FIAP

FIAP

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

DISRUPTIVE ARCHITECTURES: IOT, IOB & IA

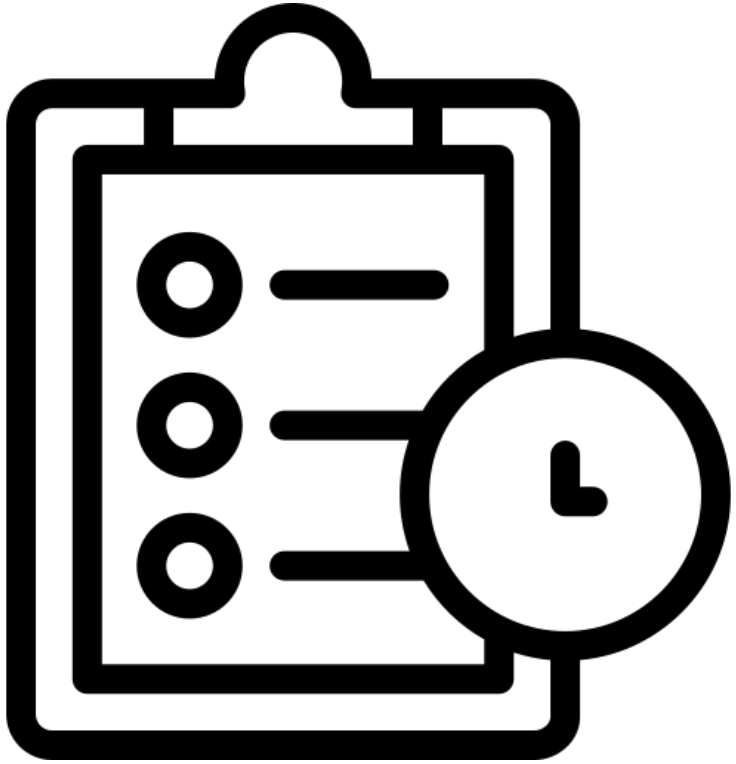
01 – Introdução ao Python

Prof. Airton Y. C. Toyofuku

profairton.toyofuku@fiap.com.br

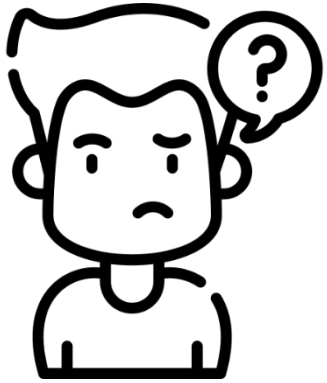
Agenda

- Introdução;
- Ambiente de desenvolvimento;
- Variáveis e operações;
- Strings e operações;
- Operadores lógicos;
- Estruturas de condição;
- Estruturas de repetição;
- Listas;
- Tuplas;
- Dicionários;
- Funções;
- Exercicícos;



Introdução

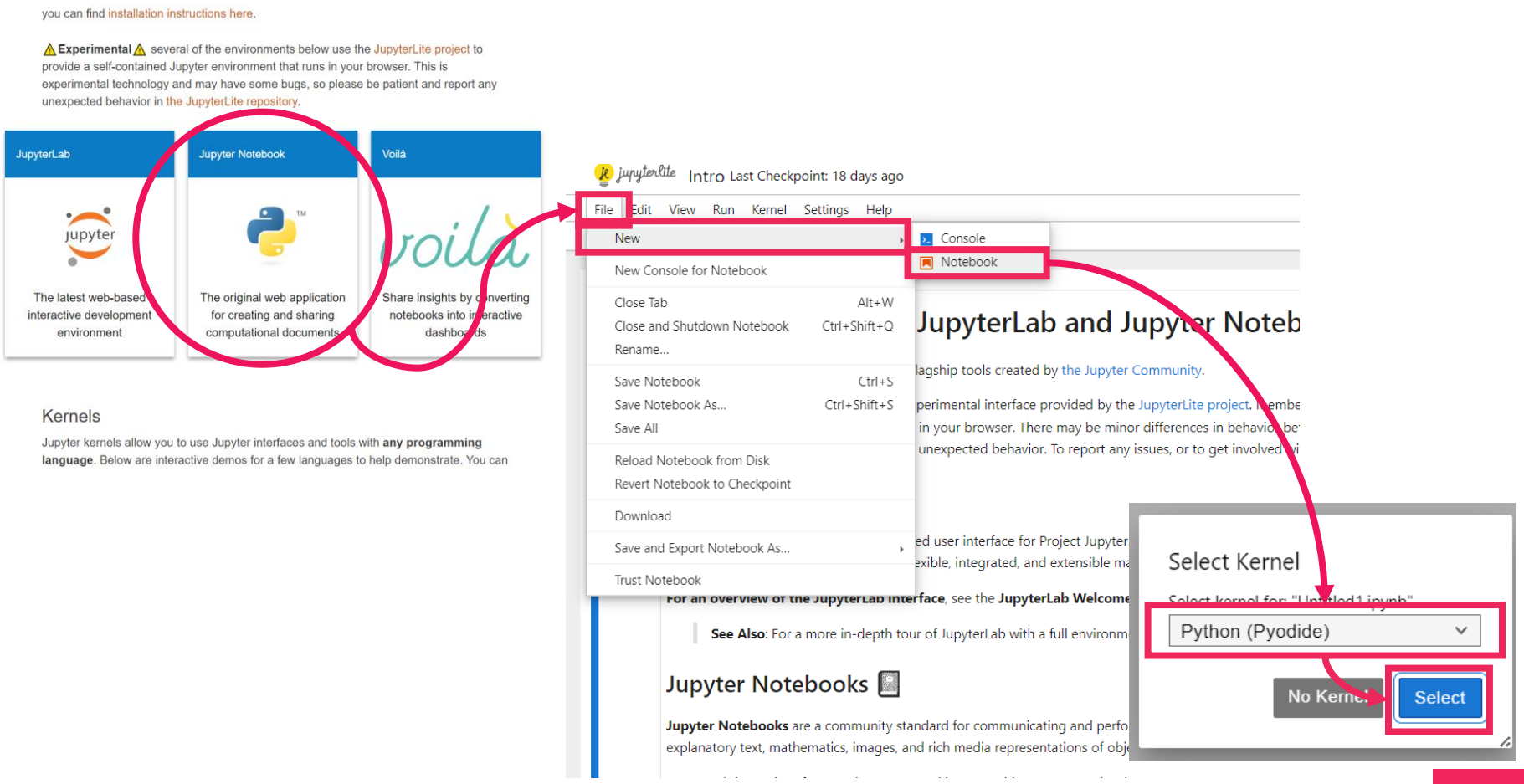
O que é o Python?



