INF 1514

Introdução à Análise de Dados

Material 9

PUC
RIO

Manipulação de strings

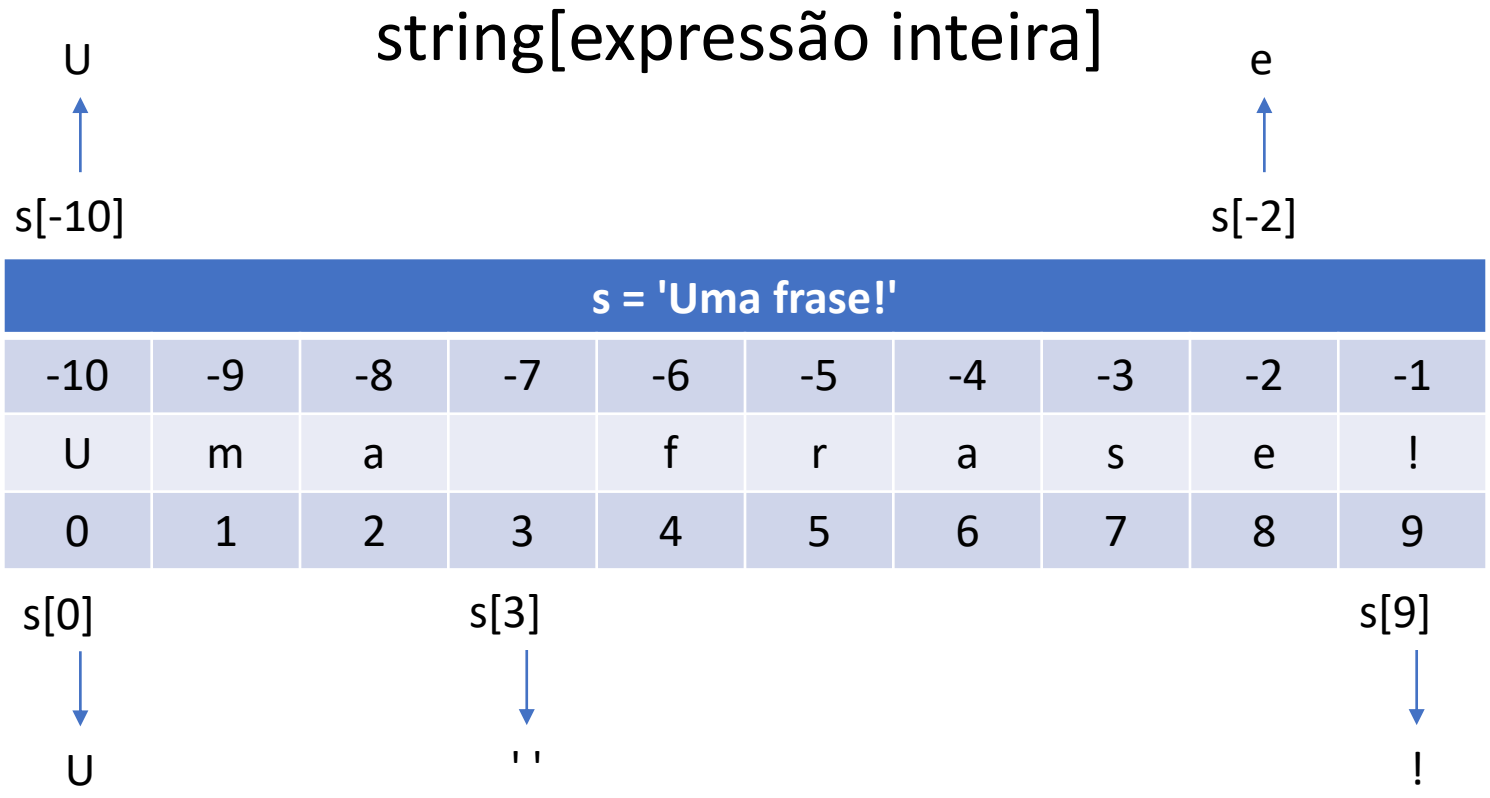
- Strings são **sequências de caracteres**.
- As **posições dos caracteres** na sequência são numeradas por **índices**
  - da esquerda para direita iniciando em 0.
  - da direita para a esquerda iniciando em -1.
- É possível **acessar um caractere ou um intervalo** (fatia/slice) da string, mas não modificá-los.

s = 'Uma frase!'									
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
U	m	a		f	r	a	s	e	!
0	1	2	3	4	5	6	7	8	9

- Um caractere é uma string de comprimento 1.

Manipulação de strings

- Um elemento pode ser selecionado, indexado pelo seu índice, utilizando o operador de indexação[].



Manipulação de strings

- O fatiamento de uma string seleciona um intervalo (fatia) da string da posição **a** (inclusive) até a posição **b** (exclusive) de ***n*** em ***n***.

string[a:b:n]

  - Se **a** não for definido será considerado como zero.
  - Se **b** não for definido, será considerado o tamanho da string.
  - Se o intervalo ***n*** (entre os caracteres), não for definido, será 1.

s = 'Uma frase!'									
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
U	m	a		f	r	a	s	e	!
0	1	2	3	4	5	6	7	8	9

- s[0:2] → 'Um', s[5:] → 'rase!', s[:5] → 'Uma f', s[0:4:2] → 'Ua'
- s[-3:-1] → 'se', s[::2] → 'Uafae', s[:] → 'Uma frase!'

