INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

Python - Aula 02

Objetivos da aula:

- Coleções - Listas
  - Introdução as coleções
  - Listas
  - Slicing
  - Funções (coleções)
  - Funções (list)
  - Mudando itens
  - Adicionar elementos
  - Remover e Ordenar
  - Copiando Listas
- Coleções - Tuplas
  - Tuplas
  - Mais sobre tuplas
  - Loops

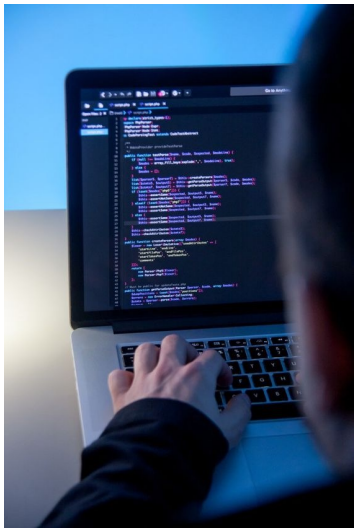
Python - Aula 02

Sequências em Python

Existem três tipos básicos de sequência na linguagem Python que são: `listas`, `tuplas` e `objetos range`.

Já vimos como o `objeto range` funciona. Caso não se lembre, você sabe o que fazer.

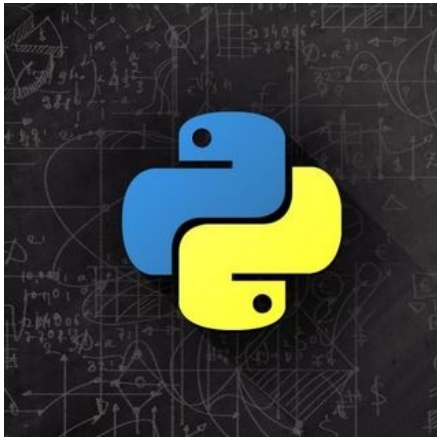
1. Acesse o *python shell*
2. Digite o comando `help(range)`
3. A descrição da função será exibida no terminal



IN
INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Python - Aula 02

Sequências em Python



Operações comuns de sequências

- **in** (`elemento in sequencia`) : *True* caso o elemento esteja na sequência, caso contrário receberá *False*
- **not in** (`elemento not in sequencia`) : *False* caso o elemento não esteja na sequência, caso contrário *True*
- **+** (`seq1 + seq2`): concatenação de seq1 e seq2
- * (`seq1 * n`) : equivale a adicionar seq1 a si mesmo n vezes
- **[ ]** (`seq1[n]`) : representa o *enésimo* elemento de seq1

OBS : Sequências e iteráveis em Python tem a indexação iniciada por 0

- **[:]** (`seq1[n:m]`) : *slicing* ou *fatiamento*. Fatia a sequência de n até m, mas sem incluir o valor de m
- **[: :]** (`seq1[n:m:p]`) : fatia a sequência de n até m sem incluir m, mas com passo p
- **len** (`len(seq1)`) : retorna o tamanho da sequência
- **min** (`min(seq1)`) : retorna o menor valor da sequência
- **max** (`max(seq1)`) : retorna o maior valor da sequência
- **sum** (`sum(seq1)`) : retorna a soma dos elementos da sequência quando o tipo de dado for inteiro ou ponto flutuante
- **index** (`seq1.index(elemento)`) : retorna o índice da primeira ocorrência de elemento na sequência
- **count** (`seq1.count(elemento)`) : retorna o número total de ocorrências de elemento na sequência

Python - Aula 02

Sequências em Python

Sequências do mesmo tipo também suportam comparações.

Tuplas e Listas são comparadas lexicograficamente pela comparação de elementos correspondentes.