- Linguagem de programação de alto nível e interpretada;
- Possui uma sintaxe simples e fácil de ler e entender;
- Suporta múltiplos paradigmas (OOP, Funcional e Procedural);
- Grande váridade de bibliotecas e uma comunidade ativa;
- Disponível em WIN, Linux e Mac OS;
- Aplicada em processos de Automação, Análise de Dados, Visão Computacional e Inteligência Artificial;
- Criado em 1991 por Guido van Rossum e batizada em homenagem ao grupo de comédia Monty Python's Flying Circus;

Ambiente de Desenvolvimento

https://jupyter.org/try



Variáveis e Operações

FIAP

❖ Tipos de variáveis:

- Números Inteiros (int);
- Números em ponto flutuante (float);
- Strings (str)
- Números Complexos (complex)
- Booleanos (bool)

```
[15]: a = 10

      b = 3.145

      c = "Eu sou um exemplo de String"

      d = 3 + 20j

      e = True
```

❖ Operações básicas:

- Soma;
- Subtração;
- Multiplicação;
- Divisão;
- Resto;
- Exponenciação;

```
[17]: soma = a + b
      subtracao = 22 - a
      multiplicacao = b * 2
      divisao = a / 2
      resto = a % 3
      expQuadrado = 10 ** 2
      expCubo = 10 ** 3

      print(soma)
      print(subtracao)
      print(multiplicacao)
      print(divisao)
      print(resto)
      print(expQuadrado)
      print(expCubo)
```

```
13.145
12
6.29
5.0
1
100
1000
```

Strings e Operações

- Qual o conteúdo de s7?

```
s1 = "KA"
s2 = "ME"
s3 = "HA"
s4 = s2
s5 = s3 + s3[1:]*5

s6 = s1 + s2 + '-' + s3 + s4 + '-' + s5

s7 = s6 + "!!!!!!"

print(s7)
```

Operadores lógicos

Operador	Significado	Exemplo

Operadores lógicos

Operador	Significado	Exemplo
not		

Operadores lógicos

Operador	Significado	Exemplo

not	“nega” ou “inverte” a situação	`x = True`
		`y = not x`
		`print(x)`
		`print(y)`
		
		`True`
		`False`

Operadores lógicos

Operador	Significado	Exemplo

not	“nega” ou “inverte” a situação	`x = True`
		`y = not x`
		`print(x)`
		`print(y)`
		
		`True`
		`False`

and		

Operadores lógicos

Operador	Significado	Exemplo

not	“nega” ou “inverte” a situação	`x = True`
		`y = not x`
		`print(x)`
		`print(y)`
		
		`True`
		`False`

and	Usado para verificar se duas ou mais condições são verdadeiras
	Só mostra “true” se todas as situações forem “verdadeiras”
		`a = 10`
		`b = 11`
		`if((a==10) and (b==11)):`
		`  print("true")`
		`else:`
		`  print("false")`
		
		`true`

Operadores lógicos

FIAP

Operador	Significado	Exemplo

not	"nega" ou "inverte" a situação	`x = True`
		`y = not x`
		`print(x)`
		`print(y)`
		
		`True`
		`False`

and	Usado para verificar se duas ou mais condições são verdadeiras
	Só mostra "true" se todas as situações forem "verdadeiras"
		`a = 10`
		`b = 11`
		`if((a==10) and (b==11)):`
		`  print("true")`
		`else:`
		`  print("false")`
		