Manipulação de strings

- O operador **+** concatena strings.

```
In [1]: "Olá" + "mundo!"
Out[1]: 'Olámundo!'
```

- O operador * replica strings.

```
In [1]: "Olá"*3
Out[1]: 'OláOláOlá'
```

- A função **len** retorna o número de caracteres (tamanho) da string.

```
In [1]: len("Olá mundo!")
Out[1]: 10
```

- A função **str** converte um número para string.

```
In [1]: str(10)
Out[1]: '10'
```

Manipulação de strings

- O operador booleano **in** verifica se uma dada string pertence (ou está presente ou contida) em outra string. Analogamente, o **in** verifica se uma string é uma substring de outra string.

```
testeOla = "Olá" in "Olá mundo!"
```
→ testeOla assume o valor True.

```
testeOi = "Oi" in "Olá mundo!"
```
→ testeOi assume o valor False.

- Para testar a ausência de uma substring, pode-se usar o operador booleano **not**.

```
testeOla = "Olá" not in "Olá mundo!"
```
→ testeOla assume o valor False.

```
testeOi = "Oi" not in "Olá mundo!"
```
→ testeOi assume o valor True.

- A string vazia é uma substring de toda string.

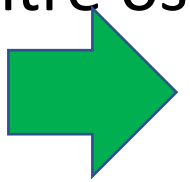
Função print

- Função da biblioteca padrão do Python usada para exibir valores de variáveis e/ou mensagens na tela.

print (valor1, valor2, valor3, ...)

- A saída começa sempre em uma nova linha, colocando automaticamente um espaço em branco (" ") como padrão entre os valores.

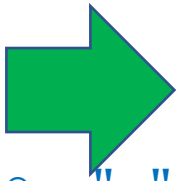
```
dia = 10
ano = 2023
print("Hoje é dia", dia, "de Julho de", ano, ".")
```



```
Hoje é dia 10 de Julho de 2023 .
```

- O parâmetro **sep** permite definir o separador dos objetos, padrão sendo " ".

```
dia = 10
ano = 2023
print("Hoje é dia", dia, "de Julho de", ano, ".", sep = '')
```



```
Hoje é dia10de Julho de2023.
```

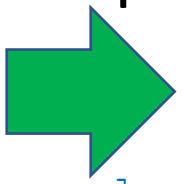
Função print

- Composição da saída.

print ('texto **%{}** texto **%{}** texto **%{}**.' **%** (**valor1**, **valor2**, **valor3**))

- No comando acima, **%{}** representa um **marcador de posição** cujos principais tipos são **%s (string)**, **%d (int)** e **%f (float)**. As variáveis devem estar em ordem, entre parênteses e separadas por vírgula após o operador de composição **(%)**.

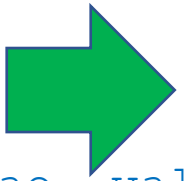
```
acao = "PETR4"
valor = 26.68
print("O valor da ação %s é %f." % (acao, valor))
```



```
O valor da ação PETR4 é 26.680000.
```

- Formatando a saída para 1 casa decimal.

```
acao = "PETR4"
valor = 26.68
print("O valor da ação %s é %.1f." % (acao, valor))
```

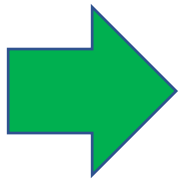


```
O valor da ação PETR4 é 26.7.
```

Função print

- Saída formatada.

```
valor = 345

# Sem formatação.
print("[%d]" % (valor))

# Ocupando 5 posições e alinhado à direita.
print("[%5d]" % (valor))

# Ocupando 5 posições e alinhado à esquerda (sinal -).
print("[%-5d]" % (valor))

# Ocupando 5 posições e alinhado à direita tendo espaços com 0.
print("[%05d]" % (valor))
```



```
[345]

[  345]

[345  ]

[00345]
```

Função input

- Função da biblioteca padrão do Python usada para capturar um valor digitado pelo usuário.

input(texto)

- O texto é exibido na tela, e o que for digitado a seguir é retornado pela função.
- O valor retornado sempre será do tipo string.
- O valor deve ser referenciado por uma variável se o programa precisar utilizá-lo depois.

```
valor = input("Digite o valor do produto:")
```

- Para valores inteiros use a função **int** para realizar a conversão e para valores ponto flutuante use a função **float**:

```
valor = float(input("Digite o valor do produto:"))
```

Exercício 1

- Construa um programa em Python para obter o resultado da divisão de dois números quaisquer fornecidos pelo usuário.

Exercício 1

- Construa um programa em Python para obter o resultado da divisão de dois números quaisquer fornecidos pelo usuário.

```
dividendo = 0.0
divisor = 0.0
resultado = 0.0
dividendo = float(input("Dividendo: "))
divisor = float(input("Divisor: "))
if (divisor != 0):
    resultado = dividendo / divisor
    print(resultado)
else:
    print("Divisão não efetuada.")
```

Exercício 2

- Construa uma função chamada **calculaDivisao** em Python que recebe como parâmetros dois números quaisquer fornecidos pelo usuário e retorna o resultado da divisão desses números. Caso a divisão não seja possível, a função deverá apresentar na tela a mensagem “Divisão não efetuada.”