Isto significa que para que a comparação ocorra com sucesso, cada elemento é comparado entre si, e as duas sequências devem ser do mesmo tipo e, também, possuírem o mesmo tamanho

VALE LEMBRAR #1 : concatenar sequências imutáveis sempre resultará em um novo objeto. Veremos isso quando falarmos de sequências imutáveis *tuplas*

VALE LEMBRAR #2 : sequências do tipo *range* suportam **apenas** sequências de itens que seguem padrões específicos e, portanto, não suportam concatenação ou repetição de sequência

Operações em sequências mutáveis

- `seq1[n] = valor` : o elemento n da sequência é substituído por valor
- `seq1[n:m] = seq2` : esta fatia da sequência será substituída pelo conteúdo da sequência 2 ou do iterável
- `del seq1[n:m]` ou `seq1[n:m] = []` : remove da sequência os valores contidos neste fatiamento ou *slicing*
- `seq1[n:m:p] = seq2` : os elementos desta fatia serão substituídos pelos elementos pelos elementos da sequência 2 ou do iterável 2
- `del seq1[n:m:p]` : remove da sequência os elementos dos índices especificados ao passo p

OBS : seq2 deve ter o mesmo tamanho que a fatia ou *slice* a qual ele substituirá

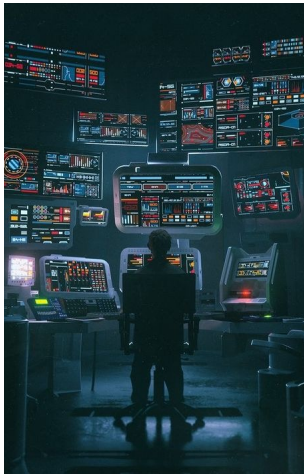
Python - Aula 02

Listas

São sequências mutáveis, normalmente usadas para armazenar coleções de itens homogêneos.

As listas podem ser declaradas e inicializadas de várias maneiras como:

- usando apenas um par de colchetes para criarmos uma lista vazia : `minha_lista = [ ]`
- usando colchetes e separando os elementos por vírgulas : `minha_lista = [1, 2, 3]`
- usando *list comprehension* : `[num for num in iterável]`
- usando o construtor : `list()` ou `list(iterável)`



INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Python - Aula 02

Listas

Usando apenas um par de colchetes

```
[ ]  minha_lista = []
     minha_lista
```

```
[]
```

Usando colchetes e separando os elementos por vírgula

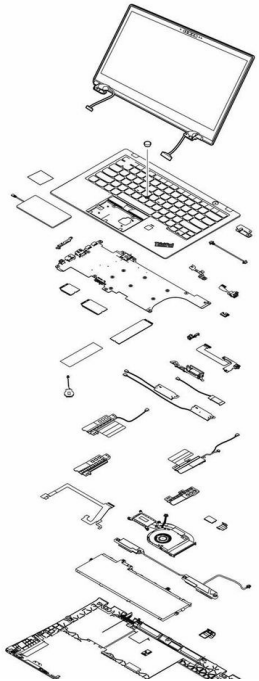
```
[ ]  minha_lista = [1, 2, 3]
     minha_lista
```

```
[1, 2, 3]
```

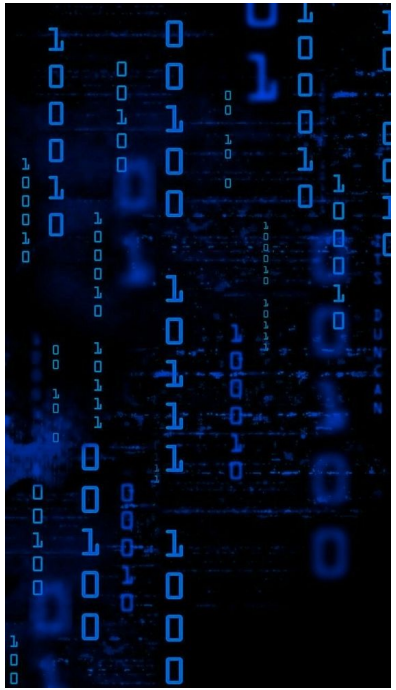
Usando *list comprehension*

```
[ ]  minha_lista = [ num for num in range(1, 4)]
     minha_lista
```

```
[1, 2, 3]
```



Métodos de Listas



Append

Adiciona um valor ao final da sequência

SINTAXE

```
seq1.append(valor)
```

```
[ ]  seq1 = [1, 2, 3]
     seq1.append([4, 5, 6])
     seq1
```

```
[1, 2, 3, [4, 5, 6]]
```

