

Unidade IV

7 LISTAS E DICIONÁRIOS

As listas e dicionários são estruturas cada vez mais usadas para resolver problemas em computação. Neste tópico estudaremos de forma um pouco mais detalhada listas, aprenderemos o conceito de dicionários e, por fim, como trabalhar com eles.

7.1 Principais métodos para manipulação de listas

Com o que conhecemos até aqui sobre listas, já é possível resolver a maioria dos problemas de forma satisfatória. Com o aprendizado e práticas das funções e métodos de manipulação de listas, o trabalho de resolver um problema e escrever um algoritmo se tornará bem mais fácil.

Seguem os métodos e exemplo de seu uso em listas:

Quadro 10

Método	Descrição
append()	Adiciona um elemento ao final da lista
clear()	Remove todos os elementos da lista
copy()	Retorna uma cópia da lista
count()	Retorna o número de elementos que são iguais ao valor especificado
extend()	Adiciona os elementos de uma lista ao final da lista que chamou o método
index()	Retorna o índice do primeiro elemento encontrado com o valor especificado
insert()	Adiciona um elemento na posição pretendida
pop()	Remove o último elemento da lista ou posição pretendida
remove()	Remove o item com o valor especificado
reverse()	Inverte a ordem da lista
sort()	Ordena a lista

Exemplo 1:

```
carros = ["monza", "kombi", "fusca"]  
# insere um carro no fim da lista  
carros.append("opala")  
print(carros)
```

Saída:

```
['monza', 'kombi', 'fusca', 'opala']
```

Exemplo 2:

```
carros = ["monza", "kombi", "fusca"]  
# insere um carro no início da lista (posição 0)  
carros.insert(0, "opala")  
print(carros)
```

Saída:

```
['opala', 'monza', 'kombi', 'fusca']
```

Exemplo 3:

```
carros = ["monza", "kombi", "fusca"]  
# remove o último elemento da lista  
carros.pop()  
print(carros)
```

Saída:

```
['monza', 'kombi']
```

Exemplo 4:

```
carros = ["monza", "kombi", "fusca"]  
# remove a kombi da lista  
carros.remove('kombi')  
print(carros)
```

Saída:

```
['monza', 'fusca']
```

Exemplo 5:

```
carros = ["monza", "kombi", "fusca"]  
# remove todos os elementos da lista  
carros.clear()  
print(carros)
```

Saída:

```
[]
```

Exemplo 6:

```
carros = ["monza", "kombi", "fusca"]
carros_2 = carros
# carros e carros_2 apontam para a mesma lista
# o que alterar em um altera a outra também
# é necessário fazer uma cópia explícita da lista
carros_3 = carros.copy()
carros.clear()
# apagou carros e carros_2
print(carros)
print(carros_2)
print(carros_3)
```

Saída:

```
[]
```

```
[]
```

```
['monza', 'kombi', 'fusca']
```

Exemplo 7:

```
carros = ["monza", "kombi", "fusca", "kombi"]
# conta quantas kombis tem
kombis = carros.count('kombi')
print(kombis)
```

Saída:

```
2
```

Exemplo 8:

```
carros = ["monza", "kombi", "fusca"]
carros_2 = ["ka", "elantra", "tucson"]
# junta duas listas
carros.extend(carros_2)
print(carros)
```

Saída:

```
['monza', 'kombi', 'fusca', 'ka', 'elantra', 'tucson']
```

Exemplo 9:

```
carros = ["monza", "kombi", "fusca"]  
# retorna o índice da kombi  
print(carros.index('kombi'))
```

Saída:

1

Exemplo 10:

```
carros = ["monza", "kombi", "fusca"]  
# retorna a lista ordenada  
carros.sort()  
print(carros)  
# coloca a lista em ordem reversa  
carros.reverse()  
print(carros)
```

Saída:

['fusca', 'kombi', 'monza']

['monza', 'kombi', 'fusca']

Com o conhecimento de manipulação de listas deste tópico, é possível criar uma estrutura de dados mais sofisticados como filas e pilhas.