Exercício 2

```
# Definição da função
def calculaDivisao (divisor, dividendo):
    if (divisor != 0):
        resultado = dividendo / divisor
    else:
        resultado = None
        print("Divisão não efetuada.")
    return resultado

# Corpo do programada
dividendo = float(input("Dividendo: "))
divisor = float(input("Divisor: "))

resultado = calculaDivisao(divisor, dividendo)
if (resultado != None):
    print('Resultado: ', resultado)
```

Para definirmos uma função usamos a palavra reservada **def**.
Observe o uso dos **:** e o **recuo** no corpo da função.

None "equivale" ao valor nulo (**null**).

Podemos iniciar as variáveis com o valor **None**.

Alguns tipos de dados

list

Sequência ou coleção de dados homogêneos ou heterogêneos.

Os elementos de uma lista são delimitados por colchetes [] e separados por vírgulas.

São mutáveis, ou seja, a qualquer momento, um item pode ser incluído, removido e alterado.

tupla

Sequências imutáveis (não permite a adição ou remoção de elementos), tipicamente usadas para armazenar coleções de dados heterogêneos.

Sequência de valores separados por vírgulas e envolvidos por parênteses ().

É possível criar tuplas que contenham objetos mutáveis, como listas.

set (conjuntos)

Coleção desordenada e mutável de elementos, sem elementos repetidos.

Suportam operações matemáticas como união, interseção, diferença e diferença simétrica.

Chaves {} ou a função set() podem ser usados para criar conjuntos.

dict (dicionários)

Coleção não-ordenada de pares chave:valor, onde as chaves são únicas em uma dada instância do dicionário.

Os itens do dicionário são ordenáveis, alteráveis e não permitem duplicatas.

Dicionários são delimitados por chaves { } e contém uma sequência de pares chave:valor separadas por vírgulas.

Listas

- É uma **sequência** ou coleção de **dados homogêneos ou heterogêneos**.
- Os **valores de uma lista**, também chamados de **elementos** ou **itens**, podem ser de **tipos diferentes** e até mesmo **outras listas** (**sublistas**), sendo delimitados por colchetes [] e separados por vírgulas.

```
indice = [3.99, -2.45, 1.23, 2.34, 0.89, 3.57, -1.07]
acao = ['AZUL4', 'AMER3', 'BBAS3', 'BBDC4']
variacao = [['ABEV3', 1.23], ['CRFB3', 2.03], ['BRKM5', 0.35]]
carteira = ['Pedro Sanchez', 900.00, [['ABEV3', 25], ['AZUL4', 75]]]

print (indice) # lista inteira
print (acao[0]) # primeiro item da lista
print (variacao[1]) # segundo item da lista
print (variacao[1][0]) # na segunda sublista o primeiro item
print (carteira[2]) # terceiro item
print (carteira[2][1]) # no terceiro item a segunda sublista
print (carteira[2][1][1]) # no terceiro item, na segunda sublista, o segundo item
```

Listas

```
indice = [3.99, -2.45, 1.23, 2.34, 0.89, 3.57, -1.07]
acao = ['AZUL4', 'AMER3', 'BBAS3', 'BBDC4']
variacao = [['ABEV3', 1.23], ['CRFB3', 2.03], ['BRKM5', 0.35]]
carteira = ['Pedro Sanchez', 900.00, [['ABEV3', 25], ['AZUL4', 75]]]

print (indice) # lista inteira
print (acao[0]) # primeiro item da lista
print (variacao[1]) # segundo item da lista
print (variacao[1][0]) # na segunda sublista o primeiro item
print (carteira[2]) # terceiro item
print (carteira[2][1]) # no terceiro item a segunda sublista
print (carteira[2][1][1]) # no terceiro item, na segunda sublista, o segundo item
```

Listas

```
indice = [3.99, -2.45, 1.23, 2.34, 0.89, 3.57, -1.07]
acao = ['AZUL4', 'AMER3', 'BBAS3', 'BBDC4']
variacao = [['ABEV3', 1.23], ['CRFB3', 2.03], ['BRKM5', 0.35]]
carteira = ['Pedro Sanchez', 900.00, [['ABEV3', 25], ['AZUL4', 75]]]

print (indice) # lista inteira
print (acao[0]) # primeiro item da lista
print (variacao[1]) # segundo item da lista
print (variacao[1][0]) # na segunda sublista o primeiro item
print (carteira[2]) # terceiro item
print (carteira[2][1]) # no terceiro item a segunda sublista
print (carteira[2][1][1]) # no terceiro item, na segunda sublista, o segundo item
```

Listas

```
indice = [3.99, -2.45, 1.23, 2.34, 0.89, 3.57, -1.07]
acao = ['AZUL4', 'AMER3', 'BBAS3', 'BBDC4']
variacao = [['ABEV3', 1.23], ['CRFB3', 2.03], ['BRKM5', 0.35]]
carteira = ['Pedro Sanchez', 900.00, [['ABEV3', 25], ['AZUL4', 75]]]

print (indice) # lista inteira
print (acao[0]) # primeiro item da lista
print (variacao[1]) # segundo item da lista
print (variacao[1][0]) # na segunda sublista o primeiro item
print (carteira[2]) # terceiro item
print (carteira[2][1]) # no terceiro item a segunda sublista
print (carteira[2][1][1]) # no terceiro item, na segunda sublista, o segundo item
```