		`true`

or		

Operadores lógicos

FIAP

Operador	Significado	Exemplo

not	"nega" ou "inverte" a situação	`x = True`
		`y = not x`
		`print(x)`
		`print(y)`
		
		`True`
		`False`

and	Usado para verificar se duas ou mais condições são verdadeiras
	Só mostra "true" se todas as situações forem "verdadeiras"
		`a = 10`
		`b = 11`
		`if((a==10) and (b==11)):`
		`  print("true")`
		`else:`
		`  print("false")`
		
		`true`

or	Usado para verificar se pelo menos uma condição é verdadeira
	Mostra "true" se uma ou mais condições forem "verdadeiras"
		`a = 10`
		`b = 11`
		`if((a==10) or (b==0)):`
		`  print("true")`
		`else:`
		`  print("false")`
		
		`true`

Estruturas de condição

FIAP

- Condicional if:

```
[1]: a = 10
     if(a > 9):
         print("a é maior que 9")
```

```
a é maior que 9
```

- Condicional if – else:

```
[2]: a = 8
     if(a > 9):
         print("a é maior que 9")
     else:
         print("a é menor que 9")
```

```
a é menor que 9
```

- Condicional if – elif – else:

```
[3]: nota = 5
     if(nota >= 6 and nota <= 10):
         print("Aprovado")
     elif(nota < 6 and nota >= 4):
         print("Está de Exame")
     else:
         print("Reprovado")
```

```
Está de Exame
```

Estruturas de repetição

❖ Loop while:

```
[5]: x = 0
     while( x <= 10):
         print(x)
         x += 1
```

```
0
1
2
3
4
5
6
7
8
9
10
```

- Enquanto (while) o x for menor ou igual a 10, o loop será realizado;
- Se o x inicial for maior que 10, o loop nem é inicializado!

❖ Loop do...while

```
[6]: x = 0
     while True:
         print(x)
         x += 2
         if(x > 10):
             break
```

```
0
2
4
6
8
10
```

- O loop é iniciado independente do valor inicial de x;
- Quando o x atingir um valor maior que 10, ou se ele já for maior de 10, o loop é encerrado com o comando **break**

Estruturas de repetição

❖ Loop for:

```
[7]: x = 0
     for i in range(10):
         print(x)
         x +=1
```

```
0
1
2
3
4
5
6
7
8
9
```

- Para (for) cada valor de i, sendo que i vai de 0 até 10, mostre o valor de x e em seguida incremente o x;

```
[8]: salarios = [1000, 2000, 2500, 500]
     total = 0
     for s in salarios:
         total += s

     total
```

```
[8]: 6000
```

- Outra forma de usar o loop for, usando a quantidade de itens em um vetor;
- O loop será realizado 4 vezes, sendo essa a quantidade de valores no vetor!

Listas

- São estruturas de dados compostas;
- Permitem armazenar uma coleção de elementos de diferentes tipos;
- A declaração de uma lista é feita entre colchetes [];
- O acesso aos elementos da lista é feito através de seu índice;
- O primeiro elemento tem índice 0, o segundo tem índice 1 e assim por diante, até n-1;
- Pode se alterar o valor de um elemento de uma lista, acessando seu índice e atribuindo um novo valor;
- Pode ser incluir novos itens na lista usando o método “append()”;
- Pode se verificar o tamanho da lista com o operador “len”;
- Pode ser remover um item da lista usando o método “remove()”;

```
[3]: cavaleirosBronze = ["pegaso", "dragao", "cisne", "andromeda", "fenix"]
     cavaleirosBronze

[3]: ['pegaso', 'dragao', 'cisne', 'andromeda', 'fenix']
```

Listas

- São estruturas de dados compostas;
- Permitem armazenar uma coleção de elementos de diferentes tipos;
- A declaração de uma lista é feita entre colchetes [];
- O acesso aos elementos da lista é feito através de seu índice;
- O primeiro elemento tem índice 0, o segundo tem índice 1 e assim por diante, até n-1;
- Pode se alterar o valor de um elemento de uma lista, acessando seu índice e atribuindo um novo valor;
- Pode ser incluir novos itens na lista usando o método “append()”;
- Pode se verificar o tamanho da lista com o operador “len”;
- Pode ser remover um item da lista usando o método “remove()”;

```
[3]: cavaleirosBronze = ["pegaso", "dragao", "cisne", "andromeda", "fenix"]
     cavaleirosBronze

[3]: ['pegaso', 'dragao', 'cisne', 'andromeda', 'fenix']

[4]: cavaleirosBronzeCoadjuvantes = ["unicornio", "lobo", "urso", "hydra", "leonete"]
     cavaleirosBronzeCoadjuvantes

[4]: ['unicornio', 'lobo', 'urso', 'hydra', 'leonete']
```

