PYTHON BÁSICO

Prof. Peter Jandl Junior

4

DIA 4 [ROTEIRO]

- Listas
- Sequências : strings, tuplas e ranges
- Conjuntos
- Dicionários

Not so Long ago
In a galaxy that isn't very far away.....

4

LISTAS

29/10/2020

LISTAS

- É uma estrutura de dados que permite o armazenamento de muitos valores do mesmo tipo ou de tipos diferentes.

- Uma lista é uma sequência de valores onde é possível, dentre outra operações:
  - Adicionar novos elementos
  - Consultar elementos
  - Contar elementos armazenados
  - Remover elementos

- Para o Python, uma lista é um tipo embutido, **heterogêneo**, **mutável** e navegável.

LISTAS

- Definição de lista vazia:
  lista = [ ]
- Adição de elemento na lista:
  lista.**append**(1) # inteiro
  lista.**append**('Peter') # string
  lista.**append**(False) # lógico
  lista.**append**(4.44) # real
  lista.**append**("Jandl") # string
- Uma lista pode conter elementos de tipos misturados, mas pode conter elementos de um mesmo tipo.
  O uso é livre!

Listas são definidas com colchetes

Uma lista chamada **lista**

append adiciona um elemento na lista operada

```
>>> lista = [ ]
>>> lista.append(1) # inteiro
>>> lista.append('Peter') # string
>>> lista.append(False) # lógico
>>> lista.append(4.44) # real
>>> lista.append("Jandl") # string
>>>
>>> lista
[1, 'Peter', False, 4.44, 'Jandl']
>>>
```

Listas podem ser criadas **vazias** ou **com elementos**

LISTAS

```
lista = [1, 'Peter', False, 4.44, 'Jandl']
```

- Cada elemento pode ser ***acessado individualmente*** por meio de um índice (posição):
  - O índice inicial é **zero**
  - Os índices são inteiros indicados entre colchetes
  - Índices negativos indicam as posições do final para o início
- Atribuir valores para posições existentes altera o conteúdo da lista:

  lista[2] = True

  lista[0] = "Início"

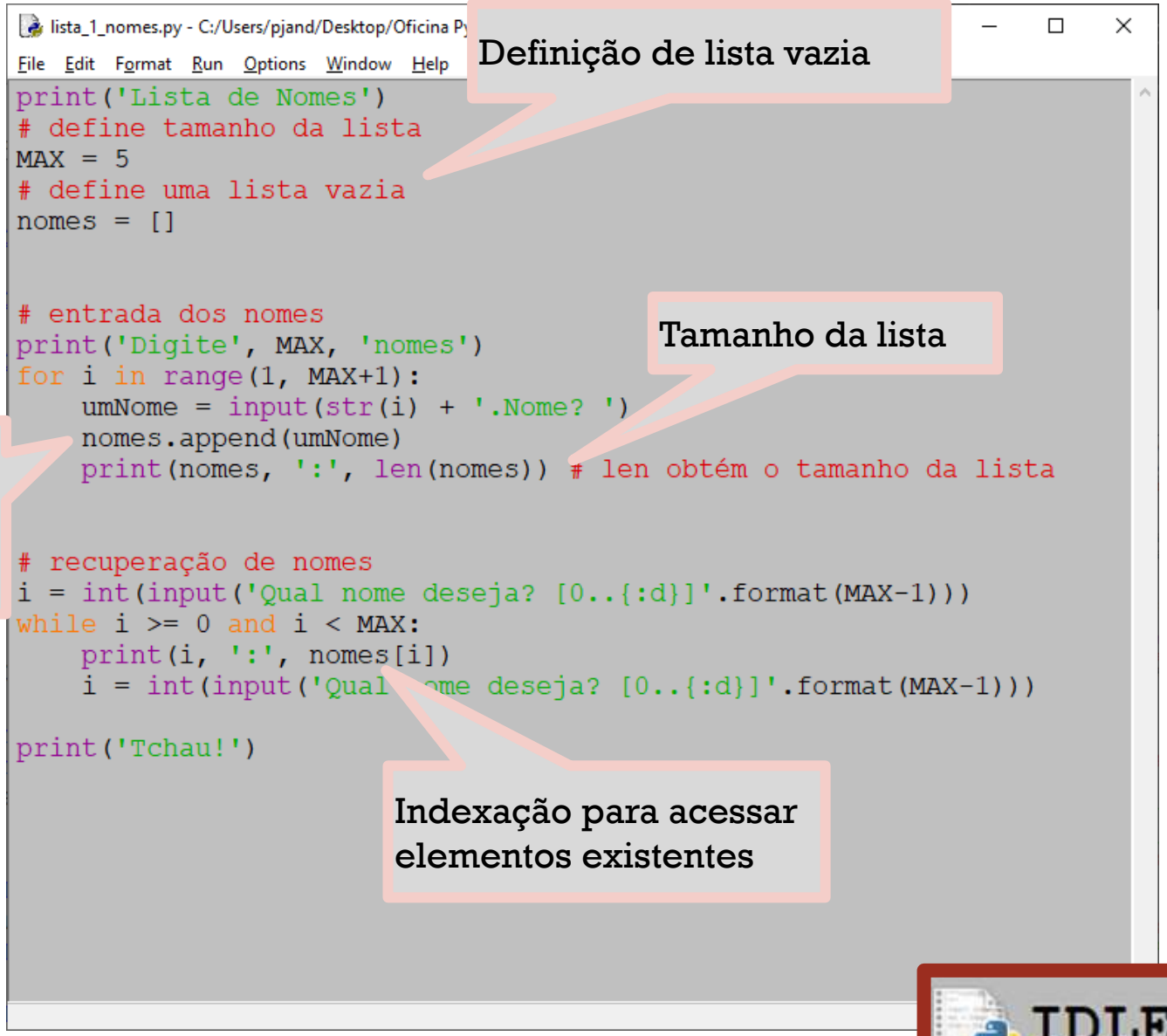
Alterações no conteúdo são permitidas

```
>>> lista = [1, 'Peter', False, 4.44, 'Jandl']
>>>
>>> lista[0]
1
>>> lista[1]
'Peter'
>>> print(lista[2])
False
>>> lista[-1]
'Jandl'
>>> lista[-2]
4.44
>>> lista[6]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>>
```

Acesso indexado das posições existentes

Acessar posições inexistentes é um **erro!!**

S
C
R



USO DE LISTAS

- Usar listas é bastante direto.
- Listas podem conter qualquer quantidade de elementos.
- A função **len(lista)** permite retornar o tamanho (comprimento ou número de elementos de uma lista).
- *Use listas sempre que precisar! Elas substituem os arrays/vetores!*