Listas

```
indice = [3.99, -2.45, 1.23, 2.34, 0.89, 3.57, -1.07]
acao = ['AZUL4', 'AMER3', 'BBAS3', 'BBDC4']
variacao = [['ABEV3', 1.23], ['CRFB3', 2.03], ['BRKM5', 0.35]]
carteira = ['Pedro Sanchez', 900.00, [['ABEV3', 25], ['AZUL4', 75]]]

print (indice) # lista inteira
print (acao[0]) # primeiro item da lista
print (variacao[1]) # segundo item da lista
print (variacao[1][0]) # na segunda sublista o primeiro item
print (carteira[2]) # terceiro item
print (carteira[2][1]) # no terceiro item a segunda sublista
print (carteira[2][1][1]) # no terceiro item, na segunda sublista, o segundo item
```

Listas

```
indice = [3.99, -2.45, 1.23, 2.34, 0.89, 3.57, -1.07]
acao = ['AZUL4', 'AMER3', 'BBAS3', 'BBDC4']
variacao = [['ABEV3', 1.23], ['CRFB3', 2.03], ['BRKM5', 0.35]]
carteira = ['Pedro Sanchez', 900.00, [['ABEV3', 25], ['AZUL4', 75]]]

print (indice) # lista inteira
print (acao[0]) # primeiro item da lista
print (variacao[1]) # segundo item da lista
print (variacao[1][0]) # na segunda sublista o primeiro item
print (carteira[2]) # terceiro item
print (carteira[2][1]) # no terceiro item a segunda sublista
print (carteira[2][1][1]) # no terceiro item, na segunda sublista, o segundo item
```

Listas

```
indice = [3.99, -2.45, 1.23, 2.34, 0.89, 3.57, -1.07]
acao = ['AZUL4', 'AMER3', 'BBAS3', 'BBDC4']
variacao = [['ABEV3', 1.23], ['CRFB3', 2.03], ['BRKM5', 0.35]]
carteira = ['Pedro Sanchez', 900.00, [['ABEV3', 25], ['AZUL4', 75]]]

print (indice) # lista inteira
print (acao[0]) # primeiro item da lista
print (variacao[1]) # segundo item da lista
print (variacao[1][0]) # na segunda sublista o primeiro item
print (carteira[2]) # terceiro item
print (carteira[2][1]) # no terceiro item a segunda sublista
print (carteira[2][1][1]) # no terceiro item, na segunda sublista, o segundo item
```

Listas

```
indice = [3.99, -2.45, 1.23, 2.34, 0.89, 3.57, -1.07]
acao = ['AZUL4', 'AMER3', 'BBAS3', 'BBDC4']
variacao = [['ABEV3', 1.23], ['CRFB3', 2.03], ['BRKM5', 0.35]]
carteira = ['Pedro Sanchez', 900.00, [['ABEV3', 25], ['AZUL4', 75]]]

print (indice) # lista inteira
print (acao[0]) # primeiro item da lista
print (variacao[1]) # segundo item da lista
print (variacao[1][0]) # na segunda sublista o primeiro item
print (carteira[2]) # terceiro item
print (carteira[2][1]) # no terceiro item a segunda sublista
print (carteira[2][1][1]) # no terceiro item, na segunda sublista, o segundo item
```

Listas

- Para definir uma lista como **vazia** ou **limpar** uma lista fazemos **nomeVariavelLista = []**.
- A função **len** retorna o **número de elementos** de uma lista, sendo que uma sublista é contada como um elemento da lista que a contém.
- **Índices fora do intervalo provocam um erro de execução**. Lembrando que o primeiro índice é **0** e o último índice é o número de elementos da lista **menos 1**.
- Para percorrer uma lista podemos usar o comando **for**.

```
for elemento in acao:
    print(elemento)
```

Percorre toda a lista **acao** elemento por elemento.

```
for i in range(len(acao)):
    print(acao[i])
```

Percorre toda a lista **acao** usando o índice da lista. A função range gera uma sequência de **0** ao tamanho da lista **menos 1**.

Listas

- Assim como as strings, **listas podem ser fatiadas**.
- O **fatiamento** de uma lista **cria uma nova lista** selecionando um intervalo (fatia) da lista da posição **a** (inclusive) até a posição **b** (exclusive) de ***n*** em ***n***.

lista[**a**:**b**:**n**]

- Possibilidades de fatiamento:
  - lista[a:b] - cria uma cópia de **a** (inclusive) até **b** (exclusive).
  - lista[a:] - cria uma cópia a partir de **a** (inclusive).
  - lista[:b] - cria uma cópia até **b** (exclusive).
  - lista[:] - cria uma cópia de todos os elementos.
  - lista[a:b:n] - cria uma cópia de **a** (inclusive) até **b** (exclusive) de ***n*** em ***n*** elementos

Listas

- **São mutáveis**, ou seja, a qualquer momento, um item pode ser **incluído**, **removido** e **alterado**.
- Alguns métodos disponíveis para listas.

Método	Descrição
append(x)	Insere um elemento ao final da lista.
insert(i, x)	Insere um elemento em uma posição específica na lista.
remove(x)	Remove um elemento da lista.
del lista[a:b]	Remove os elementos de índice a até b menos 1.
pop(i)	Remove e retorna o elemento de índice i da lista.
index(x)	Busca um elemento na lista e retorna seu índice.
sort()	Ordena a lista (os itens devem ser do mesmo tipo).

