



IN

# INFINITY SCHOOL

V I S U A L   A R T   C R E A T I V E   C E N T E R

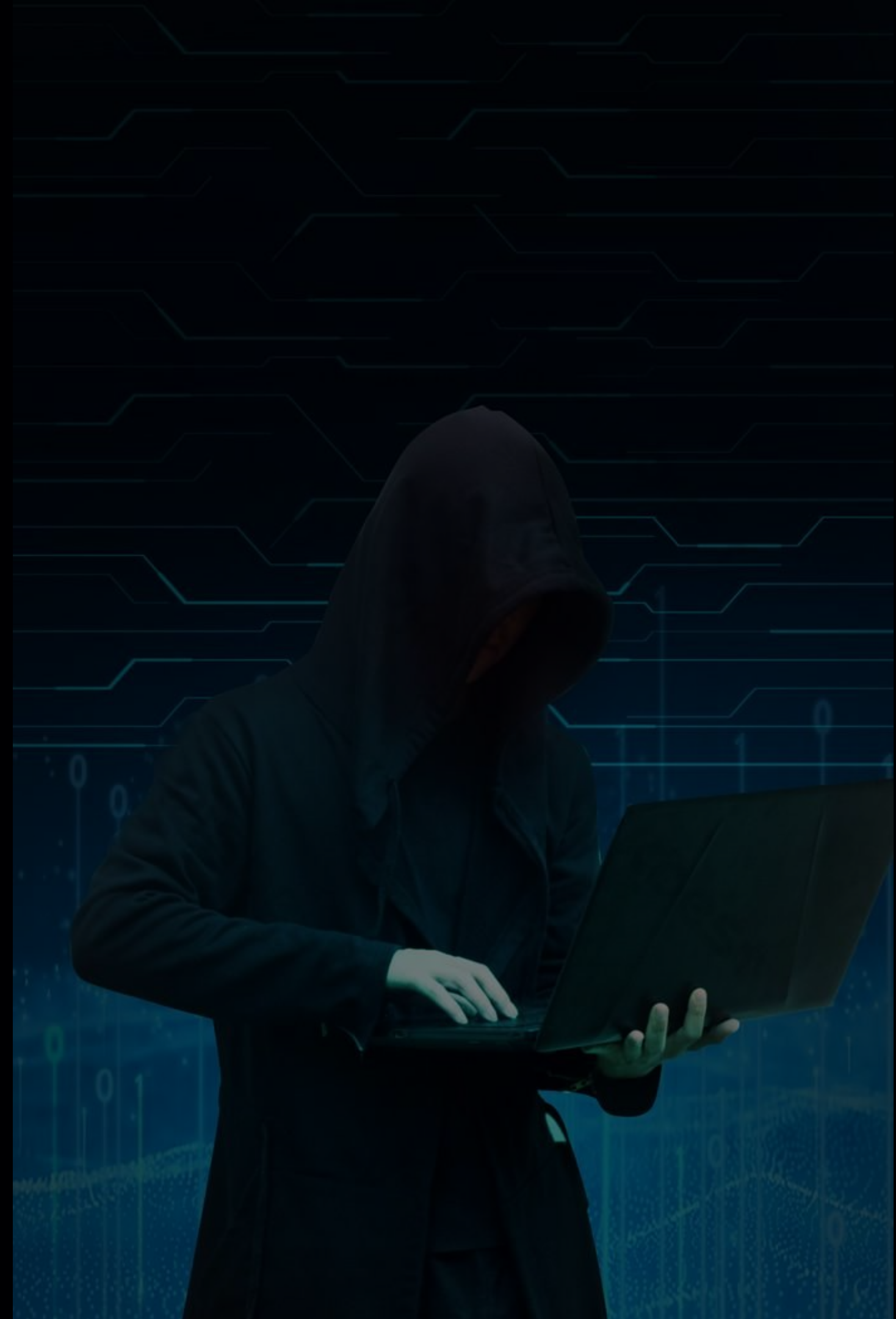
# PY – Listas e Tuplas

## 01 Coleções – Listas

- introdução as coleções
- listas
- funções(coleções)
- funções(list)

