenda["Pessoa 2"]))`.

Agora que já temos a função de exibir apenas um único contato em forma de texto, podemos pensar na função que exibe a nossa agenda completa.

A função que exibe a agenda completa pode receber um dicionário, percorrê-lo e chamar a função `contato_para_texto` para cada um dos contatos encontrados, armazenando em uma variável string o conteúdo que está sendo recebido.

A estrutura dessa função é:

```
def agenda_para_texto(**agenda_completa):
    """Recebe um dicionário de dicionários com a agenda
    que será exibida e
    retorna uma string com este dicionário formatado"""
```

Código-fonte 9 – Criação da estrutura da função `agenda_para_texto`
Fonte: Elaborado pelo autor (2022)

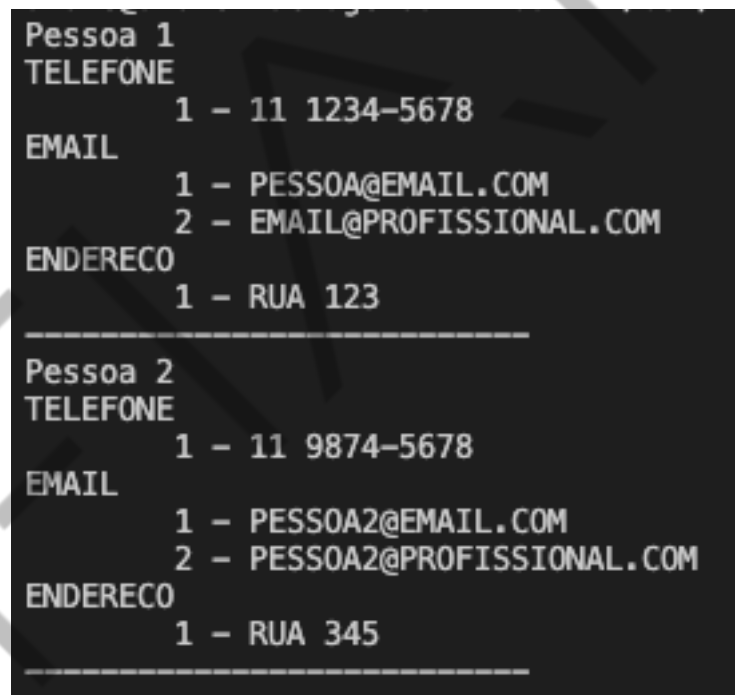
Implementando a variável `formato_texto` e o loop que percorre a agenda recebida, teremos o seguinte código para a função:

```
def agenda_para_texto(**agenda_completa):
    """Recebe um dicionário de dicionários com a agenda
    que será exibida e
    retorna uma string com este dicionário formatado"""
    formato_texto = ""
    for nome_contato, formas_contato in
agenda_completa.items():
        formato_texto =
f"{formato_texto}{contato_para_texto(nome_contato,
**formas_contato)}\n"
        formato_texto = f"{formato_texto}-----
-----\n"
    return formato_texto
```

Código-fonte 10 – Função agenda_para_texto

Fonte: Elaborado pelo autor (2022)

Executando essa função com a linha `print(agenda_para_texto(**agenda))`, teremos a seguinte saída como resultado:



```
Pessoa 1
TELEFONE
    1 - 11 1234-5678
EMAIL
    1 - PESSOA@EMAIL.COM
    2 - EMAIL@PROFISSIONAL.COM
ENDERECO
    1 - RUA 123
-----
Pessoa 2
TELEFONE
    1 - 11 9874-5678
EMAIL
    1 - PESSOA2@EMAIL.COM
    2 - PESSOA2@PROFISSIONAL.COM
ENDERECO
    1 - RUA 345
-----
```

Figura 2 – Resultado a função agenda_para_texto

Fonte: Elaborado pelo autor (2022)

Agora que nossas funções de exibição estão criadas, vamos partir para as funções que alteram a nossa agenda!

1.3.2 Criando as funções de alteração

Temos duas funções distintas para realizar alterações na nossa agenda: uma para realizar alterações no nome do contato que foi cadastrado, outra para alterar algum dos números de telefone, e-mail ou endereço.

Em ambos os casos estamos falando sobre alterar o dado original, aquele que está em nosso dicionário *agenda* e, por essa razão, não poderemos trabalhar com ****kwargs**. Será preciso receber o dicionário original para alterarmos o nome do contato, ou a lista contendo as formas de contato que se deseja alterar.

Para não nos confundirmos, vamos criar primeiro a função que altera o nome de um contato. Essa função receberá a agenda original, o nome original e o novo nome. Caso o nome original esteja na agenda original (verificação realizada com o método `keys()`), faremos a cópia das formas de contato para um outro dicionário, removeremos o contato com base no nome original e incluiremos o novo contato como chave e o dicionário copiado como valor:

```
def altera_nome_contato(agenda_original:dict,
nome_original:str, nome_atualizado:str):
    """Recebe a agenda original em forma de dicionário, o
nome_original e o nome_atualizado em forma
de string.
Busca o nome original no dicionário e retorna False
se não encontrar.
Retorna True se encontrar o nome original no
dicionário e fizer a exclusão do contato antigo e inclusão do
novo"""
    if nome_original in agenda_original.keys():
        copia_contatos = agenda_original[nome_original].copy()
        agenda_original.pop(nome_original)
        agenda_original[nome_atualizado] = copia_contatos
        return True
    return False
```

Código-fonte 11 – Função `altera_nome_contato`

Fonte: Elaborado pelo autor (2022)

Essa função é apenas uma das muitas abordagens para resolver o mesmo problema, e você também pode encontrar a sua.