Python - Aula 02

Métodos de Listas

Clear

Remove todos os elementos da sequência

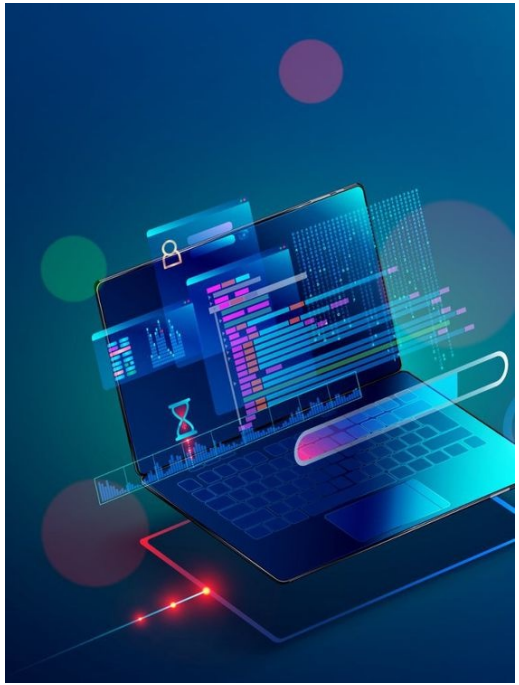
SINTAXE

`seq1.clear()`

OBS : comando equivalente a este `del seq1[:]`

```
[ ]  seq1.clear()
     seq1

     []
```



INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Métodos de Listas

Copy

Cria uma cópia rasa - shallow copy - da sequência

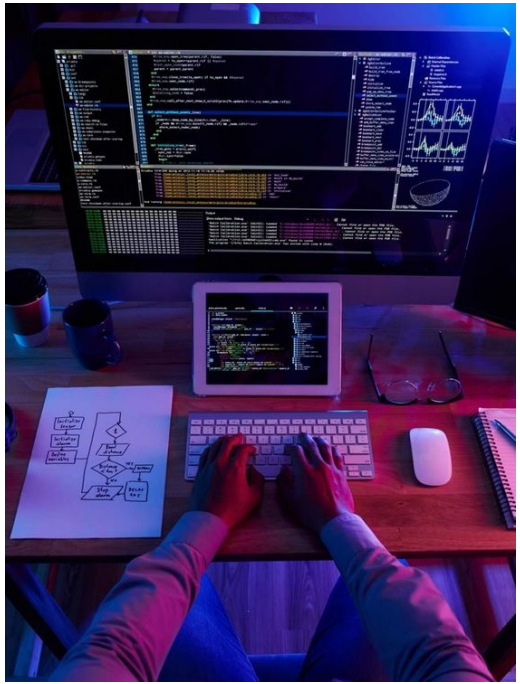
SINTAXE

`seq1.copy()`

OBS: comando equivalente a este `seq1[:]`

```
[ ]  seq1 = ['a', 'b', 'c']
     seq2 = seq1.copy()
     print('seq1', seq1)
     print('seq2', seq2)
```

```
seq1 ['a', 'b', 'c']
seq2 ['a', 'b', 'c']
```



IN
INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Python - Aula 02

Métodos de Listas

Extend

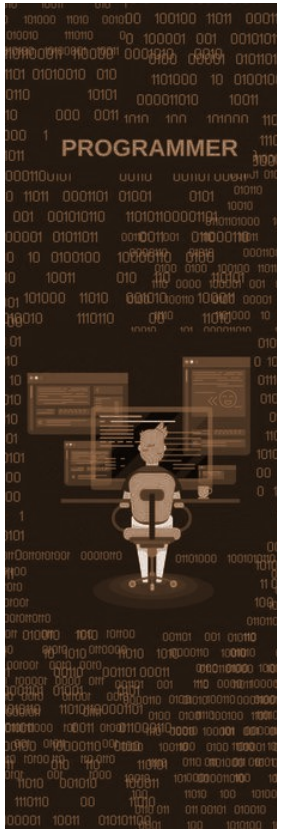
Estende a sequência com o conteúdo de uma nova sequência ou iterável

SINTAXE

```
seq1.extend(iterável)
```

```
[ ]  seq1.extend(['d', 'e', 'f'])
     seq1
```

```
['a', 'b', 'c', 'd', 'e', 'f']
```



