Essa é uma versão dos Guias de Estudo desenvolvida para auxiliar o processo de impressão e para facilitar a leitura dos guias por programas que fornecem a leitura automatizada para suporte a todos os alunos que necessitem. Dessa forma, não apresentaremos ilustrações nesse arquivo.* *#ParaTodosLerem***.

Módulo #2

Guias de estudo

Listas

Até o momento, além de saber desenvolver códigos em Python, você aprendeu algumas estruturas que podemos armazenar os dados durante a execução de um programa. Vamos relembrar o que é uma lista: as listas possibilitam a armazenagem de dados em sequência.

Exemplo: `lista=[1,2,3,4,5,6,7,8,9,10]`

Criamos uma lista com números de 1 a 10. Agora, imagine uma lista com 10.000.000 de números. Caso precisarmos percorrer todos estes dados usando estrutura de repetição, este código demoraria muito para ser executado. Então, pensando em otimizar a utilização das listas foi criado o ***list comprehension****.*

Vamos ver agora como podemos criar uma nova lista com os elementos da lista acima multiplicados por 2:

```
listaNova=[]
for x in lista:
listaNova.append(x*2)


listaNova
resultado:
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Agora vamos ver como podemos fazer isso com ***list comprehension***

```
[x*2 for x in lista]

resultado:
[x*2 for x in lista]
```

Muito mais prático não é mesmo?

Matrizes

Uma outra estrutura de dados muito famosa são as matrizes. Elas nos permite armazenar dados em 2 dimensões, como mostra a imagem a seguir:

25.0	20.0	19.3	15.9
18.7	24.5	17.8	19.9
22.4	19.2	15.9	23.8

Suponha que na tabela temos notas de 3 alunos em 4 provas. Cada prova vale 25.0 pontos. Nas linhas temos os alunos e nas colunas cada uma das provas. Para armazenar e organizar melhor esses dados é necessário uma estrutura com 2 dimensões (linhas e colunas). Vamos criar esta matriz:

```
matriz=[[25.0,20.0,19.3,15.9],[18.7, 24.5, 17.8, 19.9], [22.4, 19.2, 15.9, 23.8]]
```

Para percorrer os dados de uma matriz é necessário utilizar 2 estruturas de repetição. Neste caso vamos percorrer os valores, multiplicar por 2 e adicionar os novos elementos em um nova matriz. Veja:

```
novaMatriz=[]
for linha in matriz:
listaNotas=[]
for elemento in linha:
listaNotas.append(elemento*2)
novaMatriz.append(listaNotas)

novaMatriz
```

Vamos desenvolver a mesma ideia acima usando list comprehension:

```
novaMatriz=[[elemento*2 for elemento in linha]for linha in matriz]

novaMatriz
```

Magic Methods

Os métodos mágicos em Python são métodos especiais , já implementados, que começam e terminam com sublinhados duplos. Eles também são chamados de ***métodos dunder****.*

Os métodos mágicos não são invocados diretamente pelo programador, a invocação ocorre internamente. Por exemplo, quando você quer somar dois números e utiliza o operador +, internamente, o método __add__() é acionado. Veja:

```
numero1=10
numero2=5

numero1 + numero2
```

Resultado:
15

Agora teste:

```
numero1.__add__(numero2)
```

Resultado:
15

Um outro exemplo de método mágico: em uma classe, quando você utiliza o magic method ***__init__****, ele é executado automaticamente ao criar um objeto da classe, sem que você precise chamá-lo.*

Mas os magic methods não param por aí veja uma lista da documentação oficial do Python:

https://docs.python.org/pt-br/3/library/unittest.mock.html#magicmock-and-magic-method-support

Iterators, generators e decorators

Agora, vamos falar um pouquinho sobre ***iterators****,* ***generators*** *e* ***decorators****. Já pararam para pensar como a estrutura de repetição* ***for*** *consegue, automaticamente, processar cada um dos elementos de uma lista? Pois bem, tudo isso se dá graças aos* ***iterators****, neste caso o iterator utilizado é o next. Veja:*

Suponha uma lista de 1 a 10. Vamos criar esta lista utilizando o range:

```
lista= list(range(1,11))
```

Agora, vamos percorrer os elementos da lista e printar na tela usando for:

```
for x in lista:
 print(x)
```

Veja que é possível percorrer os elementos usando o next:

```
next(lista)
next(lista)
next(lista)
```

*Os **iterators** (**Iteradores**) são métodos que iteram estruturas de dados como **listas**, **tuplas** e outras. Com isso, podemos percorrer um objeto e retornar seus elementos.*

*Os **generators** (**geradores**) são métodos que retornam um iterador que produz uma sequência de valores. Os geradores são úteis quando queremos produzir uma sequência de valores. Lembram do range?*

*E o que são os **decorators**?*

*Os **decorators** recebem uma função, adicionam alguma funcionalidade e a retorna. Ou seja, eles modificam uma função e adicionam novas funcionalidades.*

Lembre-se

Tudo que você aprendeu neste módulo te faz ter uma compreensão melhor de como implementar métodos mais eficientes e como muitas funções do Python foram implementadas.