

# Dicionários

- Aula 11 -  
Pensamento Computacional

Prof. Me. Lucas R. C. Pessutto



# Na aula de hoje...

01

## Dicionários

- Definição
- Declaração
- Criação
- Acesso aos Valores

02

## Operações

- Adicionar dados
- Remoção
- Iteração
- Ordenação

03

## Aninhamento de Dicionários

04

## Visões sobre Dicionários

# Recapitulando

Uma **estrutura de dados** é uma coleção de dados **organizados** de forma que possam ser **acessados eficientemente**.

--Irv Kalb, *Learn to Program with Python*.

## Listas (list)

Coleção ordenada de dados

Armazena qualquer tipo de dados

Acesso via índices

Permite inserção, remoção e alteração

Suporta fatiamento, concatenação e iteração

## Strings (str)

Coleção ordenada de dados

Armazena apenas texto

Acesso via índices

Imutável

Suporta fatiamento, concatenação e iteração

# Problema 1: IDH



Enunciado de um problema:

Você quer fazer um estudo sobre o [Índice de Desenvolvimento Humano \(IDH\)](#) no Brasil, então você resolve fazer um programa que armazena os valores de IDH por estado e realiza duas operações:

- 1) dada a sigla de um estado, informar seu IDH;
- 2) dado um valor de IDH, mostrar os estados com IDH acima desse valor.

# Problema 1: IDH



Como armazenar os dados de IDH por estado?

Descrição dos Dados  
e Algoritmos

a) **1 lista** intercalando cada sigla de estado e seu respectivo IDH?

```
estados_idh = ["AC", 0.71, "AL", 0.684, ..., "TO", 0.731]
```

b) **2 listas** uma de siglas e outra de IDHs relacionando pelos índices?

```
estados = ["AC", "AL", "AM", "AP", "BA", ..., "TO"]  
idhs     = [0.71, 0.684, 0.7, 0.688, 0.691, ..., 0.731]
```

# Problema 1: IDH



Implementação do  
Programa

**Solução (a)**

```
estados_idh = ["AC", 0.71, "AL", 0.684, ..., "TO", 0.731]

op = input(f"Busca por [E]stado ou por [I]DH? ").upper()

if op == "E":
    sigla = input("Informe a sigla do estado: ")
    # Acesso somente aos índices pares
    for i in range(0, len(estados_idh), 2):
        if sigla == estados_idh[i]:
            print(f"IDH de {sigla} = {estados_idh[i + 1]}")
elif op == "I":
    idh = float(input("Informe o valor do IDH: "))
    # Acesso somente aos índices ímpares
    for i in range(1, len(estados_idh), 2):
        if idh < estados_idh[i]:
            print(estados_idh[i - 1])
else:
    print("Opção Inválida!")
```

É uma boa  
solução?



A semântica dos dados  
fica misturada com a  
organização da estrutura.

# Problema 1: IDH



Implementação do  
Programa

**Solução (b)**

```
estados = ["AC", "AL", "AM", "AP", "BA", ..., "TO"]
idhs     = [0.71, 0.684, 0.7, 0.688, 0.691, ..., 0.731]
```

É uma boa  
solução?



```
op = input(f"Busca por [E]stado ou por [I]DH? ").upper()
if op == "E":
    sigla = input("Informe a sigla do estado: ")
    # Assumindo que o usuário digitou uma sigla válida!
    idh = idhs[estados.index(sigla)]
    print(f"IDH de {sigla} = {idh}")
elif op == "I":
    idh = float(input("Informe o valor do IDH: "))
    for i in range(0, len(estados)):
        if idh < idhs[i]:
            print(estados[i])
else:
    print("Opção Inválida!")
```

Economizamos um if e  
um for, o código ficou um  
pouco mais simples!

Esse for também ficou  
um pouco mais simples!

# Problema 1: IDH

Não seria melhor poder **associar** cada IDH ao seu respectivo estado?



Como armazenar os dados de IDH por estado?

Descrição dos Dados  
e Algoritmos

~~a) 1 lista intercalando cada sigla de estado e seu respectivo IDH?~~

```
estados_idh = ["AC", 0.71, "AL", 0.684, ..., "TO", 0.731]
```

~~b) 2 listas uma de siglas e outra de IDHs relacionando pelos índices?~~

```
estados = ["AC", "AL", "AM", "AP", "BA", ..., "TO"]  
idhs     = [0.71, 0.684, 0.7, 0.688, 0.691, ..., 0.731]
```

c) Usando um **dicionário**!