IN

INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

Python - Aula 02

Métodos de Listas

Insert

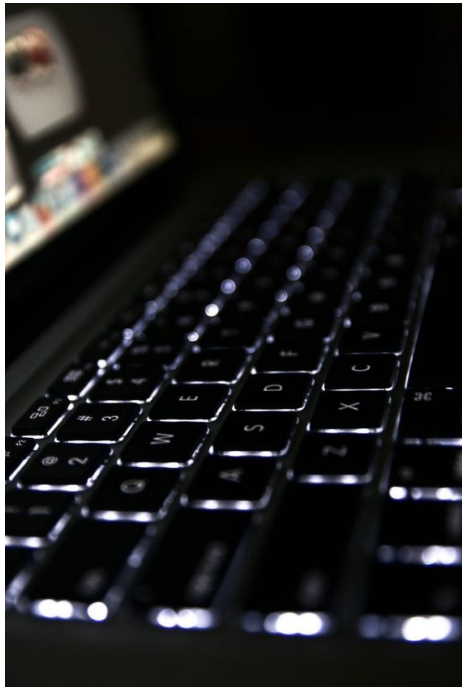
Insere um elemento na sequência no índice especificado

SINTAXE

```
seq1.insert(indice, valor)
```

```
[ ]  seq1.insert(0, 'w')
     seq1
```

```
['w', 'a', 'b', 'c', 'd', 'e', 'f']
```



INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Python - Aula 02

Métodos de Listas

Pop

Retorna e remove o item da sequência

SINTAXE

`seq1.pop()` ou `seq1.pop(indice)`

OBS : o argumento opcional *indice* tem como padrão -1, logo, o último elemento da sequência será removido e retornado

```
[ ] seq1.pop()

    'f'
```

```
[ ] seq1

    ['w', 'a', 'b', 'c', 'd', 'e']
```

```
[ ] seq1.pop(0)

    'w'
```

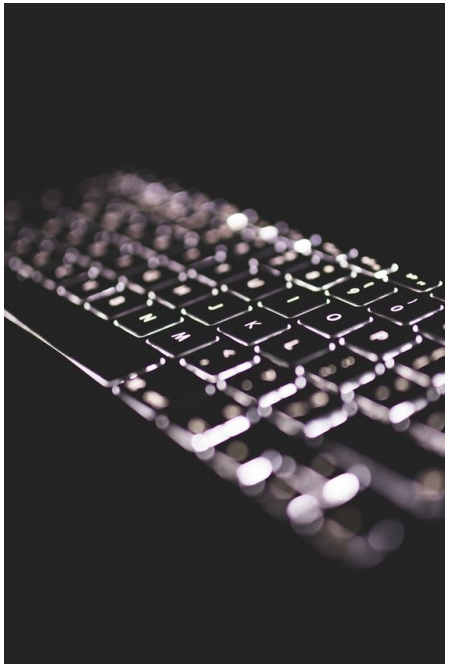
```
[ ] seq1

    ['a', 'b', 'c', 'd', 'e']
```

```
[ ] letra_removida = seq1.pop(1)
```

```
[ ] print(f'{letra_removida} foi removida da nossa lista')

    b foi removida da nossa lista
```



IN
INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Python - Aula 02

Métodos de Listas

Remove

Remove a primeira ocorrência do elemento na sequência

SINTAXE

`seq1.remove(elemento)`

OBS : este método "levanta" uma exceção *ValueError* quando o elemento não é encontrado na sequência

```
[ ]  seq1
```

```
['a', 'c', 'd', 'e']
```

```
[ ]  seq1.remove('d')
```

```
[ ]  seq1
```

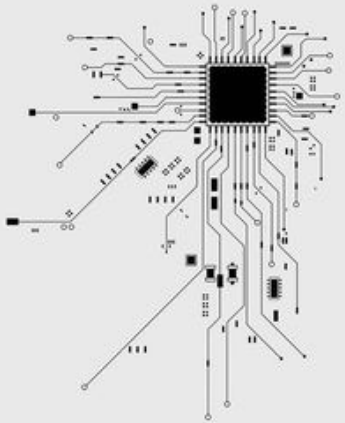
```
['a', 'c', 'e']
```

```
[ ]  'f' in seq1
```

```
False
```

```
[ ]  if 'f' in seq1:
       seq1.remove('f')
     else:
       print(f'não tá na lista')
```

```
não tá na lista
```



INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Métodos de Listas

Reverse

Inverte os elementos da sequência *in-place*, ou seja, não há a necessidade de reatribuição

```
[ ]  seq1
```