Para a função que altera um dos valores guardados nas listas que contém os telefones, e-mails ou endereços, receberemos a lista original, o valor antigo e o novo

valor. Caso o valor antigo esteja na lista, armazenaremos seu índice, removeremos o valor antigo e inseriremos o novo valor na mesma posição:

```
def altera_forma_contato(lista_contatos:list,
valor_antigo:str, novo_valor:str):
    """Recebe uma list lista_contatos, o valor antigo que
    será substituído e o novo valor
    Caso o valor antigo não esteja na lista, retornará
    False.
    Caso o valor antigo esteja na lista, será removido, o
    novo valor será incluído e retornará True
    """
    if valor_antigo in lista_contatos:
        posicao_valor_antigo = lista_contatos.index(valor_antigo)
        lista_contatos.pop(posicao_valor_antigo)
        lista_contatos.insert(posicao_valor_antigo,
novo_valor)
    return True
    return False
```

Código-fonte 12 – Função altera_forma_contato

Fonte: Elaborado pelo autor (2022)

Podemos testar essas funções rodando as linhas abaixo:

```
altera_nome_contato(agenda, "Pessoa 2", "Super Pessoa")
print(agenda_para_texto(**agenda))
altera_forma_contato(agenda["Pessoa 1"]["telefone"], "11
1234-5678", "123")
print(agenda_para_texto(**agenda))
```

Código-fonte 13 – Teste das funções de alteração

Fonte: Elaborado pelo autor (2022)

Você deve ter reparado que nossas funções de alteração retornam valores booleanos. Eles poderão ser usados futuramente para verificar se a alteração foi bem-sucedida ou não.

Ainda precisamos criar uma forma de o usuário incluir novos contatos na agenda, mas antes disso, vamos criar a funcionalidade de excluir um contato com base em seu nome.

1.3.3 Criando a função de exclusão

A função de exclusão de contatos é extremamente mais simples que as anteriores. Ela não precisa iterar nenhuma estrutura e nem fazer grandes checagens, bastando receber o dicionário com a agenda original e o nome que será excluído.

Caso o nome esteja nas chaves do dicionário, basta excluí-lo e retornar True. Caso não esteja, basta retornar False:

```
def exclui_contato(agenda:dict, nome_contato:str):  
    """Recebe uma agenda completa como dicionário e o nome  
    do contato como string.  
    Caso o nome do contato não esteja nas chaves do  
    dicionário, retornará False  
    Caso o nome do contato esteja nas chaves, o registro  
    correspondente será removido e retornará True"""  
    if nome_contato in agenda.keys():  
        agenda.pop(nome_contato)  
        return True  
    return False
```

Código-fonte 14 – Função excluir
Fonte: Elaborado pelo autor (2022)

Agora é hora de implementar uma das funções mais complexas da nossa agenda: a inclusão!

1.3.4 Criando as funções de inclusão

A função para incluir um contato, de acordo com as exigências que reunimos há pouco, deve ser capaz de incluir um novo contato (chave) na nossa agenda (dicionário), assim como incluir obrigatoriamente alguma forma de contato (dicionário cujas chaves são formas de contato e os valores são listas).

Caso pensemos muito em como o *usuário* vai interagir com nossos sistemas, podemos acabar criando funções muito acopladas com a situação atual e com baixo reuso.

Por essa razão, continuaremos deixando o usuário de lado, por enquanto, para criar essa função de forma que possa ser chamada por outra. Portanto, criaremos como argumentos a agenda original, o nome do novo contato e as formas de contato como ****kwargs**:

```
def inclui_contato(agenda:dict, nome_contato:str,  
**formas_contato):  
    """Recebe uma agenda completa como dicionário, o nome  
    do novo contato como string e as formas de contato  
    em um dicionário como **kwargs.  
    Não é feita nenhuma verificação, portanto se já existir  
    um contato com o mesmo nome, será sobrescrito"""  
    #print(formas_contato)
```

```
agenda[nome_contato] = formas_contato
```

Código-fonte 15 – Função inclui_contato

Fonte: Elaborado pelo autor (2022)

Para testar essa função, podemos utilizar as linhas abaixo:

```
inclui_contato(agenda, "Juquinha", telefone=["123456"],  
email=["a@b.com"])  
print(agenda_para_texto(**agenda))
```

Código-fonte 16 – Teste da função inclui_contato

Fonte: Elaborado pelo autor (2022)

Agora precisamos criar a função que inclui uma nova forma de contato, e será ligeiramente mais complexa, pois existem três cenários possíveis:

A forma de contato talvez ainda não exista. Nesse caso, precisaremos incluí-la e incluir o novo valor em uma lista.

A forma de contato talvez já exista. Nesse caso, precisaremos incluir o novo valor na lista já existente.

A forma de contato pode ser inválida (algo que não seja endereço, telefone ou e-mail), portanto não pode ser incluída.

Como talvez lidemos com um dicionário de contatos já existente, o mais seguro é que essa função receba um dicionário de formas de contato, uma string com a forma que manipularemos/incluiremos, e o valor da nova forma:

```
def inclui_forma_contato(formas_contato:dict,  
forma_incluida:str, valor_incluido:str):  
    """Recebe um dicionário com as formas de contato, a  
    forma de contato que será incluída ou  
    alterada e o valor que será incluído.  
    Caso a forma de contato já possua valores, o novo  
    valor será adicionado na lista e retornará True.  
    Caso a forma de contato ainda não exista e estiver  
    presente na tupla de formas de contatos suportados  
    será incluída e o novo valor será incluído em uma  
    lista, retornando True.  
    Caso a forma de contato ainda não exista e não  
    estiver presente na tupla de formas de contato suportados,  
    retornará False"""
```

Código-fonte 17 – início da função inclui_forma_contato

Fonte: Elaborado pelo autor (2022)

Uma vez que já temos a estrutura básica dessa função, precisamos seguir o fluxo das três formas apontadas anteriormente:

```
def inclui_forma_contato(formas_contato:dict,
forma_incluida:str, valor_incluido:str):
    """Recebe um dicionário com as formas de contato, a
    forma de contato que será incluída ou
    alterada e o valor que será incluído.
    Caso a forma de contato já possua valores, o novo
    valor será adicionado na lista e retornará True.
    Caso a forma de contato ainda não exista e estiver
    presente na tupla de formas de contatos suportados
    será incluída e o novo valor será incluído em uma
    lista, retornando True.
    Caso a forma de contato ainda não exista e não
    estiver presente na tupla de formas de contato suportados,
    retornará False"""
    if forma_incluida in formas_contato.keys():
formas_contato[forma_incluida].append(valor_incluido)
        return True
    elif forma_incluida in contatos_suportados:
formas_contato[forma_incluida]
[valor_incluido]
        return True
    return False
```

Código-fonte 18 – início da função inclui_forma_contato

Fonte: Elaborado pelo autor (2022)

Para testar essa função, podemos rodar as linhas abaixo:

```
#teste anterior
inclui_contato(agenda, "Juquinha", telefone=["123456"],
email=["a@b.com"])
print(agenda_para_texto(**agenda))

#teste atual
Inclui_forma_contato(agenda["Juquinha"], "endereco",
"Rua dos Jucas")
Print(agenda_para_texto(**agenda))
```

Código-fonte 19 – Teste da função inclui_forma_contato

Fonte: Elaborado pelo autor (2022)

1.3.5 Pensando um pouco no usuário

Agora que já criamos todas as nossas funções principais sem nos preocuparmos com o usuário, é hora de pensarmos na interação que ele terá com o nosso sistema.