# Na aula de hoje...

Como armazenar conjuntos de dados associados?

## Dicionários! (dict)

```
>>> d = {"a": "alpha", "b": "bravo", "c": "charlie"}  
>>> type(d)  
<class 'dict'>
```

Intuitivamente, podemos pensar em listas como uma **coleção de dados** de duas dimensões associadas, sendo que uma delas pode ser usada como “chave” na relação

# Definição

Um **dicionário** é uma **coleção não ordenada** de dados associados aos **pares** na forma de **chave** e **valor**.

*(Outras linguagens de programação se referem a esse conceito como hash ou hashmap).*

- \* Diferentemente de listas, dicionários **não estabelecem uma ordem** entre os elementos armazenados
  - A partir da versão 3.6 o Python preserva a ordem de inserção
- \* Dicionários estruturam os dados em duas dimensões chamadas de **chave** (key) e **valor** (value):
  - As **chaves** são **únicas** e podem ser de qualquer tipo de dado **imutável** (ex., str, int, float, bool)
  - Os **valores** podem ser de **qualquer tipo** inclusive outras estruturas como listas ou mesmo dicionários

# Exemplos

- \* Coleções de dados associativos que podemos implementar com dicionários:
  - Palavras e seus respectivos significados (um dicionário ou glossário)
    - **Chave:** palavra (str) → **Valor:** significado (str)
  - Nomes dos times de um campeonato de futebol e sua lista de títulos conquistados
    - **Chave:** nomes (str) → **Valor:** lista de títulos (list -> str)
  - Número de matrícula e notas de alunos de uma turma
    - **Chave:** matrícula (int) → **Valor:** notas (float)
  - Catálogo de CEP e endereços completos (logradouro, número, etc.)
    - **Chave:** número do CEP (int) → **Valor:** endereço completo (dict)
  - Valores de entrada em um experimento e os resultados obtidos (sucesso ou falha)
    - **Chave:** número real (float) → **Valor:** resultado (bool)

É possível misturar dados de vários tipos tanto nas chaves quanto nos valores, mas não é muito comum.

# Criando dicionários

Dicionário vazio: `dic = {}`

Dicionário com dados:

Palavras e seus respectivos significados (um dicionário ou glossário):

```
glossario = {"algoritmo": "maneira chique de falar passo a passo",  
             "bug": "problema no código do programa",  
             "estrutura de dados": "coleção de dados organizados",  
             "indentação": "recuos usados para organizar código",  
             "variável": "rótulo que referencia um valor"}
```

Número de matrícula e notas de alunos de uma turma:

```
turma = {1756: 9.5, 2025: 7.4, 2094: 7.2, 2132: 5.9, 1822: 7.6}
```

Apesar de numérico e único, o identificador do aluno não é sequencial nem inicia em zero, por isso é preferível usar um dicionário.

# Acesso aos valores

Imprimir na tela o valor associado a chave “bug” do glossário:

```
print(glossario["bug"])
```

Para acessar cada valor utilizamos sua **chave**.

Calcular a média de 3 notas de alunos:

```
media = (turma[1756] + turma[2132] + turma[2094]) / 3
```

**Cuidado!** O acesso a chaves indefinidas gera erros:

```
print(turma["1756"])
```

O **tipo de dado** usado no acesso deve ser o mesmo utilizado na criação.

```
...  
KeyError: '1756'
```

# Adicionar/modificar valores

Adicionando uma nova palavra (chave) e seu significado no glossário:

```
glossario["exceção"] = "erro na execução do programa"
```

Basta fazer uma **atribuição** direta que uma nova chave será criada e o respectivo valor associado a ela.

Modificar a nota do aluno com a matrícula 1756

```
turma[1756] = 10.0
```

Caso a chave já **exista** o valor é **modificado**.

# Removendo pares (chave/valor)

Remover a entrada bug (chave) do glossário e seu significado (valor):

```
del glossario["bug"]
```

A instrução del pode ser usada para remover qualquer tipo de variável.

Tentar remover uma chave inexistente gera um KeyError.

