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

Módulo #2 - Parte 1

Até o momento aprendemos a armazenar valores em uma variável, seja ela do tipo ***string****,* ***float****,* ***int*** *ou* ***boolean****. As variáveis são utilizadas quando precisamos guardar apenas um valor por vez. Porém, em alguns problemas é necessário armazenar vários dados ao mesmo tempo, por isso, precisamos usar outras estruturas de dados que veremos a seguir.*

Vamos entender isso melhor?

Estrutura de Dados

Suponha um sistema de uma escola em que para um aluno passar de ano ele precisa ter nota maior ou igual a 60, no máximo 10 faltas e não ter nenhuma suspensão.

Vamos então criar um programa que verifique se os alunos desta escola passaram de ano ou não?

Para isso vamos solicitar que seja informado o nome do aluno, a nota final, quantidade total de faltas e se ele tem alguma suspensão:

```
nome=input('Informe o nome: ')
nota=float(input('Informe a nota: '))
faltas=int(input('Informe as faltas: '))
suspensao=input('Informe S se o aluno já teve suspensão ou informe N se o aluno nunca teve suspensão: ')
```

Para passar de ano o aluno precisa ter nota>=60, no máximo 10 faltas e não ter nenhuma suspensão. Vamos criar a parte do código que analisa se o aluno passou ou não:

```
if nota>=60 and faltas<=10 and suspensao=='N':
 print('O aluno passou de ano')


else:
 print('O aluno não passou de ano')
```

Quando executamos este programa, ele faz a leitura dos dados de 1 aluno, identifica se o mesmo passou de ano ou não e o programa é encerrado. Se quisermos realizar este processo para vários alunos devemos utilizar, como vimos no módulo anterior, as estruturas de repetição. Pois elas permitem executar parte de um código ou um programa mais de uma vez. Vamos ver como seria esta implementação na prática?

*Iniciamos o **while** com o True `while True`, isso significa que o **while** vai ser executado porque sua condição é verdadeira.*

Mas se a condição é True e ela não muda, o programa entrará em loop infinito? Bom, se forçarmos a parada em algum momento não acontecerá loop infinito, mas se não fizermos isso, sim! Pois a condição nunca será falsa.

A maneira utilizada para forçar a parada será a seguinte:
*Vamos perguntar para o usuário se ele deseja informar os dados de mais algum aluno, depois, verificamos se o usuário informou que não quer continuar. Se esta condição for verdadeira, usamos a função `break` que faz a repetição **while** parar. Se o usuário quiser continuar, o **while** continua a repetição. Veja:*

```
while True:
 nome=input('Informe o nome: ')
 nota=float(input('Informe a nota: '))
 faltas=int(input('Informe as faltas: '))
 suspensao=input('Informe S se o aluno já teve suspensão ou informe
N se o aluno nunca teve suspensão: ')

 if nota>=60 and faltas<=10 and suspensao=='N':
   print('O aluno passou de ano')

 else:
   print('O aluno não passou de ano')

 opcao=input('deseja informar os dados de mais algum aluno? Digite S
se sim e N caso não.')
 if opcao=='N':
   break
```

Pronto, agora conseguimos solicitar os dados de vários alunos e verificar se todos eles foram aprovados ou não.

Observe um detalhe muito importante: *as variáveis nome, nota, faltas e suspensão serão sempre sobrescritas a cada vez que a repetição é executada.*

E se precisarmos acessar todos os dados informados? Bom, é aí que entram outras estruturas de dados.

A ***lista****, por exemplo, é uma estrutura de dados que permite armazenar uma sequência de elementos. Então, se quisermos armazenar os dados de todos os alunos, podemos criar uma* ***lista*** *com* ***todos os nomes, todas as notas****,* ***faltas*** *e* ***suspensões****. Veja:*

Primeiramente, vamos criar as ***listas****:*

```
nome=[]
nota=[]
faltas=[]
suspensao=[]
```

O interpretador entende que é uma lista por conta do símbolo ***[ ]****. As* ***listas*** *estão inicialmente vazias.*

Depois de criadas as listas, vamos preenchê-las com os dados dos alunos. Usaremos aqui a função ***append****, que é responsável por adicionar os elementos na lista. Eles serão adicionados em ordem de inserção, a começar pela* ***posição 0.***

```
while True:
 nome.append(input('Informe o nome: '))
 nota.append(float(input('Informe a nota: ')))
 faltas.append(int(input('Informe as faltas: ')))
 suspensao.append(input('Informe S se o aluno já teve suspensão ou informe N se o aluno nunca teve suspensão: '))


 opcao=input('deseja informar os dados de mais algum aluno? Digite S se sim e N caso não.')
 if opcao=='N':
  break
```

O código anterior preencheu a lista com os dados. Então, suponha que depois disso queremos realizar as seguintes análises:

a. *Quais são os nomes dos alunos reprovados?*
b. *Quais são os nomes dos alunos aprovados?*
c. *Qual foi a média das notas?*

*Podemos fazer isso acessando os dados das **listas**. Observe que os dados que estão na primeira posição de cada lista se referem ao primeiro aluno, na segunda posição ao segundo aluno e assim por diante. Observe também que já sabemos quantos alunos foram cadastrados, pois é a quantidade de elementos que tem nas **listas**.*

Vamos fazer as análises?