LISTAS

```
>>> lista = ['Yoda', 'Luke']
>>> lista
['Yoda', 'Luke']
>>> lista.append('Obi-wan')
>>> lista
['Yoda', 'Luke', 'Obi-wan']
>>> lista[1] = 'Anakin'
>>> lista
['Yoda', 'Anakin', 'Obi-wan']
>>> lista.insert(2,'Luke')
>>> lista
['Yoda', 'Anakin', 'Luke', 'Obi-wan']
>>> lista.sort()
>>> lista
['Anakin', 'Luke', 'Obi-wan', 'Yoda']
>>>
```

Anexação

Alteração

Inserção

Ordenação

```
>>> lista
['Anakin', 'Luke', 'Obi-wan', 'Yoda']
>>> lista.pop(0)
'Anakin'
>>> lista
['Luke', 'Obi-wan', 'Yoda']
>>> lista.remove('Yoda')
>>> lista
['Luke', 'Obi-wan']
>>> lista.reverse()
>>> lista
['Obi-wan', 'Luke']
>>> lista.clear()
>>> lista
[]
>>>
```

Remoção por posiçao

Remoção por valor

Reversão

Limpeza geral

S C R D M

lista_2_numeros.py - C:\Users\pjand\Desktop\Oficina Python\código-fonte\Dia_4\lista_2_numeros.py (3.8.0)

File Edit Format Run Options Window Help

```
# opções
listaOpcoes = ['Localizar número', 'Alterar número',
               'Remover número', 'Adicionar número', 'Sair', ]
# define lista vazia
listaNumeros = []

# função localizar
def localizar():
    pass

# função alterar
def alterar():
    pass

# função remover
def remover():
    pass

# função adicionar
def adicionar():
    pass

# Menu
while True:
    print('======')
    print(listaNumeros)
    print('======')
    for o in range(0, len(listaOpcoes)):
        print('[:d].[:s]'.format(o, listaOpcoes[o]))
    opcao = int(input('Opção? '))
    if opcao == 0:
        localizar()
    elif opcao == 1:
        alterar()
    elif opcao == 2:
        remover()
    elif opcao == 3:
        adicionar()
    else:
        break
```

Lista pré-definida

Pré-definição de funções

Menu de opções

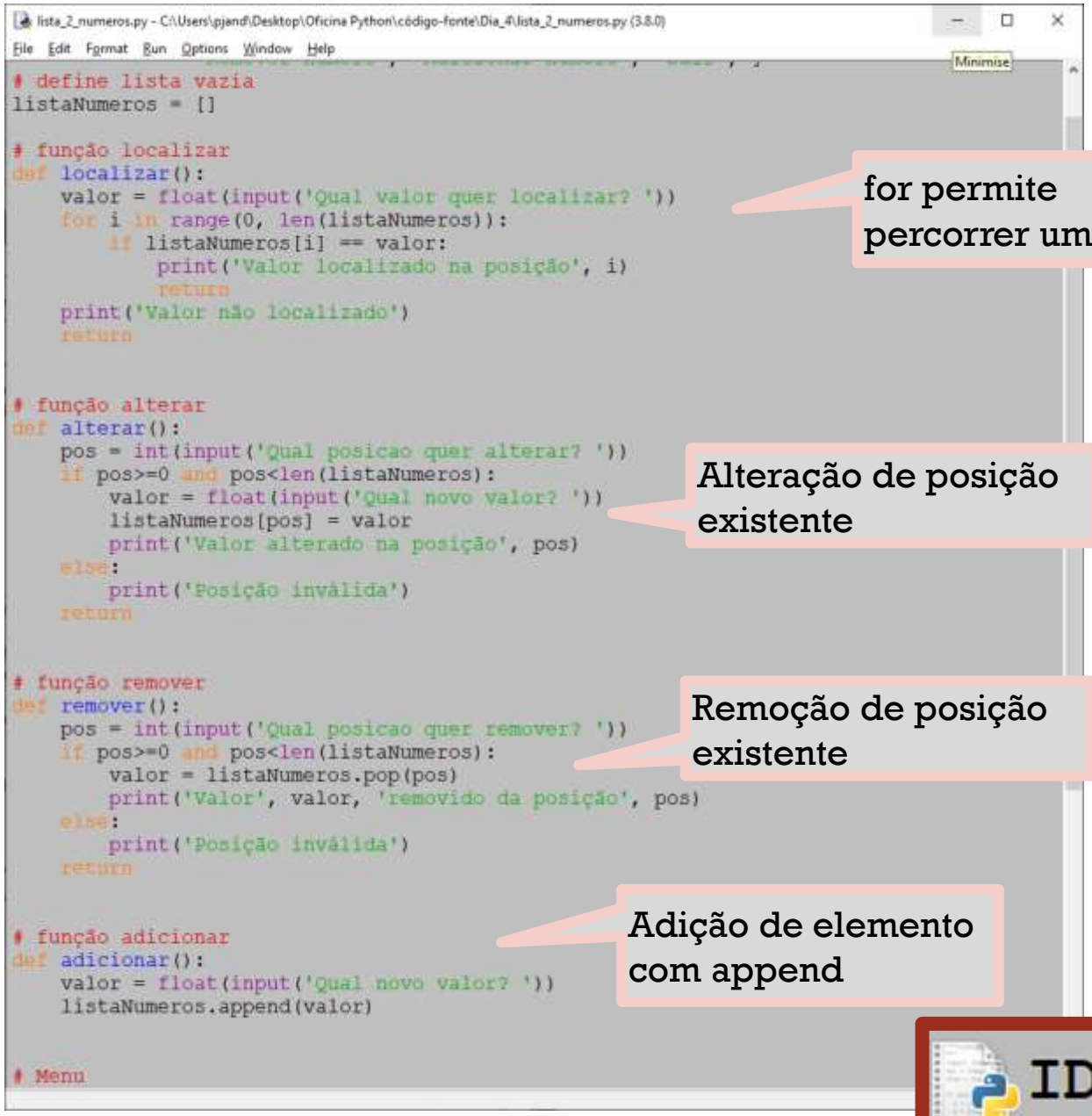
Acionamento das opções do menu

IDLE

USO DE LISTAS

- Este programa exibe um menu com opções para lidar com uma lista de número:
  - Adicionar novo elemento
  - Alterar elemento
  - Localizar elemento
  - Remover elemento
- *Use listas sempre que precisar! Elas substituem os arrays/vetores!*