Existem diversas abordagens possíveis, cada uma com os seus benefícios e desvantagens. Para a primeira versão do nosso programa, criaremos as seguintes funções de interação com o usuário:

Função	Efeito
exibe_menu()	Exibe as opções do menu.
manipulador_agenda()	Chama a função de exibição do menu, captura a opção do usuário e chama as funções correspondentes.
usuario_inclui_contato(agenda)	Solicita as informações do contato e chama a função inclui_contato() construída anteriormente.
usuario_inclui_forma_contato(agenda)	Solicita as informações de forma de contato e chama a função inclui_forma_contato() construída anteriormente.
usuario_altera_nome_contato(agenda)	Solicita o nome do contato e chama a função altera_nome_contato() construída anteriormente.
usuario_altera_forma_contato(agenda)	Solicita informações sobre forma de contato e chama a função altera_forma_contato() construída anteriormente.
usuario_contato_para_texto(agenda)	Solicita o nome do contato e chama a função contato_para_texto() construída anteriormente.
usuario_exclui_contato(agenda)	Solicita o nome do contato e chama a função exclui_contato() construída anteriormente.

Quadro 1 – Funções de interação com o usuário
Fonte: Elaborado pelo autor (2022)

Para que você consiga visualizar bem o código de cada uma das funções, vamos apresentá-las separadamente e, depois, todo o código atual do arquivo `manipulacao_agenda.py`.

```
def exibe_menu():
    print("\n\n")
    print("1 - Incluir contato na agenda")
    print("2 - Incluir uma forma de contato")
    print("3 - Alterar o nome de um contato")
    print("4 - Alterar uma forma de contato")
    print("5 - Exibir um contato")
    print("6 - Exibir toda a agenda")
    print("7 - Excluir um contato")
    print("8 - Sair")
    print("\n")
```

Código-fonte 20 – Função `exibe_menu()`
Fonte: Elaborado pelo autor (2022)

```
def manipulador_agenda():
    agenda = {}
    op = 1
    while op != 8:
        exibe_menu()
        op = int(input("Informe a opção desejada: "))
        if op == 1:
            usuario_inclui_contato(agenda)
        elif op == 2:
            usuario_inclui_forma_contato(agenda)
        elif op == 3:
            usuario_altera_nome_contato(agenda)
        elif op == 4:
            usuario_altera_forma_contato(agenda)
        elif op == 5:
            usuario_contato_para_texto(agenda)
        elif op == 6:
            print(agenda_para_texto(**agenda))
        elif op == 7:
            usuario_exclui_contato(agenda)
        elif op == 8:
            print("Saindo do sistema")
        else:
            print("Opção inválida! Informe uma opção existente.")
```

Código-fonte 21 – Função manipulador_agenda()

Fonte: Elaborado pelo autor (2022)

```
def usuario_inclui_contato(agenda:dict):
    nome = input("Informe o nome do novo contato que será
inserido na agenda: ")
    dicionario_formas = {}
    for forma in contatos_suportados:
        resposta = input(f"Deseja inserir um {forma} para
{name.upper()}? \nSIM ou NÃO -> ")
        lista_contatos = []
        while "S" in resposta.upper():
            lista_contatos.append(input(f"Informe um
{forma}: "))
            resposta = input(f"Deseja inserir outro
{forma} para {nome.upper()}?\nSIM ou NÃO -> ")
        if len(lista_contatos) > 0:
            dicionario_formas[forma] =
lista_contatos.copy()
            lista_contatos.clear()
        if len(dicionario_formas.keys()) > 0:
            inclui_contato(agenda, nome,
**dicionario_formas)
            print("Inclusão bem sucedida!")
        else:
```

```
print("É necessário incluir pelo menos uma forma de contato!\nA agenda não foi alterada.")
```

Código-fonte 22 – Função usuário_inclui_contato ()

Fonte: Elaborado pelo autor (2022)

```
def usuario_inclui_forma_contato(agenda:dict):
    #inclui_forma_contato(formas_contato:dict,
    forma_incluida:str, valor_incluido:str):
    nome = input("Informe o nome do contato para o qual deseja incluir formas de contato ")
    if nome in agenda.keys():
        print(f"As formas de contato suportadas pelo sistema são: {contatos_suportados}")
        forma_incluida = input("Qual forma de contato deseja incluir? ")
        if forma_incluida in contatos_suportados:
            valor_incluido = input(f"Informe o {forma_incluida} que deseja incluir: ")
            if inclui_forma_contato(agenda[nome], forma_incluida, valor_incluido):
                print("Operação bem sucedida! A nova forma de contato foi incluída! ")
            else:
                print("Ocorreu um erro durante a inserção. A agenda não foi alterada.")
        else:
            print("A forma de contato indicada não é suportada pelo sistema. A agenda não foi alterada.")
    else:
        print("O contato informado não existe na agenda. Não foram feitas alterações. ")
```

Código-fonte 23 – Função usuário_inclui_forma_contato ()

Fonte: Elaborado pelo autor (2022)

```
def usuario_altera_nome_contato(agenda:dict):
    #altera_nome_contato(agenda_original:dict,
    nome_original:str, nome_atualizado:str):
    nome_original = input("Informe o nome do contato que deseja alterar: ")
    nome_atualizado = input("Informe o nome do novo contato: ")
    if altera_nome_contato(agenda, nome_original, nome_atualizado):
        print(f"O contato foi atualizado e agora se chama {nome_atualizado}")
    else:
        print(f"O contato original não foi localizado. A agenda não foi alterada.")
```

Código-fonte 24 – Função usuário_altera_nome_contato ()

Fonte: Elaborado pelo autor (2022)