Lembrete

A lista em Python é uma coleção de elementos ordenada e mutável. Ordenada quer dizer que existe uma sequência e ordem; mutável significa que podemos modificar seus elementos.

7.2 Conceito de dicionários

O dicionário em Python é uma coleção de elementos ordenada e mutável. Ordenada quer dizer que existe uma sequência e ordem, e mutável significa que podemos modificar seus elementos. O dicionário não permite a inclusão de elementos duplicados.

Dicionários são criados com abre e fecha chaves. São constituídos de pares chamados de chave e valor. Hoje, os dicionários são usados como estruturas de registros de base de dados.

Todos os tipos estruturados de dados que estudamos até o momento tiveram inteiros como índices; os dicionários podem ter qualquer tipo imutável como índice. Os índices são suas chaves.

Exemplo 1:

```
carro = {  
    "marca": "Ford",  
    "modelo": "Ka",  
    "ano": 2005,  
    "cor": "preto"  
}  
print(carro["modelo"])  
print('Tamanho: ', len(carro))  
print('Tipo: ', type(carro))
```

Saída:

Ka

Tamanho: 3

Tipo: <class 'dict'>

Exemplo 2:

```
carro = {  
    "marca": "Ford",  
    "modelo": "Ka",  
    "ano": 2005,  
    "cor": "preto"  
}  
# retorna as chaves dos dicionários  
print(carro.keys())  
# retorna os valores dos dicionários  
print(carro.values())  
# retorna a chave e valor como tuplas  
print(carro.items())
```

Saída:

dict_keys(['marca', 'modelo', 'ano', 'cor'])

dict_values(['Ford', 'Ka', 2005, 'preto'])

dict_items([('marca', 'Ford'), ('modelo', 'Ka'), ('ano', 2005), ('cor', 'preto')])

Exemplo 3:

```
carro = {  
    "marca": "Ford",  
    "modelo": "Ka",  
    "ano": 2005,  
    "cor": "preto"  
}  
# modifica o modelo  
carro["modelo"] = 'Corsa'  
print(carro)
```

Saída:

```
{'marca': 'Ford', 'modelo': 'Corsa', 'ano': 2005, 'cor': 'preto'}
```

É importante mostrar também como acessamos os dicionários através da iteração do laço for. Seguem os exemplos para essa tarefa.

Exemplo 1:

```
carro = {  
    "marca": "Ford",  
    "modelo": "Ka",  
    "ano": 2005,  
    "cor": "preto"  
}  
# imprime todas as chaves  
for i in carro:  
    print(i)
```

Saída:

```
marca  
modelo  
ano  
cor
```

Exemplo 2:

```
carro = {  
    "marca": "Ford",  
    "modelo": "Ka",  
    "ano": 2005,  
    "cor": "preto"  
}  
# imprime todos os valores  
for i in carro:  
    print(carro[i])
```

Saída:

Ford

Ka

2005

preto

Exemplo 3:

```
carro = {  
    "marca": "Ford",  
    "modelo": "Ka",  
    "ano": 2005,  
    "cor": "preto"  
}  
# imprime todos as chaves usando .keys()  
for i in carro.keys():  
    print(carro[i])
```

Saída:

marca

modelo

ano

cor

Exemplo 4:

```
carro = {  
    "marca": "Ford",  
    "modelo": "Ka",  
    "ano": 2005,  
    "cor": "preto"  
}  
# imprime todos os valores usando .values()  
for i in carro.values():  
    print(carro[i])
```

Saída:

Ford

Ka

2005

preto

Exemplo 5:

```
carro = {  
    "marca": "Ford",  
    "modelo": "Ka",  
    "ano": 2005,  
    "cor": "preto"  
}  
# imprime todos os valores usando .items()  
for x, y in carro.items():  
    print(x, y)
```

Saída:

```
marca Ford  
modelo Ka  
ano 2005  
cor preto
```

Existem várias funções que facilitam o programador a trabalhar com dicionários, tais como: adição, remoção, procura de elementos e outros. Essas facilidades serão apresentadas na próxima seção.



