

Programação Estruturada Linguagem Python

Professor Adjenor Cristiano
Queiroz
FAPAM



Termos de Uso

Todo conteúdo desta apresentação é para uso exclusivo em sala de aula. Este material não poderá ser reproduzido, copiado, publicado, transmitido, vendido ou distribuído e nem sequer modificado, seja ou não para fins comerciais, sem prévia autorização expressa e formal da empresa.

Aula 5



HORA DA
CORREÇÃO!



Dicas do Programador



Utilize o comando `upper()` para converter strings em maiúscula:

```
pais = "Brasil"  
print(pais.upper())  
#BRASIL
```

Utilize o comando `lower()` para converter strings em minúscula:

```
pais = "ESPANHA"  
print(pais.lower())  
#espanha
```



EXERCÍCIOS:

- 1 - Desenvolva um software que receba do usuário uma quantidade e imprima na tela todos os números pares de 0 até o número digitado.
- 2 - Altere o software do exercício anterior para que o sistema receba também um tipo (P para Pares e I para Impares) e imprima apenas os números de acordo com a escolha.
- 3 - Valide a opção digitada, caso o usuário digite um valor diferente se P ou I, solicite a ele que digite novamente. DICA: Utilize repetições aninhadas.

```
1 resp=""
2 cont=0
3 while((resp!="s") or (resp!="S")):
4
5     while((resp!="P") and (resp!="I") and (resp!="S")):
6         resp = input("Escolha o Tipo [P - Par / I - Impar / S - Sair]\n").upper()
7         if((resp!="P") and (resp!="I") and (resp!="S")):
8             print("Valor Inválido")
9         elif(resp!="S"):
10             qtd=int(input("Digite a Quantidade\n"))
11
12     if(resp=="S"):
13         break
14
15     while(cont<=qtd):
16         if((resp=="P") and (cont%2==0)):
17             print(cont)
18         elif((resp=="I") and (cont%2==1)):
19             print(cont)
20         cont+=1
21     resp=""
22     cont=0
23 print("Obrigado por Utilizar nosso Software!")
```



EXERCÍCIOS:

4 - Escreva um programa que leia um número e verifique se é ou não um número primo. Para fazer essa verificação, calcule o resto da divisão do número por 2 e depois por todos os números ímpares até o número lido. Se o resto de uma dessas divisões for igual a zero, o número não é primo. Observe que 0 e 1 não são primos e que 2 é o único número primo que é par.


```
1 nun = int(input("Digite um Número\n"))
2 cont = 3
3 resp = 1
4 if(nun==2):
5     print("O número %d é primo"%nun)
6 elif((nun%2==0)):
7     print("O número %d não é primo"%nun)
8 else:
9     while(cont<nun):
10         if(nun%cont==0):
11             print("O número %d não é primo"%nun)
12             resp = 0
13             break
14         cont+=1
15     if(resp):
16         print("O número %d é primo"%nun)
```



EXERCÍCIOS:

5 - Desenvolva um software de restaurante que apresente ao menos 4 tipos de produtos e a opção de finalizar o pedido. O software deve perguntar ao usuário se ele deseja adicionar outro produto. Ao final ele deve mostrar na tela todos os produtos que o cliente comprou e o total do pedido.