Exemplo 1

- Elabore uma função chamada **intersecao** que receba duas listas X e Y contendo valores numéricos e retorne a interseção entre elas.

Exemplo 1

- Elabore uma função chamada **intersecao** que receba duas listas X e Y contendo valores numéricos e retorne a interseção entre elas.

```
def intersecao (lista1, lista2):
    listaIntersecao = [] # Inicailiza a lista de retorno como vazia.
    for elemento in lista1: # Percorre a lista1 elemento por elemento.
        if (elemento in lista2): # Verifica (in) se o elemento está na lista2.
            listaIntersecao.append(elemento) # Insere ao final da lista.
    return (listaIntersecao)

X = [1.2, 2, 3.9, 4, 5] # Cria a lista X.
Y = [1, 3.9, 5, 4.2, 6, 15] # Cria a lista Y.
print (intersecao (X, Y)) # Chama a função e imprime a lista retornada.
```

Exercício 3

- Sabendo que a função **abs** retorna o valor absoluto da diferença entre dois números, elabore uma função que receba uma lista contendo apenas números e retorne o valor mais próximo da média dos elementos.

Exercício 3

```
def calculaMedia(lista): # Função para calcular a média da lista.
    soma = 0
    for elemento in lista: # Percorre os elementos.
        soma += elemento
    media = soma / len(lista)
    return (media)

def valorMaisProximo(lista):
    media = calculaMedia(lista) # Calcula a média.
    maisProximo = lista[0] # Seleciona o primeiro elemento da lista.
    maisProximoDiferenca = abs(maisProximo - media) # Calcula a diferença.
    for i in range(1, len(lista)): # Percorre a lista a partir do segundo elemento.
        diferenca = abs(lista[i] - media) # Calcula a diferença.
        if (diferenca < maisProximoDiferenca): # Verifica se a diferença é menor.
            maisProximoDiferenca = diferenca # Guarda a nova diferença.
            maisProximo = lista[i] # Guarda o novo elemento de menor diferença.

    return (maisProximo)

lista = [6] # Executa a função para uma lista com um único elemento.
print(valorMaisProximo(lista))
lista = [-1, 2, 5, 3.5] # Executa a função para uma lista com cinco elementos.
print(valorMaisProximo(lista))
```

O método **mean()** do pacote **statistics** é uma alternativa mais simples.

Tuplas

- Consistem em uma **sequência** de valores **separados por vírgulas** e **envolvidos por parênteses**.
- Enquanto as **listas** são **sequências mutáveis**, normalmente usadas para **armazenar coleções de itens homogêneos**, **tuplas** são **sequências imutáveis**, tipicamente **usadas para armazenar coleções de dados heterogêneos**.
- **É possível criar tuplas que contenham objetos mutáveis, como listas**.

```
"""
 Tupla que armazena o código do cliente dentro de uma seguradora e uma
 lista contendo número de chassi e ano de compra de seus veículos.
"""
t = (768, [['9BGRD08X04G117974', 2017], ['9BHTG97Z73J236723', 2018]])

print(t[0]) # Primeiro elemento.
print(t) # Tupla inteira.
t[1][0][1] = 2016 # Alterando 2017 para 2016 (o elemento 1 de t é uma lista).
print(t)
t[0] = 123 # Tentando mudar o código do cliente.
```

Tuplas

```
"""
 Tupla que armazena o código do cliente dentro de uma seguradora e uma
 lista contendo número de chassi e ano de compra de seus veículos.
"""
t = (768, [['9BGRD08X04G117974', 2017], ['9BHTG97Z73J236723', 2018]])

print(t[0]) # Primeiro elemento.

print(t) # Tupla inteira.

t[1][0][1] = 2016 # Alterando 2017 para 2016 (o elemento 1 de t é uma lista).
print(t)

t[0] = 123 # Tentando mudar o código do cliente.
```

Tuplas

```
"""
 Tupla que armazena o código do cliente dentro de uma seguradora e uma
 lista contendo número de chassi e ano de compra de seus veículos.
"""
t = (768, [['9BGRD08X04G117974', 2017], ['9BHTG97Z73J236723', 2018]])

print(t[0]) # Primeiro elemento.

print(t) # Tupla inteira.

t[1][0][1] = 2016 # Alterando 2017 para 2016 (o elemento 1 de t é uma lista).
print(t)

t[0] = 123 # Tentando mudar o código do cliente.
```

Tuplas

```
"""
 Tupla que armazena o código do cliente dentro de uma seguradora e uma
 lista contendo número de chassi e ano de compra de seus veículos.
"""
t = (768, [['9BGRD08X04G117974', 2017], ['9BHTG97Z73J236723', 2018]])

print(t[0]) # Primeiro elemento.

print(t) # Tupla inteira.

t[1][0][1] = 2016 # Alterando 2017 para 2016 (o elemento 1 de t é uma lista).
print(t)

t[0] = 123 # Tentando mudar o código do cliente.
```

```
(768, [['9BGRD08X04G117974', 2016], ['9BHTG97Z73J236723', 2018]])
```