Observação

Os dicionários a partir do Python 3.7 são ordenados. Na versão do Python 3.6 e nas inferiores, eles não são estruturas de dados ordenadas.

7.2.1 Principais métodos para manipulação de dicionários

Como vimos em listas, os dicionários também têm métodos que podem ajudar nas tarefas comuns de manipulação de dados. Nesta seção, vamos estudar vários métodos dos dicionários que utilizam algoritmos internos otimizados e ajudam muito na resolução de problemas.

Seguem os métodos e exemplos de seu uso em dicionários. Note que os métodos `keys()`, `values()` e `items()` já foram demonstrados em exemplos na seção anterior.

Quadro 11

Método	Descrição
<code>clear()</code>	Remove todos os elementos do dicionário
<code>copy()</code>	Retorna uma cópia do dicionário
<code>fromkeys()</code>	Retorna um dicionário com as chaves especificadas e valor
<code>get()</code>	Retorna o valor de uma chave especificada

Método	Descrição
items()	Retorna uma lista contendo uma tupla para cada chave e valor
keys()	Retorna uma lista contendo as chaves do dicionário
pop()	Remove o elemento com a chave especificada
popitem()	Remove a última chave e o valor inserido
setdefault()	Retorna o valor da chave especificada. Se ela não existe, é inserida
update()	Atualiza o dicionário com o par chave-valor especificado
values()	Retorna uma lista de todos os valores no dicionário

Exemplo 1:

```
carro = {  
    "marca": "Ford",  
    "modelo": "Ka",  
    "ano": 2005,  
    "cor": "preto"  
}  
# remove todas as chaves e valores do dicionário  
carro.clear()  
print(carro)
```

Saída:
{ }

Exemplo 2:

```
carro = {  
    "marca": "Ford",  
    "modelo": "Ka",  
    "ano": 2005,  
    "cor": "preto"  
}  
# faz uma cópia do dicionário, já que a atribuição somente cria referência  
# ao dicionário existente  
carro_2 = carro.copy()  
print(carro_2)
```

Saída:
{'marca': 'Ford', 'modelo': 'Ka', 'ano': 2005, 'cor': 'preto'}

Exemplo 3:

```
# retorna um novo dicionário das chaves especificadas
carro = dict.fromkeys(['marca', 'modelo'])
print(carro)
```

Saída:

```
{'marca': None, 'modelo': None}
```

Exemplo 4:

```
carro = {
    "marca": "Ford",
    "modelo": "Ka",
    "ano": 2005,
    "cor": "preto"
}
# retorna o valor da marca
print(carro.get('marca'))
```

Saída:

```
Ford
```

Exemplo 5:

```
carro = {
    "marca": "Ford",
    "modelo": "Ka",
    "ano": 2005,
    "cor": "preto"
}
# remove o ano
carro.pop('ano')
print(carro)
```

Saída:

```
{'marca': 'Ford', 'modelo': 'Ka', 'cor': 'preto'}
```

Exemplo 6:

```
carro = {  
    "marca": "Ford",  
    "modelo": "Ka",  
    "ano": 2005,  
    "cor": "preto"  
}  
# remove o último item  
carro.popitem()  
print(carro)
```

Saída:

```
{'marca': 'Ford', 'modelo': 'Ka', 'ano': 2005}
```

Exemplo 7:

```
carro = {  
    "marca": "Ford",  
    "modelo": "Ka",  
    "ano": 2005,  
    "cor": "preto"  
}  
# retorna o valor da chave, caso não exista adiciona a chave-valor  
print(carro.setdefault('ano', '2020'))
```

Saída:

```
2005
```

Exemplo 8:

```
carro = {  
    "marca": "Ford",  
    "modelo": "Ka",  
    "ano": 2005,  
    "cor": "preto"  
}  
# insere nova chave-valor  
carro.update({"fabricacao": "2004"})  
print(carro)
```

Saída:

```
{'marca': 'Ford', 'modelo': 'Ka', 'ano': 2005, 'cor': 'preto', 'fabricacao': '2004'}
```

É importante executar todos os exemplos em Python para fixação. Além disso, é recomendado fazer todos ou a maior quantidade de exercícios possível. Só a prática forma bons desenvolvedores.