S C R D M



for permite percorrer uma lista

Alteração de posição existente

Remoção de posição existente

Adição de elemento com append

IDLE

USO DE LISTAS

- Este programa trabalha várias possibilidades com uma lista:
  - Adicionar novo elemento append()
  - Alterar elemento indexação
  - Localizar elemento percurso
  - Remover element pop()
- *Use listas sempre que precisar! Elas substituem os arrays/vetores!*

S C R D M

lista_3_forca.py - C:\Users\pjand\Desktop\Oficina Python\código-fonte\Dia_4\lista_3_forca.py (3.8.0)

File Edit Format Run Options Window Help

```
import random

'''
Testa acertos na palavra secreta
'''
def verificaAcerto(palavra, letra, tentativa):
    posicao = 0
    letrasAcertadas = 0
    for l in palavra:
        if letra.lower() == l.lower():
            tentativa[posicao] = letra
        if tentativa[posicao] == palavra[posicao]:
            letrasAcertadas += 1
        posicao += 1
    return len(palavra) == letrasAcertadas

# preparação
# define uma lista de palavras "secretas"
palavras = ['Python', 'banana', 'ideia', 'elemento', 'zebra']
# sorteia uma palavra e "organiza" tentativa
segredo = palavras[random.randrange(0, len(palavras))].lower()
tentativa = []
for i in range(0, len(segredo)):
    tentativa.append('_')
enforcado = False
acertou = False
erros = 0
print('Mini-Forca')

# entrada das letras
while (not enforcado and not acertou):
    letra = input('Digite uma letra: ')
    if letra in segredo:
        acertou = verificaAcerto(segredo, letra, tentativa)
    else:
        erros += 1
        print('Você errou!', erros, 'erro(s).')
    print(len(segredo)*'-----')
    print(tentativa)
    print(len(segredo)*'-----')
    enforcado = erros == 4

# situação final
if acertou:
    print('Você venceu!')
else:
    print('Você foi enforcado!')
```

Ln: 1 Col: 0

Função que testa acertos das letras em relação ao segredo

Preparação do jogo

Laço de repetição das ações do jogo

Resultado da jogada

Resultado do jogo

IDLE

USO DE LISTAS

- Listas podem ser usadas para manter:
  - Nomes, palavras ou quaisquer strings;
  - Valores de qualquer tipo;
- E também pode conter outras estruturas de dados, como:
  - Tuplas,
  - Conjuntos,
  - Listas e
  - Dicionários!

13

SEQUÊNCIAS

29/10/2020

SEQUÊNCIAS

- Correspondem a definição mais formal de várias estruturas de dados do Python.
- Uma sequência é um *container* que permite organizar uma série de elementos.
- As sequências podem ser **mutáveis** ou **imutáveis**.

- Existem três tipos básicos de sequências no Python:
  - Listas (tipo ***list***)
  - Tuplas (tipo ***tuple***)
  - Intervalos (tipo ***range***)

Já tratamos dos **ranges** quando falamos do **for**

SEQUÊNCIAS::OPERAÇÕES

Operação	Funcionalidade
x **in** seq	True se x está presente na sequência
x **not in** seq	True se x **não** está presente na sequência
s + t	Concatenação da sequência s com sequência t
seq * n n * seq	Concatenação n vezes da sequência s
seq [i]	Elemento da posição i da sequência
seq [i : j]	Fatia da posição i (inclusa) até j (não inclusa) da sequência
seq [i : j : p]	Fatia da posição i (inclusa) até j (não inclusa) da sequência com passo p
len (seq)	Comprimento (número de elementos) da sequência
min (seq)	Menor elemento da sequência
max (seq)	Maior elemento da sequência
seq.**count**(x)	Contagem das ocorrências de x na sequência

16

STRINGS

STRINGS

- São como listas, mas cujos elementos são exclusivamente caracteres individuais.
- Constituem o tipo embutido **str**.
- No entanto, as strings são **imutáveis**, ou seja, não podem ser alteradas.
- Depois de criada, uma string mantém seu valor até ser destruída.
- Isto ocorre para permitir uma série de otimizações no Python, dado que o uso de strings é bastante intenso na maior parte dos programas.

STRINGS

- Toda função que *aparentemente altera* uma string, na verdade, retorna uma ***nova*** string com a alteração realizada.
- Como qualquer sequência, as strings podem ter seus elementos (caracteres individuais) acessados por um índice, podem ser fatiadas ou percorridas/navegadas.
- Oferecem várias funções que possibilitam obter transformações diversas.

```
>>> palavra = 'paralelepipedo'
>>> len(palavra)
14
>>> palavra[9]
'i'
>>> palavra[2:12:3]
'repe'
>>> 'a' in palavra
True
>>> 'x' not in palavra
True
>>> palavra[0] = 'P'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>>
```

Fatias da string

STRINGS

```
>>> 'StarWars'.isalpha()
True
>>> 'Star Wars'.isalpha()
False
>>> palavra = 'Star Wars'
>>> palavra.capitalize()
'Star wars'
>>> palavra.endswith('ars')
True
>>> palavra.find('W')
5
>>> palavra.index('a')
2
>>> palavra.isalnum()
False
>>> palavra.lower()
'star wars'
>>> palavra
'Star Wars'
>>> palavra.upper()
'STAR WARS'
>>> palavra
'Star Wars'
>>>
```