## 02 Coleções – Tuplas

- Tuplas
- Desempacotamento

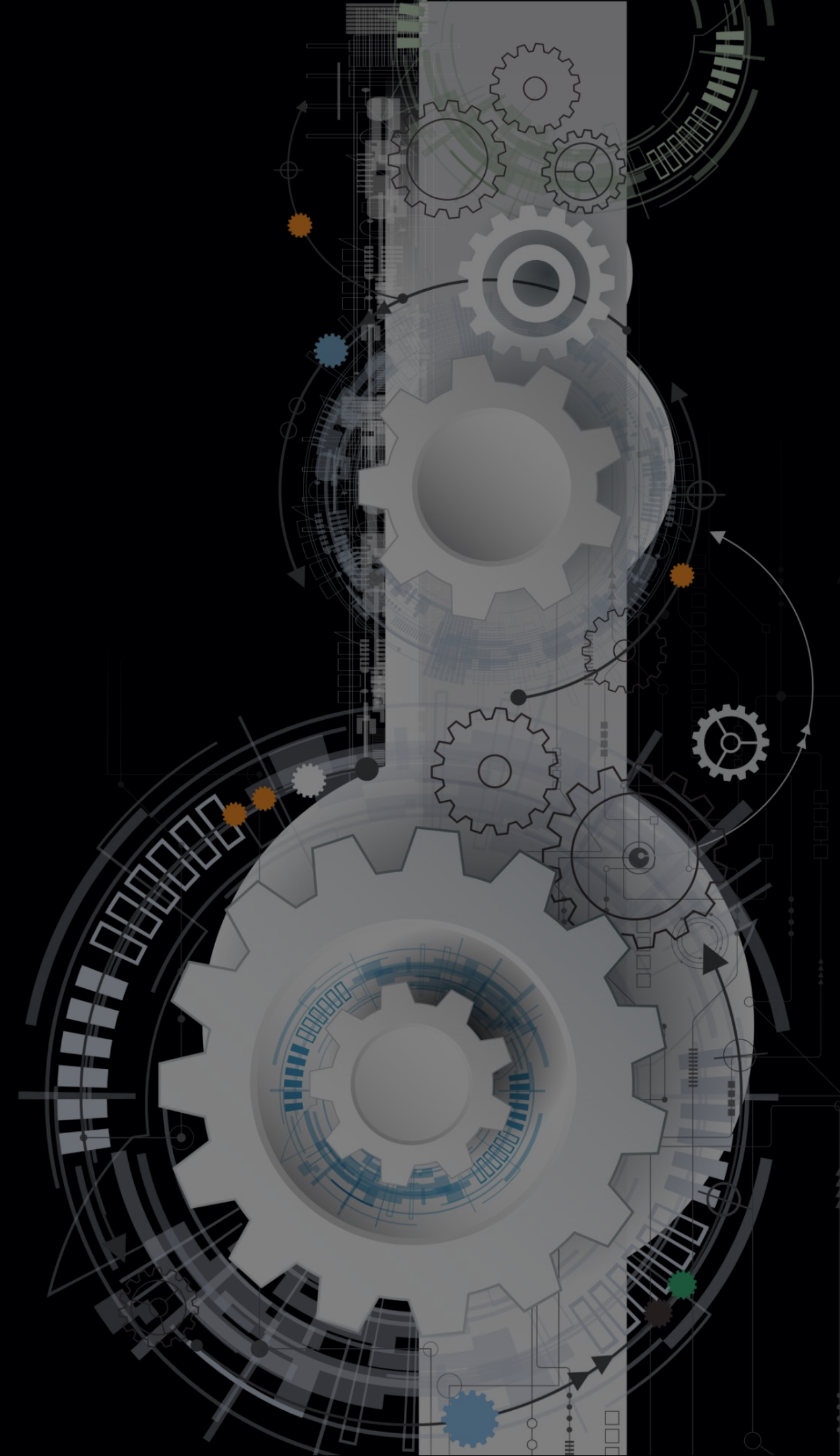


# PY – Listas e Tuplas

## INTRODUÇÃO: Sequências em Python

Nessa aula introduziremos o conceito de *lista* do **tipo list**, uma estrutura sequencial indexada muito utilizada e uma das principais estruturas básicas do Python.

Uma lista em Python é uma sequência ou coleção ordenada de valores de qualquer tipo ou classe tais como: int, float, bool, str e mesmo list, entre outros.