No próximo tópico, aprofundaremos o conhecimento, aprendendo sobre módulos e bases de dados.

8 MÓDULOS E BANCO DE DADOS

No último tópico deste livro-texto, estudaremos dois conteúdos mais avançados: módulos e acesso programático aos bancos de dados. Uma biblioteca Python é uma coleção de módulos. Já utilizamos módulos externos ao Python – um deles é o NumPy –, mas o intuito desta seção é capacitá-lo a criar seus próprios módulos e dar início à reutilização de código. Sobre banco de dados, não é objetivo ensinar a modelar o banco, existem disciplinas inteiras sobre o assunto. Contudo, daremos aqui o ferramental necessário que deverá ser empregado quando chegar o momento de utilizar o Python em conjunto com o gerenciador de banco de dados MySQL.

8.1 Introdução aos módulos

Podemos entender um módulo como uma biblioteca de códigos prontos, um arquivo contendo uma série de funções que queremos incluir em nosso aplicativo e que pode ser reutilizado em outro aplicativo que desenvolveremos no futuro.

Para construir um módulo, basta criar um arquivo com suas funções e códigos, e salvá-lo com o nome do seu módulo mais a extensão .py.

Exemplo:

```
# salvo com meu_modulo.py
def imprime(texto):
    Print(texto)
```

Para utilizarmos o meu_modulo.py, em outro arquivo no mesmo diretório, ou no próprio console do Python:

Exemplo 1:

```
# utilizando a instrução import
import meu_modulo
meu_modulo.imprime('Meu primeiro módulo!')
```

Exemplo 2:

```
# arquivo carros.py
carro = {
    "marca": "Ford",
    "modelo": "Ka",
    "ano": 2005
}
# arquivo script.py na mesma pasta de carros.py
import carros
x = carros.carro['modelo']
print(x)
```

Saída:

Ka

É possível dar apelidos aos módulos usando a instrução `as`. Sendo assim, pode ser um nome menor, o que economiza bastante digitação, caso o módulo seja muito usado.

Exemplo:

```
# arquivo carros.py
carro = {
    "marca": "Ford",
    "modelo": "Ka",
    "ano": 2005
}
# arquivo script.py na mesma pasta de carros.py
import carros as cs
x = cs.carro['modelo']
print(x)
```

Saída:

Ka



Observação

Quando utilizamos uma função de um módulo, usamos a sintaxe: `nome_modulo.nome_funcao`. O mesmo acontece com uma variável de um módulo.



Lembrete

Em Python, podemos entender um módulo como uma biblioteca de códigos prontos ou ele pode se referir ao operador módulo, que retorna o resto de uma divisão matemática.

8.1.1 SYS

Na linguagem Python, existem vários módulos que já vêm por padrão na instalação. Um deles é o `sys`, que permite acesso a algumas variáveis usadas ou mantidas pelo interpretador e contém funções que interagem com o Python.

Muitas vezes, programas em Python precisam processar argumentos passados no terminal quando a aplicação é chamada. A lista de argumentos passados na linha de comando fica armazenada na variável `argv` dentro do módulo `sys`.

Exemplo:

```
# O nome do arquivo deste programa é sys.py
import sys
print(sys.argv)
```

Quando a linha é executada da seguinte forma: `python sys.py 1 2 3 abc`

Temos a saída:

```
['sys.py', '1', '2', '3', 'abc']
```

```

C:\Users\Public>python sys.py 1 2 3 abc
['sys.py', '1', '2', '3', 'abc']
C:\Users\Public>_

```

Figura 62 – Execução do programa `sys.py`

O módulo `sys` também dispõe de acesso à saída, entrada e erro padrão (`stdin`, `stdout` e `stderr`), além de ter uma função para encerrar a execução do programa `sys.exit()`.