Funções utilitárias

String 'original' não é alterada após uso das funções

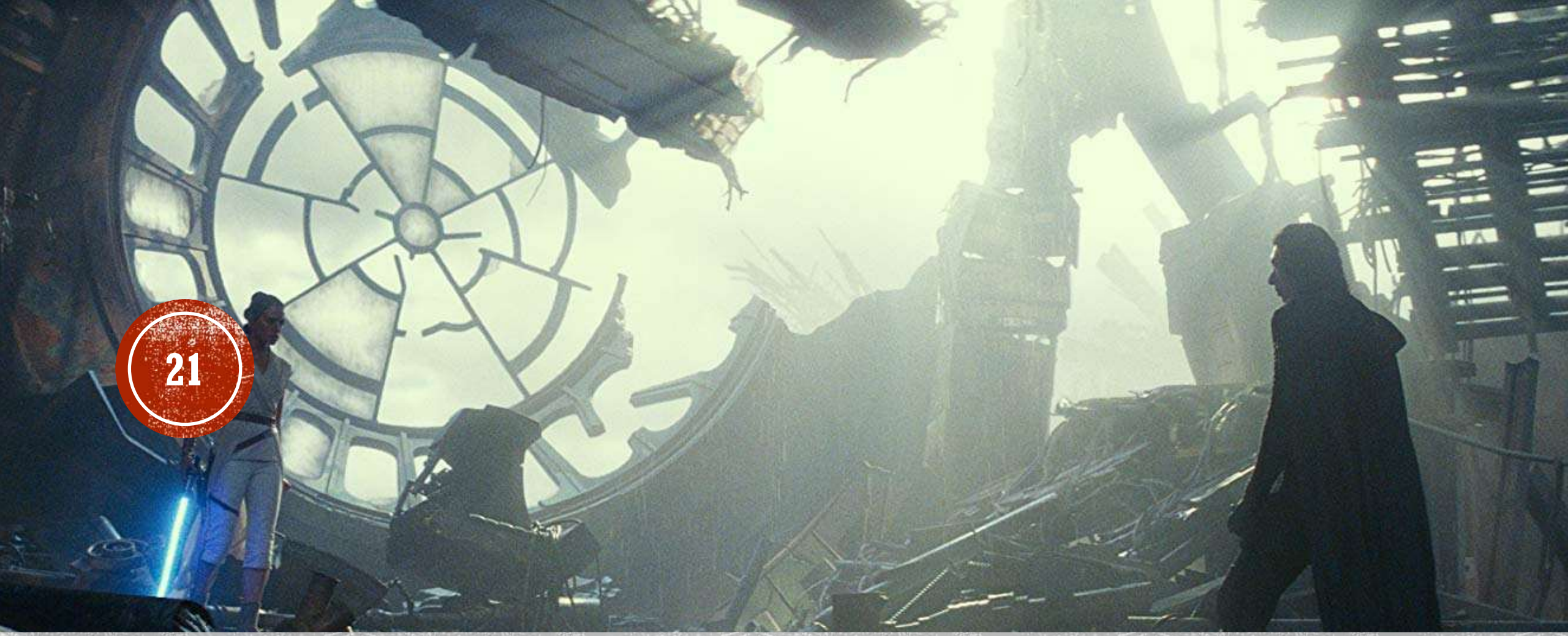
```
>>> frase = 'Star Wars é uma das sequên
cias de maior sucesso do cinema.'
>>> frase.split()
['Star', 'Wars', 'é', 'uma', 'das', 'se
quências', 'de', 'maior', 'sucesso', 'd
o', 'cinema.']
>>> lista = frase.split()
>>> for p in lista: print(p)
...
Star
Wars
é
uma
das
sequências
de
maior
sucesso
do
cinema.
>>>
```

Divisão de string

Iteração por elementos extraídos da divisão da string

STRINGS::FUNÇÕES

Função	Propósito
capitalize ( )	Faz primeiro caractere maiúsculo e os demais minúsculos
count (sub [, ini [, fim]])	Conta o número de ocorrências da substring (entre ini e fim)
endswith (sufixo)	Retorna True se string termina com sufixo indicado
find (sub [, ini])	Retorna a posição da primeira ocorrência de sub (a partir de ini)
index (sub [, ini])	Retorna a posição da primeira ocorrência de sub (a partir de ini)*
join (list)	Retorna a concatenação de uma lista de string
lower ( )	Retorna uma cópia da string em minúsculas
replace (old, new)	Retorna uma cópia da string onde ocorrências de old são substituídas por new
split ([sep])	Retorna uma lista contendo a divisão da string (dividida por sep)
startswith (prefixo)	Retorna True se string inicia com prefixo indicado
strip ( )	Remove uma cópia da string sem whitechars no início e fim
upper ( )	Retorna uma cópia da string em maiúsculas



TUPLAS

TUPLAS

- São coleções ordenadas e **imutáveis** de elementos, ou seja, não podem ser alteradas, mas que podem conter elementos em qualquer quantidade e em qualquer combinação de tipo (são **heterogêneas**).
- Constituem o tipo embutido **tuple**.
- A ordenação não significa classificação, mas que a posição de cada elemento na tupla é sempre fixa, pois não pode ser alterada.
- Servem para agrupar dados, usualmente associados, que mantém o valor e a ordem com que foram definidos.

TUPLAS

- Definição de tupla vazia:
  tuplaVazia = ( )
- Definição de tupla com um elemento:
  una = ("The Force",)
- Definição de tupla com dois elementos:
  dupla = ('Qui-Gon', 'Obi-Wan')
- Definição de tupla com três elementos:
  trio = ('Alpha', 'Beta', 'Gama')
- E por aí vai!

Tuplas são definidas com parênteses

´Tupla vazia, ok

Tupla com um elemento, sem vírgula, se torna uma variável simples

Tuplas com 1 elemento requer vírgula

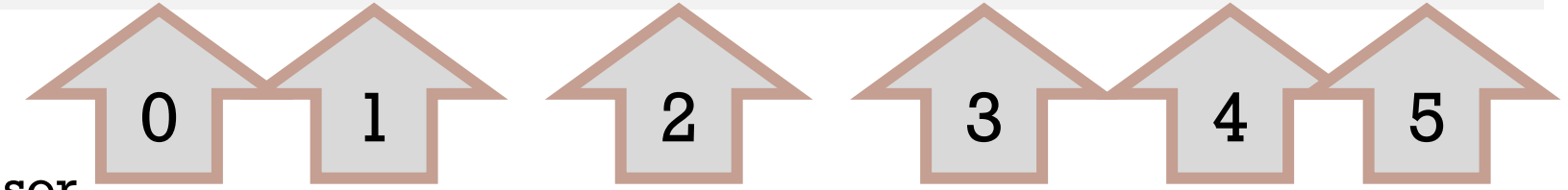
Tuplas com 2 ou mais elementos, ok

```
>>> tuplaVazia = ()
>>> tuplaVazia
()
>>> type(tuplaVazia)
<class 'tuple'>
>>> una = ("The Force")
>>> una
'The Force'
>>> type(una)
<class 'str'>
>>> una = ("The Force", )
>>> una
('The Force',)
>>> type(una)
<class 'tuple'>
>>> dupla = ('Qui-Gon', 'Obi-Wan')
>>> dupla
('Qui-Gon', 'Obi-Wan')
>>> type(dupla)
<class 'tuple'>
>>> print(una, 'e', dupla)
('The Force',) e ('Qui-Gon', 'Obi-Wan')
>>> trio = ('Alpha', 'Beta', 'Gama')
>>> especial = (una, dupla, trio)
>>> especial
(('The Force',), ('Qui-Gon', 'Obi-Wan'), ('Alpha', 'Beta', 'Gama'))
>>>
```

Tuplas, assim com listas, podem conter elementos de tipos diferentes

TUPLAS

```
tupla = (1, 'Peter', 'Jandl', 'True', 1.69, 84.5)
```



- Cada elemento pode ser ***acessado individualmente*** por meio de um índice (posição):
  - O índice inicial é **zero**
  - Os índices são inteiros indicados entre colchetes
  - Índices negativos indicam as posições do final para o início
- Não é possível atribuir valores para posições existentes, pois tuplas são imutáveis. Isso produz um **TypeError**.