# PY – Listas e Tuplas



## Listas

Uma Lista ( list ) em Python, nada mais é que uma **coleção ordenada de valores, separados por vírgula e dentro de colchetes []** . Elas são utilizadas para armazenar diversos itens em uma única variável. Entender este conteúdo é de extrema importância para dominar a linguagem por completo!



# PY – Listas e Tuplas

## Listas

Diariamente utilizamos listas para organizar informação, como a lista de coisas a fazer, lista de compras, lista de filmes em cartaz etc.

Existem várias maneiras de criarmos uma lista. A maneira mais simples é envolver os elementos da lista por colchetes [ ].



```
# Podemos criar uma lista de vários objetos de tipos distintos:  
minha_lista = [15, "hello", 0.5, True]
```

```
# Podemos criar uma lista contendo informações de uma pessoa:  
registro = ["José", "masculino", 30, 173, "brasileiro", "solteiro"]
```

```
# ou resultados de operações ou verificações:  
primos = [2, 3, 5, 7, 11]
```

# PY – Listas e Tuplas

## Comprimento de uma lista

A função `len()` retorna o comprimento (o número de elementos ou objetos) de uma lista.



```
1 # Terminal
2 print(len(minha_lista)) # 4
3 print(len(registro))   # 6
4 print(len(primos))     # 5
```



# PY – Listas e Tuplas

## Índices

Cada valor da lista é identificado por um índice.

Dizemos que uma lista é uma estrutura sequencial indexada pois os seus elementos podem ser acessados sequencialmente utilizando índices. O primeiro elemento da lista tem índice 0, o segundo tem índice 1 e assim por diante.



```
registro = ["José", "masculino", 30, 173, "brasileiro", "solteiro"]  
print(registro[0]) # Terminal: José  
print(registro[1]) # Terminal: masculino  
print(registro[2]) # Terminal: 30  
print(registro[-1])# Terminal: solteiro
```



# PY – Listas e Tuplas

## Fatias de Listas

A operação de fatiar (*slice*) que vimos com strings também pode ser aplicada sobre listas. Lembre que o primeiro índice indica o ponto do início da fatia e o segundo índice é um depois do final da fatia (o elemento com esse índice não faz parte da fatia).

Teste esse código em seu computador:



```
registro = ["José", "masculino", 30, 173, "brasileiro", "solteiro"]  
print(registro[0:3]) # Terminal: ['José', 'masculino', 30]  
print(registro[3:6]) # Terminal: [173, 'brasileiro', 'solteiro']
```



# PY – Listas e Tuplas

## Listas são mutáveis

Diferentemente de strings, listas são mutáveis. Podemos alterar um item em uma lista acessando-o diretamente pelo seu índice e utilizando o operador de atribuição alterando o valor ou objeto indexado.

Teste esse código em seu computador:



```
frutas = ["Laranja", "Maçã", "Pera", "uva"]  
print(frutas)
```

```
frutas[0] = "Jaca"  
frutas[-1] = "Melancia"  
print(frutas)
```

# PY – Listas e Tuplas

## Métodos de Listas

O operador ponto também pode ser usado para acessar métodos nativos (*built-in*) de objetos que são do tipo list e é através desses métodos que manipulamos a lista.

```
frutas = ["Laranja", "Maçã", "Pera", "uva"]  
frutas.
```

- append
- clear
- copy
- count
- extend
- index
- insert
- pop
- remove
- reverse
- sort



# PY – Listas e Tuplas

## Principais Métodos de Manipulação de Listas

O método `append` é um método de listas que insere o argumento passado para ele no final da lista. Execute este código em seu computador!



```
lista_de_nomes = []  
# inserindo dados de forma hardcoding  
lista_de_nomes.append("Maria")  
lista_de_nomes.append("Antonia")  
  
# inserindo dados atravez do input  
nome = input("digite um nome: ")  
lista_de_nomes.append(nome)
```

# PY – Listas e Tuplas

## Principais Métodos de Manipulação de Listas

O método insert exige dois argumentos para inserir dados em uma lista. Primeiro a posição (index) e segundo o item (dados).

Onde será que o nome José será inserido ?

