Já fizemos alguns programas para ler 2 ou 3 notas e calcular a média. Inclusive já fomos além e aprendemos a verificar se o aluno passou ou não. Vamos rever um exemplo desses:

nota1 = float(input('Digite a primeira nota: '))

nota2 = float(input('Digite a segunda nota: '))

media = (nota1 + nota2)/2

print(media)

Simples, certo? Mas e se a regra da escola mudasse, e agora cada professor precisasse aplicar 4 provas? Modificaríamos nosso programa:

nota1 = float(input('Digite a primeira nota: '))

nota2 = float(input('Digite a segunda nota: '))

nota3 = float(input('Digite a terceira nota: '))

nota4 = float(input('Digite a quarta nota: '))

media = (nota1 + nota2 + nota3 + nota4)/4

print(media)

Até aqui tudo bem. Mas e se o objetivo fosse testar o quanto o professor consegue ensinar? Para isso, poderíamos calcular a média das médias de todos os alunos do professor. Mas e se o professor trabalha em uma faculdade muito grande e suas turmas têm 80 alunos?

aluno1 = float(input('Digite a nota do aluno 1: '))

aluno2 = float(input('Digite a nota do aluno 2: '))

aluno3 = float(input('Digite a nota do aluno 3: '))

aluno4 = float(input('Digite a nota do aluno 4: '))

aluno5 = float(input('Digite a nota do aluno 5: '))

aluno6 = float(input('Digite a nota do aluno 6: '))

aluno7 = float(input('Digite a nota do aluno 7: '))

aluno8 = float(input('Digite a nota do aluno 8: '))

aluno9 = float(input('Digite a nota do aluno 9: '))

aluno10 = float(input('Digite a nota do aluno 10: '))

# ...

```

Nota do programador: eu me demito.

Não ganho bem o suficiente para ISSO!

```

Para trabalhar com poucos valores, é fácil e conveniente criar uma variável para cada valor e realizar operações individualmente sobre cada uma. Porém, dizemos que esse tipo de solução não é *escalável*: o programa não está preparado para lidar com variações no tamanho da base de dados, e modificá-lo para comportá-las pode ser difícil, trabalhoso ou mesmo inviável.

Imagine se para cada novo perfil em uma rede social o estagiário precisasse criar uma variável nova para o nome, uma para o e-mail, uma para a data de nascimento, e assim sucessivamente... E depois ainda precisasse de linhas novas de código para ler cada um desses valores do novo usuário!

**1. Listas**

É aí que entram as listas. Listas são *coleções de objetos* em Python. Falando de maneira simplificada, são variáveis que comportam diversos valores ao mesmo tempo. Vejamos alguns jeitos de criar listas em Python:

primeira\_lista = [] # cria uma lista vazia

segunda\_lista = list() # cria uma lista vazia

terceira\_lista = [1, 3.14, 5, 7, 9, 'onze'] # lista com valores

Note que podemos misturar tipos de dados. A terceira\_lista possui 4 **int**, um **float** e uma **str**.

Bom, e agora, como fazemos para acessar cada valor? Podemos imaginar a lista da seguinte maneira: imagine que ao invés de ter uma caixa para guardar cada item, temos uma cômoda com várias gavetas. Cada item está em uma gaveta. Não estamos acostumados a dizer que algo está na terceira gaveta do armário? A ideia é a mesma: a lista é uma coleção **indexada**, ou seja, podemos acessar cada elemento através de **índices**, que são números indicando a posição. A indexação é automática e começa a partir do zero:

| **elemento** | **1** | **3.14** | **5** | **7** | **9** | **11** |
| --- | --- | --- | --- | --- | --- | --- |
| índice | 0 | 1 | 2 | 3 | 4 | 5 |

Portanto, para acessar o elemento "7" da nossa lista, utilizaríamos o índice 3. Informamos o índice entre colchetes:

terceira\_lista = [1, 3.14, 5, 7, 9, 'onze'] # lista com valores

print(terceira\_lista[3])

A lista é mutável. Isso significa que podemos modificar os valores já existentes:

terceira\_lista = [1, 3.14, 5, 7, 9, 'onze'] # lista com valores

terceira\_lista[3] = 'sete' # troca 7 por 'sete' na lista

print(terceira\_lista)

É possível utilizar índices negativos. lista[-1] pega o último elemento, lista[-2] o penúltimo, e assim sucessivamente. Mas **não** é possível acessar índices iguais ou superiores ao tamanho da lista. A tentativa de acessar um índice inexistente resultará em erro.