Alterações no conteúdo **não** são permitidas

```
>>> tupla = (1, 'Peter', 'Jandl', 'True', 1.69, 84.5)
>>> tupla[0]
1
>>> tupla[3]
'True'
>>> tupla[1:2]
('Peter',)
>>> tupla[1:3]
('Peter', 'Jandl')
>>> tupla[5]/tupla[4]**2
29.58579881656805
>>> for e in tupla: print(e)
...
1
Peter
Jandl
True
1.69
84.5
>>>
```

Acesso indexado das posições existentes

Fatiamento

Navegação/percurso

TUPLAS

- Permitem que uma função retorne 2 ou mais valores.
- Permitem organizar conjuntos de valores que podem ser colocados em listas (ou outras estruturas de dados).
- Por exemplo:

```
:
X = 7
Y = 12
return (X,Y)
```

- Listas podem ser transformadas em tuplas.
- lista = [1, 2, 3, 4]
- tupla = tuple(lista)

tupla_1_distancia.py - C:\Users\pjand\Desktop\Oficina Python\código-fonte\Dia_4\tupla_1_distancia.py (3.8.0)

File Edit Format Run Options Window Help

```
import math

# Função que lê um ponto x,y retornando-o como uma tupla
def lePonto(nomePonto):
    x = float(input('{:s}.coordenada x? '.format(nomePonto)))
    y = float(input('{:s}.coordenada y? '.format(nomePonto)))
    return (x, y) # cria e retorna uma tupla com valores de x e y


# Função que calcula a distância no eixo X entre dois pontos p1 e p2
def deltaX(p1, p2):
    delta = abs(p1[0] - p2[0])
    return delta


# Função que calcula a distância no eixo Y entre dois pontos p1 e p2
def deltaY(p1, p2):
    delta = abs(p1[1] - p2[1])
    return delta


# Função que calcula a distância entre dois pontos p1 e p2
def distancia(p1, p2):
    return math.sqrt(deltaX(p1, p2)**2 + deltaY(p1, p2)**2)


# Programa principal
ponto1 = lePonto('Ponto 1')
ponto2 = lePonto('Ponto 2')

print('Distancia entre {:s} e {:s} é {:.3f}.'
      .format(str(ponto1), str(ponto2), distancia(ponto1, ponto2)))
```

Função que retorna tupla

Função que recebe tuplas

Função que recebe tuplas

Conversão de tupla em string

IDLE

USO DE TUPLAS

- Tuplas podem ser usadas para manter conjuntos fixos (*imutáveis*) de dados de qualquer tipo (*heterogêneos*).
- E também pode conter outras estruturas de dados, como:
  - Tuplas,
  - Conjuntos,
  - Listas e
  - Dicionários!

27

CONJUNTOS

CONJUNTOS

- São coleções não ordenadas e **mutáveis** de elementos que não aceitam duplicatas, ou seja, não contém elementos repetidos (iguais).
- Podem conter elementos em qualquer quantidade e em qualquer combinação de tipo (são **heterogêneas**).
- Constituem o tipo embutido **set**.
- Servem para criar e manter conjuntos de dados distintos (sem repetição).

CONJUNTOS

- Definição de conjunto vazio:
  vazio = set( )
- Definição de conjunto com um elemento:
  uno = { "The Force" }
- Definição de conjunto com dois elementos:
  dupla = { 'Qui-Gon', 'Obi-Wan' }
- Definição de conjunto com três elementos:
  trio = { 'Ahsoka', 'Luminara', 'Satele' }
- E por aí vai!

Conjuntos **com** elementos são definidos com chaves

Conjunto vazio requer função set()

```
>>>
>>> conj = set()
>>> conj
set()
>>> type(conj)
<class 'set'>
>>> uno = { "The Force" }
>>> uno
{'The Force'}
>>> type(uno)
<class 'set'>
>>> dupla = { 'Qui-Gon', 'Obi-Wan' }
>>> trio = { 'Ahsoka', 'Luminara', 'Satele' }
>>> print(dupla,'e',trio)
{'Qui-Gon', 'Obi-Wan'} e {'Luminara', 'Satele', 'Ahsoka'}
>>>
```

Conjuntos com 1 ou mais elementos usam chaves

Tipo embutido **set**

Conjunto, assim com listas e tuplas, podem conter elementos de tipos diferentes

CONJUNTOS

```
conj = { 'Darth', 'Sidius', 'Maul', 'Tyranus', 'Vader' }
```

- Os elementos ***não podem*** ser *acessados individualmente* por meio de índices.
- Não possível atribuição de valores para posições existentes, o que causa TypeError.
- Conjuntos podem ser navegados, proporcionando acesso aos seus elementos.

```
>>>
>>> conj = { 'Darth', 'Sidius', 'Maul', 'Tyranus', 'Vader' }
>>> for syth in conj: print(syth)
...
Maul
Vader
Darth
Tyranus
Sidius
>>> 'Maul' in conj
True
>>> 'Bane' in conj
False
>>> conj.add('Bane')
>>> conj
{'Maul', 'Bane', 'Vader', 'Darth', 'Tyranus', 'Sidius'}
>>> 'Bane' in conj
True
>>>
```

Navegação/Percurso

Pertencimento

Adição de elemento

CONJUNTOS::OPERAÇÕES

Operação	Resultado
A | B	União dos conjuntos A e B
A – B	Diferença dos conjuntos A e B
A & B	Intersecção dos conjuntos A e B
A ^ B	Diferença simétrica entre A e B

```
>>> A = { 1, 2, 3 }
>>> B = { 3, 4, 5, 6 }
>>> A | B
{1, 2, 3, 4, 5, 6}
>>> A & B
{3}
>>> A - B
{1, 2}
>>> A ^ B
{1, 2, 4, 5, 6}
>>> A.add(4)
>>> A
{1, 2, 3, 4}
>>> A.add(4)
>>> A
{1, 2, 3, 4}
>>> A.discard(5)
>>> A
{1, 2, 3, 4}
>>> A.remove(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 5
>>>
```