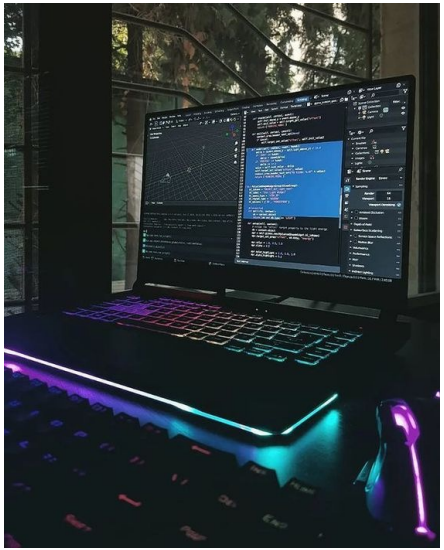
```
['a', 'c', 'e']
```

```
[ ]  seq1.reverse()
```

```
[ ]  seq1
```

```
['e', 'c', 'a']
```

Python - Aula 02



Métodos de Listas

Sort

Este método classifica a lista *in-place*.

Suporta 2 argumentos que **só** podem ser passados como *argumentos nomeados*

- **key** : especifica uma função de um argumento que é usado para extrair uma chave de comparação de cada elemento da lista

A chave correspondente a cada item na lista é calculada uma vez e depois usada para todo o processo de classificação.

O valor padrão `None` significa que os itens da lista são classificados diretamente sem calcular um valor de chave separado

- **reverse** : é um valor *booleano*. Se definido como **True**, então os elementos da lista são classificados como se cada comparação estivesse invertida

Python - Aula 02

Métodos de Listas

Sort

Este método modifica a sequência *in-place* para economizar espaço ao classificar uma sequência grande.

```
[ ] seq1
```

```
['e', 'c', 'a']
```

```
[ ] seq1.sort()
```

```
[ ] seq1
```

```
['a', 'c', 'e']
```

```
[ ] seq1.sort(reverse=True)
```

```
[ ] seq1
```

```
['e', 'c', 'a']
```

```
[ ] nomes = ['Cinthia', 'Camila', 'Amanda', 'Gisele']
    nomes.sort()
    nomes
```

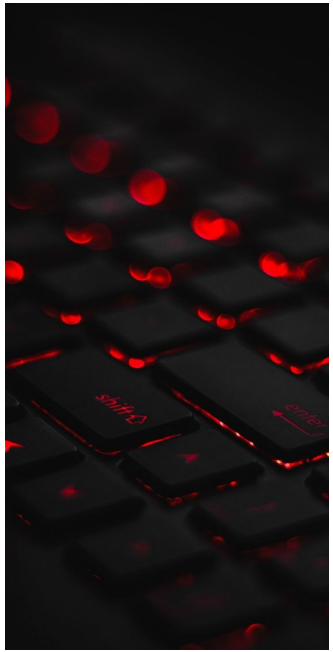
```
['Amanda', 'Camila', 'Cinthia', 'Gisele']
```

```
[ ] nomes = ['Cinthia', 'Camila', 'amanda', 'Gisele']
    nomes.sort()
    nomes
```

```
['Camila', 'Cinthia', 'Gisele', 'amanda']
```

```
[ ] nomes = ['Cinthia', 'Camila', 'amanda', 'Gisele']
    nomes.sort(key=str.lower)
    nomes
```

```
['amanda', 'Camila', 'Cinthia', 'Gisele']
```



INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Python - Aula 02

Tuplas

São sequências imutáveis, tipicamente usadas para armazenar coleções de dados heterogêneos.

Também são utilizadas para casos em que seja necessária uma sequência imutável de dados homogêneos.

As tuplas podem ser declaradas e inicializadas de várias maneiras como:

- usando um par de parêntesis para criarmos uma tupla vazia : `minha_tupla = ()`
- usando uma vírgula à direita para uma tupla *singleton*, ou seja, de apenas um único elemento : `minha_tupla = ('a',)` ou `minha_tupla = 'a',`
- separando os elementos com vírgulas : `minha_tupla = a, b, c` ou `minha_tupla = (a, b, c)`
- usando o construtor : `tuple()` ou `tuple(iterável)`

OBS : O iterável pode ser uma sequência, um contâiner que suporte iteração ou um objeto iterador.

