**1**

**Estruturas em memória (dicionários e listas)**

Lucas Mendes Marques Gonçalves

***Resumo***

*Revisão de listas e dicionários. Tratamos da inserção e acesso de elementos em uma lista ou dicionário, da verificação de pertencimento e dos erros mais comuns que o python pode retornar.*

*Você tem domínio dessas operações? Sabe ler os erros que o python te fornece? Com esse conteúdo, poderá codificar melhor e debugar seu código de forma muito mais rápida*

*Exibimos também os exercícios resolvidos em vídeo.*

**1.1. Listas**

Listas são uma estrutura fundamental para o armazenamento e acesso de dados em memória. Veja a seguir como se dão essas operações.

**Figura 1.1. Operações em listas**

**>>> #para criar uma lista vazia**

**>>> lista = []**

**>>> #para inserir alguns elementos**

**>>> lista.append(“banana”)**

**>>> lista.append(“melancia”)**

**>>> lista.append(“morango”)**

**>>> lista**

**[‘banana’, ‘melancia’, ‘morango’]**

**>>> #também podemos criar uma lista já com algumas coisas dentro**

**>>> lista = [“gato”,”cachorro”,”peixe palhaço”]**

**>>> lista**

**[‘gato’, ‘cachorro’, ‘peixe palhaço’]**

**>>> #os elementos são acessados com números, que começam do 0**

**>>> lista[0]**

**‘gato’**

**>>> lista[1]**

**‘cachorro’**

**>>> lista[2]**

**‘peixe palhaço’**

**Fonte: do autor, 2021**

**Figura 1.2. Acessos incorretos e respectivos erros**

**>>> #os elementos são acessados com números, que começam do 0**

**>>> lista[0]**

**‘gato’**

**>>> lista[1]**

**‘cachorro’**

**>>> lista[2]**

**‘peixe palhaço’**

**>>> #e se eu for longe demais, a lista reclama**

**>>> lista[3]**

**Traceback (most recent call last):**

**File “/usr/lib/python3.8/idlelib/run.py”, line 559, in runcode**

**exec(code, self.locals)**

**File “<pyshell#15>”, line 1, in <module>**

**IndexError: list index out of range**

**>>> #da mesma forma, ela reclama se eu tentar acessá-la passando uma string em vez**

**>>> #de um número**

**Traceback (most recent call last):**

**File “/usr/lib/python3.8/idlelib/run.py”, line 559, in runcode**

**exec(code, self.locals)**

**File “<pyshell#18>”, line 1, in <module>**

**TypeError: list indices must be integers or slices, not str**

**>>> “gato” in lista #esse é o jeito certo de conferir se um elemento aparece na lista**

**True**

**Fonte: do autor, 2021**

Um erro **IndexError** indica que fizemos um acesso a um elemento que a lista não tem.

Por exemplo, a lista **h=[‘banana’,’abacate’]** tem dois elementos, **h[0]**, cujo valor é **banana** e **h[1]**, cujo valor é **abacate.** Se tentarmos obter **h[2]**, receberemos um **IndexError.**

Um erro TypeError indica que fizemos um acesso com um tipo inválido.

Em **h[1]**, o tipo em questão é inteiro, pois 1 é um inteiro.

Ao fazer **h[“gato”]**, temos um **TypeError**, pois **“gato”** é uma string, um tipo incompatível com o acesso de listas.

**1.2. Dicionários**

**Figura 1.3. Criação e acesso de dicionário**

**>>> #criar um dicionário é fácil**

**>>> agenda = {}**

**>>> #para inserir, basta dizer a chave e o valor**

**>>> agenda[“maria”] = 998223322**

**>>> agenda[“antonieta”] = 35543321**

**>>> agenda[“chave”] = “valor”**

**>>> agenda**

**{‘maria’: 998223322, ‘antonieta’: 35543321, ‘chave’: ‘valor’}**

**>>> #para consultar, basta dizer a chave**

**>>> agenda[“maria”]**

**998223322**

**>>> #Mas cuidado, se a chave não existir, teremos um erro!**

**>>> agenda[“antonio”]**

**Traceback (most recent call last):**

**File “/usr/lib/python3.8/idlelib/run.py”, line 559, in runcode**

**exec(code, self.locals)**

**File “<pyshell#10>”, line 1, in <module>**

**KeyError: ‘antonio’**

**>>> #Vale a pena verificar antes se a chave existe!**

**>>> “antonio” in agenda**

**False**

**>>> “maria” in agenda**

**True**

**Fonte: do autor, 2021**

O dicionário também é fundamental em python (e todas as linguagens de programação modernas). Permite armazenamento e acesso a dados mais estruturados. No exemplo acima, temos uma agenda telefônica.

Um dicionário armazena pares chave-valor. Por exemplo, em:

**{'maria': 998223322, 'antonieta': 35543321, 'chave': 'valor'}**

Temos a chave **“maria”** associada ao valor **998223322.**

Quando vamos acessar uma chave do dicionário, escrevemos, por exemplo, agenda["maria"]. Se cometermos um erro e usarmos uma chave inexistente, como no exemplo da agenda["antônio"], o python lançará um erro chamado **KeyError**.