Execute esse código em seu computador!




```
lista_de_nomes = ["Maria", "Antonia", "Antonio"]  
lista_de_nomes.insert(2, "José")  
print(lista_de_nomes)
```




# PY – Listas e Tuplas

## Principais Métodos de Manipulação de Listas

O método `pop` é utilizado para deletar e retornar o item da lista que foi removido. Esse método possui argumento opcional, ou seja, quando não é inserido um argumento o método retorna e exclui o último item da lista, mas quando se é referenciado o item pelo index da lista no argumento o método retorna o item removido da lista.



```
lista_de_nomes = ["Maria", "Antonia", "José", "Antonio"]
# método sem argumento!
exclui_ultimo_nome = lista_de_nomes.pop()
print(f'este item {exclui_ultimo_nome} foi removido dessa lista {lista_de_nomes}')
```



```
lista_de_nomes = ["Maria", "Antonia", "José", "Antonio"]
# passando index como argumento!
exclui_ultimo_nome = lista_de_nomes.pop(2)
print(f'este item {exclui_ultimo_nome} foi removido dessa lista {lista_de_nomes}')
```



# PY – Listas e Tuplas

## Principais Métodos de Manipulação de Listas

O método `remove` é um método utilizado para remover a primeira ocorrência do item dentro de uma lista.



```
lista_de_nomes = ["Maria", "Antonia", "José", "Antonio"]  
print(lista_de_nomes)  
question = input("Qual nome você deseja remover da lista: ")  
lista_de_nomes.remove(question)  
print(lista_de_nomes)
```



# PY – Listas e Tuplas

## Principais Métodos de Manipulação de Listas

O método `remove` é um método utilizado para remover a primeira ocorrência do item dentro de uma lista.



```
lista_de_nomes = ["Maria", "Antonia", "José", "Antonio"]  
print(lista_de_nomes)  
question = input("Qual nome você deseja remover da lista: ")  
lista_de_nomes.remove(question)  
print(lista_de_nomes)
```

# PY – Listas e Tuplas

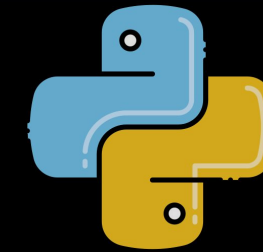
## Principais Métodos de Manipulação de Listas

A tabela a seguir mostra um resumo dos métodos de listas mostrados acima. Faça testes com esses métodos para ganhar uma melhor compreensão do que eles fazem.

Método	Parâmetros	Resultado	Descrição
append	item	mutador	Acrescenta um novo item no final da lista
insert	posição, item	mutador	Insere um novo item na posição dada
pop	nenhum	híbrido	Remove e retorna o último item
pop	posição	híbrido	Remove e retorna o item da posição
sort	nenhum	mutador	Ordena a lista
reverse	nenhum	mutador	Ordena a lista em ordem reversa
index	item	retorna posição	Retorna a posição da primeira ocorrência do item
count	item	retorna qtd	Retorna o número de ocorrências do item
remove	item	mutador	Remove a primeira ocorrência do item



# PY – Listas e Tuplas



```
1 tupla = ("oque", "são", "tuplas", "?")
```

# PY – Listas e Tuplas

## Tuplas

Tuplas são uma sequência de **objetos imutáveis**, em outras palavras, uma vez criadas, tuplas não podem ser modificadas, normalmente são usadas para guardar **dados protegidos**.

As tuplas são escritas entre parênteses **()**.

Uma tupla em Python é semelhante a uma lista. A diferença entre os dois é que não podemos alterar os elementos de uma tupla depois de atribuída, enquanto podemos alterar os elementos de uma lista.





# PY – Listas e Tuplas

## Criando uma Tupla

Podemos definir uma tupla da seguinte maneira:



```
linguagens = ("Python", "JavaScript", "PHP", "Java", "C#")  
print(type(linguagens))
```

As tuplas também podem ser inicializadas com uma atribuição seguida de uma vírgula, exatamente dessa forma:



```
linguagens = "Python",  
print(type(linguagens))
```