Tópicos Avançados

```
def usuario_altera_forma_contato(agenda:dict):
    nome = input("Informe o nome do contato que deseja
alterar: ")
    if nome in agenda.keys():
        print(f"As formas de contato suportadas pelo
sistema são: {contatos_suportados}")
        forma_incluida = input("Qual forma de contato
deseja incluir? ")
        if forma_incluida in contatos_suportados:
            print(contato_para_texto(nome,
**agenda[nome]))
            valor_antigo = input(f"Informe o
{forma_incluida} que deseja alterar ")
            nova_valor = input(f"Informe o novo
{forma_incluida} ")
            if
altera_forma_contato(agenda[nome][forma_incluida],
valor_antigo, nova_valor):
                print("Contato alterado com sucesso!")
            else:
                print("Ocorreu um erro durante a
alteração do contato. A agenda não foi alterada.")
        else:
            print(f"{forma_incluida} não é uma forma de
contato suportada pelo sistema. A agenda não foi alterada.")
    else:
        print(f"O contato {nome} não está na agenda. A
agenda não foi alterada.")
```

Código-fonte 25 – Função usuário_altera_forma_contato ()

Fonte: Elaborado pelo autor (2022)

```
def usuario_contato_para_texto(agenda:dict):
    nome = input("Informe o nome do contato que deseja
exibir: ")
    if nome in agenda.keys():
        print(contato_para_texto(nome, **agenda[nome]))
    else:
        print("O contato informado não está na agenda.")
```

Código-fonte 26 – Função usuário_contato_para_texto ()

Fonte: Elaborado pelo autor (2022)

```
def usuario_exclui_contato(agenda:dict):
    nome = input("Informe o nome do contato que deseja
excluir: ")
    if exclui_contato(agenda, nome):
        print("Usuário excluído com sucesso!")
    else:
        print("Nome do usuário não localizado na agenda.
Não foram feitas alterações.")
```

Código-fonte 27 – Função usuário_exclui_contato ()

Fonte: Elaborado pelo autor (2022)

Tópicos Avançados

```
contatos_suportados = ("telefone", "email", "endereço")

agenda = {
    "Pessoa 1":{
        "telefone":["11 1234-5678"],
        "email":["pessoa@email.com",
"email@profissional.com"],
        "endereço":["Rua 123"]
    },
    "Pessoa 2":{
        "telefone":["11 9874-5678"],
        "email":["pessoa2@email.com",
"pessoa2@profissional.com"],
        "endereço":["Rua 345"]
    }
}

def          contato_para_texto(nome_contato:str,
**formas_contato):
    """Recebe um nome de contato com string e um
dicionário
    com as formas de contato.
    Retorna uma string com os dados recebidos"""
    formato_texto = f"{nome_contato}"
    for meio_contato, contato in formas_contato.items():
        formato_texto
    =
f"{formato_texto}\n{meio_contato.upper()}"
        contador_formas = 1
        for valor in contato:
            formato_texto
            =
f"{formato_texto}\n\t{contador_formas} - {valor.upper()}"
            contador_formas = contador_formas + 1

    return formato_texto

def agenda_para_texto(**agenda_completa):
    """Recebe um dicionário de dicionários com a agenda
que será exibida e
    retorna uma string com este dicionário formatado"""
    formato_texto = ""
    for          nome_contato,          formas_contato          in
agenda_completa.items():
        formato_texto
        =
f"{formato_texto}{contato_para_texto(nome_contato,
**formas_contato)}\n"
        formato_texto = f"{formato_texto}-----
-----\n"
    return formato_texto
```


Tópicos Avançados

```
def altera_nome_contato(agenda_original:dict,
nome_original:str, nome_atualizado:str):
    """Recebe a agenda original em forma de dicionário, o
nome_original e o nome_atualizado em forma
de string.
Busca o nome original no dicionário e retorna False
se não encontrar.
Retorna True se encontrar o nome original no
dicionário e fizer a exclusão do contato antigo e inclusão do
novo"""
    if nome_original in agenda_original.keys():
        copia_contatos = agenda_original.copy()
        agenda_original.pop(nome_original)
        agenda_original[nome_atualizado] = copia_contatos
        return True
    return False

def altera_forma_contato(lista_contatos:list,
valor_antigo:str, novo_valor:str):
    """Recebe uma list lista_contatos, o valor antigo que
será substituído e o novo valor
Caso o valor antigo não esteja na lista, retornará
False.
Caso o valor antigo esteja na lista, será removido, o
novo valor será incluído e retornará True
"""
    if valor_antigo in lista_contatos:
        posicao_valor_antigo = lista_contatos.index(valor_antigo)
        lista_contatos.pop(posicao_valor_antigo)
        lista_contatos.insert(posicao_valor_antigo,
novo_valor)
        return True
    return False

def exclui_contato(agenda:dict, nome_contato:str):
    """Recebe uma agenda completa como dicionário e o nome
do contato como string.
Caso o nome do contato não esteja nas chaves do
dicionário, retornará False
Caso o nome do contato esteja nas chaves, o registro
correspondente será removido e retornará True"""
    if nome_contato in agenda.keys():
        agenda.pop(nome_contato)
        return True
    return False
```

Tópicos Avançados

```
def inclui_contato(agenda:dict, nome_contato:str,
**formas_contato):
    """Recebe uma agenda completa como dicionário, o nome
do novo contato como string e as formas de contato
em um dicionário como **kwargs.
Não é feita nenhuma verificação, portanto se já existir
um contato com o mesmo nome, será sobrescrito"""
    #print(formas_contato)
    agenda[nome_contato] = formas_contato

def inclui_forma_contato(formas_contato:dict,
forma_incluida:str, valor_incluido:str):
    """Recebe um dicionário com as formas de contato, a
forma de contato que será incluída ou
alterada e o valor que será incluído.
Caso a forma de contato já possua valores, o novo
valor será adicionado na lista e retornará True.
Caso a forma de contato ainda não exista e estiver
presente na tupla de formas de contatos suportados
será incluída e o novo valor será incluído em uma
lista, retornando True.
Caso a forma de contato ainda não exista e não
estiver presente na tupla de formas de contato suportados,
retornará False"""
    if forma_incluida in formas_contato.keys():
formas_contato[forma_incluida].append(valor_incluido)
        return True
    elif forma_incluida in contatos_suportados:
formas_contato[forma_incluida] =
[valor_incluido]
        return True
    return False

def usuario_inclui_contato(agenda:dict):
    nome = input("Informe o nome do novo contato que será
inserido na agenda: ")
    dicionario_formas = {}
    for forma in contatos_suportados:
        resposta = input(f"Deseja inserir um {forma} para
{nome.upper()}? \nSIM ou NÃO -> ")
        lista_contatos = []
        while "S" in resposta.upper():
            lista_contatos.append(input(f"Informe um
{forma}: "))
            resposta = input(f"Deseja inserir outro
{forma} para {nome.upper()}? \nSIM ou NÃO -> ")
        if len(lista_contatos) > 0:
            dicionario_formas[forma] =
lista_contatos.copy()
        lista_contatos.clear()
```