Listas

- São estruturas de dados compostas;
- Permitem armazenar uma coleção de elementos de diferentes tipos;
- A declaração de uma lista é feita entre colchetes [];
- O acesso aos elementos da lista é feito através de seu índice;
- O primeiro elemento tem índice 0, o segundo tem índice 1 e assim por diante, até n-1;
- Pode se alterar o valor de um elemento de uma lista, acessando seu índice e atribuindo um novo valor;
- Pode ser incluir novos itens na lista usando o método “append()”;
- Pode se verificar o tamanho da lista com o operador “len”;
- Pode ser remover um item da lista usando o método “remove()”;

```
[3]: cavaleirosBronze = ["pegaso", "dragao", "cisne", "andromeda", "fenix"]
     cavaleirosBronze

[3]: ['pegaso', 'dragao', 'cisne', 'andromeda', 'fenix']

[4]: cavaleirosBronzeCoadjuvantes = ["unicornio", "lobo", "urso", "hydra", "leonete"]
     cavaleirosBronzeCoadjuvantes

[4]: ['unicornio', 'lobo', 'urso', 'hydra', 'leonete']

[19]: cavaleirosBronze[3]

[19]: 'andromeda'
```

Listas

- São estruturas de dados compostas;
- Permitem armazenar uma coleção de elementos de diferentes tipos;
- A declaração de uma lista é feita entre colchetes [];
- O acesso aos elementos da lista é feito através de seu índice;
- O primeiro elemento tem índice 0, o segundo tem índice 1 e assim por diante, até n-1;
- Pode se alterar o valor de um elemento de uma lista, acessando seu índice e atribuindo um novo valor;
- Pode ser incluir novos itens na lista usando o método “append()”;
- Pode se verificar o tamanho da lista com o operador “len”;
- Pode ser remover um item da lista usando o método “remove()”;

```
[3]: cavaleirosBronze = ["pegaso", "dragao", "cisne", "andromeda", "fenix"]
     cavaleirosBronze

[3]: ['pegaso', 'dragao', 'cisne', 'andromeda', 'fenix']

[4]: cavaleirosBronzeCoadjuvantes = ["unicornio", "lobo", "urso", "hydra", "leonete"]
     cavaleirosBronzeCoadjuvantes

[4]: ['unicornio', 'lobo', 'urso', 'hydra', 'leonete']

[19]: cavaleirosBronze[3]

[19]: 'andromeda'

[20]: cavaleirosBronze[3] = "Shun"
      cavaleirosBronze

[20]: ['pegaso', 'dragao', 'cisne', 'Shun', 'fenix']
```

Listas

- São estruturas de dados compostas;
- Permitem armazenar uma coleção de elementos de diferentes tipos;
- A declaração de uma lista é feita entre colchetes [];
- O acesso aos elementos da lista é feito através de seu índice;
- O primeiro elemento tem índice 0, o segundo tem índice 1 e assim por diante, até n-1;
- Pode se alterar o valor de um elemento de uma lista, acessando seu índice e atribuindo um novo valor;
- Pode ser incluir novos itens na lista usando o método “append()”;
- Pode se verificar o tamanho da lista com o operador “len”;
- Pode ser remover um item da lista usando o método “remove()”;

```
[3]: cavaleirosBronze = ["pegaso", "dragao", "cisne", "andromeda", "fenix"]
     cavaleirosBronze

[3]: ['pegaso', 'dragao', 'cisne', 'andromeda', 'fenix']

[4]: cavaleirosBronzeCoadjuvantes = ["unicornio", "lobo", "urso", "hydra", "leonete"]
     cavaleirosBronzeCoadjuvantes

[4]: ['unicornio', 'lobo', 'urso', 'hydra', 'leonete']

[19]: cavaleirosBronze[3]

[19]: 'andromeda'

[20]: cavaleirosBronze[3] = "Shun"
      cavaleirosBronze

[20]: ['pegaso', 'dragao', 'cisne', 'Shun', 'fenix']

[21]: cavaleirosBronze.append(cavaleirosBronzeCoadjuvantes)

[22]: cavaleirosBronze

[22]: ['pegaso',
       'dragao',
       'cisne',
       'Shun',
       'fenix',
       ['unicornio', 'lobo', 'urso', 'hydra', 'leonete']]
```

Listas

- São estruturas de dados compostas;
- Permitem armazenar uma coleção de elementos de diferentes tipos;
- A declaração de uma lista é feita entre colchetes [];
- O acesso aos elementos da lista é feito através de seu índice;
- O primeiro elemento tem índice 0, o segundo tem índice 1 e assim por diante, até n-1;
- Pode se alterar o valor de um elemento de uma lista, acessando seu índice e atribuindo um novo valor;
- Pode ser incluir novos itens na lista usando o método “append()”;
- Pode se verificar o tamanho da lista com o operador “len”;
- Pode ser remover um item da lista usando o método “remove()”;

```
[3]: cavaleirosBronze = ["pegaso", "dragao", "cisne", "andromeda", "fenix"]
     cavaleirosBronze

[3]: ['pegaso', 'dragao', 'cisne', 'andromeda', 'fenix']

[4]: cavaleirosBronzeCoadjuvantes = ["unicornio", "lobo", "urso", "hydra", "leonete"]
     cavaleirosBronzeCoadjuvantes

[4]: ['unicornio', 'lobo', 'urso', 'hydra', 'leonete']

[19]: cavaleirosBronze[3]

[19]: 'andromeda'

[20]: cavaleirosBronze[3] = "Shun"
      cavaleirosBronze

[20]: ['pegaso', 'dragao', 'cisne', 'Shun', 'fenix']

[21]: cavaleirosBronze.append(cavaleirosBronzeCoadjuvantes)

[22]: cavaleirosBronze

[22]: ['pegaso',
       'dragao',
       'cisne',
       'Shun',
       'fenix',
       ['unicornio', 'lobo', 'urso', 'hydra', 'leonete']]

[23]: len (cavaleirosBronzeCoadjuvantes)

[23]: 5
```