DICA: Utilize uma variável String para armazenar os produtos escolhidos e uma variável float para armazenar o total do pedido. Apresente o total formatado para Real.

```
1 opcao = 0
2 total = 0
3 pedido = ""
4 while (opcao!=5) :
5     menu = "Menu:\n1 - Coxinha = R$ 6,00 \n"
6     menu += "2 - Pão de Queijo = R$ 4,00"
7     menu += "\n3 - Pastel = R$ 4,50"
8     menu += "\n4 - Empada = R$5,00 "
9     menu += "\n5 - Encerrar Pedido\n"
10    opcao = int(input(menu))
```


```
10 opcao = int(input(menu))
11 if(opcao==1):
12     total+=6
13     pedido+="1 - Coxinha \n"
14 elif(opcao==2):
15     total+=4
16     pedido += "1 - Pão de Queijo = R$ 4,00 \n"
17 elif(opcao==3):
18     total+=4.5
19     pedido += "1 - Pastel = R$ 4,50 \n"
20 elif(opcao==4):
21     total+=5
22     pedido += "1 - Empada = R$ 5,00 \n"
23 elif(opcao==5):
24     break
25 else:
26     print("Opção Inválida!")
27 print("Deseja acrescentar algo mais?")
```

```
28
29 totalReal = str("%.2f"%total).replace(".", ",")
30 print("Seu pedido: \n%s"%pedido)
31 print("O total do seu pedido é R$ %s"%totalReal)
```


```
1 opcao = 0
2 total = 0
3 pedido = ""
4 while(opcao!=5):
5     menu = "Menu:\n1 - Coxinha = R$ 6,00 \n"
6     menu += "2 - Pão de Queijo = R$ 4,00"
7     menu += "\n3 - Pastel = R$ 4,50"
8     menu += "\n4 - Empada = R$5,00 "
9     menu += "\n5 - Encerrar Pedido\n"
10    opcao = int(input(menu))
11    if(opcao==1):
12        total+=6
13        pedido+="1 - Coxinha \n"
14    elif(opcao==2):
15        total+=4
16        pedido += "1 - Pão de Queijo = R$ 4,00 \n"
17    elif(opcao==3):
18        total+=4.5
19        pedido += "1 - Pastel = R$ 4,50 \n"
20    elif(opcao==4):
21        total+=5
22        pedido += "1 - Empada = R$ 5,00 \n"
23    elif(opcao==5):
24        break
25    else:
26        print("Opção Inválida!")
27        print("Deseja acrescentar algo mais?")
28
29 totalReal = str("%.2f"%total).replace(".", ",")
30 print("Seu pedido: \n%s"%pedido)
31 print("O total do seu pedido é R$ %s"%totalReal)
```

Introdução a Listas (Arrays)






Listas são um tipo de variável que permite o armazenamento de vários valores, acessados por um índice. Uma lista pode conter zero ou mais elementos de um mesmo tipo ou de tipos diversos, podendo inclusive conter outras listas. O tamanho de uma lista é igual à quantidade de elementos que ela contém.



Podemos imaginar uma lista como um edifício de apartamentos, onde o térreo é o andar zero, o primeiro andar é o andar 1 e assim por diante. O índice é utilizado para especificarmos o “apartamento” onde guardaremos nossos dados.



Em um prédio de seis andares, teremos números de andar variando entre 0 e 5.

Se chamarmos nosso prédio de `P`, teremos `P[0]` como o endereço do térreo, `P[1]` como endereço do primeiro andar, continuando assim até `P[5]`. Em Python, `P` seria o nome da lista; e o número entre colchetes, o índice.

Listas são mais flexíveis que prédios e podem crescer ou diminuir com o tempo.

Vamos criar uma lista em Python.



Criar uma lista vazia:

`L= []`

Essa linha cria uma lista chamada L com zero elemento, ou seja, uma lista vazia.

Os colchetes (`[]`) após o símbolo de igualdade servem para indicar que L é uma lista.



Criar uma lista vazia:

`L= []`

Essa linha cria uma lista chamada L com zero elemento, ou seja, uma lista vazia.

Os colchetes (`[]`) após o símbolo de igualdade servem para indicar que L é uma lista.



Criar Uma lista com três elementos:

$Z = [15, 8, 9]$

A lista Z foi criada com três elementos: 15, 8 e 9. Dizemos que o tamanho da lista Z é 3. Como o primeiro elemento tem índice 0, temos que o último elemento é $Z[2]$.



► Listagem 6.3 – Acesso a uma lista

```
>>> Z=[15,8,9]
```

```
>>> Z[0]
```


```
15
```

```
>>> Z[1]
```

```
8
```


```
>>> Z[2]
```

```
9
```



Utilizando o nome da lista e um índice, podemos mudar o conteúdo de um elemento:

```
>>> Z=[15,8,9]
>>> Z[0]
15
>>> Z[0]=7
>>> Z[0]
7
>>> Z
[7, 8, 9]
```




Quando criamos a lista Z , o primeiro elemento era o número 15. Por isso, $Z[0]$ era 15. Quando executamos $Z[0]=7$, alteramos o conteúdo do primeiro elemento para 7. Isso pode ser verificado quando pedimos para exibir Z , agora com 7, 8 e 9 como elementos.