Tuplas

```
"""
 Tupla que armazena o código do cliente dentro de uma seguradora e uma
 lista contendo número de chassi e ano de compra de seus veículos.
"""
t = (768, [['9BGRD08X04G117974', 2017], ['9BHTG97Z73J236723', 2018]])

print(t[0]) # Primeiro elemento.

print(t) # Tupla inteira.

t[1][0][1] = 2016 # Alterando 2017 para 2016 (o elemento 1 de t é uma lista).
print(t)

t[0] = 123 # Tentando mudar o código do cliente.
```

Observe que a **lista de veículos do cliente** pode e deve ser alterada, mas seu **código de cliente**, que é o seu **identificador** dentro da seguradora, não deve ser alterado.

```
TypeError: 'tuple' object does not support item assignment
```

Tuplas

- Apesar de **tuplas** serem **similares a listas**, elas são frequentemente utilizadas em **situações diferentes** e com **propósitos distintos**.
- Uma **lista** pode ter **elementos adicionados** ou **removidos** a qualquer momento, enquanto uma **tupla**, uma vez definida, **não permite a adição** ou **remoção de elementos**.
- Sua característica de **imutabilidade** oferece segurança nas informações armazenadas e, por isso, uma das **finalidades da tupla** é **armazenar uma sequência de dados que não será modificada em outras partes do código**.

Tuplas

- Seus elementos são **acessados via índice** (visto anteriormente) ou **desempacotamento**.

```
t = (768, [['9BGRD08X04G117974', 2017], ['9BHTG97Z73J236723', 2018]])

codigo, listaVeiculo = t # Desempacotamento

print(codigo)
```

```
768
```

```
print(listaVeiculo)
```

```
[['9BGRD08X04G117974', 2017], ['9BHTG97Z73J236723', 2018]]
```

No desempacotamento, a variável **codigo** irá receber o **código do cliente** e a variável **listaVeiculo** irá receber a **lista de veículos do cliente**.

Tuplas

- Uma vez criada, a **tupla não pode ter seus elementos reordenados**.
- A função **len** retorna o **número de elementos** de uma tupla.
- O operador **in** permite verificar se um **elemento existe em uma tupla**, retornando **True** ou **False** de acordo com o resultado da pesquisa.
- **Índices fora do intervalo provocam um erro de execução.** Lembrando que o primeiro índice é **0** e o último índice é o número de elementos da tupla **menos 1**.
- Para percorrer uma tupla podemos usar o comando **for**.

```
for elemento in t:
    print(elemento)
```

Percorre toda a tupla **t** elemento por elemento.

```
for i in range((len(t))):
    print(t[i])
```

Percorre toda a tupla **t** usando o **índice da tupla**. A função **range** gera uma sequência de **0** ao tamanho da tupla **menos 1**.

Tuplas

- Assim como as listas, **tuplas podem ser fatiadas**.
- O fatiamento de uma tupla cria uma nova tupla selecionando um intervalo (fatia) da tupla da posição **a** (inclusive) até a posição **b** (exclusive) de ***n*** em ***n***.

tupla[**a**:**b**:**n**]

- Possibilidades de fatiamento:
  - tupla[a:b] - cria uma cópia de **a** (inclusive) até **b** (exclusive).
  - tupla[a:] - cria uma cópia a partir de **a** (inclusive).
  - tupla[:b] - cria uma cópia até **b** (exclusive).
  - tupla[:] - cria uma cópia de todos os elementos.
  - tupla[a:b:n] - cria uma cópia de **a** (inclusive) até **b** (exclusive) de ***n*** em ***n*** elementos.

Conjuntos

- Um **conjunto** é uma **coleção desordenada** de elementos, **sem elementos repetidos**.
- Suportam operações matemáticas como **união**, **interseção**, **diferença** e **diferença simétrica** e, por isso, são comumente **usados na verificação eficiente da existência de objetos** e **na eliminação de itens duplicados**.
- **Chaves** ou a função **set()** podem ser usados para criar conjuntos, sendo que para criar um **conjunto vazio** deve ser usado **set()** e não **{}**.

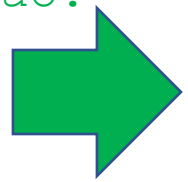
```
A = {'Dolar', 'Euro', 'Real', 'Libra', 'Real', 'Dolar', 'Peso'}
B = {'Euro', 'Libra', 'Franco'}
print(A) # Sem repetição.
print('Dolar' in A)
print('Rupia' in A)
print(A - B) # Moeda em A mas não em B.
print(A | B) # Moeda em A ou em B ou em ambos.
print(A & B) # Moeda em A e em B.
print(A ^ B) # Moeda em A ou em B mas não em ambos.
```

Conjuntos

```
A = {'Dolar', 'Euro', 'Real', 'Libra', 'Real', 'Dolar', 'Peso'}
B = {'Euro', 'Libra', 'Franco'}

print(A) # Sem repetição.

print('Dolar' in A)

print('Rupia' in A)

print(A - B) # Moeda em A mas não em B.

print(A | B) # Moeda em A ou em B ou em ambos.

print(A & B) # Moeda em A e em B.

print(A ^ B) # Moeda em A ou em B mas não em ambos.
```

