02

# Organizando melhor nosso código

## app.py e lógica da aplicação

E se agora eu pedir para você fechar o terminal e abrir novamente? Muito provavelmente você ficará magoado porque perderá todo o código que acabamos de fazer. Sabemos que não precisa ser assim e que podemos criar nossa função em um arquivo em separado.

Dentro da nossa pasta `python` criaremos um arquivo `app.py` e dentro dela definiremos a função `cadastrar`. Vamos apelar para o famoso CONTROL + C e CONTROL + V do código do nosso terminal para dentro do arquivo:

```
# python/app.py

def cadastrar(nomes):
    print 'Digite: o nome:'
    nome = raw_input()
    nomes.append(nome)
```

E agora? Teremos que importar o arquivo e toda aquela burocracia de `import`? Não exatamente, faremos algo diferente. No lugar de executarmos nosso código pelo terminal, via linha de comando, passaremos para o interpretador do Python nosso arquivo `app.py`, mas sem esquecer de entrar no diretório onde salvamos o arquivo antes:

```
$ cd Desktop/python/
$ python app.py
```

Ops! Nada aconteceu porque não chamamos nossa função, apenas a declaramos! Quem sabe faz a hora e não espera acontecer, mas nesse caso, precisamos preparar o terreno antes de chamá-la.

## Menu da aplicação e o laço while

Toda aplicação que se preza possui um menu para que possamos interagir. Com nossa pequena aplicação não será diferente. Desta maneira, o usuário poderá executar a função `cadastrar` quando ele quiser. Para isso criaremos dentro do arquivo `app.py` uma nova função `menu()`. Ela será responsável pela captura da escolha do usuário (cadastrar, remover e por aí vai):

```
# imediatamente abaixo da função cadastrar:

def menu():
    nomes = []
    escolha = ''
```

A variável `escolha` representa a ação que o usuário gostaria de executar. Vamos dizer que a escolha `1` significa cadastrar um novo perfil de usuário e `0` significa terminar o programa.

Ou seja, *enquanto o usuário não digitar `0` a aplicação continuará executando*. Utilizamos a palavra "enquanto", que nos lembra o famoso `while` que executará infinitamente nosso código enquanto a variável `escolha` não for 0. Por enquanto fica apenas uma saída dentro do laço para explicar as opções disponíveis e guardar a escolha do usuário através do `raw_input()`:

```
def menu():
    nomes = []
    escolha = ''
    while(escolha != '0'):
        print 'Digite: 1 para cadastrar, 0 para terminar'
        escolha = raw_input()

        if(escolha == '1'):
            cadastrar(nomes)
```

No fim do arquivo `app.py` adicione a chamada da função `menu` para começar a aplicação. Nosso arquivo completo ficará assim:

```
#arquivo app.py

def cadastrar(nomes):
    print 'Digite: o nome:'
    nome = raw_input()
    nomes.append(nome)

def menu():
    nomes = []
    escolha = ''
    while(escolha != '0'):
        print 'Digite: 1 para cadastrar, 0 para terminar'
        escolha = raw_input()

        if(escolha == '1'):
            cadastrar(nomes)

menu()
```

Agora é só testarmos nosso programa:

```
$ python app.py
```

Podemos cadastrar vários nomes, mas como saberemos se todos estão sendo armazenados? Que tal listarmos o conteúdo da nossa lista? Para isso, precisamos adicionar mais uma opção para o usuário.

## Varrendo nossa lista com a instrução for

Vamos adicionar mais uma opção para listar os nomes já cadastrados. Estipularemos que o número 2 representa essa ação:

```
def menu():
    nomes = []
    escolha = ''
    while(escolha != '0'):
        print 'Digite: 1 para cadastrar, 2 para listar, 0 para terminar'
        escolha = raw_input()

        if(escolha == '1'):
```

```
            cadastrar(nomes)

        if(escolha == '2'):
            listar(nomes)
```

Só que a função `listar` não existe ainda. Vamos adicioná-la no arquivo `app.py`. Igual à função `cadastrar`, a função `listar` recebe também a lista de nomes como parâmetro. Vamos imprimir uma mensagem que confirma a ação do usuário:

```
def listar(nomes):
    print “Listando nomes:”
```

Depois disso precisaremos imprimir cada nome da lista. Para isso usaremos um laço do tipo `for`. Para *cada nome dentro da lista nomes* imprima o nome:

```
def listar(nomes):
    print “Listando nomes:”
    for nome in nomes:
        print nome
```

Vamos salvar o arquivo `app.py` e verificar sempre a sintaxe e indentação. Para testarmos, executaremos o arquivo com o comando `python app.py`

Segue uma vez o código completo:

```
def cadastrar(nomes):
    print 'Digite: o nome:'
    nome = raw_input()
    nomes.append(nome)

def menu():
    nomes = []
    escolha = ''
    while(escolha != '0'):
        print 'Digite: 1 para cadastrar, 2 para listar, 0 para terminar'
        escolha = raw_input()

        if(escolha == '1'):
            cadastrar(nomes)

        if(escolha == '2'):
            listar(nomes)

def listar(nomes):
    print 'Listando nomes:'
    for nome in nomes:
        print nome


menu()
```

Excelente, será que tudo funciona? Aparentemente sim. Vamos cadastrar mais um nome, desta vez "Flávio Almeida", com acento mesmo. Que saída é essa?

```
'Fl\xc3\xa1vio Almeida'
```

Será que "Flávio Almeida" é um nome tão especial assim para sair desse jeito? Vamos tentar outro, desta vez "Rômulo Henrique":

```
'R\xc3\xb4mulo Henrique'
```

Não é possível, dois nomes especiais? Na verdade, o que é especial é a cadeia de caracter utilizada, é como se o Python não a entendesse. Aliás, qual a cadeia de caracteres utilizamos? UTF-8. Existem outros como LATIN1 e por aí vai. O Python não consegue sozinho inferir o tipo, por isso precisamos dar uma pista para ele através do comentário especial adicionado no início do nosso arquivo **app.py**:

```
# -*- coding: UTF-8 -*-

def cadastrar(nomes):
    print 'Digite: o nome:'
    nome = raw_input()
    nomes.append(nome)

def menu():
    nomes = []
    escolha = ''
    while(escolha != '0'):
        print 'Digite: 1 para cadastrar, 2 para listar, 0 para terminar'
        escolha = raw_input()

        if(escolha == '1'):
            cadastrar(nomes)

        if(escolha == '2'):
            listar(nomes)

def listar(nomes):
    print 'Listando nomes:'
    for nome in nomes:
        print nome


menu()
```

Vamos parar e rodar nossa aplicação novamente.

Agora sim, nomes acentuados são impressos corretamente. Nos exercícios criaremos mais funções e melhoraremos a saída. Vamos lá!