Curso de Computação 1
Introdução à Programação em Python

AULA 6

Funções para manipulação de iteráveis

Nas aulas anteriores foram apresentados diversos tipos iteráveis (strings, tuplas, listas e dicionários). Nesta aula, consolidaremos o entendimento das estruturas iteráveis e como usar funções já prontas, manipulando dados não triviais, para definir funções mais sofisticadas.

1. Manipulação de sequências (revisão)

Manipular um dado é uma maneira de dizer que podemos trabalhar o dado, fazer operações sobre ele, a fim de atender a alguma demanda. Além de algumas operações muito comuns, como por exemplo a soma aritmética entre inteiros, muitas vezes precisamos usar funções pré-definidas para conseguir manipular os dados de uma maneira mais fácil ou confortável.

Algumas operações são comuns a vários tipos de dados iteráveis. A operação de **indexação** é comum a strings, tuplas, listas e dicionários; para strings tuplas de listas os índices serão números inteiros, enquanto que para dicionários, os índices são as chaves dos mapeamentos já inseridos. Já o **fatiamento** pode ser aplicado a strings, tuplas e listas, porém *não* é aplicável a dicionários (que não pressupõem ordem entre os mapeamentos inseridos).

O operador **in** (e consequentemente a expressão **not in**), que relaciona elementos com um dado iterável, também pode ser aplicado à strings, tuplas, listas e dicionários. Lembrando que esse operador retorna True caso o elemento esteja no iterável, e falso caso contrário (o oposto ocorre com a expressão **not in**).

Os operadores de **concatenação** e **multiplicação** podem ser aplicados em tuplas, listas e strings. Não se aplicam diretamente a dicionários.

2. Funções específicas para manipulação de strings

Python provê funções básicas de manipulação de dados específicas para cada tipo. No caso das strings, podemos citar, por exemplo, **str.lower()**, **str.upper()**, **str.count()** e **str.index()**. No vídeo a seguir, vamos falar sobre estas funções e como fazer o uso

adequado delas. Sugerimos que após ver o vídeo com a explicação, abra a shell do IDLE e leia a documentação de cada uma dessas funções disponíveis via comando help.

Manipulação de Strings

Talvez tenha ficado alguma dúvida sobre como essas funções funcionam na prática. Uma maneira de tirar essas dúvidas é fazer alguns testes na shell do python e ver os resultados.

Em Python existem funções diversas para manipulação de Strings. Apresentamos nesta seção mais algumas funções que podem lhe ser úteis na manipulação de strings. No exercício a seguir, **faça testes na shell do IDLE** para entender melhor o funcionamento delas. Exercite sua curiosidade fazendo outros testes a mais para entender bem estas funções.

Exercício: Na Shell do IDLE, execute os exemplos abaixo e tente entender o que acontece para cada situação.

Qual será o retorno de cada execução abaixo?

```
>>> str.count('banana', 'a')
>>> str.count('batata', 'n')
```

Para os próximos exercícios, iremos criar uma variável *s* que irá conter a string *"quem parte e reparte, fica com a maior parte"*.

```
>>> s = "quem parte e reparte, fica com a maior  parte"
```

Qual será o retorno de cada execução da função str.index() para a string contida em s?

```
>>> str.index(s, "parte")
>>> str.index(s, "parte", 13)
>>> str.index(s, "parcela")
```

Repita o exercício anterior usando a função *str.find()* passando os mesmo parâmetros. Há alguma diferença no retorno?