Listas

- São estruturas de dados compostas;
- Permitem armazenar uma coleção de elementos de diferentes tipos;
- A declaração de uma lista é feita entre colchetes [];
- O acesso aos elementos da lista é feito através de seu índice;
- O primeiro elemento tem índice 0, o segundo tem índice 1 e assim por diante, até n-1;
- Pode se alterar o valor de um elemento de uma lista, acessando seu índice e atribuindo um novo valor;
- Pode ser incluir novos itens na lista usando o método “append()”;
- Pode se verificar o tamanho da lista com o operador “len”;
- Pode ser remover um item da lista usando o método “remove()”;

```
[3]: cavaleirosBronze = ["pegaso", "dragao", "cisne", "andromeda", "fenix"]
     cavaleirosBronze

[3]: ['pegaso', 'dragao', 'cisne', 'andromeda', 'fenix']

[4]: cavaleirosBronzeCoadjuvantes = ["unicornio", "lobo", "urso", "hydra", "leonete"]
     cavaleirosBronzeCoadjuvantes

[4]: ['unicornio', 'lobo', 'urso', 'hydra', 'leonete']

[19]: cavaleirosBronze[3]

[19]: 'andromeda'

[20]: cavaleirosBronze[3] = "Shun"
      cavaleirosBronze

[20]: ['pegaso', 'dragao', 'cisne', 'Shun', 'fenix']

[21]: cavaleirosBronze.append(cavaleirosBronzeCoadjuvantes)

[22]: cavaleirosBronze

[22]: ['pegaso',
       'dragao',
       'cisne',
       'Shun',
       'fenix',
       ['unicornio', 'lobo', 'urso', 'hydra', 'leonete']]

[23]: len (cavaleirosBronzeCoadjuvantes)

[23]: 5

[24]: cavaleirosBronze.remove(cavaleirosBronzeCoadjuvantes)
      cavaleirosBronze

[24]: ['pegaso', 'dragao', 'cisne', 'Shun', 'fenix']
```

Listas

- Podemos concatenar duas listas, gerando uma nova lista;

```
[27]: todosCavaleirosBronze = cavaleirosBronze + cavaleirosBronzeCoadjuvantes
      todosCavaleirosBronze
```

```
[27]: ['pegaso',
       'dragao',
       'cisne',
       'andromeda',
       'fenix',
       'unicornio',
       'lobo',
       'urso',
       'hydra',
       'leonete']
```

- Também podemos percorre a lista livremente, de várias maneiras diferentes;

```
[28]: todosCavaleirosBronze[2:4]
```

```
[28]: ['cisne', 'andromeda']
```

```
[29]: todosCavaleirosBronze[6:]
```

```
[29]: ['lobo', 'urso', 'hydra', 'leonete']
```

```
[30]: todosCavaleirosBronze[:6]
```

```
[30]: ['pegaso', 'dragao', 'cisne', 'andromeda', 'fenix', 'unicornio']
```

```
[31]: todosCavaleirosBronze[-1:]
```

```
[31]: ['leonete']
```

Listas

FIAP

❖ Criando Listas com a função range()

- A função **range()** define um interval de valores inteiros;
- Associada com a função **list()**, cria uma lista com valores dentro do interval definido pela **range();**
- A função **range()** pode ter até três parâmetros:
  - **range(n):** gera um intervalo de 0 a n-1;
  - **range(i, n):** gera um intervalo de i a n-1;
  - **range(i, n, p):** gera um intervalo de i a n-1 com passo p

```
lista = list(range(10))
lista
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
lista = list(range(5,10))
lista
```

```
[5, 6, 7, 8, 9]
```

```
lista = list(range(2, 10, 2))
lista
```

```
[2, 4, 6, 8]
```