Primeiro, entenda que para recuperar os dados em uma lista precisamos acessar a posição em que ele se encontra, por exemplo:

- *Qual é o nome do primeiro aluno cadastrado?*

```
nome[0]
```

→ é o dado que está na lista `nome` na primeira posição, que é `0`.

> *E se eu quiser acessar todos os nomes da lista?*
> *Nesse caso, é preciso acessar cada uma das posições da lista. Perceba que nesse processo há uma repetição.*

Dessa forma, podemos utilizar uma estrutura de repetição para resolver isso. Já sabemos quantas vezes queremos repetir, pois é a quantidade de alunos cadastrados. Qual estrutura de repetição vamos utilizar?

*Você já deve ter se lembrado, usaremos o **for**. Veja:*

```
for x in range(len(nome)):

 if nota[x]>=60 and faltas[x]<=10 and suspensao[x]=='N':
  print("O aluno",nome[x], "passou de ano")

 else:
   print("O aluno",nome[x], "não passou de ano")
```

*A função **len()** identifica quantos elementos foram cadastrados e a função **range** vai criar uma lista com esta quantidade de elementos, que se inicia no 0 até a quantidade de elementos - 1 – justamente porque começamos no 0.*
*Então x, na primeira execução da repetição **for**, vale **0**; depois x vale **1,** depois **2**, até percorrer todos os elementos. Sendo assim, no código acima, analisamos se cada aluno cadastrado passou de ano ou não. Agora vamos calcular a média das notas:*

```
sum(nota)/len(nota)
```

*Uma maneira muito prática de fazer isso é usando a função **sum()** para somar todos os elementos da lista de notas e depois dividir pela quantidade de notas cadastradas. A função **len()** já conhecemos anteriormente. Note que este código foi adicionado fora da repetição, pois estas duas funções realizam as operações em toda lista.*

Existe uma outra maneira de fazer isso sem usar estas funções? Sim, e para isso precisamos somar todas as notas dentro da repetição e dividir pela quantidade de alunos.

Primeiro vamos criar dois contadores que vão controlar o somatório de todas as notas e a quantidade de alunos cadastrados.

```
somaNotas=0
quantidadeAluno=0
```

*Feito isso, dentro da repetição, vamos somar no contador de notas a nota de cada aluno percorrido. Além disso, a cada iteração do **for** vamos somar **+1** no contador `quantidadeAluno`, assim conseguimos obter ao final quantos alunos foram cadastrados:*

```
for x in range(len(nome)):

 if nota[x]>=60 and faltas[x]<=10 and suspensao[x]=='N':
  print("O aluno",nome[x], "passou de ano")

 else:
   print("O aluno",nome[x], "não passou de ano")

 somaNotas+=nota[x]
 quantidadeAluno+=1
```

Por fim, fora da repetição ***for****, vamos calcular a média das notas:*
`somaNotas/quantidadeAluno`

Veja como na programação existem mais de uma maneira de implementar a solução de um problema. Mostramos por aqui apenas dois exemplos, mas você pode criar sua própria lógica também. Porém, observe que sempre existe uma melhor solução. No primeiro exemplo não foi preciso criar os contadores, o que torna a solução mais simples de implementar e mais eficiente de executar, uma vez que tem menos linhas de código.