Agora vamos testar exemplos com a função *str.replace()*. Ainda usando a string contida na variável *s*, o que será retornado após a execução das linhas abaixo?

```
>>> str.replace(s,"parte","parcela")
```

Curso de Computação 1
Introdução à Programação em Python

```
>>> str.replace(s, "parte","parcela",2)
```

Lembrando para que serve a função *str.partition()*, qual o retorno das seguintes chamadas dessa função usando a variável *s* como parâmetro?

```
>>> str.partition(s,"t")
>>> s.partition("z")
```

Sobre o que você viu no vídeo sobre a função *str.split()*, qual o resultado das seguintes execuções? Repare que foi atribuído um novo valor à variável s.

```
>>> s = "xxx yyy zzz xxx yyy zzz"
>>> str.split(s)
>>> str.split(s, 'zzz')
```

Agora, vamos testar alguns exemplos usando a função *str.join()*. O que será retornado para cada execução abaixo?

```
>>> str.join("/",("usr","bin","python"))
```

Agora vamos testar com uma lista. O que irá acontecer após executar a linha abaixo?

```
>>> str.join("/",["usr","bin","python‟])
```

O que acontece com os demais exemplos abaixo?

```
>>> str.join("*", "teste")
>>> str.join("Q", (1,2,3,4,5))
>>> str.join("Q",('1','2','3','4','5'))
```

Importante! Como o *str.split()* gera como saída uma sequência de strings, é comum que essa saída seja usada como entrada para *str.join*. Vamos testar com o seguinte exemplo:

```
>>>  str.join("/", str.split('19-09-2020', ‘-’))
```

Por último, vamos realizar alguns testes com a função *str.strip()*. Dado os parâmetros dos exemplos abaixo, o que será retornado? Repare que temos três funções diferentes nas linhas de execução, são elas: *str.strip()*, *str.rstrip()* e *str.lstrip()*.

```
>>> str.strip("   xxx afdsfa   ")
>>> str.strip("xxx yyy zzz xxx","xy ")
```

Curso de Computação 1
Introdução à Programação em Python

```
>>> str.strip("xxx yyy zzz xxx yyy","xy")

>>> str.rstrip("   xxx   ")
>>> str.lstrip("   xxx   ")
```

Você é livre para fazer outros testes a mais que achar conveniente para entender estas funções.

Na tabela abaixo, apresentamos um resumo das funções de strings estudadas até agora:

Strings

Função	**Para que serve**	**Exemplo**
upper	retorna a string passada como parâmetro em letras maiúsculas.	str.upper(<<nome da string>>)
lower	retorna a string passada como parâmetro em letras minúsculas.	str.lower(<<nome da string>>)
count	conta o número de ocorrências de uma substring dentro de uma string. Lembrando que os parâmetros de *inicio* e *fim* são opcionais.	str.count(<<nome da string>>, <<substring>>, <<início>>, <<fim>>)
index	retorna o índice da primeira ocorrência de uma substring dentro de uma string. Lembrando que os parâmetros de *início* e *fim* são opcionais.	str.index(<<nome da string>>, <<substring>>, <<início>>, <<fim>>)
find	retorna o índice da primeira ocorrência de uma substring em uma string. Lembrando que os parâmetros de *início* e *fim* são opcionais e que não é levantado um erro quando a substring não é encontrada, como acontece com *str.index()*.	str.find(<<nome da string>>, <<substring>>, <<início>>, <<fim>>)
replace	substitui as n primeiras	str.replace(<<nome da

Curso de Computação 1
Introdução à Programação em Python

Strings

	ocorrências de uma substring por uma nova substring dentro de uma string. Lembrando que o número de ocorrências é opcional e, se não for definido, são trocadas todas as ocorrências.	string>>,<<substring a ser substituída>>,<<substring de substituição>>, <<número de ocorrências>>)
strip	retorna uma substring de uma string passada como parâmetro sem os caracteres pertencentes à substring passada como segundo parâmetro. Se a substring *caractere* não for especificada, retira caracteres em branco.	str.strip(<<nome da string>>, <<caractere>>)
partition	Divide a *string* em 3 partes: o que vem antes do *separador*, *separador* e o que vem depois do *separador*. Se *separador* não pertence à *string*, a *string* é retornada seguida por duas strings vazias.	str.partition(<<string>>, <<separador>>)
split	Retorna uma lista com as substrings da *string* presentes entre cópias de uma outra string *separador*. Se *separador* não for especificado, assume-se sequências de caracteres em branco, tabs ou newlines.	str.split(<<string>>, <<separador>>)
join	Concatena uma sequência de strings (fornecidas em uma tupla ou lista ou string), intercalados com cópias do separador.	str.join(<<separador>>, <<sequencia>>)

Curso de Computação 1
Introdução à Programação em Python

3. *Alias* x cópia de listas

Tecnicamente, a mutabilidade de listas, com a consequente possibilidade de alteração de parte de uma lista sem a necessidade de se criar uma nova lista, influencia o comportamento das atribuições da forma var2 = var1 quando var1 é uma lista.