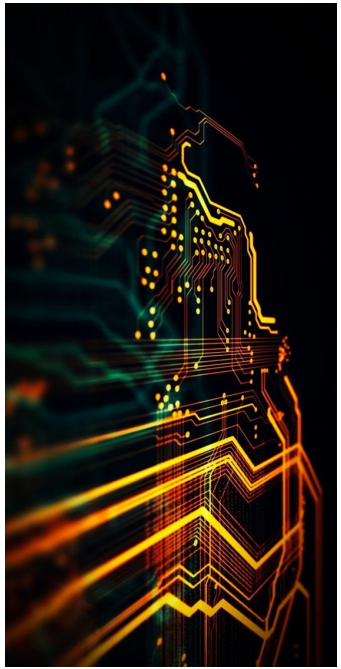
- Exemplo
  - `tuple('rafael')` retornará `('r', 'a', 'f', 'a', 'e', 'l')`
  - `tuple([1, 2, 3])` retornará `(1, 2, 3)`
  - `tuple()` retornará `()`

  ATENÇÃO : Se você reparou bem, é a vírgula quem faz uma tupla e não os parêntesis. Estes são opcionais, **exceto** nos casos em que desejamos criar uma tupla vazia, ou quando são necessários para evitar ambiguidades sintáticas.

  Veja este exemplo:

  - `func(a, b, c)` : é uma chamada da função `func` com três argumentos
  - `func((a, b, c))` : é uma chamada da função `func` com uma tupla de 3 elementos com um único argumento

DICA : as tuplas implementam todas as operações comuns de sequência. Retorna para a seção *Operações comuns de sequências* e relembre!



Python - Aula 02

Desempacotamento

Vimos que uma tupla pode armazenar tipos variados de dados. O acesso a eles se dá pelo índice e como sabemos, python indexa suas estruturas compostas a partir do zero.

Mas, também, podemos desembrulhar ou desempacotar uma tupla. Esta é uma operação beeemm poderosa que temos a mão quando trabalhamos com a linguagem **Python**.

```
[ ]  dados_de_contato = ('Will', 'Silva', 26, 'will@email.com')
```

```
[ ]  type(dados_de_contato)
```

```
tuple
```

```
[ ]  nome, sobrenome, idade, email = dados_de_contato
```

```
[ ]  print(nome, sobrenome)
     print(idade, email)
```

```
Will Silva
26 will@email.com
```

INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Python - Aula 02

Desempacotamento

ATENÇÃO : se podemos *desempacotar*, logo, podemos *empacotar* diferentes variáveis em uma tupla

```
[ ]  turma = 315
     modulo = 'Python'
     instituicao = 'Infinity School'

     info = turma, modulo, instituicao
```

```
[ ]  type(info)

     tuple
```

```
[ ]  info

     (315, 'Python', 'Infinity School')
```

IN

INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

Python - Aula 02

Operador _

E se quisermos pegar apenas alguns elementos da tupla e ignorar os demais?

Podemos fazer isso com o operador `_`

```
[ ]  nome, sobrenome, _, _ = dados_de_contato
```

```
[ ]  nome, sobrenome = dados_de_contato
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-42-858f8baa79e1> in <module>()
----> 1 nome, sobrenome = dados_de_contato

ValueError: too many values to unpack (expected 2)
```

SEARCH STACK OVERFLOW

```
[ ]  print(nome, sobrenome)
```

```
Rafael Puyau
```

```
[ ]  nome, _, idade, _ = dados_de_contato
```

```
[ ]  nome
```

```
'Rafael'
```

```
[ ]  idade
```

```
46
```

IN

INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

Expressão estrelada

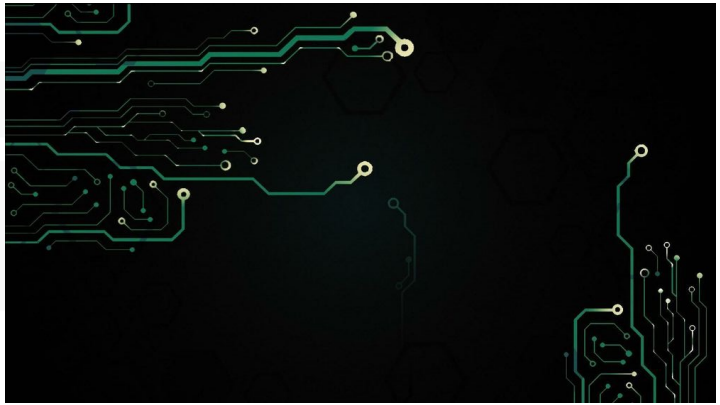
Mas, e se quisermos pegar somente o nome e sobrenome, deixando as demais informações "empactadas"?