Tópicos Avançados

```
        if len(dicionario_formas.keys()) > 0:
            inclui_contato(agenda, nome,
**dicionario_formas)
            print("Inclusão bem sucedida!")
        else:
            print("É necessário incluir pelo menos uma forma
de contato!\nA agenda não foi alterada.")

    def usuario_inclui_forma_contato(agenda:dict):
        #inclui_forma_contato(formas_contato:dict,
forma_incluida:str, valor_incluido:str):
        nome = input("Informe o nome do contato para o qual
deseja incluir formas de contato ")
        if nome in agenda.keys():
            print(f"As formas de contato suportadas pelo
sistema são: {contatos_suportados}")
            forma_incluida = input("Qual forma de contato
deseja incluir? ")
            if forma_incluida in contatos_suportados:
                valor_incluido = input(f"Informe o
{forma_incluida} que deseja incluir: ")
                if inclui_forma_contato(agenda[nome],
forma_incluida, valor_incluido):
                    print("Operação bem sucedida! A nova
forma de contato foi incluída! ")
                else:
                    print("Ocorreu um erro durante a
inserção. A agenda não foi alterada.")
            else:
                print("A forma de contato indicada não é
suportada pelo sistema. A agenda não foi alterada.")
        else:
            print("O contato informado não existe na agenda.
Não foram feitas alterações. ")

    def usuario_exclui_contato(agenda:dict):
        nome = input("Informe o nome do contato que deseja
excluir: ")
        if exclui_contato(agenda, nome):
            print("Usuário excluído com sucesso!")
        else:
            print("Nome do usuário não localizado na agenda.
Não foram feitas alterações.")

    def usuario_altera_forma_contato(agenda:dict):
        nome = input("Informe o nome do contato que deseja
alterar: ")
        if nome in agenda.keys():
            print(f"As formas de contato suportadas pelo
sistema são: {contatos_suportados}")
```

Tópicos Avançados

```
        forma_incluida = input("Qual forma de contato  
deseja incluir? ")  
        if forma_incluida in contatos_suportados:  
            print(contato_para_texto(nome,  
**agenda[nome]))  
            valor_antigo = input(f"Informe o  
{forma_incluida} que deseja alterar ")  
            nova_valor = input(f"Informe o novo  
{forma_incluida} ")  
            if  
altera_forma_contato(agenda[nome][forma_incluida],  
valor_antigo, nova_valor):  
                print("Contato alterado com sucesso!")  
            else:  
                print("Ocorreu um erro durante a  
alteração do contato. A agenda não foi alterada.")  
        else:  
            print(f"{forma_incluida} não é uma forma de  
contato suportada pelo sistema. A agenda não foi alterada.")  
        else:  
            print(f"O contato {nome} não está na agenda. A  
agenda não foi alterada.")  
  
    def usuario_altera_nome_contato(agenda:dict):  
        #altera_nome_contato(agenda_original:dict,  
nome_original:str, nome_atualizado:str):  
        nome_original = input("Informe o nome do contato que  
deseja alterar: ")  
        nome_atualizado = input("Informe o nome do novo  
contato: ")  
        if altera_nome_contato(agenda, nome_original,  
nome_atualizado):  
            print(f"O contato foi atualizado e agora se chama  
{nome_atualizado}")  
        else:  
            print(f"O contato original não foi localizado. A  
agenda não foi alterada.")  
  
    def usuario_contato_para_texto(agenda:dict):  
        nome = input("Informe o nome do contato que deseja  
exibir: ")  
        if nome in agenda.keys():  
            print(contato_para_texto(nome, **agenda[nome]))  
        else:  
            print("O contato informado não está na agenda.")  
  
    def exibe_menu():  
        print("\n\n")  
        print("1 - Incluir contato na agenda")  
        print("2 - Incluir uma forma de contato")
```

```
print("3 - Alterar o nome de um contato")
print("4 - Alterar uma forma de contato")
print("5 - Exibir um contato")
print("6 - Exibir toda a agenda")
print("7 - Excluir um contato")
print("8 - Sair")
print("\n")

def manipulador_agenda():
    agenda = {}
    op = 1
    while op != 8:
        exibe_menu()
        op = int(input("Informe a opção desejada: "))
        if op == 1:
            usuario_inclui_contato(agenda)
        elif op == 2:
            usuario_inclui_forma_contato(agenda)
        elif op == 3:
            usuario_altera_nome_contato(agenda)
        elif op == 4:
            usuario_altera_forma_contato(agenda)
        elif op == 5:
            usuario_contato_para_texto(agenda)
        elif op == 6:
            print(agenda_para_texto(**agenda))
        elif op == 7:
            usuario_exclui_contato(agenda)
        elif op == 8:
            print("Saindo do sistema")
        else:
            print("Opção inválida! Informe uma opção existente.")

manipulador_agenda()
```

Código-fonte 28 – Conteúdo atual do arquivo manipulacao_agenda.py
Fonte: Elaborado pelo autor (2022)

Para que o nosso projeto cumpra com todas as etapas que havíamos planejado, está faltando ser capaz de persistir dados em formato de texto, em formato json e carregar os dados nesse formato.

1.3.6 Exportando e importando dados

Nosso projeto contará com 3 funções referentes à manipulação de arquivos: uma função para exportar a agenda para o formato txt, outra para exportar a agenda para um json e uma terceira para fazer a importação no formato json.