Devemos verificar se uma determinada chave existe, antes do acesso, para evitar tais erros. Observe como na próxima figura.

**Figura 1.4. Verificação de pertencimento**

**>>> #Vale a pena verificar antes se a chave existe!**

**>>> “antonio” in agenda**

**False**

**>>> “maria” in agenda**

**True**

**>>> #Atenção, o in verifica apenas chaves**

**>>> “maria” in agenda**

**True**

**>>> 998223322 in agenda**

**False**

**>>> #Se quiser, podemos verificar valores, mas temos que ser explicitos**

**>>> 998223322 in agenda.vales()**

**True**

**>>> #Aliás, não é uma má ideia sermos explicitos também na hora das chaves**

**>>> “maria” in agenda.keys()**

**True**

**>>> “maria” in agenda**

**True**

**Fonte: do autor, 2021**

Alguns programadores cometem um erro, e em vez de adicionar um elemento em um dicionário, sobrescrevem o dicionário. Veja o erro na ilustração.

**Figura 1.5. Sobrescrita acidental**

**>>> agenda = {“maria”: 998223322, “antonieta”: 35543321, “chave”: “valor”}**

**>>> agenda**

**{‘maria’: 998223322, ‘antonieta’: 35543321, ‘chave’: ‘valor’}**

**>>> #Cuidado, a sintaxe de adicionar valores é**

**>>> agenda[“bruno”] = 44443232**

**>>> agenda**

**{‘maria’: 998223322, ‘antonieta’: 35543321, ‘chave’: ‘valor’, ‘bruno’: 44443232}**

**>>> #Algumas pessoas as vezes fazem**

**>>> agenda = {“bruno”: 44443232}**

**>>> #mas isso sobreescreve o dicionário**

**>>> agenda**

**{‘bruno’: 44443232}**

**Fonte: do autor, 2021**

**1.3. Exercícios do vídeo**

No vídeo, resolvemos alguns exercícios práticos para reforçar e esclarecer os conceitos.

Aqui estão eles. Na próxima sessão teremos as respostas. **Não deixe de tentar fazer antes de ver as respostas!** Escreva suas respostas no computador ou em um papel antes.

**Codificação 1.1. Exercício**

dic = {

"alimentos": {

"pizzas": ["margueritta", "mussarella",

"frango com catupiry", "portuguesa"],

"bolos": ("floresta negra",

"red velvet",

"de laranja", "dá vó"),

"calorias": {

"leite": 129, "fatia pizza": 320,

"agua": 0, "maça": 95

}

},

"linguagens": [

{"nome": "javascript", "criacao": 1996,

"paradigma": ["eventos","funcional"]},

{"nome": "python", "criacao": 1991,

"paradigma": ["orientada a objetos","estruturada"]},

{"nome": "haskell", "criacao": 1990,

"paradigma": ["funcional"]}

]

}

#1. quantas chaves tem o dicionario dic?

print("r1",len(dic))

#2. dic['linguagens'] é uma tupla, um dicionário ou uma lista?

print("r2", type(dic['linguagens']))

#3. Como eu faço para mostrar todos os bolos?

# (escreva o código!)

#4. Qual o tipo da lista de todos os bolos?

print("r4", type(dic['alimentos']['bolos']))

#5. O que o seguinte acesso imprime? Se ele dá erro, qual o erro? Se dá erro, como corrigir?

print("r5", dic["linguagens"]["javascript"]["criacao"])

#6 O que o seguinte acesso imprime? Se ele dá erro, qual o erro? Se dá erro, como corrigir?

print("r6", dic["linguagens"][2] == "haskell")

#7 O que o seguinte acesso imprime? Se ele dá erro, qual o erro? Se dá erro, como corrigir?

print("r7", dic["alimentos"]["pizzas"][2] == "mussarella")

#8 O que o seguinte acesso imprime? Se ele dá erro, qual o erro? Se dá erro, como corrigir?

print("r8", 1996 in dic['linguagens'][0])

#9 O que o seguinte acesso imprime? Se ele dá erro, qual o erro? Se dá erro, como corrigir?

print("r9", "criacao" in dic['linguagens'][0])

#9 O que o seguinte acesso imprime? Se ele dá erro, qual o erro? Se dá erro, como corrigir?

print("pudim" in dic["sobremesas"]["doces"])

#10 Escreva uma função "mais velha" que

# recebe um dicionário como dic e

# retorna (isso é diferente de imprimir!) a linguagem de programação mais velha da nossa lista

#11 Escreva uma função que retorna uma lista (sem repetições) de paradigmas de linguagens de programação