Dois conjuntos

União

Intersecção

Diferença

Diferença simétrica

Adição de elemento repetido é desprezada

Adição de elemento

Remoção de elemento

Remoção estrita de elemento

S C R M

conjunto_1_palavras.py - C:/Users/pjand/Desktop/Oficina Python/código-fonte/Dia_4/conjunto_1_palavras.py (3.8.0

File Edit Format Run Options Window Help

```
# Cria conjunto de palavras
def criaConjunto():
    conjunto = set()     # conjunto vazio
    palavra = input('Digite uma palavra (enter finaliza): ')
    while len(palavra) > 0:
        print(conjunto,'<--',palavra)
        conjunto.add(palavra)
        palavra = input('Digite uma palavra (enter finaliza): ')
    print(conjunto)
    return conjunto


# Programa principal
A = criaConjunto()  # cria um conjunto de palavras A
B = criaConjunto()  # cria um conjunto de palavras B

# Operações sobre os conjuntos
print('União A | B:', A | B)
print('-----------:', A.union(B))

print('Intersecção A & B:', A & B)
print('-----------------:', A.intersection(B))

print('Diferença A - B:', A - B)
print('---------------:', A.difference(B))
print('Diferença B - A:', B - A)
print('---------------:', B.difference(A))

print('Diferença simétrica A ^ B:', A ^ B)
print('-------------------------:', A.symmetric_difference(B))
print('Diferença simátrica B ^ A:', B ^ A)
print('-------------------------:', B.symmetric_difference(A))
```

Função que cria e retorna conjunto

Uso da função

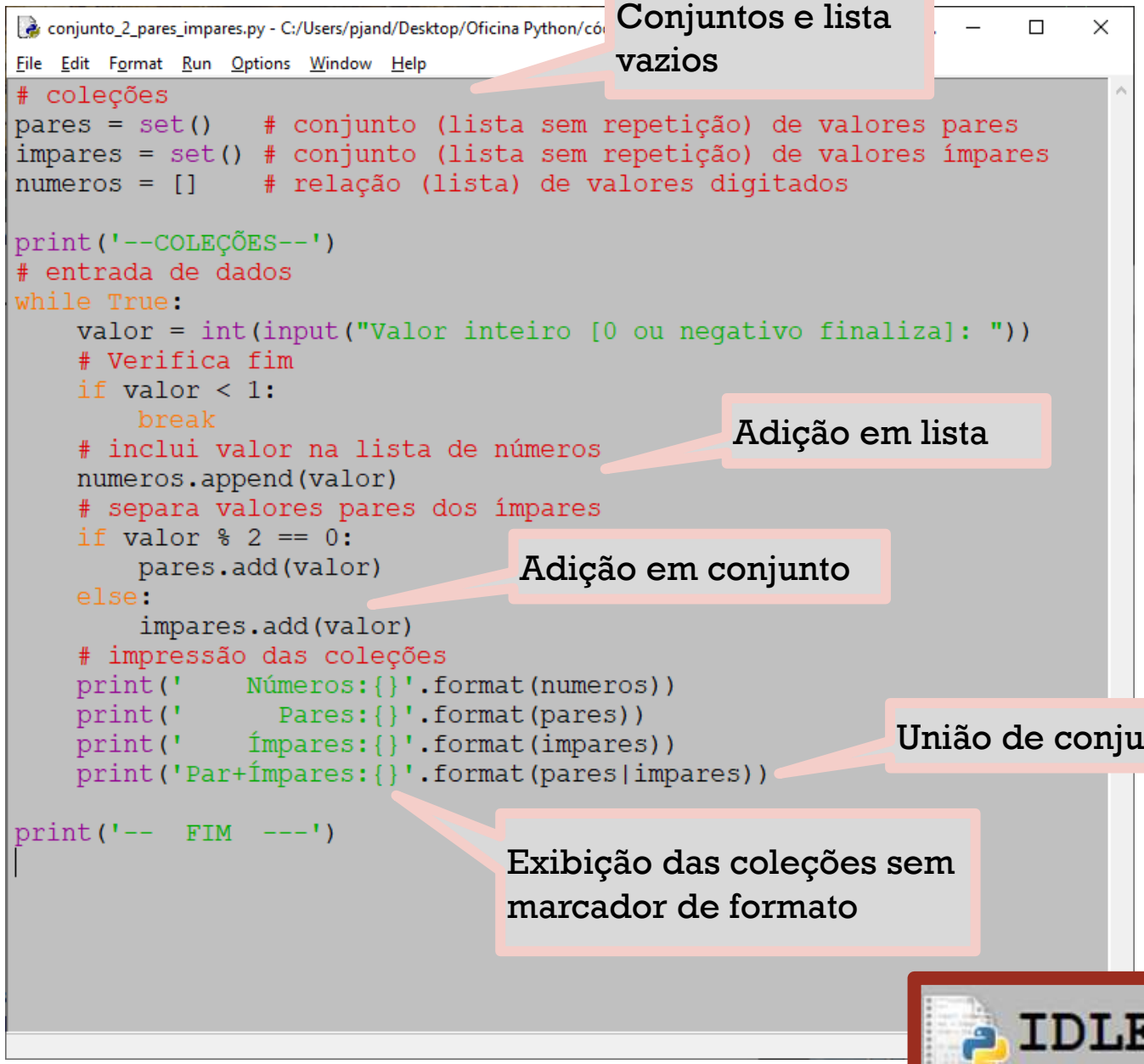
Operações via operador e função equivalente