Vejamos um exemplo onde um aluno tem cinco notas e no qual desejamos calcular sua média aritmética.




EXEMPLO - Média

```
notas=[6,7,5,8,9] ❶  
soma=0  
x=0  
while x<5: ❷  
    soma += notas[x] ❸  
    x+=1 ❹  
print("Média: %5.2f" % (soma/x))
```



Criamos a lista de notas em ❶. Em ❷, criamos a estrutura de repetição para variar o valor de x e continuar enquanto este for menor que 5. **Lembre-se de que uma lista de cinco elementos contém índices de 0 a 4.** Por isso inicializamos `x=0` na linha anterior. Em ❸, adicionamos o valor de `notas[0]` à soma e depois `notas[1]`, `notas[2]`, `notas[3]` e `notas[4]`, um elemento a cada repetição.




Para isso, utilizamos o valor de x como índice e o incrementamos de 1 em ④. Uma grande vantagem desse programa foi que não precisamos declarar cinco variáveis para guardar as cinco notas. Todas as notas foram armazenadas na lista, utilizando um índice para identificar ou acessar cada valor.



Trabalhando com índices:

Vejamos outro exemplo: um programa que lê cinco números, armazena-os em uma lista e depois solicita que o usuário escolha um número a mostrar. O objetivo é, por exemplo, ler 15, 12, 5, 7 e 9 e armazená-los na lista.



Depois, se o usuário digitar 2, ele imprimirá o segundo número digitado, 3, o terceiro, e assim sucessivamente.

Observe que o índice do primeiro número é 0, e não 1: essa pequena conversão será feita no programa da listagem 6.7

► Listagem 6.7 – Apresentação de números

```
números=[0,0,0,0,0]
```

```
x=0
```

```
while x<5:
```

```
    números[x]=int(input("Número %d:" % (x+1))) ❶
```

```
    x+=1
```

```
while True:
```

```
    escolhido=int(input("Que posição você quer imprimir (0 para sair): "))
```

```
    if escolhido == 0:
```

```
        break
```

```
    print("Você escolheu o número: %d" % (números[escolhido-1])) ❷
```



Cópia e fatiamento de listas:

Embora listas em Python sejam um recurso muito poderoso, todo poder traz responsabilidades. Um dos efeitos colaterais de listas aparece quando tentamos fazer cópias. Vejamos um teste no interpretador na listagem 6.8



► Listagem 6.8 – Tentativa de copiar listas

```
>>> L=[1,2,3,4,5]
```

```
>>> V=L
```

```
>>> L
```

```
[1, 2, 3, 4, 5]
```

```
>>> V
```

```
[1, 2, 3, 4, 5]
```


```
>>> V[0]=6
```

```
>>> V
```

```
[6, 2, 3, 4, 5]
```

```
>>> L
```

```
[6, 2, 3, 4, 5]
```

Veja que, ao modificarmos `V`, modificamos também o conteúdo de `L`. Isso porque uma lista em Python é um objeto e, quando atribuímos um objeto a outro, estamos apenas copiando a mesma referência da lista, e não seus dados em si. Nesse caso, `V` funciona como um apelido de `L`, ou seja, `V` e `L` são a mesma lista. Vejamos o que acontece no gráfico da figura 6.1.

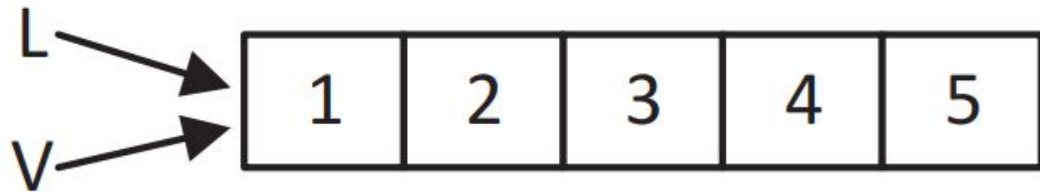




Figura 6.1 – Duas variáveis referenciando a mesma lista.

