Iteráveis, Iteradores e Geradores

quinta-feira, 25 de agosto de 2022 14:44

Existem vários objetos iteráveis dentro de python alguns exemplos são listas,strings e outros.

quando usamos o laço for em uma lista, ele transforma essa lista em um iterador para atribuir o valor a variável que esta sendo jogada.

Exemplo:

usamos a função hasattr(objeto,"**__iter__**") -> para saber se o objeto é iteravel

```
lista = [0,1,2,3,4]

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

lista é um objeto iteravel, usando o for lista usa um iterador para a variavel v.

para saber se o objeto é um iterador usamos hasattr(objeto,"**__next__**")

para transformar um objeto em um iterador usamos assim:

fazemos um cast com iter()

```
lista = [0,1,2,3,4]
lista = iter(lista)

print(hasattr(lista,"__next__"))
```

então para ver cada valor na lista usamos a função next

```
lista = [0,1,2,3,4]
lista = iter(lista)

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

Resultado:

```
0
1
2
3
4

Process finished with exit code 0
```

Geradores são criados quando queremos criar algo que irá usar muito de nossa memória e provavelmente o programa irá ficar lento

As listas quando criadas retem cada valor acumulando, porem se criadas com geradores, os geradores pega apenas 1 valor de cada vez, utilizando um valor específico da memória extremamente menor que o das listas sem gerador.

Existem 2 metodos para criar um gerador:

```
def gerador():
    for n in range(100):
        yield n

g = gerador()
```

ou podemos criar usando list compreesion:

```python
import sys

lista = [valor for valor in range(1000)]
lista2 = (valor for valor in range(1000))

print(sys.getsizeof(lista))
print(sys.getsizeof(lista2))
```

e para obter os valores podemos ou usar um for

quando usamos o next consome os valores do iterador, porém se usado apenas até uma parte o restante será pego no for se usado.

Zip ,Zip_longest

Quando usamos a função zip ela cria um gerador parecido com o yield, unindo listas, e quando usamos o next ou o for mostra os valores unidos.

exemplo:

```python
lista1 = [1,2,3,5,4,6]
lista2 = [1,2,5,6,4,8]
lista3 = [1,2,4,5]

lista4 = zip(lista1,lista2,lista3)

for v in lista4:
    print(v,end=" ")
```

unindo indice com indice e unindo apenas até a menor lista.

Resultado:

```
(1, 1, 1) (2, 2, 2) (3, 5, 4) (5, 6, 5)
Process finished with exit code 0
```

ou então ao inves do for usar a função next

```python
lista1 = [1,2,3,5,4,6]
lista2 = [1,2,5,6,4,8]
lista3 = [1,2,4,5]

lista4 = zip(lista1,lista2,lista3)

print(next(lista4))
print(next(lista4))
print(next(lista4))
print(next(lista4))
```

podemos também inverter as ordens das listas no zip para por exemplo criar dicionários.

```python
lista = ['nome','idade','tamanho']
lista2 = ['geronimo',22,1.80]

lista4 = dict([(v,y) for v,y in zip(lista2,lista)])

print(lista4)
```

ou assim:

```python
lista = ['nome','idade','tamanho']
lista2 = ['geronimo',22,1.80]

lista4 = dict([(v,y) for v,y in zip(lista,lista2)])

print(lista4)
```

com a função zip o python apenas vai unir até o tamanho da menor lista, ou seja se a lista menor tiver 4 itens e a maior 7, so vai criar um gerador até o 4 item.

O outro método para unir as listas ou iteráveis é usando o modulo zip_longest da biblioteca itertools, ao contrário do zip ela uni os iteraveis até o maior porém se faltar algum valor na menor lista usamos a função fillvalue = valorquedesejar

```python
import itertools
lista = {'nome': 'geronimo','idade':17,'comprando':'material'}
lista2 = {'paciente':'novopaciente','droga':'rupinou'}


lista4 = itertools.zip_longest(lista,lista2,fillvalue=0)
for v in lista4:
    print(v)
```

Count()

A função count é da biblioteca itertools e é um iterador no python, gerando números.

```python
from itertools import count

contador = count()

for v in contador:
    print(v)
    if v ==10:
        break
```

Existem funções nomeadas dentro da função count também

```python
contador = count(start = 0, step = 2 )
```

Start é onde inicia, step é o passo

o passo também pode ser ponto flutuante

```python
contador = count(start = 0, step = 0.5 )

for v in contador:
    print(v)
    if v ==10:
        break
```

Resultado:

```
0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
```

Podemos então criar um contador em uma variável e uma lista, usando para adicionar indices usando o zip para iterar entre essas 2.