# PY – Listas e Tuplas

## Atualizando Tuplas

Lembrando que as tuplas são imutáveis, portanto se tentarmos modificá-la, nos será retornado um erro do tipo **TypeError**:



```
linguagens = ("Python", "JavaScript", "PHP", "Java", "C#")  
linguagens[0] = "Ruby"
```

```
File "c:\Users\José\Documents\testepy\testepy\estudos_artigos\teste.py", line 2, in <module>  
    linguagens[0] = "Ruby"  
TypeError: 'tuple' object does not support item assignment
```



# PY – Listas e Tuplas

## Descompactando uma tupla

No Python, há um recurso de atribuição de tupla muito poderoso que atribui o lado direito dos valores ao lado esquerdo. De outra forma, é chamado de desempacotamento de uma tupla de valores em uma variável. **Isso também se aplica a listas!**



```
linguagens = ("Python", "JavaScript", "PHP", "Java", "C#")
#desempacotamento
(py, js, php, java, c_sharp) = linguagens
print(f"""
1 - {py}
2 - {js}
3 - {php}
4 - {java}
5 - {c_sharp}
""")
```

# PY – Listas e Tuplas

## Iterando sob uma coleção(objeto iterável):

A execução repetida de uma sequência de instruções é chamada de **iteração** (*iteration*). Como iterar é muito comum, Python tem várias características para torná-la mais fácil. Nós já vimos o comando **for** em Lógica da Programação. Esta é uma forma muito comum de iteração em Python.

Lembre-se que o comando **for** processa cada item em uma lista. Cada item, por sua vez, é reatribuído a variável de contagem, e o corpo do laço é executado.



```
linguagens = ("Python", "JavaScript", "PHP", "Java", "C#")  
for dados in linguagens:  
    print(dados)
```



Também podemos usar Listas!

```
linguagens = ["Python", "JavaScript", "PHP", "Java", "C#"]  
for dados in linguagens:  
    print(dados)
```



# PY – Listas e Tuplas

## Listas VS Tuplas

Quais estruturas de dados devo usar, Listas ou Tuplas?

A execução do programa é mais rápida quando manipulamos uma tupla do que uma equivalente lista.

Às vezes desejamos que os dados não sejam modificados, se determinados que valores em uma coleção devem ser constantes no programa, utilizar uma Tupla nos protege contra acidentes de modificação.

Por outro lado, as listas nos permitem ser mais flexíveis com alterações, facilitando a resolução e trazendo soluções para seu programa.

O interessante é saber em quais situações será necessário o uso de cada uma dessas estruturas.

# PY – Listas e Tuplas

## Exercícios:

1. Considere a seguinte lista de números: [2, 5, 8, 11, 14]. Escreva um programa que itere sobre essa lista e exiba cada número elevado ao quadrado.
1. Dada a tupla de nomes de países: ("Brasil", "Canadá", "Austrália", "Espanha", "Índia"), crie um programa que itere sobre a tupla e exiba na tela cada país seguido pelo número de caracteres presentes em seu nome.
1. Você recebeu uma lista de tuplas, onde cada tupla contém o nome de um produto e seu preço. Por exemplo: [("Maçã", 2.50), ("Banana", 1.75), ("Laranja", 3.00)]. Escreva um programa que itere sobre essa lista e calcule o valor total dos produtos, exibindo-os na tela.
1. Considere a seguinte lista de palavras: ["Python", "é", "uma", "linguagem", "poderosa"]. Escreva um programa que itere sobre essa lista e exiba apenas as palavras que possuem mais de 4 letras.



# PY – Listas e Tuplas

## Extra: List Comprehension

List Comprehension, ou Compreensão de Lista, é uma construção sintática em várias linguagens de programação, incluindo Python, que permite criar listas de forma concisa a partir de uma expressão ou de uma sequência iterável. Ela é uma maneira elegante e eficiente de criar, filtrar e transformar listas em uma única linha de código.



```
dez_a_trinta = [x for x in range(10,30)]  
print(dez_a_trinta)
```

```
#mostrando apenas numeros pares
```

```
so_numeros_pares = [x for x in range(10,30) if x % 2 == 0]  
print(so_numeros_pares)
```

```
#exemplo com strings
```

```
apenas_maiusculos = [nomes for nomes in ["JOSÉ","maria","TEREZA"] if nomes == nomes.upper()]  
print(apenas_maiusculos)
```



IN

# INFINITY SCHOOL

V I S U A L   A R T   C R E A T I V E   C E N T E R