

LIST COMPREHENSIONS

List Comprehension foi concebida na [PEP 202](#) e é uma forma concisa de criar e manipular listas.

```
listanumeros = [1,4,5,7,10]
```

```
lc1 = [multiplica * 2 for multiplica in listanumeros]
```

```
print(lc1)
```

```
listapares = [p for p in range(20) if p % 2 == 0]
```

```
print(listapares)
```

LIST COMPREHENSIONS

```
listanomes = [ 'Joao', "alberto", "Juca", "afonso"]
```

```
lc2 = [ troca.replace( 'a','@') for troca in listanomes]
```

```
print(lc2)
```

DICT COMPREHENSIONS

O PEP 202 introduz uma extensão sintática no Python chamada *List Comprehension* . Este é uma extensão sintática semelhante chamada de *dict Comprehension*. Você pode usar de maneiras muito semelhantes às compreensões de lista, exceto que elas produzem objetos de dicionário Python em vez de objetos de lista.

```
listadic = [  
    ('valor1', 10),  
    ('valor2', 8),  
    ('valor3', 30),  
]
```

```
d1 = { c: v for c,v in listadic }  
print(d1)
```

```
d2 = { c: v*2 for c,v in listadic }  
print(d2)
```

DICT COMPREHENSIONS

```
d3 = { c.upper(): v for c,v in listadic }  
print(d3)
```

ITERADORES EM PYTHON

OBJETOS ITERAVEIS

Em Python Listas, Tuplas, Dicionarios e Strings são objetos iteráveis.

Exemplo:

```
lista = [ 'Joao', 1 , "Pedro", 45]
```

Como checar se um objeto é iterável. Utilizando método `__iter__`

```
print(hasattr(lista, '__iter__'))
```

```
string = "Florianópolis"
```

```
print(hasattr(string, '__iter__'))
```

ITERADORES EM PYTHON

Executando um iterador em um iterável.

```
for s in string:  
    print(s)
```

ITERADORES

For é um dos iteradores mais utilizados. Porém podemos transformar nossos iteráveis em iteradores e geradores. Aplicamos o método **iter** em um objeto **Iteravel**.

```
nome= "João Ricardo"
```

```
lista_iterador = iter(nome)
```

```
print(type(lista_iterador))
```


ITERADORES EM PYTHON

Utilizamos o método next para iterar em memória.

```
print(hasattr(listaiterador, '__next__'))
```

```
print(next(listaiterador))
```

```
print(next(listaiterador))
```

```
print(next(listaiterador))
```

```
print(next(listaiterador))
```

```
print( '''Separador  
de Iteração ''' )
```

Utilizamos for para iterar o restante da sequencia

```
for n in listaiterador:  
    print(n)
```

ITERADORES EM PYTHON

Geradores

Os geradores trabalham com otimização de memória onde não é necessário carregar todos os elementos de uma função ou estrutura de dados.

Vamos utilizar uma função normal sem ser uma função geradora.

```
import time
```

```
def funcgerador():
```

```
    l = []
```

```
    for n in range(100):
```

```
        l.append(n)
```

```
        time.sleep(0.1)
```

```
    return l
```


ITERADORES EM PYTHON

```
gen = funcgerador()
```

Testando o tipo da função

```
print(funcgerador)
```

Iterando a Função

```
for i in gen:  
    print(i)
```

Declarando mesma função como tipo geradora. Utilizando o método **yield**

```
def funcgerador():  
    for n in range(100):  
        yield n  
        time.sleep(0.1)
```

```
gen = funcgerador()
```

ITERADORES EM PYTHON

Testando Tipo da Função

```
print(gen)
```

Iterando a Função Geradora

```
for i in gen:  
    print(i)
```

Criando Geradores através de uma lista.

```
lc1 = (l for l in range(100))
```

Iterando a lista.

```
for l in lc1:  
    print(l)
```

ITERADORES EM PYTHON

Transformando a lista em gerador.

```
gen1= (l for l in range(100))
```

Analisando o tipo dos dados

```
print(type(gen1))  
print(type(lc1))
```

Iterando o Gerador

```
print(next(gen1))
```

```
for l in gen1:  
    print(l)
```