Para a função de exportação para txt, basta chamarmos a função que converte a agenda em texto e depois usar a função open para salvar o arquivo:

```
def agenda_para_txt(nome_arquivo:str, agenda):
    if "txt" not in nome_arquivo:
        nome_arquivo = f"{nome_arquivo}.txt"
    with open(nome_arquivo, "w", encoding="utf-8") as
arquivo:
        arquivo.write(agenda_para_texto(**agenda))
        print("Agenda exportada com sucesso!")
```

Código-fonte 29 – Função agenda_para_txt

Fonte: Elaborado pelo autor (2022)

Para a função de exportação para json, teremos basicamente o mesmo código anterior, com a diferença que a fonte dos dados que serão gravados no nosso arquivo não será a função que gera a agenda como texto, e sim a o método .dumps() do módulo json:

```
def agenda_para_json(nome_arquivo:str, agenda):
    if ".json" not in nome_arquivo:
        nome_arquivo = f"{nome_arquivo}.json"
    with open(nome_arquivo, "w", encoding="utf-8") as
arquivo:
        arquivo.write(json.dumps(agenda, indent=4,
ensure_ascii=False))
        print("Agenda exportada com sucesso!")
```

Código-fonte 30 – Função agenda_para_json

Fonte: Elaborado pelo autor (2022)

Para completar a nossa trinca de manipulação de arquivos, vamos criar uma função que retorne um dicionário com base no json lido:

```
def json_para_agenda(nome_arquivo:str):
    with open(nome_arquivo, "r", encoding="utf-8") as
arquivo:
        conteudo = arquivo.read()
        print("Agenda carregada com sucesso!")
        return json.loads(conteudo)
```

Código-fonte 31 – Função agenda_para_json

Fonte: Elaborado pelo autor (2022)

Tópicos Avançados

Para podermos considerar este projeto concluído para os nossos objetivos atuais, falta apenas incluir as três funções que acabamos de criar no método `manipulador_agenda()` e fazer os devidos ajustes no menu.

O código completo do projeto é este:

```
import json

contatos_suportados = ("telefone", "email", "endereço")

agenda = {
    "Pessoa 1":{
        "telefone":["11 1234-5678"],
        "email":["pessoa@email.com",
"email@profissional.com"],
        "endereço":["Rua 123"]
    },
    "Pessoa 2":{
        "telefone":["11 9874-5678"],
        "email":["pessoa2@email.com",
"pessoa2@profissional.com"],
        "endereço":["Rua 345"]
    }
}

def agenda_para_txt(nome_arquivo:str, agenda):
    if "txt" not in nome_arquivo:
        nome_arquivo = f"{nome_arquivo}.txt"
    with open(nome_arquivo, "w", encoding="utf-8") as
arquivo:
        arquivo.write(agenda_para_texto(**agenda))
        print("Agenda exportada com sucesso!")

def json_para_agenda(nome_arquivo:str):
    with open(nome_arquivo, "r", encoding="utf-8") as
arquivo:
        conteudo = arquivo.read()
        print("Agenda carregada com sucesso!")
        return json.loads(conteudo)

def agenda_para_json(nome_arquivo:str, agenda):
    if ".json" not in nome_arquivo:
        nome_arquivo = f"{nome_arquivo}.json"
```

Tópicos Avançados

```
with open(nome_arquivo, "w", encoding="utf-8") as
arquivo:
    arquivo.write(json.dumps(agenda, indent=4,
ensure_ascii=False))
    print("Agenda exportada com sucesso!")

def contato_para_texto(nome_contato:str,
**formas_contato):
    """Recebe um nome de contato com string e um
dicionário
    com as formas de contato.
    Retorna uma string com os dados recebidos"""
    formato_texto = f"{nome_contato}"
    for meio_contato, contato in formas_contato.items():
        formato_texto =
f"{formato_texto}\n{meio_contato.upper()}"
        contador_formas = 1
        for valor in contato:
            formato_texto =
f"{formato_texto}\n\t{contador_formas} - {valor.upper()}"
            contador_formas = contador_formas + 1

    return formato_texto

def agenda_para_texto(**agenda_completa):
    """Recebe um dicionário de dicionários com a agenda
que será exibida e
    retorna uma string com este dicionário formatado"""
    formato_texto = ""
    for nome_contato, formas_contato in
agenda_completa.items():
        formato_texto =
f"{formato_texto}{contato_para_texto(nome_contato,
**formas_contato)}\n"
        formato_texto = f"{formato_texto}-----
-----\n"
    return formato_texto

def altera_nome_contato(agenda_original:dict,
nome_original:str, nome_atualizado:str):
    """Recebe a agenda original em forma de dicionário, o
nome_original e o nome_atualizado em forma
    de string.
    Busca o nome original no dicionário e retorna False
se não encontrar.
    Retorna True se encontrar o nome original no
dicionário e fizer a exclusão do contato antigo e inclusão do
novo"""
    if nome_original in agenda_original.keys():
```


Tópicos Avançados

```
        copia_contatos = agenda_original[nome_original].copy()
        agenda_original.pop(nome_original)
        agenda_original[nome_atualizado] = copia_contatos
        return True
    return False

def altera_forma_contato(lista_contatos:list,
valor_antigo:str, novo_valor:str):
    """Recebe uma list lista_contatos, o valor antigo que
    será substituído e o novo valor
    Caso o valor antigo não esteja na lista, retornará
    False.
    Caso o valor antigo esteja na lista, será removido, o
    novo valor será incluído e retornará True
    """
    if valor_antigo in lista_contatos:
        posicao_valor_antigo = lista_contatos.index(valor_antigo)
        lista_contatos.pop(posicao_valor_antigo)
        lista_contatos.insert(posicao_valor_antigo,
novo_valor)
        return True
    return False

def exclui_contato(agenda:dict, nome_contato:str):
    """Recebe uma agenda completa como dicionário e o nome
    do contato como string.
    Caso o nome do contato não esteja nas chaves do
    dicionário, retornará False
    Caso o nome do contato esteja nas chaves, o registro
    correspondente será removido e retornará True"""
    if nome_contato in agenda.keys():
        agenda.pop(nome_contato)
        return True
    return False

def inclui_contato(agenda:dict, nome_contato:str,
**formas_contato):
    """Recebe uma agenda completa como dicionário, o nome
    do novo contato como string e as formas de contato
    em um dicionário como **kwargs.
    Não é feita nenhuma verificação, portanto se já existir
    um contato com o mesmo nome, será sobrescrito"""
    #print(formas_contato)
    agenda[nome_contato] = formas_contato

def inclui_forma_contato(formas_contato:dict,
forma_incluida:str, valor_incluido:str):
```