Dicionários também possuem um método pop que remove e retorna o valor associado a uma chave:

```
nota = turma.pop(1756)
```

Não é possível remover apenas a chave ou apenas o valor. Em um dicionário uma **chave sempre terá um valor associado** e vice-versa.

# Iteração sobre dicionários

Na iteração com **for** a variável de controle do laço vai assumir o valor de cada uma das chaves do dicionário:

```
for chave in glossario:  
    print(f"{chave} -> {glossario[chave]}")
```

Lembrando que com listas a variável do laço assume os valores contidos e não os índices.

```
algoritmo -> maneira chique de falar passo a passo  
estrutura de dados -> coleção de dados organizados  
indentação -> recuos usados para organizar código  
variável -> rótulo que referencia um valor  
exceção -> erro na execução do programa
```



# Retomando o Problema 1: IDH



Enunciado de um problema:

Você quer fazer um estudo sobre o [Índice de Desenvolvimento Humano \(IDH\)](#) no Brasil, então você resolve fazer um programa que armazena os valores de IDH por estado e realiza duas operações:

- 1) dada a sigla de um estado, informar seu IDH;
- 2) dado um valor de IDH, mostrar os estados com IDH acima desse valor.

# Problema 1: IDH



## Descrição dos Dados e Algoritmos

Como armazenar os dados de IDH por estado?

Sabemos que vamos usar um dicionário, mas como definir a estrutura?

- \* **Chaves:** podemos usar a sigla do estado (str) que são únicas
- \* **Valores:** os valores de IDH (float) para cada respectivo estado

O dicionário será definido *a priori*, então precisamos apenas implementar as duas operações.

# Problema 1: IDH

```
estados_idh = {"AC": 0.71, "AL": 0.684, ..., "TO": 0.731}
```

```
op = input(f"Busca por [E]stado ou por [I]DH? ").upper()
```

```
if op == "E":  
    sigla = input("Informe a sigla do estado: ")  
    # Assumindo que a sigla digitada é válida  
    print(f"IDH de {sigla} = {estados_idh[sigla]}")
```

```
elif op == "I":  
    idh = float(input("Informe o valor do IDH: "))  
    for estado in estados_idh:  
        if idh < estados_idh[estado]:  
            print(estado)
```

```
else:  
    print("Opção Inválida!")
```

Não é preciso percorrer a estrutura porque podemos acessar o valor diretamente pela chave.

Esse for é necessário porque estamos procurando pelos valores que respeitam a condição lógica.

É uma boa solução?



# Ordenação de dicionários

- \* Dicionários **não possuem relação de ordem** nem entre chaves nem entre valores. Podemos ordenar um dicionário para percorrer ou imprimir seu conteúdo usando a função embutida `sorted`:

```
sorted(iterable, [key=None, reverse=False])
```

- \* **Retorna uma lista** classificada dos itens em iterable. O argumento **key** recebe uma função com um argumento usado na comparação dos elementos. O argumento **reverse**, se definido como True, classifica os elementos em ordem reversa.
- \* Importante notar que um **dicionário é iterável** a partir das suas **chaves**. Portanto, uma lista com chaves ordenadas será retornada, mas o dicionário em si **não será modificado**.

Documentação da função `sorted` em: <https://docs.python.org/pt-br/3/library/functions.html#sorted>

## Problema 2: IDH, o retorno



Enunciado de um problema:

Vamos fazer uma nova versão do programa do **Índice de Desenvolvimento Humano (IDH)** no Brasil com **duas novas operações**:

- 1) imprimir uma **lista de estados ordenada** pela **sigla** com os respectivos IDHs em ordem crescente ou decrescente
- 2) imprimir uma **lista de estados ordenada** pelo **IDH** em ordem crescente ou decrescente

## Problema 2: IDH, o retorno

```
estados_idh = {"RN": 0.728, "RS": 0.771, "RO": 0.7, ..., "MG": 0.774}
```

```
op = input(f"Listar ordenado por [E]stado ou por [I]DH? ").upper()
inverter = input(f"Ordem reversa (sim ou não)? ").lower() == "sim"
```

```
if op == "E":
    for estado in sorted(estados_idh, reverse=inverter):
        print(f"{estado}: {estados_idh[estado]}")
elif op == "I":
    for estado in sorted(estados_idh, key=estados_idh.get, reverse=inverter):
        print(f"{estado}: {estados_idh[estado]}")
else:
    print("Opção Inválida!")
```

Passando o dicionário como iterável vai ordenar pelas chaves.