IDLE

USO DE CONJUNTOS

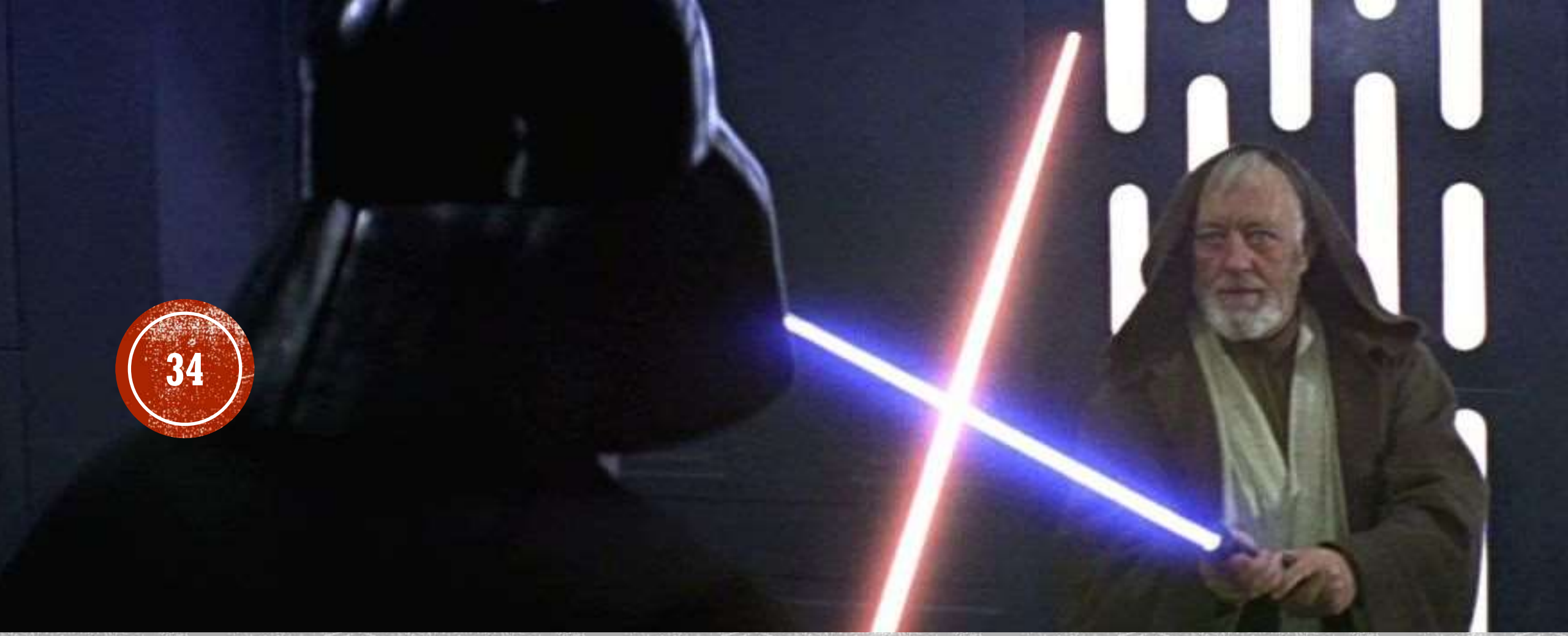
- Conjuntos podem ser usados para manter coleções variáveis (*mutáveis*) de elementos **distintos** de qualquer tipo (*heterogêneos*).
- E também pode conter outras estruturas de dados, como:
  - Tuplas,
  - Conjuntos,
  - Listas e
  - Dicionários!

S C R D



USO DE CONJUNTOS

- Conjuntos podem ser usados para manter coleções variáveis (*mutáveis*) de elementos **distintos** de qualquer tipo (*heterogêneos*).
- E também pode conter outras estruturas de dados, como:
  - Tuplas,
  - Conjuntos,
  - Listas e
  - Dicionários!



34

DICIONÁRIOS

DICIONÁRIOS

- São coleções não ordenadas, **mutáveis** de elementos associados em pares de chave e valor.
- Cada chave é associada (mapeada) em um valor.
- As chaves não se repetem (como num conjunto).
  Pode existir qualquer número de chaves.
  Cada chave tem um valor, que pode se repetir (em chaves diferentes)
- As chaves podem ser de qualquer tipo ***imutável***, enquanto os valores associados podem ser de qualquer tipo, caracterizando uma estrutura **heterogêneas**.
- Constituem o tipo embutido **dict**.
- Servem para recuperar, de maneira eficiente, as informações associadas às chaves.

DICIONÁRIOS

Dicionários são definidos com chaves

- Definição de dicionário vazio:
  vazio = { }
- Definição de dicionário com um par chave-valor:

  dicio1 = { 1 : "Yoda" }

  Chave e valor são separados por **dois-pontos**
- Definição de dicionário com dois elementos:
  dicio2 = { 'Qui-Gon':'Jinn' ,
  'Obi-Wan':'Kenobi' }

  Pares chave:valor são separados por **vírgulas**
- E por aí vai!

Dicionário com um elemento

```
>>> dicio1 = { 1 : "Yoda" }
>>> dicio1
{1: 'Yoda'}
>>> type(dicio1)
<class 'dict'>
>>> dicio1[1]
'Yoda'
>>> dicio1.keys()
dict_keys([1])
>>> dicio1.values()
dict_values(['Yoda'])
>>> dicio1[1] = 'Master Yoda'
>>> dicio1
{1: 'Master Yoda'}
>>>
```

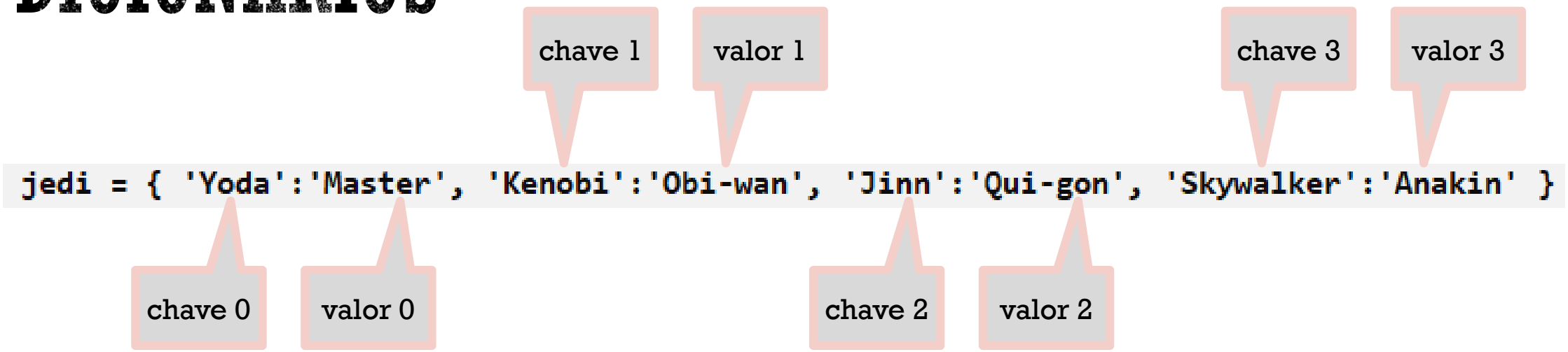
Tipo embutido **dict**

Indexação com chave recupera valor associado

Possui listas de chaves e valores

Permite alterar valor de uma chave

DICIONÁRIOS



Chave	Valor
Yoda	Master
Kenobi	Obi-wan
Jinn	Qui-gon
Skywalker	Anakin

São como *tabelas associativas*, onde a **chave**, sempre distinta, indexa/mapeia/associa um **valor**