Tópicos Avançados

```
"""Recebe um dicionário com as formas de contato, a
forma de contato que será incluída ou
alterada e o valor que será incluído.
Caso a forma de contato já possua valores, o novo
valor será adicionado na lista e retornará True.
Caso a forma de contato ainda não exista e estiver
presente na tupla de formas de contatos suportados
será incluída e o novo valor será incluído em uma
lista, retornando True.
Caso a forma de contato ainda não exista e não
estiver presente na tupla de formas de contato suportados,
retornará False"""
if forma_incluida in formas_contato.keys():
formas_contato[forma_incluida].append(valor_incluido)
return True
elif forma_incluida in contatos_suportados:
formas_contato[forma_incluida] =
[valor_incluido]
return True
return False

def usuario_inclui_contato(agenda:dict):
nome = input("Informe o nome do novo contato que será
inserido na agenda: ")
dicionario_formas = {}
for forma in contatos_suportados:
resposta = input(f"Deseja inserir um {forma} para
{nome.upper()}? \nSIM ou NÃO -> ")
lista_contatos = []
while "S" in resposta.upper():
lista_contatos.append(input(f"Informe um
{forma}: "))
resposta = input(f"Deseja inserir outro
{forma} para {nome.upper()}? \nSIM ou NÃO -> ")
if len(lista_contatos) > 0:
dicionario_formas[forma] =
lista_contatos.copy()
lista_contatos.clear()
if len(dicionario_formas.keys()) > 0:
inclui_contato(agenda, nome,
**dicionario_formas)
print("Inclusão bem sucedida!")
else:
print("É necessário incluir pelo menos uma forma
de contato!\nA agenda não foi alterada.")

def usuario_inclui_forma_contato(agenda:dict):
#inclui_forma_contato(formas_contato:dict,
forma_incluida:str, valor_incluido:str):
```

Tópicos Avançados

```
    nome = input("Informe o nome do contato para o qual
deseja incluir formas de contato ")
    if nome in agenda.keys():
        print(f"As formas de contato suportadas pelo
sistema são: {contatos_suportados}")
        forma_incluida = input("Qual forma de contato
deseja incluir? ")
        if forma_incluida in contatos_suportados:
            valor_incluido = input(f"Informe o
{forma_incluida} que deseja incluir: ")
            if inclui_forma_contato(agenda[nome],
forma_incluida, valor_incluido):
                print("Operação bem sucedida! A nova
forma de contato foi incluída! ")
            else:
                print("Ocorreu um erro durante a
inserção. A agenda não foi alterada.")
        else:
            print("A forma de contato indicada não é
suportada pelo sistema. A agenda não foi alterada.")
    else:
        print("O contato informado não existe na agenda.
Não foram feitas alterações. ")

def usuario_exclui_contato(agenda:dict):
    nome = input("Informe o nome do contato que deseja
excluir: ")
    if exclui_contato(agenda, nome):
        print("Usuário excluído com sucesso!")
    else:
        print("Nome do usuário não localizado na agenda.
Não foram feitas alterações.")

def usuario_altera_forma_contato(agenda:dict):
    nome = input("Informe o nome do contato que deseja
alterar: ")
    if nome in agenda.keys():
        print(f"As formas de contato suportadas pelo
sistema são: {contatos_suportados}")
        forma_incluida = input("Qual forma de contato
deseja incluir? ")
        if forma_incluida in contatos_suportados:
            print(contato_para_texto(nome,
**agenda[nome]))
            valor_antigo = input(f"Informe o
{forma_incluida} que deseja alterar ")
            nova_valor = input(f"Informe o novo
{forma_incluida} ")
            if
altera_forma_contato(agenda[nome][forma_incluida],
valor_antigo, nova_valor):
```

Tópicos Avançados

```
        print("Contato alterado com sucesso!")
    else:
        print("Ocorreu um erro durante a
alteração do contato. A agenda não foi alterada.")
    else:
        print(f"{forma_incluida} não é uma forma de
contato suportada pelo sistema. A agenda não foi alterada.")
    else:
        print(f"O contato {nome} não está na agenda. A
agenda não foi alterada.")

def usuario_altera_nome_contato(agenda:dict):
    #altera_nome_contato(agenda_original:dict,
nome_original:str, nome_atualizado:str):
    nome_original = input("Informe o nome do contato que
deseja alterar: ")
    nome_atualizado = input("Informe o nome do novo
contato: ")
    if altera_nome_contato(agenda, nome_original,
nome_atualizado):
        print(f"O contato foi atualizado e agora se chama
{nome_atualizado}")
    else:
        print(f"O contato original não foi localizado. A
agenda não foi alterada.")

def usuario_contato_para_texto(agenda:dict):
    nome = input("Informe o nome do contato que deseja
exibir: ")
    if nome in agenda.keys():
        print(contato_para_texto(nome, **agenda[nome]))
    else:
        print("O contato informado não está na agenda.")

def exibe_menu():
    print("\n\n")
    print("1 - Incluir contato na agenda")
    print("2 - Incluir uma forma de contato")
    print("3 - Alterar o nome de um contato")
    print("4 - Alterar uma forma de contato")
    print("5 - Exibir um contato")
    print("6 - Exibir toda a agenda")
    print("7 - Excluir um contato")
    print("8 - Exportar agenda para txt")
    print("9 - Exportar agenda para JSON")
    print("10 - Importar agenda de JSON")
    print("11 - Sair")
    print("\n")

def manipulador_agenda():
```

```
agenda = {}
op = 1
while op != 11:
    exibe_menu()
    op = int(input("Informe a opção desejada: "))
    if op == 1:
        usuario_inclui_contato(agenda)
    elif op == 2:
        usuario_inclui_forma_contato(agenda)
    elif op == 3:
        usuario_altera_nome_contato(agenda)
    elif op == 4:
        usuario_altera_forma_contato(agenda)
    elif op == 5:
        usuario_contato_para_texto(agenda)
    elif op == 6:
        print(agenda_para_texto(**agenda))
    elif op == 7:
        usuario_exclui_contato(agenda)
    elif op == 8:
        nome_arquivo = input("Informe o nome ou
caminho do arquivo: ")
        agenda_para_txt(nome_arquivo, agenda)
    elif op == 9:
        nome_arquivo = input("Informe o nome ou
caminho do arquivo: ")
        agenda_para_json(nome_arquivo, agenda)
    elif op == 10:
        nome_arquivo = input("Informe o nome ou
caminho do arquivo: ")
        agenda = json_para_agenda(nome_arquivo)
    elif op == 11:
        print("Saindo do sistema")
        break
    else:
        print("Opção inválida! Informe uma opção
existente.")

manipulador_agenda()
```

Código-fonte 32 – Função agenda_para_json

Fonte: Elaborado pelo autor (2022)

1.4 E o futuro?

Com este projeto, conseguimos reunir boa parte dos conteúdos que trabalhamos ao longo deste curso, mas ele está longe de ser perfeito!