Nestes casos, devemos usar a expressão estrela

```
[ ]  nome, sobrenome, *demais_info = dados_de_contato
```

```
[ ]  print(nome, sobrenome)
     print(type(demais_info))
     print(demais_info)
```

```
Rafael Puyau
<class 'list'>
[46, 'puyau@protonmail.com']
```



Python - Aula 02

Aprofundando conhecimento

Agora você já possui conhecimento suficiente para trabalhar **confortavelmente** com estruturas compostas mutáveis (*listas*) e imutáveis (*tuplas*) em python.

Desta forma, podemos avançar um pouco em nossos estudos!

INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Python - Aula 02

Enumerate()

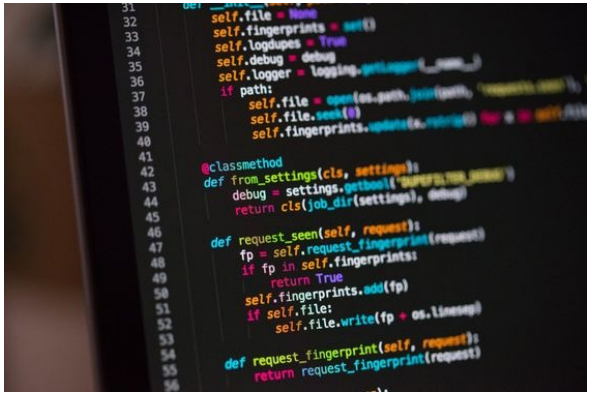
É uma função *built-in* do python que basicamente devolve um objeto enumerado.

SINTAXE

`enumerate(iterável)`

OBS : O *iterável* deve ser uma sequência, um iterador ou algum outro objeto que suporte iteração.

O retorno será uma tupla contendo a contagem e os valores obtidos na iteração sobre o iterável.



```
[ ]  nomes = ['Cinthia', 'Amanda', 'Camila', 'Gisele']
     enumerate(nomes)
```

```
<enumerate at 0x7fa3cec925f0>
```

```
[ ]  list(enumerate(nomes))
```

```
[(0, 'Cinthia'), (1, 'Amanda'), (2, 'Camila'), (3, 'Gisele')]
```

```
[ ]  for indice, nome in list(enumerate(nomes)):
       print(indice, nome)
```

```
0 Cinthia
1 Amanda
2 Camila
3 Gisele
```

Quando usar então?

Quando queremos pegar o índice de um elemento e seu valor ao mesmo tempo

INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Python - Aula 02

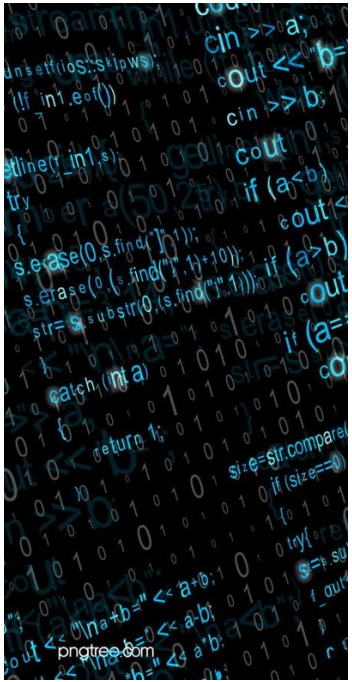
Zip

É uma função *built-in* do python que retorna um iterador de tuplas, onde a enésima tupla contém o enésimo elemento e cada um dos iteráveis do argumento.

Esta função é preguiçosa, ou seja, os elementos não serão processados até que o iterável seja iterado por um *loop for* ou por uma *lista*.

ATENÇÃO : Vale considerar que os iteráveis passados para zip() podem ter comprimentos diferentes; às vezes por design e às vezes por causa de um bug no código que preparou esses iteráveis.