Tuplas

- São estruturas de dados compostas semelhantes a lista;
- Permitem armazenar uma coleção de elementos de diferentes tipos;
- A declaração de uma tupla é feita entre parênteses ();
- O acesso aos elementos da lista é feito através de seu índice;
- O primeiro elemento tem índice 0, o segundo tem índice 1 e assim por diante, até n-1;
- Os valores dos elementos das tuplas são imutáveis. Uma vez criada, não dá para modificar;
- São mais rápidas que as listas, pois seu processamento é mais eficiente;

```
: cavaleriosBronze = ("pegaso", "dragao", "cisne", "andromeda", "fenix")
  cavaleriosBronze

: ('pegaso', 'dragao', 'cisne', 'andromeda', 'fenix')

: cavaleriosBronze[3]

: 'andromeda'

: cavaleriosBronze[3] = "Shun"

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[40], line 1
----> 1 cavaleriosBronze[3] = "Shun"

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

Tuplas

FIAP

- Empacotamento e Desempacotamento de tuplas
  - Empacotamento é a técnica que permite que várias variáveis sejam atribuídas a uma única tupla;

```
: a = "pegaso"
  b = "dragao"
  c = "cisne"
  d = "andromeda"
  e = "fenix"
  cavaleirosBronze = (a,b,c,d,e,)
  cavaleirosBronze

: ('pegaso', 'dragao', 'cisne', 'andromeda', 'fenix')
```

  - Desempacotamento é a técnica que permite que os valores armazenados em uma tupla sejam atribuídos a várias variáveis;

```
Seiya, Shiryu, Hyoga, Shun, Ikki = cavaleriosBronze
print("Seiya=", Seiya, "Shiryu=", Shiryu, "Hyoga=" , Hyoga, "Shun=", Shun, "Ikki=", Ikki)
```

```
Seiya= pegaso Shiryu= dragao Hyoga= cisne Shun= andromeda Ikki= fenix
```

Dicionários

- São conjuntos de dados estruturados entre Chave e Valor;
- Cada Valor é identificado por uma Chave;
- Tanto Chave quanto Valor podem ser de qualquer tipo;
- A declaração de um dicionário é através de {};
- Cada Valor-Chave é separado por virgula (,);

```
dicionario = {"nome": "João", "idade": 30, "cidade": "São Paulo"}
dicionario
```

```
{'nome': 'João', 'idade': 30, 'cidade': 'São Paulo'}
```

- Para acessar um valor, basta especificar a sua chave entre [ ];

```
idade = dicionario["idade"]
idade
```

```
30
```

- Para atribuir um novo Valor-Chave, basta atribui-lo:

```
dicionario["profissão"] = "programador"
dicionario
```

```
{'nome': 'João',
 'idade': 30,
 'cidade': 'São Paulo',
 'profissão': 'programador'}
```

Dicionários


```
cavaleirosOuro = {"Mu":"Aries", "Saga":"Gemeos", "Aiolia":"Leao", "Milo":"Escorpiao"}
cavaleirosOuro
```