ITERADORES EM PYTHON

Validando a memória do uso dos objetos lista e geradores
Vamos importar o modulo **sys** para utilizar os recursos do sistema operacional.
Vamos utilizar o método getsizeof.

```
import sys
```

Vamos testar os valores de memória da listas e do geradores

```
print(sys.getsizeof(lc1))  
print(sys.getsizeof(gen1))  
print(sys.getsizeof(nome))  
print(sys.getsizeof(lista))  
print(sys.getsizeof(string))
```

EXERCICIO DE FIXAÇÃO

Crie 1 list comprehension executando algumas operação matemática

Crie 1 list comprehension listando valores impares em um range de números.

Crie 1 dict comprehension através de uma lista de strings.

Crie 1 dict comprehension através de uma lista de inteiros executando uma operação matemática.

Crie uma coleção de dados e valide se a mesma possui a propriedade de iteravel com a função `hasattr` e `__iter__`

Crie uma lista e transforme a mesma em um iterador e valide a iteração da mesma com a função `next`.

Crie uma variável de string e itere a mesma como o função `next` e o `for` e itere até a mesma gerar a exceção `StopIteration`

EXERCICIO DE FIXAÇÃO

Crie um list comprehension e faça os teste com o iterador for. Transforme essa lista em um gerador e itere a mesma com o for. Analise o comportamento da list x gerador. Valide com o comando type o tipo dos objetos criados.

Valide o uso de memória da lista criada anteriormente x o gerador. Aumente os valores da lista e valide se a memória irá aumentar e aumente do gerador também e valide será aumentar.

TRABALHANDO COM ARQUIVOS

Em Python podemos manipular diversos tipos de arquivos.

Métodos de manipulação de arquivos:

Método	Utilização
<code>open()</code>	Usada para abrir o arquivo
<code>read()</code>	Leitura do arquivo
<code>write()</code>	Gravação no arquivo
<code>seek()</code>	Retorna para o início do arquivo
<code>readlines()</code>	Retorna a lista de linhas do arquivo
<code>close()</code>	Fecha o arquivo

TRABALHANDO COM ARQUIVOS

Em Python podemos manipular diversos tipos de arquivos.

Métodos de manipulação de arquivos:

Character	Meaning
'r'	open for reading (default)
'w'	open for writing, truncating the file first
'x'	open for exclusive creation, failing if the file already exists
'a'	open for writing, appending to the end of the file if it exists
'b'	binary mode
't'	text mode (default)
'+'	open for updating (reading and writing)

The default mode is `'r'` (open for reading text, synonym of `'rt'`). Modes `'w+'` and `'w+b'` open and truncate the file. Modes `'r+'` and `'r+b'` open the file with no truncation.

TRABALHANDO COM ARQUIVOS

Manipulando Arquivos formatado como TXT.

Abrindo arquivo para leitura

```
arqu1 = open("arquivos/arquivo.txt", "r" )
```

Lendo o arquivo

```
print(arqu1.read())
```

Seek retorna para o inicio do arquivo

```
print(arqu1.seek(0,0))
```

Fechando o arquivo

```
arqu1.close()
```

TRABALHANDO COM ARQUIVOS

Abrindo arquivo para opção de gravação

```
arqu2 = open("arquivos/arquivo.txt", "w+" )
```

Gravando no arquivo

```
arqu2.write('Tem novo Conteudo\n')
```

```
arqu2.write('Tem novo Conteudo novamente\n')
```

Fechando o arquivo

```
arqu2.close()
```

Voltando posição do cursos e lendo o arquivo

```
print(arqu2.seek(0,0))
```

```
print(arqu2.read())
```