Conjuntos

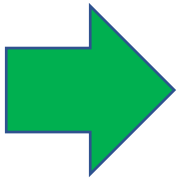
```
A = {'Dolar', 'Euro', 'Real', 'Libra', 'Real', 'Dolar', 'Peso'}
B = {'Euro', 'Libra', 'Franco'}

print(A) # Sem repetição.

print('Dolar' in A)

print('Rupia' in A)

print(A - B) # Moeda em A mas não em B.

print(A | B) # Moeda em A ou em B ou em ambos.

print(A & B) # Moeda em A e em B.

print(A ^ B) # Moeda em A ou em B mas não em ambos.
```



```
True
```

Conjuntos

```
A = {'Dolar', 'Euro', 'Real', 'Libra', 'Real', 'Dolar', 'Peso'}
B = {'Euro', 'Libra', 'Franco'}

print(A) # Sem repetição.

print('Dolar' in A)

print('Rupia' in A)
```

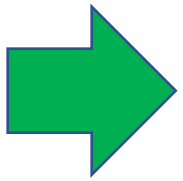


```
False
```

```
print(A - B) # Moeda em A mas não em B.

print(A | B) # Moeda em A ou em B ou em ambos.

print(A & B) # Moeda em A e em B.

print(A ^ B) # Moeda em A ou em B mas não em ambos.
```

Conjuntos

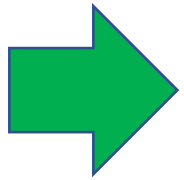
```
A = {'Dolar', 'Euro', 'Real', 'Libra', 'Real', 'Dolar', 'Peso'}
B = {'Euro', 'Libra', 'Franco'}

print(A) # Sem repetição.

print('Dolar' in A)

print('Rupia' in A)

print(A - B) # Moeda em A mas não em B.

print(A | B) # Moeda em A ou em B ou em ambos.

print(A & B) # Moeda em A e em B.

print(A ^ B) # Moeda em A ou em B mas não em ambos.
```



```
{'Real', 'Peso', 'Dolar'}
```

Conjuntos

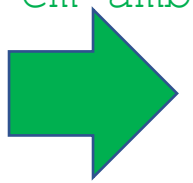
```
A = {'Dolar', 'Euro', 'Real', 'Libra', 'Real', 'Dolar', 'Peso'}
B = {'Euro', 'Libra', 'Franco'}

print(A) # Sem repetição.

print('Dolar' in A)

print('Rupia' in A)

print(A - B) # Moeda em A mas não em B.

print(A | B) # Moeda em A ou em B ou em ambos.

print(A & B) # Moeda em A e em B.

print(A ^ B) # Moeda em A ou em B mas não em ambos.
```



```
{'Real', 'Franco', 'Libra',
'Peso', 'Dolar', 'Euro'}
```

Conjuntos

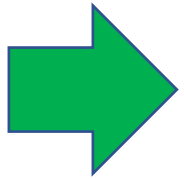
```
A = {'Dolar', 'Euro', 'Real', 'Libra', 'Real', 'Dolar', 'Peso'}
B = {'Euro', 'Libra', 'Franco'}

print(A) # Sem repetição.

print('Dolar' in A)

print('Rupia' in A)

print(A - B) # Moeda em A mas não em B.

print(A | B) # Moeda em A ou em B ou em ambos.

print(A & B) # Moeda em A e em B.

print(A ^ B) # Moeda em A ou em B mas não em ambos.
```



```
{'Euro', 'Libra'}
```

Conjuntos

```
A = {'Dolar', 'Euro', 'Real', 'Libra', 'Real', 'Dolar', 'Peso'}
B = {'Euro', 'Libra', 'Franco'}

print(A) # Sem repetição.

print('Dolar' in A)

print('Rupia' in A)

print(A - B) # Moeda em A mas não em B.

print(A | B) # Moeda em A ou em B ou em ambos.

print(A & B) # Moeda em A e em B.

print(A ^ B) # Moeda em A ou em B mas não em ambos.
```



```
{'Franco', 'Real',
'Peso', 'Dolar'}
```

Conjuntos

- Alguns métodos disponíveis para conjuntos.

Método	Descrição
s.difference_update(t)	Remove os itens que estão no conjunto **t** do conjunto **s**.
s.intersection_update(t)	Faz com que o conjunto **s** contenha a interseção dele com **t**.
s.isdisjoin(t)	Retorna **True** se **s** e **t** não tem nenhum item em comum.
s.issubset(t) (s <= t)	Retorna **True** se **s** é igual a **t** ou um subconjunto de **t**.
s.issuperset(t) (s >= t)	Retorna **True** se **s** é igual a **t** ou **t** é um subconjunto de **s**.
s.add(x)	Adiciona ao conjunto **s** o elemento **x**.
s.discard(x)	Remove o item **x** do conjunto **s**.
s.remove(x)	Remove o item **x** se ele estiver em **s** senão retorna um **KeyError**.
s.update(t)	Adiciona cada item do conjunto **t** que não está no conjunto **s** ao conjunto **s**.

Dicionários

- Coleção **não-ordenada de pares chave:valor**, onde as **chaves** são únicas em uma dada instância do dicionário. As **chaves** podem ser de qualquer tipo **imutável** (como strings e inteiros), sendo **valor** qualquer tipo de dado.
- Os itens do dicionário são **ordenáveis**, **alteráveis** e **não permitem duplicatas**.
- Dicionários são delimitados por **{ }**, e contém uma sequência de pares **chave:valor** separadas por vírgulas.