```
{'Mu': 'Aries', 'Saga': 'Gemeos', 'Aiolia': 'Leao', 'Milo': 'Escorpiao'}
```

Dicionários

FIAP

Comando	Descrição	Exemplo

Dicionários

Comando	Descrição	Exemplo
del		

Dicionários

Comando	Descrição	Exemplo

del	Exclui um item informando a chave	`del cavaleirosOuro['Mu']`
		`cavaleirosOuro`
		`{'Saga': 'Gemeos', 'Aiolia': 'Leao', 'Milo': 'Escorpiao'}`

Dicionários

Comando	Descrição	Exemplo

del	Exclui um item informando a chave	`del cavaleirosOuro['Mu']`
		`cavaleirosOuro`
		`{'Saga': 'Gemeos', 'Aiolia': 'Leao', 'Milo': 'Escorpiao'}`

in		

Dicionários

Comando	Descrição	Exemplo

del	Exclui um item informando a chave	`del cavaleirosOuro['Mu']`
		`cavaleirosOuro`
		`{'Saga': 'Gemeos', 'Aiolia': 'Leao', 'Milo': 'Escorpiao'}`

in	Verifica se uma chave existe no dicionário	`'Seiya' in cavaleirosOuro` `False`

Dicionários

Comando	Descrição	Exemplo

del	Exclui um item informando a chave	`del cavaleirosOuro['Mu']`
		`cavaleirosOuro`
		`{'Saga': 'Gemeos', 'Aiolia': 'Leao', 'Milo': 'Escorpiao'}`

in	Verifica se uma chave existe no dicionário	`'Seiya' in cavaleirosOuro` `False`
keys()		

Dicionários

FIAP

Comando	Descrição	Exemplo

del	Exclui um item informando a chave	`del cavaleirosOuro['Mu']`
		`cavaleirosOuro`
		`{'Saga': 'Gemeos', 'Aiolia': 'Leao', 'Milo': 'Escorpiao'}`

in	Verifica se uma chave existe no dicionário	`'Seiya' in cavaleirosOuro` `False`
keys()	Obtém as chaves de um dicionário	`cavaleirosOuro.keys()` `dict_keys(['Saga', 'Aiolia', 'Milo'])`

Dicionários

Comando	Descrição	Exemplo

del	Exclui um item informando a chave	`del cavaleirosOuro['Mu']`
		`cavaleirosOuro`
		`{'Saga': 'Gemeos', 'Aiolia': 'Leao', 'Milo': 'Escorpiao'}`

in	Verifica se uma chave existe no dicionário	`'Seiya' in cavaleirosOuro` `False`
keys()	Obtém as chaves de um dicionário	`cavaleirosOuro.keys()` `dict_keys(['Saga', 'Aiolia', 'Milo'])`
values()		

Dicionários

FIAP

Comando	Descrição	Exemplo

del	Exclui um item informando a chave	`del cavaleirosOuro['Mu']`
		`cavaleirosOuro`
		`{'Saga': 'Gemeos', 'Aiolia': 'Leao', 'Milo': 'Escorpiao'}`

in	Verifica se uma chave existe no dicionário	`'Seiya' in cavaleirosOuro` `False`
keys()	Obtém as chaves de um dicionário	`cavaleirosOuro.keys()` `dict_keys(['Saga', 'Aiolia', 'Milo'])`
values()	Obtém os valores de um dicionário	`cavaleirosOuro.values()` `dict_values(['Gemeos', 'Leao', 'Escorpiao'])`

Dicionários

FIAP

Comando	Descrição	Exemplo

del	Exclui um item informando a chave	`del cavaleirosOuro['Mu']`
		`cavaleirosOuro`
		`{'Saga': 'Gemeos', 'Aiolia': 'Leao', 'Milo': 'Escorpiao'}`

in	Verifica se uma chave existe no dicionário	`'Seiya' in cavaleirosOuro` `False`
keys()	Obtém as chaves de um dicionário	`cavaleirosOuro.keys()` `dict_keys(['Saga', 'Aiolia', 'Milo'])`
values()	Obtém os valores de um dicionário	`cavaleirosOuro.values()` `dict_values(['Gemeos', 'Leao', 'Escorpiao'])`
items()		

Dicionários

Comando	Descrição	Exemplo

del	Exclui um item informando a chave	`del cavaleirosOuro['Mu']`
		`cavaleirosOuro`
		`{'Saga': 'Gemeos', 'Aiolia': 'Leao', 'Milo': 'Escorpiao'}`

in	Verifica se uma chave existe no dicionário	`'Seiya' in cavaleirosOuro` `False`
keys()	Obtém as chaves de um dicionário	`cavaleirosOuro.keys()` `dict_keys(['Saga', 'Aiolia', 'Milo'])`
values()	Obtém os valores de um dicionário	`cavaleirosOuro.values()` `dict_values(['Gemeos', 'Leao', 'Escorpiao'])`
items()	Retorna uma lista dos pares de tuplas de um dicionário (chave, valor)	`cavaleirosOuro.items()` `dict_items([('Saga', 'Gemeos'), ('Aiolia', 'Leao'), ('Milo', 'Escorpiao')])`

Funções

- São trechos de código que podem ser reutilizados várias vezes;
- São definidas usando a palavra chave **def**, seguida de um nome;
- A definição da função é concluída por uma lista de parâmetros entre parênteses e um bloco de código identado;
- A função pode retornar um resultado, ou somente realizar uma tarefa;

```
def soma(a, b):
    resultado = a + b
    return resultado
```

- Para usar essa função, basta chama-la no código, passando os números que devem ser somados como parâmetros a e b;

```
resultado = soma(10, 20)
```

```
resultado
```

```
30
```

Exercícios

1) Crie uma função que conte o número de ocorrências de uma determinada letra em uma string.
2) Crie uma função que verifique se uma string é um palíndromo.
3) Crie uma função que substitua todas as vogais de uma string por outra letra.
4) Crie uma função que conte quantas vezes cada letra aparece em uma string.
5) Crie uma função que calcule a soma dos elementos de uma lista usando um loop.
6) Crie uma função que encontre o maior número de uma lista.
7) Crie uma função que junte duas listas em uma única lista.
8) Crie uma função que verifique se todos os elementos de uma lista são iguais.
9) Crie uma tupla com os nomes dos personagens principais de Dragon Ball Z (Goku, Vegeta, Piccolo, Trunks, Gohan, Freeza, Cell e Majin Buu).
    a. Escreva uma função que exiba o personagem mais forte de acordo com o seu poder de luta, para isso, você deve atribuir um valor de luta para cada personagem, por exemplo Goku = 9999, Vegeta = 9000, etc;
    b. Modifique a função criada em a. para exibir todos os personagens com poder de luta igual ou superior a 9000;
    c. Escreva uma função que conte quantos personagens são poderosos o suficiente para lutar contra o Freeza (poder maior que 8000);
    d. Modifique a função em c. para retornar uma nova tupla apenas com os personagens poderosos para lutar contra Freeza;

Exercícios

1) Crie uma função que conte o número de ocorrências de uma determinada letra em uma string.

```python
def count_letter(string, letter):
    count = 0
    for char in string:
        if char == letter:
            count += 1
    return count

print(count_letter("banana", "a")) # output: 3
```

2) Crie uma função que verifique se uma string é um palíndromo.

```python
def is_palindrome(string):
    return string == string[::-1]

print(is_palindrome("racecar")) # output: True
```

Exercícios

FIAP

3) Crie uma função que substitua todas as vogais de uma string por outra letra.