Recapitulando aqui um trecho de uma aula anterior, onde introduzimos o comando de atribuição:

> O processamento de uma atribuição em Python, então, é sempre o mesmo: avalia o resultado da expressão à direita do operador de atribuição e associa esse resultado à variável. Na realidade, quase sempre, pois existe uma exceção, que é chamada de **alias**. Alias significa **sinônimo**. Se a expressão à direita do operador de atribuição for apenas uma outra variável, como por exemplo, var2 = var1, não será realizada uma avaliação da expressão var1. Se var1 fosse avaliada e seu resultado armazenado na memória (em var2) isso significaria que uma cópia do valor de var1 seria associada à variável var2. Nesse caso particular, Python faz com que var2 e var1 indiquem a mesma posição da memória, até que uma das duas variáveis apareça como lado esquerdo em outra atribuição, passando a indicar outra posição da memória.

Então, quando se atribui uma variável que contém um valor do tipo lista a uma outra variável, na verdade está se criando um ***alias*** para a lista original. Um ***alias*** é um nova forma de chamar a mesma região de memória com um novo nome. Alterações em partes da lista representada por uma das variáveis serão na realidade alterações no valor de ambas as variáveis. Por exemplo:

```
>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> nums2 = nums
>>> nums[4] = 55
>>> nums
[10, 20, 30, 40, 55, 60, 70, 80, 90]
>>> nums2
[10, 20, 30, 40, 55, 60, 70, 80, 90]
```

Imagine agora que não seja esse o comportamento desejado, ou seja, que queremos que `nums2` seja uma lista diferente (uma região diferente da memória), contendo elementos iguais aos pertencentes à lista `nums`. Precisamos então fazer com que `nums2` seja uma

cópia de `nums`. Isso é muito comum quando queremos manipular os dados de uma lista sem perder as informações originais.

```
>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> nums2 = nums[:]
>>> nums[4] = 55
>>> nums
[10, 20, 30, 40, 55, 60, 70, 80, 90]
>>> nums2
[10, 20, 30, 40, 50, 60, 70, 80, 90]
```

Atividade: Vamos agora treinar um pouco com exercícios de fixação. Responda as perguntas da atividade “Manipulação de Strings e Listas”. Fique atento ao prazo de entrega dessa atividade!

4. Funções específicas para manipulação de listas

O Python também oferece um conjunto de funções para lidar com listas. Uma propriedade importante, e que torna a manipulação de listas um pouco diferente da manipulação de strings em Python, é o fato das listas serem **mutáveis**.

Ao lidar com tipos de dados imutáveis, estamos constantemente gerando novos dados a partir da aplicação de operações e funções a dados que já temos. Ao lidar com as listas, temos outras opções. Podemos manipular uma lista mudando uma parte dela! Isso pode ser feito através de operações de atribuição a uma posição indexada da lista, como visto anteriormente, ou pelo uso de **funções que modificam seus dados de entrada**.

Se uma lista é passada como parâmetro de entrada para uma função e ocorre de ela ser modificada dentro da função, a modificação persiste mesmo quando a execução da função termina. É preciso ter muito cuidado com isso, pois conceitualmente, o objetivo de uma função deve ser única e exclusivamente produzir seu valor de retorno, sem gerar nenhuma modificação em outros dados. Essas modificações, quando acontecem, são consideradas “efeitos colaterais” de uma função, e não são uma boa prática de programação: elas têm um efeito negativo na legibilidade do código. É muito fácil identificar o valor de retorno de uma função olhando seu código, mas não é fácil perceber se algo que está sendo feito no decorrer de uma função modifica dados que podem ser usados em outras partes do código.