Existem funções sem a devida descrição, mais funções podem ser criadas para validações, existem trechos de códigos que podem virar funções e serem reaproveitados, enfim, um mundo de oportunidades para continuar estudando programação.

E, para ajudá-lo na sua jornada, alguns tópicos que podem compor a sua agenda de estudos.

1.4.1 Boas práticas e a PEP-8

Na sua jornada pelo mundo Python, você ouvirá e lerá muito sobre a PEP-8 e outras PEPs.

As PEPs são propostas de melhorias para o Python e reúnem propostas de novos recursos, documentações de layout e design para que a comunidade possa seguir.

A PEP 8, mais especificamente, é um documento de convenções sobre a biblioteca padrão do Python. Lá está convencionado, por exemplo, que a indentação utilizada para representar que as instruções estão dentro de um bloco é de quatro espaços em branco.

É importante que você saiba onde a PEP 8 se encontra e crie o hábito de consultá-la, pois frequentemente precisamos nos remeter ao princípio da legibilidade que tanto guia o Python.

Já temos aplicada a maior parte dos conceitos desde o início do curso, mas você pode encontrar os outros diretamente no site oficial: <https://peps.python.org/pep-0008/>.

Além disso, vale a pena estar sempre em contato com outros desenvolvedores e trocar experiências e informações sobre boas práticas na escrita e manutenção dos seus códigos.

Esta seção do nosso último capítulo, portanto, é um lembrete: a formação de um desenvolvedor nunca está completa! É preciso continuar estudando, se reciclando e revendo cada linha de código que for produzida.

Reveja o projeto final e busque identificar oportunidades para, dentro do possível, aplicar as boas práticas sugeridas. Não considere este um projeto *final* no sentido de estar na versão mais perfeita, mas sim no sentido de ser um projeto que reúne o que aprendemos até então.

Se você encontrar melhorias, pode e deve implementá-las sempre!

1.4.2 Os ambientes virtuais

Para que um script em Python seja executado, na maior parte das vezes precisamos apenas que o interpretador esteja disponível para dar vida às nossas linhas de código. Mas existem cenários em que é preciso mais.

À medida que você for encontrando novos desafios, passará a usar pacotes e módulos não nativos, que dão maiores poderes ao Python. E é aí que os ambientes virtuais mostram a sua real importância.

Imagine que você está trabalhando em um projeto que usa a versão 1.0 de um módulo que gera arquivos de imagem a partir do Python. Você instala esses módulos no seu computador e segue a vida feliz e contente. Algum tempo depois, porém, você precisa utilizar a versão 1.2 do mesmo pacote em outro projeto. O que fazer?

Ficar instalando e desinstalando pacotes pode se tornar uma tarefa impossível para qualquer desenvolvedor que trabalha com diversos projetos simultaneamente e, por essa razão, é comum utilizarmos ambientes virtuais para resolvermos esse problema.

A versão atual do Python já traz um ambiente virtual chamado *venv*, que pode ser configurado independentemente da IDE utilizada (ou até na ausência de uma). No PyCharm estamos “olhando” para o *venv* desde o princípio:

Tópicos Avançados

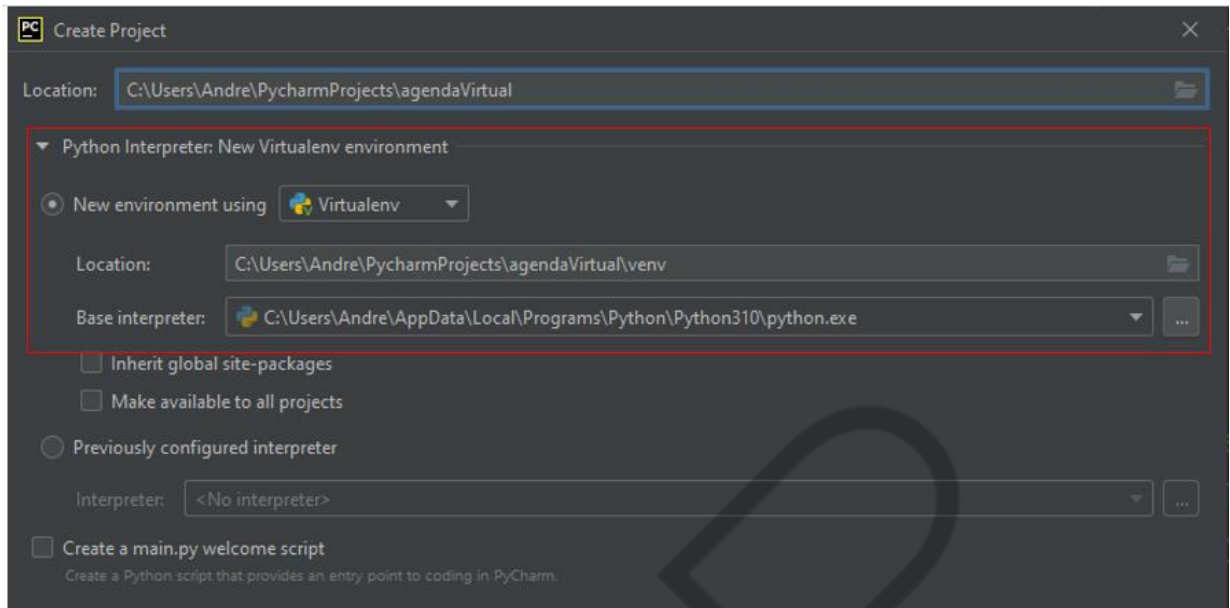


Figura 3 – Na criação de um novo projeto, o PyCharm já providencia um ambiente virtual com o venv
Fonte: Elaborado pelo autor (2022)

Portanto, quando estamos utilizando essa IDE, a nossa preocupação será mínima.

Nos seus próximos projetos, fique sempre atento aos pacotes e módulos que estão sendo utilizados para que eles estejam prontos para uso sem a necessidade de fatores externos!

REFERÊNCIAS

LUTZ, M. **Learning Python**. Sebastopol: O'Reilly Media Inc, 2013.

PUGA, S.; RISSETI, G. **Lógica de programação e estrutura de dados com aplicações em Java**. São Paulo: Pearson Prentice Hall, 2009.

EMASP