Exemplo:

```
# enviando mensagem de erro ao terminal
sys.stderr.write('Erro 404')
# terminando programa
sys.exit(-1)
```

Saída:
Erro 404



Saiba mais

Para ter acesso à lista e à documentação dos módulos inclusos por padrão na linguagem Python, acesse:

PYTHON. *Python Module Index*. [s.d.].

Disponível em: <https://cutt.ly/nmg6iZf>. Acesso em 12 fev. 2021.

8.1.2 Instalando módulos

A comunidade Python é muito ativa no desenvolvimento de módulos e manutenção dos que já existem. O universo é gigante, e para quase tudo que você pensar existe uma biblioteca em Python que pode ajudá-lo. Sabemos que uma biblioteca Python é uma coleção de módulos. Já instalamos a biblioteca NumPy para manipulação de matrizes.

Nesta seção, vamos retomar o processo de instalação através da ferramenta Pip (Package Installer for Python). Ele vem embutido na instalação do pacote padrão do Python, nas versões mais novas.

Antes de usar o Pip, é recomendado atualizar o programa, usando o comando no terminal:
`python -m pip install --upgrade pip`

Exemplo de comando para instalação da biblioteca Arrow, com funções de data e horário: `pip install arrow`

```
Prompt de Comando

C:\Users\Public>python -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\ricardo.silva\appdata\local\programs\python\python37-32\lib\site-packages (21.0.1)

C:\Users\Public>pip install arrow
Collecting arrow
  Downloading arrow-1.0.2-py3-none-any.whl (54 kB)
    | 54 kB 2.0 MB/s
Requirement already satisfied: python-dateutil>=2.7.0 in c:\users\ricardo.silva\appdata\roaming\python\python37-32\lib\site-packages (from arrow) (2.8.1)
Requirement already satisfied: typing-extensions in c:\users\ricardo.silva\appdata\local\programs\python\python37-32\lib\site-packages (from arrow) (3.7.4.3)
Requirement already satisfied: six>=1.5 in c:\users\ricardo.silva\appdata\roaming\python\python37\site-packages (from python-dateutil>=2.7.0->arrow) (1.12.0)
Installing collected packages: arrow
Successfully installed arrow-1.0.2

C:\Users\Public>
```

Figura 63 – Upgrade do Pip e instalação do Arrow no Windows



Saiba mais

Para procurar bibliotecas e módulos Python, acesse o site:

Disponível em: <https://cutt.ly/bmSuZEI>. Acesso em: 13 jul. 2021.

8.1.3 From...Import

Podemos escolher importar somente partes de um módulo. Isso ajuda ao Python otimizar o que vai ou não para a memória da máquina. Provavelmente, utilizamos nos módulos padrão da linguagem Python somente uma ou duas funções em nosso programa. Uma boa prática é usar as instruções from...import.

Exemplo:

```
# do módulo math utilizaremos somente a função sqrt() - raiz quadrada
from math import sqrt
print(sqrt(25))
```

Saída:
5.0

8.1.4 Arquivos byte-compiled .pyc

O Python é uma linguagem interpretada. Isso quer dizer que ele lê os programas fontes escritos em texto e os traduz, no momento da execução, para código de máquina. Importar um módulo grande e traduzi-lo em código de máquina no momento da execução é uma tarefa muito mais custosa.

Para minimizar esse problema de desempenho, o Python faz algumas otimizações e cria o chamado byte-compiled. É um arquivo com a extensão .pyc que contém um código intermediário otimizado, o qual é mais fácil de traduzir para o código de máquina.

Os arquivos .pyc são independentes de sistemas operacionais e podem ser movidos do Windows para o Linux, por exemplo, mas não se preocupe, esses arquivos são criados automaticamente na primeira execução do programa caso eles ainda não existam.