Para ordenar pelos valores usamos o método `get` do próprio dicionário. A função `sorted` vai chamar esse método usando as chaves para ter acesso aos valores.

## Problema 3: Hashmon®

### Enunciado de um problema:

Você resolveu criar um jogo de cartas chamado **Hashmon®**. No seu jogo, cada carta representa um personagem com as seguintes características: **nome** (str), **tipo** (grama, água ou fogo) (str), **HP** (quantidade de vida) (int) e **fraqueza** (os mesmos valores do tipo) (str). Para auxiliar na criação do baralho você resolve fazer um programa em Python onde você vai digitando iterativamente as características das cartas e o programa gera um aviso em duas situações:

1. caso você tente inserir uma carta com nome repetido
2. caso você tente inserir uma carta com as outras características repetidas

Como vamos organizar esses dados em um dicionário?

# Problema 3: Hashmon®



**Chaves:** nome (str)



**Valores:** características (dict)

Descrição dos Dados  
e Algoritmos



```
baralho = {  
    "Hashzard": {  
        "hp": 20,  
        "tipo": "fogo",  
        "fraqueza": "grama"  
    },  
    "Squirrel":  
    {...}  
}
```



# Aninhando Estruturas

- \* Podemos armazenar em dicionários valores de qualquer tipo, inclusive listas ou outros dicionários.

```
baralho = {  
    "Hashzard": {  
        "hp": 20,  
        "tipo": "fogo",  
        "fraqueza": "grama"  
    },  
    "Squirrel":  
    {...}  
}
```



- \* Por exemplo, para acessar o tipo da carta "Hashzard" usamos:

```
baralho["Hashzard"]["tipo"]
```

# Visões

- \* É possível obter **visões dinâmicas** de dicionários, o que significa que quando o dicionário muda, a visão **reflete essas mudanças**. Três métodos principais podem ser usados para obter essas visões:

`dict.keys()`

Retorna uma nova **visão das chaves** do dicionário.

`dict.values()`

Retorna uma nova **visão dos valores** do dicionário.

`dict.items()`

Retorna uma nova **visão dos itens** do dicionário na forma de pares (chave, valor).

# Verificar existência de chaves/valores

- \* Verificar a existência de uma chave no dicionário:

```
nome = input("Nome: ")
if nome in baralho.keys():
    print("Já existe uma carta com esse nome!")
```

Nesse caso até daria no mesmo omitir a chamada do método `dict.keys()`.

- \* Verificar a existência de um valor em um dicionário:

```
carta = {}
carta["tipo"] = input("Tipo: ") # grama, água ou fogo
carta["hp"] = int(input("HP: "))
carta["fraqueza"] = input("Fraqueza: ") # grama, água ou fogo

if carta in baralho.values():
    print("Já existe uma carta com essas características!")
```

# Retomando o Problema 3

Enunciado de um problema:

Você resolveu criar um jogo de cartas chamado **Hashmon®**. No seu jogo, cada carta representa um personagem com as seguintes características: **nome** (str), **tipo** (grama, água ou fogo) (str), **HP** (quantidade de vida) (int) e **fraqueza** (os mesmos valores do tipo) (str). Para auxiliar na criação do baralho você resolve fazer um programa em Python onde você vai digitando iterativamente as características das cartas e o programa gera um aviso em duas situações:

1. caso você tente inserir uma carta com nome repetido
2. caso você tente inserir uma carta com as outras características repetidas

Como vamos organizar esses dados em um dicionário?

```
baralho = {}
i = 1
continuar = True
while continuar:
    print(f"Informe os dados da carta {i}")
    nome = input("Nome: ")
    if nome in baralho.keys():
        print("Já existe uma carta com esse nome!")
        continue
    carta = {}
    carta["tipo"] = input("Tipo: ") # grama, água ou fogo
    carta["hp"] = int(input("HP: "))
    carta["fraqueza"] = input("Fraqueza: ") # grama, água ou fogo
    if carta in baralho.values():
        print("Já existe uma carta com essas características!")
        continue
    baralho[nome] = carta
    continuar = input("Inserir outra carta (s/n)? ").lower() in ['s', 'sim']
    i += 1

print("Seu baralho ficou assim:")
for hashmon, poderes in baralho.items():
    print(f"{hashmon} -> {poderes}")
```