DICIONÁRIOS

- Os elementos ***podem*** ser *acessados individualmente* por meio de sua ***chave***.
- As chaves pode ser obtidas por meio da função **keys** ( ).
- Os valores presentes pode ser obtidos por meio da função **values** ( ).
- Dicionários também podem ser navegados, proporcionando acesso direto às suas chaves (e aos seus valores).

```
>>>
>>> jedi = { 'Yoda':'Master', 'Kenobi':'Obi-wan', 'Jinn':
'Qui-gon', 'Skywalker':'Anakin' }
>>> for j in jedi: print(j)
...
Yoda
Kenobi
Jinn
Skywalker
>>> for j in jedi: print(jedi[j])
...
Master
Obi-wan
Qui-gon
Anakin
>>> len(jedi)
4
>>> jedi['Tano']='Ashoka'
>>> jedi
{'Yoda': 'Master', 'Kenobi': 'Obi-wan', 'Jinn': 'Qui-gon'
, 'Skywalker': 'Anakin', 'Tano': 'Ashoka'}
>>> jedi['Kenobi']
'Obi-wan'
>>> jedi['Tano']
'Ashoka'
>>> jedi['Windu']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Windu'
>>>
```

Navegação percorre chaves do dicionário

Chave permite recuperar valor associado

Adição de par chave:valor

Recuperação associativa

dicionario_1_definicao.py - C:\Users\pjand\Desktop\Oficina Python\código-fonte\Dia_4\dicionario_1_d

File Edit Format Run Options Window Help

```
# dados
cor = input('Carro - cor: ')
marca = input('Carro - marca: ')
modelo = input('Carro - modelo: ')

# dicionário
# uma forma de definir
dict1 = {'cor':cor, 'marca':marca, 'modelo':modelo}

dict2 = {} # outra forma de definir
dict2['cor'] = cor
dict2['marca'] = marca
dict2['modelo'] = modelo

# impressão de chaves e valores
print('dict1')
for k in dict1.keys():
    print('{}:{}'.format(k, dict1[k]))

print('dict2')
for k in dict1.keys():
    print('{}:{}'.format(k, dict2[k]))
```

Entrada de dados simples

Definição de dicionário

Outra definição de dicionário

Navegação em dicionário

IDLE

USO DE DICIONÁRIOS

- Dicionários podem ser usados para associar qualquer quantidade de pares chave-valor.
- Podem ser usados para manter dados distintos de uma entidade/elemento do programa.
- Podem ser usados para manter coleções semelhantes de dados, identificadas por chaves distintas.

S
C
R
D
M

dicionario_2_pares.py - C:\Users\pjand\Desktop\Oficina Python\código-fonte\Dia_4\dicionario_2_pares.py (3.8.0)

File Edit Format Run Options Window Help

```
# opções
listaOpcoes = ['Localizar chave-valor', 'Alterar chave-valor',
               'Remover chave-valor', 'Adicionar chave-valor', 'Sair', ]
# define dicionario vazio
dicionario = {}

# função localizar
def localizar():
    chave = input('Qual chave-valor quer localizar? ')
    if chave in dicionario:
        print('Valor associado', dicionario[chave])
    else:
        print('Chave não localizada')
    return


# função alterar
def alterar():
    chave = input('Qual chave-valor quer loc
    if chave in dicionario:
        valor = input('Qual novo valor da ch
        dicionario[chave] = valor
        print('Novo valor associado')
    else:
        print('Chave não localizada')
    return


# função remover
def remover():
    chave = input('Qual chave-valor quer loc
    if chave in dicionario:
        valor = dicionario.pop(chave)
        dicionario[chave] = valor
        print('Valor associado removido: ',
    else:
        print('Chave não localizada')
    return
```

dicionario_2_pares.py - C:\Users\pjand\Desktop\Oficina Python\código-fonte\Dia_4\dicionario_2_pares.py

File Edit Format Run Options Window Help

```
# função adicionar
def adicionar():
    chave = input('Qual nova chave? ')
    valor = input('Qual valor associado? ')
    dicionario[chave] = valor
    print('Novo par chave-valor criado')


# Menu
while True:
    print('=====')
    print(dicionario)
    print('=====')
    for o in range(0, len(listaOpcoes)):
        print('{:d}.{:s}'.format(o, listaOpcoes[o]))
    opcao = int(input('Opção? '))
    if opcao == 0:
        localizar()
    elif opcao == 1:
        alterar()
    elif opcao == 2:
        remover()
    elif opcao == 3:
        adicionar()
    else:
        break
```

USO DE DICIONÁRIOS

- E também pode conter outras estruturas de dados, como:
  - Tuplas,
  - Conjuntos,
  - Listas e
  - Dicionários!

O QUE APRENDEMOS?

- Habilidades da programação
- Uso de IDE básico
- Construção de módulos/bibliotecas
- Uso de estruturas de dados

HABILIDADES DA PROGRAMAÇÃO

- Sequenciar
- Computar
- Repetir
- Decidir
- Modularizar

- Combinar todas estas habilidades para solução de problemas

USO DE IDE BÁSICO

- IDLE é um IDE simples.
  Mas efetivo. Com ele podemos nos concentrar no aprendizado dos elementos básicos do Python.
- PyCham é uma (ótima) sugestão para aqueles que estão mais à vontade com a linguagem e desejam uma ferramenta mais profissional.

CRIAÇÃO DE MÓDULOS

- Com módulos podem organizar o código, além de promover sua reutilização.

ESTRUTURAS DE DADOS

- Listas, tuplas, conjuntos, dicionários.
- As estruturas de dados do Python são fáceis de usar e permitem resolver uma grande gama de problemas.

PARA SABER MAIS

- Arquivos, Acesso a Bancos de Dados
- Programação Web, Machine Learning
- E muito mais!

(C) 1999-2020, Jandl.

MÃOS NA MASSA

Que força esteja com você!

- Resolver a Lista **QUATRO**.
- Como pensar como um Cientista da Computação.
  Projeto Panda | IME | USP.
  https://panda.ime.usp.br/pensepy/static/pensepy/index.html
- Python e Orientação a Objetos
  Curso Py-14 | Caelum.
  https://www.caelum.com.br/apostila/apostila-python-orientacao-a-objetos.pdf
- Python 3.8. Documentação Oficial.
  https://docs.python.org/3/index.html

Obrigado!