Porém, há casos específicos onde a modificação dos dados de entrada de uma função é aceitável. Isto acontece por exemplo em funções cujo objetivo é manipular dados específicos. Algumas funções feitas para manipular listas tem essa característica. Assim sendo, diferente das funções de strings vistas anteriormente, há no Python funções de

manipulação de listas que algumas vezes modificam a lista passada como dado de entrada, e algumas vezes, nem retornam valor nenhum, já que a modificação da lista de entrada já cumpre com o objetivo da função. Fique atento a isso ao estudar as funções desta seção.

Python dispõe de algumas funções e operadores para a manipulação de listas. Algumas funções servem para a inserção de valores em uma lista. Outras funções serão utilizadas para deleção (remoção) de valores nas listas. Há também algumas funções para operações como obtenção de índices, ordenação, inversão da ordem dos elementos, etc.

Vejamos primeiramente as funções de **inserção** em listas:

Listas

Função	**Para que serve**	**Exemplo**
append	Insere um elemento no final da lista (modifica a lista de entrada, sem valor de retorno)	list.append(<<nome da lista>>, <<novo elemento>>)
extend	Insere um elemento no final da lista (modifica a lista de entrada, sem valor de retorno) Aceita mais de um argumento por vez se informados dentro de uma lista	list.extend(<<nome da lista>>, <<novo elemento>>)
insert	Insere um elemento na lista na posição informada (modifica a lista de entrada, sem valor de retorno)	list.insert(<<nome da lista>>, <<índice>>, <<novo elemento>>)

Alguns exemplos:

```
>>> lista = [1, 2, 3, 4, 5]
>>> list.append(lista, 6)
>>> lista
[1, 2, 3, 4, 5, 6]
>>> list.extend(lista, [7, 8, 9, 10])
>>> lista
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list.insert(lista, 0, 0)
>>> lista
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Agora veja o vídeo sobre funções de inserção em listas:

Curso de Computação 1
Introdução à Programação em Python

Inserção de elementos em listas

Vejamos as funções de **remoção** de elementos em listas:

Listas

Função	Para que serve	Exemplo	Observações
remove	Remove a primeira incidência do elemento na lista (modifica a lista de entrada, sem valor de retorno)	list.remove(<<nome da lista>>, <<elemento>>)	O elemento a ser removido deve estar na lista
del	Remove o elemento na posição informada (modifica a lista de entrada, sem valor de retorno)	del <<nome da lista>>[<<índice>>]	O índice deve estar dentro da faixa de índices da lista
pop	Remove e retorna o elemento na posição informada (modifica a lista de entrada)	list.pop(<<nome da lista>>, <<índice>>)	O índice deve estar dentro da faixa de índices da lista
clear	Remove todos os elementos de uma lista (sem valor de retorno)	list.clear(<<nome da lista>>)	

Alguns exemplos:

```
>>> lista = [1, 3, 9, 27]
>>> lista
[1, 3, 9, 27]
>>> list.remove(lista, 27)
>>> lista
[1, 3, 9]
>>> del lista[0]
>>> lista
[3, 9]
```

Curso de Computação 1
Introdução à Programação em Python

```
>>> list.pop(lista, -1)
9
>>> lista
[3]
>>> list.clear(lista)
>>> lista
[]
```

Veja o vídeo sobre funções de remoção em listas:

Remoção de elementos de listas

Finalmente, vejamos outras funções e operadores de listas:

Listas

Função	Para que serve	Exemplo	Observações
count	Retorna quantas vezes um elemento informado aparece na lista	list.count(<<nome da lista>>, <<elemento>>)	
index	Retorna a posição da primeira incidência do elemento na lista	list.index(<<nome da lista>>, <<elemento>>)	O elemento deve estar na lista
in	Retorna True se o elemento está na lista ou False caso contrário	<<elemento>> in <<nome da lista>>	
reverse	Inverte a ordem dos elementos da lista de entrada (modifica a lista de entrada, sem valor de retorno)	list.reverse(<<nome da lista>>)	
sort	Ordena os elementos da lista de entrada (modifica a lista de entrada, sem valor de	list.sort(<<nome da lista>>)	Se os tipos de elementos da lista forem incompatíveis pode ocorrer erro

	retorno)		

Mais alguns exemplos:

```
>>> lista = [0, 1, 2, 0, 1, 2, 3, 4, 5]
>>> list.count(lista, 0)
2
>>> list.count(lista, 9)
0
>>> list.index(lista, 0)
0
>>> list.index(lista, 5)
8
>>> 9 in lista
False
>>> 5 in lista
True
>>> list.reverse(lista)
>>> lista
[5, 4, 3, 2, 1, 0, 2, 1, 0]
>>> list.sort(lista)
>>> lista
[0, 0, 1, 1, 2, 2, 3, 4, 5]
```

Veja agora, com mais detalhes as outras funções e operadores de listas:

Ainda mais funções e operações sobre listas

Atividade: Vamos agora treinar um pouco com exercícios de fixação. Responda as perguntas da atividade “Manipulação de listas”. Fique atento ao prazo de entrega dessa atividade!

5. Funções específicas para manipulação de dicionários

Python dispõe de algumas funções para a manipulação de dicionários. Há funções para operações como obtenção de chaves, valores, itens, etc.

Curso de Computação 1
Introdução à Programação em Python

Dicionários

Função	Para que serve	Exemplo	Observações
len	Retorna o número de mapeamentos do dicionário.	len(<<nome do dicionário>>)	
keys	Retorna o conjunto de chaves do dicionário.	dict.keys(<<nome do dicionário>>)	
values	Retorna o conjunto de valores do dicionário.	dict.values(<<nome do dicionário>>)	
items	Retorna todos os itens *(chave, valor)* do dicionário.	dict.items(<<nome do dicionário>>)	
get	Retorna o valor associado à chave passada como parâmetro. Nesta função, podemos informar um *valor de retorno* e, caso a chave não pertença ao dicionário, é este *valor de retorno* que será retornado.	dict.get(<<nome do dicionário>>, <<chave>>, <<valor de retorno>>)	
del	Remove um item do dicionário.	del <<nome do dicionário>> [<<chave>>]	A chave deve pertencer ao dicionário.
clear	Remove todos os itens do dicionário.	dict.clear(<<nome do dicionário>>)	

Curso de Computação 1
Introdução à Programação em Python

copy	Copia o dicionário para outra variável.	dict.copy(<<nome do dicionário>>)	Os elementos mutáveis dentro do dicionário (listas ou dicionários) do novo dicionário serão apenas referências aos elementos do dicionário original.

Exemplos:

```
>>> produtos = {'farinha': 3.00,
                'feijão': 5.00,
                'leite': 4.25,
                'açúcar': 2.49}
>>> len(produtos)
4

>>> list(dict.keys(produtos))
['farinha', 'feijão', 'leite', 'açúcar']

>>> list(dict.values(produtos))
[3.0, 5.0, 4.25, 2.49]

>>> list(dict.items(produtos))
[('farinha', 3.0), ('feijão', 5.0), ('leite', 4.25), ('açúcar',
2.49)]

>>> dict.get(produtos, 'feijão')
5.0

>>> dict.get(produtos, 'arroz')
None

>>> dict.get(produtos, 'arroz', 'Produto não encontrado')
'Produto não encontrado'

>>> del produtos['leite']
>>> produtos
{'farinha': 3.0, 'feijão': 5.0, 'açúcar': 2.49}
```

Curso de Computação 1
Introdução à Programação em Python

```
>>> dict.clear(produtos)
>>> produtos
{}
```

Veja mais sobre as funções de dicionários no vídeo:

Manipulação de dicionários

Atividade: Faça agora a atividade “Manipulação de dicionários”, para exercitar seus conhecimentos. Fique atento ao prazo de entrega dessa atividade!

Até a próxima aula.