```python
def replace_vowels(string, letter):
    vowels = "aeiouAEIOU"
    result = ""
    for char in string:
        if char in vowels:
            result += letter
        else:
            result += char
    return result

print(replace_vowels("hello", "x")) # output: "hxllx"
```

4) Crie uma função que conte quantas vezes cada letra aparece em uma string.

```python
def count_letters(string):
    result = {}
    for char in string:
        if char in result:
            result[char] += 1
        else:
            result[char] = 1
    return result

print(count_letters("hello")) # output: {'h': 1, 'e': 1, 'l': 2, 'o': 1}
```

Exercícios

FIAP

5) Crie uma função que calcule a soma dos elementos de uma lista usando um loop.

```python
def sum_list(list):
    result = 0
    for number in list:
        result += number
    return result

print(sum_list([1, 2, 3, 4, 5])) # output: 15
```

6) Crie uma função que encontre o maior número de uma lista.

```python
def find_max(list):
    max = list[0]
    for number in list:
        if number > max:
            max = number
    return max

print(find_max([1, 2, 3, 4, 5])) # output: 5
```

Exercícios

7) Crie uma função que junte duas listas em uma única lista.

```python
def join_lists(list1, list2):
    result = list1 + list2
    return result

print(join_lists([1, 2, 3], [4, 5, 6])) # output: [1, 2, 3, 4, 5, 6]
```

8) Crie uma função que verifique se todos os elementos de uma lista são iguais.

```python
def all_equal(list):
    first = list[0]
    for item in list:
        if item != first:
            return False
    return True

print(all_equal([1, 1, 1, 1, 1])) # output: True
```

Exercícios

9) Crie uma tupla com os nomes dos personagens principais de Dragon Ball Z (Goku, Vegeta, Piccolo, Trunks, Gohan, Freeza, Cell e Majin Buu).
   a. Escreva uma função que exiba o personagem mais forte de acordo com o seu poder de luta, para isso, você deve atribuir um valor de luta para cada personagem, por exemplo Goku = 9999, Vegeta = 9000, etc;
   b. Modifique a função criada em a. para exibir todos os personagens com poder de luta igual ou superior a 9000;
   c. Escreva uma função que conte quantos personagens são poderosos o suficiente para lutar contra o Freeza poder maior que 8000);
   d. Modifique a função em c. para retornar uma nova tupla apenas com os personagens poderosos para lutar contra Freeza;

```python
# Criação da tupla com os nomes dos personagens
personagens = ('Goku', 'Vegeta', 'Piccolo', 'Trunks', 'Gohan', 'Freeza', 'Cell', 'Majin Buu')

# Criação do dicionário com o nome do personagem e o seu poder de luta
poderes = {'Goku': 9999, 'Vegeta': 9000, 'Piccolo': 8000, 'Trunks': 7500, 'Gohan': 9000,
           'Freeza': 8000, 'Cell': 8500, 'Majin Buu': 8200}

# Função que exibe o personagem mais forte de acordo com o seu poder de luta
def personagem_mais_forte():
    mais_forte = max(poderes, key=poderes.get)
    print(f'O personagem mais forte é {mais_forte} com poder de luta de {poderes[mais_forte]}.')

# Função que exibe todos os personagens com um poder de luta igual ou superior a 9000
def personagens_fortes_9000():
    for personagem, poder in poderes.items():
        if poder >= 9000:
            print(f'{personagem} tem poder de luta de {poder}.')

# Função que conta quantos personagens são poderosos o suficiente para lutar contra o Freeza (poder de luta igual ou superior a 8000)
def conta_personagens_poderosos_contra_Freeza():
    contador = 0
    for personagem, poder in poderes.items():
        if poder >= 8000:
            contador += 1
    print(f'Existem {contador} personagens com poder de luta igual ou superior a 8000 e capazes de lutar contra o Freeza.')

# Função que retorna uma nova tupla com apenas os personagens poderosos o suficiente para lutar contra o Freeza
def tupla_personagens_poderosos_contra_Freeza():
    personagens_poderosos = []
    for personagem, poder in poderes.items():
        if poder >= 8000:
            personagens_poderosos.append(personagem)
    return tuple(personagens_poderosos)
```

```python
personagem_mais_forte()
personagens_fortes_9000()
conta_personagens_poderosos_contra_Freeza()
tupla_personagens_poderosos_contra_Freeza()
```

```
O personagem mais forte é Goku com poder de luta de 9999.
Goku tem poder de luta de 9999.
Vegeta tem poder de luta de 9000.
Gohan tem poder de luta de 9000.
Existem 7 personagens com poder de luta igual ou superior a 8000 e capazes de lutar contra o Freeza.
('Goku', 'Vegeta', 'Piccolo', 'Gohan', 'Freeza', 'Cell', 'Majin Buu')
```

FIAP

This presentation has been designed using images from Flaticon.com
Images from Monty Python's Flying Circle: BBC, 1969. Netflix, 2019
Imagens from Dragon Ball, Saint Seiya: Toei Animation