A linguagem **Python** nos oferece três abordagens diferentes para lidar com este problema:



INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Python - Aula 02

Zip

Quando o iterável mais curto ou menor se esgota

Por padrão, `zip()` para quando o iterável mais curto se esgota. Ele irá ignorar os itens restantes nos iteráveis mais longos, cortando o resultado para o comprimento do iterável mais curto.

```
[ ]  list(zip(range(3), ['Python', 'R']))
```

```
[(0, 'Python'), (1, 'R')]
```

INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Python - Aula 02

Zip

Quando os iteráveis possuem o mesmo tamanho [opção **strict**]

`zip()` é frequentemente usado em casos onde os iteráveis são considerados de tamanho igual.

Nestes casos, é recomendado usar a opção `strict=True`.

```
[ ]  # Este código roda na versão 3.10 onde o argumento strict foi adicionado
     #list(zip(range(3), ['Python', 'R', 'SQL'], strict=True))
```

Ao contrário do comportamento padrão, ele verifica se os comprimentos dos iteráveis são idênticos, levantando uma exceção `ValueError` se não forem.

Sem o argumento `strict=True`, qualquer bug que resulte em iteráveis de diferentes comprimentos será silenciado, possivelmente se manifestando como um bug difícil de encontrar em outra parte do programa

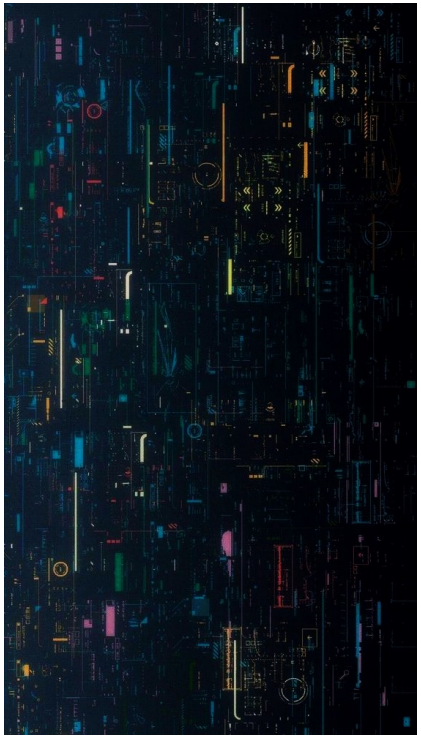
```
[ ]  nomes = ['Cinthia', 'Amanda', 'Camila', 'Gisele']
     emails = ['cinthia@email.com', 'amanda@email.com', 'camila@email.com', 'gisele@email.com']
```

```
[ ]  list(zip(nomes, emails))
```

```
[('Cinthia', 'cinthia@email.com'),
 ('Amanda', 'amanda@email.com'),
 ('Camila', 'camila@email.com'),
 ('Gisele', 'gisele@email.com')]
```

```
[ ]  for nome, email in list(zip(nomes, emails)):
       print(f'{nome} tem este email: {email}')
```

```
Cinthia tem este email: cinthia@email.com
Amanda tem este email: amanda@email.com
Camila tem este email: camila@email.com
Gisele tem este email: gisele@email.com
```



INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Python - Aula 02

Hora de praticar

- Faça um programa que leia o nome de uma pessoa e imprima o mesmo invertido
- Faça um programa que leia 4 notas de um aluno e imprima sua média. Após a média ser calculada, informe se o aluno foi ou não aprovado.
    a. Aprovado --- média maior ou igual a 7
    b. Reprovado --- média menor que 5
    c. Em recuperação --- média entre 5 e 7
- Faça um programa que leia 10 números e depois mostre o maior e o menor números lidos
- Faça um programa que leia 10 números inteiros e separe os mesmos em pares e ímpares. Mostre quantos pares foram lidos, bem como o maior e o menor número par. Faça o mesmo para os ímpares.
- Faça um programa que leia 10 números inteiros e imprima a lista ordenada em ordem crescente e decrescente.
- Faça um programa que leia o nome de 4 vendedores e o valor total de venda que cada um realizou. Imprima na tela os 2 vendedores que mais venderam, ordem decrescente.
- Faça um programa que leia os nomes dos 3 nadadores que subirão ao pódio na ordem do primeiro colocado para o terceiro. Imprima na tela a posição do nadador e seu nome.

Você pode conferir o gabarito após baixar o arquivo notebook que está disponível para download, na página da aula.

Você concluiu a aula 02 do seu módulo de Python.
Continue praticando e até a próxima aula!

IN

INFINITY SCHOOL

VISUAL ART CREATIVE CENTER