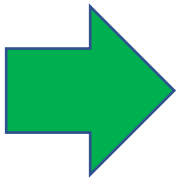
```
# Coleção de clientes representados por código (chave) e nome (valor).
cliente = {345:'Pedro Santos', 675:'Caio Freitas', 879:'Carmem Rios'}
print(cliente) # Todos os clientes.
print(cliente[675]) # Nome do cliente de código 675.
cliente[649] = 'Carlos Lopes' # Incluindo um cliente.
print(cliente)
print(cliente[123]) # Retorna erro de execução pois 123 não existe como chave.
```

Dicionários

```
# Coleção de clientes representados por código (chave) e nome (valor).
cliente = {345:'Pedro Santos', 675:'Caio Freitas', 879:'Carmem Rios'}

print(cliente) # Todos os clientes.

print(cliente[675]) # Nome do cliente de código 675.

cliente[649] = 'Carlos Lopes' # Incluindo um cliente.
print(cliente)

print(cliente[123]) # Retorna erro de execução pois 123 não existe como chave.
```

Dicionários

```
# Coleção de clientes representados por código (chave) e nome (valor).
cliente = {345:'Pedro Santos', 675:'Caio Freitas', 879:'Carmem Rios'}

print(cliente) # Todos os clientes.

print(cliente[675]) # Nome do cliente de código 675.

cliente[649] = 'Carlos Lopes' # Incluindo um cliente.
print(cliente)

print(cliente[123]) # Retorna erro de execução pois 123 não existe como chave.
```



```
Caio Freitas
```

Dicionários

```
# Coleção de clientes representados por código (chave) e nome (valor).
cliente = {345:'Pedro Santos', 675:'Caio Freitas', 879:'Carmem Rios'}

print(cliente) # Todos os clientes.

print(cliente[675]) # Nome do cliente de código 675.

cliente[649] = 'Carlos Lopes' # Incluindo um cliente.
print(cliente)
```

```
{345: 'Pedro Santos', 675: 'Caio Freitas', 879: 'Carmem Rios', 649: 'Carlos Lopes'}
```

```
print(cliente[123]) # Retorna erro de execução pois 123 não existe como chave.
```

Dicionários

```
# Coleção de clientes representados por código (chave) e nome (valor).
cliente = {345:'Pedro Santos', 675:'Caio Freitas', 879:'Carmem Rios'}

print(cliente) # Todos os clientes.

print(cliente[675]) # Nome do cliente de código 675.

cliente[649] = 'Carlos Lopes' # Incluindo um cliente.
print(cliente)

print(cliente[123]) # Retorna erro de execução pois 123 não existe como chave.
```

```
KeyError: 123
```

Observe que o dicionário **cliente** não possui a chave **123**.

Dicionários

- Para definir uma dicionário vazio fazemos **nomeVariavel = {}**.
- A função **dict** permite criar dicionários a partir de listas.

```
listaEstado = [('RJ', 'Rio de Janeiro'), ('SP', 'São Paulo')]
estado = dict(listaEstado)
```

- A função **len** retorna o número de elementos do dicionário.
- Pare recuperar um valor do dicionário também pode ser usado o método **get** que recebe o valor da chave para busca.

```
print(estado.get('RJ')) # Retorna 'Rio de Janeiro'
print(estado.get('ES')) # Retorna None, pois 'ES' não existe como chave.
```

- Para remover um elemento do dicionário pode ser usado o comando **pop**.

```
valor = estado.pop('SP')
print(valor)
```

- O método **update** permite unir dois dicionários sem que sejam criadas duplicidades de chaves.

Dicionários

- Os operadores **in** e **not in** permitem identificar se um elemento está em um dicionário ou não.

```
print('RJ' in estado)
print('PE' not in estado)
```

- O método **keys** permite obter uma lista das chaves de um dicionário, o método **values** a lista de valores e o método **items** lista de tuplas, onde cada tupla é composta por dois valores: a chave e o valor.
- Para percorrer uma lista podemos usar o comando **for**.

```
for chave in estado:
    print(chave)
    print(estado[chave])
```

Percorre todo o dicionário **estado** elemento por elemento.

```
for chave, valor in estado.items():
    print(chave)
    print(valor)
```

Percorre todo o dicionário **estado** elemento por elemento, sendo que o método **items** permite obter chave e valor simultaneamente.

Exemplo 2

- Escreva a função chamada **recuperarMensagem**, que recebe uma lista com números inteiros correspondentes a uma mensagem secreta e retornar a mensagem de acordo com a tabela de codificação abaixo:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
' '	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Espaço em branco.

Exemplo 2

```
def recuperarMensagem(mensagemCodificada):
    tabelaDecodificacao = " abcdefghijklmnopqrstuvwxyz" # Tabela de decodificação.
    mensagemDecodificada = '' # Mensagem a ser retornada.
    for codigo in mensagemCodificada: # Percorre os códigos da mensagem codificada passada.
        mensagemDecodificada = mensagemDecodificada + tabelaDecodificacao[codigo] # Decodificação.
    return (mensagemDecodificada)

# Corpo do programa.

# Mensagem secreta.
mensagemSecreta = [20, 5, 14, 8, 1, 0, 21, 13, 1, 0, 2, 15, 1, 0, 20, 1, 18, 4, 5]

# Mensagem original.
mensagem = recuperarMensagem(mensagemSecreta)

print(mensagem)
```