Exemplo:

```
from itertools import count

contador = count()

lista = ["Geronimo","Eunice","Stanley"]
lista = ((x,y) for x,y in zip(contador,lista))

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

Combinations,permutations e Product

As funções da biblioteca itertools são usadas para fazer combinações.

combinations uni valores de uma iteravel fazendo combinações sem importar a ordem, ou seja valores não vão se repetir.

```
from itertools import combinations,permutations,product

lista = ['geronimo','morais','lima','neto']

for v in combinations(lista,2):
    print(v)
```

o número 2 são as combinações, porém no combinations valores exemplo geronimo morais não se repete no inverso morais geronimo, se quisermos focar nisso usamos o permutations

```
from itertools import combinations,permutations,product

lista = ['geronimo','morais','lima','neto']

for v in permutations(lista,2):
    print(v)
```

Dessa forma os valores como morais geronimo e geronimo morais irão aparecer na lista, porém também existe outro método que équando precisamos também repetir o mesmo valor, exemplo: geronimo geronimo morais morais, usamos a função product

```
from itertools import combinations,permutations,product

lista = ['geronimo','morais','lima','neto']

for v in product(lista,repeat = 2):
    print(v)
```

Diferente dos outros 2 combinations e permutations precisamos usar o repeat para saber o número de combinações.

Groupby

A função groupby funciona para agrupar valores de um dicionário porém para isso precisamos talvez ordenar usando uma função lambda, após ordenar uma lista ou dicionário usamos ela.

Exemplo:

```python
lista = [
    {'nome': "Geronimo",'nota':10},
    {'nome': "Eunice",'nota':8},
    {'nome': "Stanley",'nota':7},
    {'nome': "Robson",'nota':2},
    {'nome': "Moacyr",'nota':7},
    {'nome': "Rosemary",'nota':6},
    {'nome': "Morais",'nota':8},
    {'nome': "Monica",'nota':10},
    {'nome': "Carol",'nota':7},
    {'nome': "Joaquim",'nota':9},
    {'nome': "Mora",'nota':5},
]
```

```python
from itertools import groupby,tee

lista = [...]

#Usando função Lambda para ordenar os dicionários pelas notas.
ordem = lambda item : item['nota']
lista.sort(key=ordem,reverse=True)

#Usando a função Groupby para agrupar os valores
lista = groupby(lista,ordem)

#Iterando sobre o iterador groupby
for nomes,notas in lista:

    #Fazendo uma cópia dos valores do iterador
    nota1,nota2 = tee(notas)
    print(f"Alunos com nota: {nomes}")
    #iterando sobre o iterador
    for valor in nota1:
        print(f"{valor}")
    quantidade = len(list(nota2))
    print(f"Os alunos com nota {valor['nota']} foram: {quantidade}")
    print()
```

Map()

A função map como próprio nome diz faz um mapeamento, recebendo uma função seja lambda ou ou função criada como parametro para fazer modificação em listas ou dicionários(principalmente)

A função map tem uma função parecida com o list compreension, em alguns casos iremos ver o map em outros o List compreension

exemplo:

```python
lista1 = [
    {'nome':'geronimo','idade':17},
    {'nome':'geronimo','idade':25},
    {'nome':'geronimo','idade':20},
    {'nome':'geronimo','idade':22},
    {'nome':'geronimo','idade':21},
    {'nome':'geronimo','idade':19},
    {'nome':'geronimo','idade':18},
    {'nome':'geronimo','idade':15},
    {'nome':'geronimo','idade':14},
    {'nome':'geronimo','idade':12}
]

idadex2 = map(lambda i_: i['idade']*2,lista1)

for nova_idade in idadex2:
    print(nova_idade)
```

A função filter dos iteradores funciona dessa forma filtrando alguma condição, parecida com a função map recebendo uma função ou uma lambda.
Exemplo:

```python
lista1 = [
    {'nome':'geronimo','idade':17},
    {'nome':'geronimo','idade':25},
    {'nome':'geronimo','idade':20},
    {'nome':'geronimo','idade':22},
    {'nome':'geronimo','idade':21},
    {'nome':'geronimo','idade':19},
    {'nome':'geronimo','idade':18},
    {'nome':'geronimo','idade':15},
    {'nome':'geronimo','idade':14},
    {'nome':'geronimo','idade':12}
]

nova_lista = filter(lambda x:x['idade']>18 ,lista1)

for nova_idade in nova_lista:
    print(nova_idade)
```

podemos também criar uma função e usar dentro do filter para pegar algo porém devemos usar um valor booleano no retorno.

```python
]
def encontre(valor):
    if valor['idade'] > 18:
        return True

nova_lista = filter(lambda x:x['idade']>18 ,lista1)

for nova_idade in nova_lista:
    print(nova_idade)
```

Reduce()

A função reduce vem de outra biblioteca que é a functools e serve para acumular valores