**Fonte: do autor, 2021**

**1.4. Exercícios corrigidos**

**Codificação 1.2. Exercício - 2**

dic = {

"alimentos": {

"pizzas": ["margueritta", "mussarella",

"frango com catupiry", "portuguesa"],

"bolos": ("floresta negra",

"red velvet",

"de laranja", "dá vó"),

"calorias": {

"leite": 129, "fatia pizza": 320,

"agua": 0, "maça": 95

}

},

"linguagens": [

{"nome": "javascript", "criacao": 1996,

"paradigma": ["eventos","funcional"]},

{"nome": "python", "criacao": 1991,

"paradigma": ["orientada a objetos","estruturada"]},

{"nome": "haskell", "criacao": 1990,

"paradigma": ["funcional"]}

]

}

# #Só se aprende fazendo. PAUSE O VIDEO E TENTE RESPONDER!

# #Se possível, FAÇA JUNTO NO SEU COMPUTADOR

# #1. quantas chaves tem o dicionario dic?

print("r1",len(dic))

# duas, a chave "alimentos" e a chave "linguagens"

# #2. dic['linguagens'] é uma tupla, um dicionário ou uma lista?

print("r2", type(dic['linguagens']))

# É uma lista, repare nos colchetes inicial e final

# #3. Como eu faço para mostrar todos os bolos?

# # (escreva o código!)

dic["alimentos"]["bolos"]

# #4. Qual o tipo da lista de todos os bolos?

print("r4", type(dic['alimentos']['bolos']))

# Tupla

# #5. O que o seguinte acesso imprime? Se ele dá erro, qual o erro? Se dá erro, como corrigir?

print("r5", dic["linguagens"]["javascript"]["criacao"])

# TypeError: list indices must be integers or slices, not str

# O erro ocorre porque dic["linguagens"] é uma lista

# O acesso correto seria dic["linguagens"][0] em vez de dic["linguagens"]["javascript"]

# #6 O que o seguinte acesso imprime? Se ele dá erro, qual o erro? Se dá erro, como corrigir?

print("r6", dic["linguagens"][2] == "haskell")

# False

# dic["linguagens"][2] é o dicionário {"nome": "haskell", "criacao": 1990, "paradigma": ["funcional"]}

# Esse dicionário não é a mesma coisa que a string "haskell"

# #7 O que o seguinte acesso imprime? Se ele dá erro, qual o erro? Se dá erro, como corrigir?

print("r7", dic["alimentos"]["pizzas"][2] == "mussarella")

# False

# dic["alimentos"]["pizzas"] é a lista ["margueritta", "mussarella", "frango com catupiry", "portuguesa"]

# A sua posição 0 é "margueritta"

# A sua posição 1 é "mussarella"

# A sua posição 2 é "frango com catupiry"

# #8 O que o seguinte acesso imprime? Se ele dá erro, qual o erro? Se dá erro, como corrigir?

print("r8", 1996 in dic['linguagens'][0])

#False, pois 1996 não é uma \*\*chave\*\* do dicionário dic['linguagens'][0]

#Alias, dic['linguagens'][0] é {"nome": "javascript", "criacao": 1996,

# "paradigma": ["eventos","funcional"]}

# #9 O que o seguinte acesso imprime? Se ele dá erro, qual o erro? Se dá erro, como corrigir?

print("r9", "criacao" in dic['linguagens'][0])

#True, pois criação é uma chave do dicionário dic['linguagens'][0]

#Alias, dic['linguagens'][0] é {"nome": "javascript", "criacao": 1996,

# "paradigma": ["eventos","funcional"]}

# Não há como corrigir, os dados não existem

# #9 O que o seguinte acesso imprime? Se ele dá erro, qual o erro? Se dá erro, como corrigir?

print("ex9b", "pudim" in dic["sobremesas"]["doces"])

#KeyError: 'sobremesas', pois o dicionario dic só tem as chaves "alimentos" e "linguagens"

#10 Escreva uma função "mais velha" que

# recebe um dicionário como dic e

# retorna (isso é diferente de imprimir!) a linguagem de programação mais velha da nossa lista

def mais\_velha(dic):

lista\_linguagens = dic['linguagens']

ling\_velha = lista\_linguagens[0]

for linguagem in lista\_linguagens:

if linguagem['criacao'] < ling\_velha['criacao']:

ling\_velha = linguagem

return ling\_velha

#11 Escreva uma função que retorna uma lista (sem repetições) de paradigmas de linguagens de programação

def todos\_paradigmas(dic):

lista\_linguagens = dic['linguagens']

paradigmas = []

for linguagem in lista\_linguagens:

paradigmas\_da\_linguagem = linguagem['paradigma']

for p in paradigmas\_da\_linguagem:

if p not in paradigmas:

paradigmas.append(p)

return paradigmas

**Fonte: do autor, 2021**

**1.5. Finalização**

Agora você relembrou e tem mais desenvoltura para usar listas e dicionários em python. Essas estruturas são fundamentais, e você deve dominar todos os elementos apresentados nessa aula para ter sucesso como programador(a) python.

**Referências**

PYTHON. **Estruturas de dados**. Python, 27 jul. 2021. Disponível em:<https://docs.python.org/pt-br/3/tutorial/datastructures.html>. Acesso em: 29 jul. 2021.