Uma outra estrutura de dados muito similar às listas são as ***tuplas****. A diferença entre elas é que a* ***tupla*** *é imutável, ou seja, depois de inserido os valores não é possível alterar.*

Veja o exemplo a seguir:

Vou criar uma lista chamada `listaIdade` *e uma* ***tupla*** *chamada* `tuplaIdade`*. Ambas receberão as idades* `11,5,18` *e* `90`*. Depois vou printar na tela as duas estruturas:*

```
listaIdade=[11,5,18,90]
tuplaIdade=(11,5,18,90)

print('Valores da lista : ', listaIdade)
print('Valores da tupla : ', tuplaIdade)
```

O resultado dos prints são:

```
Valores da lista :  [11, 5, 18, 90]
Valores da tupla :  (11, 5, 18, 90)
```

Observe que o interpretador sabe o que é uma ***lista*** *e o que é uma* ***tupla*** *pelo símbolo:*

[ ] - Lista
( ) - Tupla

Agora vamos modificar a idade 11, que está na primeira posição da lista e da tupla para 55 anos:

```
listaIdade[0]= 55
print('Valores da lista : ', listaIdade)
```

Vamos selecionar o elemento na posição 0 `listaIdade[0]` e atribuir o valor 55 à ela, os elementos da lista agora são:

```
Valores da lista :  [55, 5, 18, 90]
```

Vamos então tentar alterar na tupla:

```
tuplaIdade[0]= 55
print('Valores da tupla : ', tuplaIdade)
```

TypeError Traceback (most recent call last)
<ipython-input-3-3f7d2ab7a019> in <module>
----> 1 tuplaIdade[0]= 55
2 print('Valores da tupla : ', tuplaIdade)

TypeError: 'tuple' object does not support item assignment

*Uma **mensagem de erro** aparece e a **tupla não é modificada** pois ela é **imutável**.*

*No exemplo acima foi preciso criar **4 listas para armazenar os dados dos alunos**, que são: **nome**, **nota**, **faltas** e **suspensão**. Será que não existe uma maneira de armazenar tudo isso em uma única estrutura? A resposta é: **sim!***

*Para isso precisamos introduzir o conceito de **dicionários**. Esta estrutura permite armazenar uma coleção de dados que contém em sua estrutura um conjunto de chave/valor, em que cada chave individual tem um valor associado. Vamos ver na prática como seria:*

Criaremos a seguir um dicionário com os dados de um aluno. Nesta estrutura temos as chaves: nome, nota, faltas e suspensão e, cada chave está associada a um valor.

```
dados_aluno1 = {
    'Nome': 'Bruno',
    'nota': 80.78,
    'faltas': 3,
    'suspensao':'N'

}
```

Podemos criar o dicionário de um outro aluno da escola também e quantos mais quisermos:

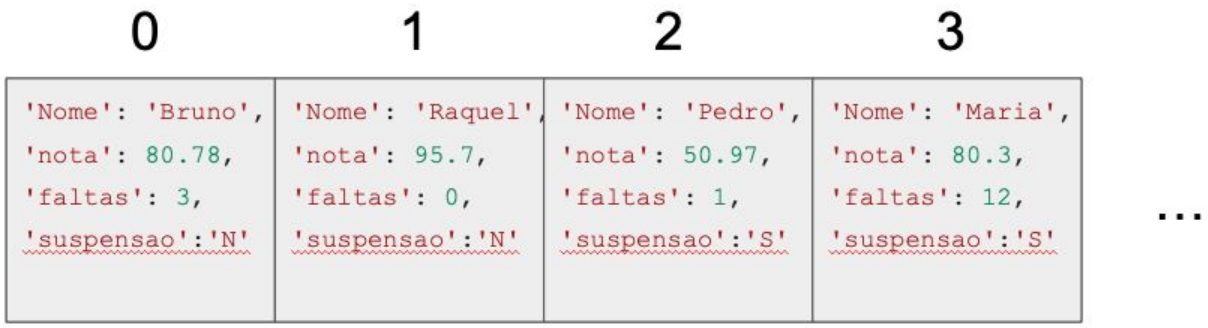
```
dados_aluno2 = {
    'Nome': 'Raquel',
    'nota': 95.7,
    'faltas': 0,
    'suspensao':'N'

}
```

Observe que o interpretador sabe que estamos criando um dicionário por conta do símbolo ***{ }.***

Agora você deve estar se perguntando: será necessário criar uma coleção para cada aluno? Mas o que queremos, na verdade, é criar uma única estrutura que comporte todos os dados de todos os alunos. Como faremos isso?

Entenda que cada ***dicionário é um conjunto com os dados de um aluno.*** *Logo, este conjunto pode ser considerado um único elemento.*

Se as listas permitem inserir em sua estrutura uma sequência de elementos e o dicionário permite inserir em sua estrutura uma coleção de dados, podemos criar uma lista em que cada elemento é o dicionário de um aluno. Isso resolveria nosso problema, veja:



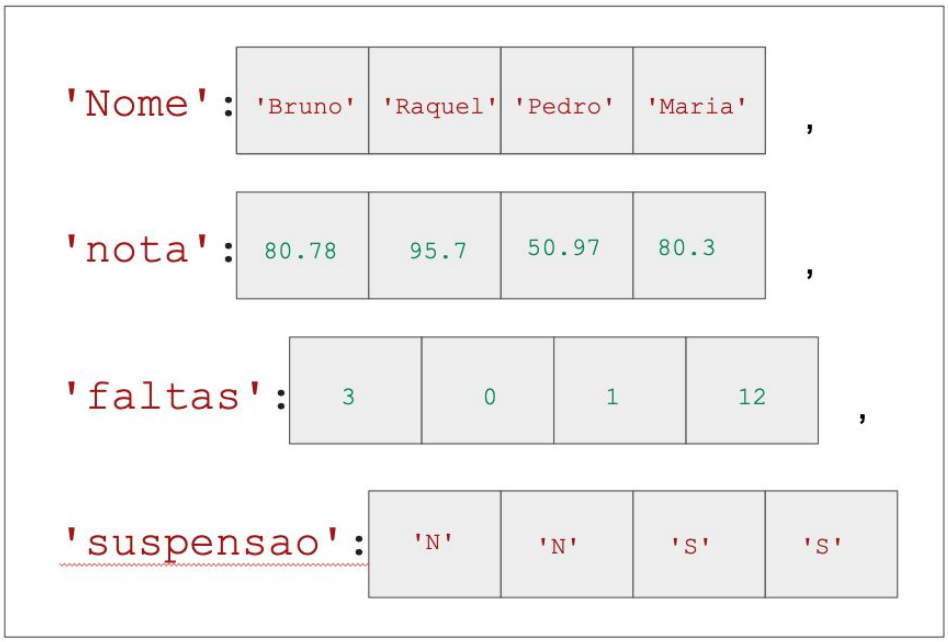
A codificação da ilustração acima seria:

```
dicionario_lista_alunos=[
    {
    'Nome': 'Bruno',
    'nota': 80.78,
    'faltas': 3,
    'suspensão':'N'
    },
    {
    'Nome': 'Raquel',
    'nota': 95.7,
    'faltas': 0,
    'suspensão':'N'
    },
    {
    'Nome': 'Pedro',
    'nota': 50.97,
    'faltas': 1,
    'suspensão':'S'
    },
    {
    'Nome': 'Maria',
    'nota': 80.3,
    'faltas': 12,
    'suspensão':'S'
    }
]
```

Se printarmos `dicionario_lista_alunos` o resultado do print é:

```
[{'Nome': 'Bruno', 'nota': 80.78, 'faltas': 3, 'suspensão': 'N'},
 {'Nome': 'Raquel', 'nota': 95.7, 'faltas': 0, 'suspensão': 'N'},
 {'Nome': 'Pedro', 'nota': 50.97, 'faltas': 1, 'suspensão': 'S'},
 {'Nome': 'Maria', 'nota': 80.3, 'faltas': 12, 'suspensão': 'S'}]
```

Podemos também criar a estrutura de outra maneira. Que tal um dicionário de listas em que os valores relacionados às chaves são listas com os dados:



Nesse caso, todos os dados que estão na primeira posição das listas diz respeito aos dados do Bruno; os que estão na segunda posição aos da Raquel e assim sucessivamente...

A codificação da ilustração acima seria:

```
dicionario_lista_alunos={

    'Nome':['Bruno','Raquel', 'Pedro', 'Maria'],
    'nota':[80.78,95.7,50.97,80.3],
    'faltas':[3,0,1,12],
    'suspensão':['N', 'N', 'S','S']

}
```

Se printarmos `dicionario_lista_alunos` o resultado do print é:

```
{'Nome': ['Bruno', 'Raquel', 'Pedro', 'Maria'],
 'nota': [80.78, 95.7, 50.97, 80.3],
 'faltas': [3, 0, 1, 12],
 'suspensão': ['N', 'N', 'S', 'S']}
```

Lembre-se

Saber estruturar os dados em nossos programas é fundamental para a profissão de programador. Por isso é importante conhecer as diferentes opções de estruturas de dados que existem para você decidir qual utilizar em cada situação.

Se o dado que você precisa armazenar será utilizado apenas nas próximas linhas de código e pode ser sobrescrito, utilize ***variável****.*
Nesse contexto, não faz sentido usar lista, por exemplo.

Se você precisa armazenar uma sequência de dados que não podem ser sobrescritos e que você precisará de todos eles no decorrer do programa, que tal usar uma ***lista****?*

Se você precisa armazenar uma coleção de dados sobre uma entidade, use um ***dicionário****.*

E caso precise armazenar várias coleções de dados, mescle listas com dicionários.

É importante que você pratique programação, o quanto puder, para conseguir adquirir esta habilidade de saber qual estrutura utilizar.