**2. Quebrando listas**

É possível pegar subconjuntos de nossas listas utilizando o conceito de *slices*. Ao invés de passar apenas 1 valor entre colchetes (o índice desejado), podemos passar faixas de valores. Veja o exemplo abaixo:

impares = [1, 3, 5, 7, 9, 11, 13, 15, 17]

meio = impares[3:6]

print(meio) # resultado na tela: [7, 9, 11]

O primeiro valor é o índice inicial da *sublista* a ser gerada, e o segundo é o índice final (exclusivo). Podemos omitir um desses valores para indicar que será desde o início ou até o final:

impares = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

primeira\_metade = impares[:5]

segunda\_metade = impares[5:]

print(primeira\_metade) # resultado: [1, 3, 5, 7, 9]

print(segunda\_metade) # resultado: [11, 13, 15, 17, 19]

Além de índices inicial e final, podemos também passar um *passo* para os índices. Veja o exemplo abaixo:

numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

# múltiplos de 3 abaixo de 10:

mult3\_sub10 = numeros[3:10:3]

print(mult3\_sub10) # resultado: [3, 6, 9]

**Atenção:** quando nós atribuímos uma lista a outra variável, a lista **não** é copiada. Observe o exemplo abaixo:

lista1 = [1, 3, 5]

lista2 = lista1

lista2.append(7)

print(lista1) # resultado: [1, 3, 5, 7]

Modificações aplicadas em lista2 também afetarão a lista1. Isso ocorre porque não foi criada uma lista. O Python apenas fez com que ambas as variáveis (lista1 e lista2) referenciassem a mesma estrutura na memória. Quando utilizamos *slices*, isso não ocorre. O Python cria uma lista contendo os valores restritos pelos índices. Sendo assim, uma estratégia fácil para copiar uma lista para outra é utilizar um *slice* indo da primeira à última posição:

lista1 = [1, 3, 5]

lista2 = lista1[:]

lista2.append(7)

print(lista1) # resultado: [1, 3, 5]

print(lista2) # resultado: [1, 3, 5, 7]

**3. Percorrendo listas**

Suponha que você queira acessar cada elemento de sua lista individualmente. Digitar todos os índices manualmente cancelaria a *escalabilidade* do programa, certo? Portanto, podemos usar um *loop* para gerar os índices:

pares = [0, 2, 4, 6, 8]

tamanho = len(pares) # calcula o tamanho da lista

# tamanho vale 5, logo índice recebe os valores 0, 1, 2, 3 e 4

for indice in range(tamanho):

print(pares[indice])

Porém, tem um jeito ainda **mais** fácil de percorrer a lista. O *for* não serve apenas para gerar sequências numéricas junto do *range*: ele serve para percorrer coleções. Portanto, podemos trocar o *range* pela própria lista:

pares = [0, 2, 4, 6, 8]

for elemento in pares:

print(elemento)

Assim como no caso das contagens, "elemento" é apenas uma variável que será criada de forma automática e poderia ter qualquer nome. Em cada repetição do *loop*, um valor diferente da lista será **copiado** para *elemento*.

**Importante:** Como os elementos são copiados, caso você modifique o valor de *elemento* você **não** irá modificar o valor na lista, e sim uma cópia dele. Além disso, como este *loop* serve especificamente para percorrer listas, se dentro dele você fizer operações que alterem o tamanho da lista (*append* ou *remove*, por exemplo), o *loop* poderá executar incorretamente, pulando ou repetindo elementos.

O for serve, primariamente, para percorrer coleções. Ou seja, para **iterar** coleções. O **range** age, na prática, como se fosse uma lista contendo os valores determinados por seu parâmetro. Dizemos que a função **range** gera um **iterável**, ou seja, um tipo especial de dado que pode ser percorrido através de um loop.

**4. Testando a existência de valores**

Existe um comando que temos visto bastante recentemente: o *in*. Ele costuma aparecer no *for* para indicar a lista ou a sequência a ser percorrida. Mas ele também possui outra utilidade.

O *in* pode ser utilizado para informar se um elemento está presente em uma lista ou não. Observe a saída do código abaixo.

linguagens = ['Python', 'JavaScript', 'C#', 'Java']

existe\_html = 'HTML' in linguagens

existe\_java = 'Java' in linguagens

print('HTML:', existe\_html) # HTML: False

print('Java:', existe\_java) # Java: True

Normalmente utilizamos esse comando junto de um *if* quando precisamos checar se um elemento existe:

linguagem\_desejada = input('Digite a linguagem que você gostaria de aprender: ')

linguagens = ['Python', 'JavaScript', 'C#', 'Java']

if linguagem\_desejada in linguagens:

print('Faça o curso conosco! :)')

else:

print('Não temos esse curso disponível no momento :(')