Quando modificamos $V[0]$, estamos modificando o mesmo valor de $L[0]$, pois ambos são referências, ou apelidos para a mesma lista na memória.



Dependendo da aplicação, esse efeito pode ser desejado ou não. Para criar uma cópia independente de uma lista, utilizaremos outra sintaxe. Vejamos o resultado das operações da listagem 6.9.



► Listagem 6.9 – Cópia de listas

```
>>> L=[1,2,3,4,5]
```

```
>>> V=L[:]
```

```
>>> V[0]=6
```

```
>>> L
```

```
[1, 2, 3, 4, 5]
```

```
>>> V
```

```
[6, 2, 3, 4, 5]
```

Ao escrevermos `L[:]`, estamos nos referindo a uma nova cópia de `L`. Assim `L` e `V` se referem a áreas diferentes na memória, permitindo alterá-las de forma independente, como na figura 6.2.

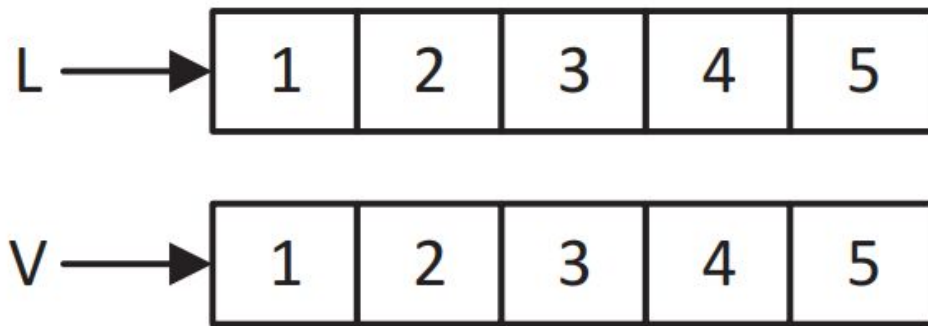



Figura 6.2 – Duas variáveis referenciando duas listas.



Podemos também fatiar uma lista, da mesma forma que fizemos com strings. Vejamos alguns exemplos:

```
>>> L=[1,2,3,4,5]
>>> L[0:5]
[1, 2, 3, 4, 5]
>>> L[3:]
[4, 5]
>>> L[:3]
[1, 2, 3]
>>> L[-1]
5
>>> L[1]
2
```

Tamanho de listas:

Podemos usar a função `len` com listas. O valor retornado é igual ao número de elementos da lista.

```
>>> L=[12,9,5]
```

```
>>> len(L)
```

```
3
```

```
>>> V=[]
```

```
>>> len(V)
```

```
0
```



Tamanho de listas:

Podemos usar a função `len` para interagir com a nossa lista no `while`.

```
L=[1,2,3]
x=0
while x < len(L):
    print(L[x])
    x+=1
```




Adição de elementos:

Uma das principais vantagens de trabalharmos com listas é poder adicionar novos elementos durante a execução do programa.

Para adicionar um elemento ao fim da lista, utilizaremos o método `append`.

Em Python, chamamos um método escrevendo o nome dele após o nome do objeto. Como listas são objetos, sendo `L` a lista, teremos `L.append(valor)`.



► Listagem 6.14 – Adição de elementos à lista

```
>>> L=[]
>>> L.append("a")
>>> L
['a']
>>> L.append("b")
>>> L
['a', 'b']
>>> L.append("c")
>>> L
['a', 'b', 'c']
>>> len(L)
```

Adição de elementos:

Outra forma de adicionarmos elementos a uma lista é com adição de listas.

```
>>> L=[]
>>> L+= [1]
>>> L
[1]
>>> L+= [2]
>>> L
[1, 2]
>>> L+= [3, 4, 5]
>>> L
[1, 2, 3, 4, 5]
```




Adição de elementos:

Quando adicionamos apenas um elemento, tanto `L.append(1)` quanto `L+[1]` produzem o mesmo resultado. Perceba que em `L+[1]` escrevemos o elemento a adicionar dentro de uma lista (`[1]`), e que, em `append`, apenas `1`. Isso porque, quando adicionamos uma lista a outra, o interpretador executa um método chamado `extend` que adiciona os elementos de uma lista a outra.

```
>>> L=[]
>>> L+= [1]
>>> L
[1]
>>> L+= [2]
>>> L
[1, 2]
>>> L+= [3, 4, 5]
>>> L
[1, 2, 3, 4, 5]
```

```
>>> L=[]
>>> L.append(1)
>>> L
[1]
>>> L.append(2)
>>> L
[1, 2]
>>> L.append([3, 4, 5])
>>> L
[1, 2, [3, 4, 5]]
```



```
>>> L=["a"]
>>> L.append("b")
>>> L
['a', 'b']
>>> L.extend(["c"])
>>> L
['a', 'b', 'c']
>>> L.append(["d","e"])
>>> L
['a', 'b', 'c', ['d', 'e']]
>>> L.extend(["f","g","h"])
>>> L
['a', 'b', 'c', ['d', 'e'], 'f', 'g', 'h']
```



Adição de elementos:

O método `extend` sequer aceita parâmetros que não sejam listas. Se você utilizar o método `append` com uma lista como parâmetro, em vez de adicionar os elementos no fim da lista, `append` adiciona a lista inteira, mas como apenas um novo elemento. Teremos então listas dentro de listas (Listagem 6.18)

Adição de elementos:

► Listagem 6.18 – Adição de elementos e listas com append

```
>>> L=["a"]
>>> L.append(["b"])
>>> L.append(["c","d"])
>>> len(L)
3
>>> L[1]
['b']
>>> L[2]
['c', 'd']
>>> len(L[2])
2
>>> L[2][1]
'd'
```


Remoção de elementos da lista:

Como o tamanho da lista pode variar, permitindo a adição de novos elementos, podemos também retirar alguns elementos da lista, ou mesmo todos eles. Para isso, utilizaremos a instrução **del**:

```
>>> L=["a","b","c"]
>>> del L[1]
>>> L
['a', 'c']
>>> del L[0]
>>> L
['c']
```

Remoção de elementos da lista:

É importante notar que o elemento excluído não ocupa mais lugar na lista, fazendo com que os índices sejam reorganizados, ou melhor, que passem a ser calculados sem esse elemento. Podemos também apagar fatias inteiras de uma só vez

► Listagem 6.20 – Remoção de fatias

```
>>> L=list(range(101))  
>>> del L[1:99]  
>>> L  
[0, 99, 100]
```

EXERCÍCIOS





EXEMPLO I - Média

```
notas=[6,7,5,8,9] ❶  
soma=0  
x=0  
while x<5: ❷  
    soma += notas[x] ❸  
    x+=1 ❹  
print("Média: %5.2f" % (soma/x))
```



EXERCÍCIOS:

1 - Altere o software do Exemplo I de modo que ele agora receba as notas do usuário e calcule a média e mostre na tela um resumo das notas.

DICA: inicie a lista de notas zerada:

```
notas=[0,0,0,0,0]
```

Utilize duas estruturas while



Exemplo da Saída do Software:

```
Digite a Nota 0:10
Digite a Nota 1:15
Digite a Nota 2:18
Digite a Nota 3:20
Digite a Nota 4:15
*****
Resumo das Notas:
*****
Nota 0:  10.00
Nota 1:  15.00
Nota 2:  18.00
Nota 3:  20.00
Nota 4:  15.00
Média: 15.60
```



EXERCÍCIOS:

2 - Altere o software do restaurante desenvolvido na semana passada de modo que a cada interação o software mostrará para o cliente todos os produtos comprados e o total da compra. O software deve permitir a ele que ele remova produtos ou adicione mais produtos.

DICA: Utilize ao menos duas listas, uma para os produtos e outra para os valores.



Exercícios:

Exercício 6.2 Faça um programa que leia duas listas e que gere uma terceira com os elementos das duas primeiras.

Exercício 6.3 Faça um programa que percorra duas listas e gere uma terceira sem elementos repetidos.