TRABALHANDO COM ARQUIVOS

Atualizando o arquivos

```
arqu2 = open("arquivos/arquivo.txt", "a+" )
```

```
arqu2.write("Nova escrita")  
arqu2.close()
```

Trazer para o começo o cursor

```
arqu2.seek(0,0)
```

Abrir arquivo

```
print(arqu2.read())
```


TRABALHANDO COM ARQUIVOS

gerenciando Contexto de arquivos

with open("arquivos/arquivo1.txt", "w+") as f:

```
f.write("Teste Linha\n")  
f.write("Teste Linha\n")  
f.seek(0,0)
```

O retorno de uma leitura vai ser em lista. Vamos converter para string

```
grava = str(f.read())
```


TRABALHANDO COM ARQUIVOS

Gravar dados de um arquivo para outro

with open("arquivos/arquivo2.txt", "w+") as f2:

```
f2.write(grava)  
f2.seek(0,0)  
print(f2.read())
```

EXERCÍCIO DE FIXAÇÃO

Criar um arquivo txt na pasta. Adicione algumas linhas no arquivo. E faça um script python leia esse arquivo.

Abra o arquivo criado e sobrescreva o arquivo e adicione 2 novas linhas escritas no arquivo.

No mesmo arquivo criado adicione uma nova linha sem sobrescrever o conteúdo do mesmo.

Copie o conteúdo do arquivo criado anteriormente para um novo arquivo. Utilize o gerenciado de contexto de arquivos para isso.

TRABALHANDO COM ARQUIVOS

Manipulando Arquivos formatados em CSV.

Importar pacote CSV

```
import csv
```

Criando um arquivos CSV. Criamos utilizando as funções writer e writerow

with open ("arquivos/nomes.csv","w", newline="") as fcsv:

```
    escrever = csv.writer(fcsv, delimiter = ',')
```

```
    escrever.writerow(("Nome","Sobrenome","Idade"))
```

```
    escrever.writerow(("João","Ricardo",35))
```

```
    escrever.writerow(("Juca","Souza",23,))
```

```
    escrever.writerow(("Alberto","Cunha",54))
```

TRABALHANDO COM ARQUIVOS

Ler o Arquivo CSV Criado com o método `csv.reader`

```
with open ("arquivos/nomes.csv","r") as fcsv:  
    ler = csv.reader(fcsv)
```

Transformar os valores do CSV em uma lista

```
lista1 = list(ler)
```

```
print(lista1)
```

Iterar a lista criada do CSV

```
for csv in lista1:  
    print(csv)
```

TRABALHANDO COM ARQUIVOS

Transformar a saída em um dicionário. Usamos o método DictReader

```
with open ("arquivos/nomes.csv","r") as fcsv:  
    ler_dict = csv.DictReader(fcsv)
```

Iterando os valores

```
for dic in ler_dict:  
    print(dic)
```

Iterando valores de Chaves

```
for dic in ler_dict:  
    print(dic["Nome"])
```

TRABALHANDO COM ARQUIVOS

Lendo um arquivo CSV e gravando uma lista

```
with open ("arquivos/arquivo1.csv","r") as arqu1:  
    arqu1 = csv.reader(arqu1)
```

Iterar os valores do CSV

```
for l in arqu1:  
    print(l)
```

Criar a Lista

```
lista2 = list(arqu1)
```

```
print(lista2)
```


EXERCÍCIO DE FIXAÇÃO

Crie uma arquivo CSV com cabeçalho e lista de preço de produtos.

Leia o arquivo CSV criado e itere os valores do mesmo com for e depois converta ele em uma lista.

Leia o arquivo disponibilizado chamado arquivo_ex.csv transforme o mesmo em uma lista e itere o valor da mesma.

Após isso transforme o mesmo em um dicionário e itere os valores de chave chamado "SaleNumber".