8.2 Banco de dados

Para a maioria dos técnicos de tecnologia da informação, banco de dados é sinônimo de SGBD (Sistema de Gerenciamento de Banco de Dados). O SGBD é o sistema que gerencia vários bancos de dados, e bancos de dados são coleções de tabelas, índices, procedimentos etc. É dentro das tabelas que estão os dados; essas tabelas são manipuladas por uma linguagem chamada SQL (Linguagem Estruturada de Consulta).

O objetivo desta seção é prover o básico para que seja possível a conexão entre um programa em Python e o SGBD MySQL. Não entraremos em detalhes e nem em conceitos sobre banco de dados relacionais e a linguagem SQL. Existem disciplinas em seu curso focadas somente no ensino de banco de dados e SQL.

O Python possui módulos específicos para tratar conexões com banco de dados. Vamos utilizar o popular MySQL, que é um SGBD que tem uma versão gratuita e está em crescente utilização pelas empresas.

8.2.1 Conexão com banco de dados MySQL

Antes de começar a manipular o banco de dados MySQL programaticamente pelo Python, temos que completar três passos:

1. Instalar o MySQL.
2. Instalar o módulo conector Python.
3. Construir o programa e conectar ao SGBD.

A instalação do MySQL depende do sistema operacional. Faça o download no site oficial da última versão. Escolha a versão gratuita que tem o nome de MySQL Community no link <https://dev.mysql.com/downloads/> (acesso em: 20 jun. 2021). Siga o processo de instalação, escolha uma senha para o usuário root e anote-a para não esquecer.

A instalação do conector é feita pelo utilitário Pip:

Exemplo no terminal:

```
pip install mysql-connector-python
```

Podemos fazer a criação do banco de dados, tabelas e inclusão de dados, via código Python (feita na próxima seção). Agora, vamos fazer a conexão com o MySQL:

Exemplo:

```
import mysql.connector
# localhost é a sua máquina local
# root é o usuário administrador do banco, use o seu
# use sua senha em password
bd = mysql.connector.connect(
    host="localhost",
    user="root",
    password="<senha aqui entre aspas!>"
)
print(bd)
```

Saída:

```
<mysql.connector.connection.MySQLConnection object at 0x034BFC10>
```

Pronto, estamos conectados ao SGBD MySQL.

8.2.2 Leitura, gravação, alteração e exclusão

Para iniciar a leitura, a gravação, a alteração e a exclusão de dados, devemos concluir algumas tarefas após a conexão:

1. Criar o banco de dados.
2. Conectar ao banco de dados.
3. Criar a tabela ou as tabelas.

Na seção anterior, usamos o usuário administrador (root) para fazer a conexão, o que não é uma boa prática de segurança. Nas aplicações profissionais, são criados usuários de serviço somente com as permissões necessárias, diminuindo assim os riscos de segurança. Nos exemplos, usaremos o root.

Exemplo de criação de banco de dados:

```
import mysql.connector
bd = mysql.connector.connect(
    host="localhost",
    user="root",
    password="<senha aqui entre aspas!>"
)
bd_cursor = bd.cursor()
bd_cursor.execute("CREATE DATABASE primeirobanco")
```

O objeto MySQLcursor() interage diretamente com a conexão criada ao MySQL e pode executar os comandos SQL no banco.

Exemplo para listar os bancos de dados no SGBD:

```
import mysql.connector
bd = mysql.connector.connect(
    host="localhost",
    user="root",
    password="<senha aqui entre aspas!>"
)
bd_cursor = bd.cursor()
# o comando SQL: SHOW DATABASES lista os bancos
bd_cursor.execute("SHOW DATABASES")
for banco in bd_cursor:
    print(banco)
```

Saída:

```
('information_schema',)
('mysql',)
('performance_schema',)
('primeirobanco',)
('sys',)
```

Existem duas formas mais comuns de conectar ao banco de dados para criar seus elementos, como tabelas e índices. Uma delas é usar o comando do MySQL "use <nome_do_banco>"; a outra é, já na conexão, adicionar o argumento "database". A segunda maneira só funciona caso o banco de dados já exista anteriormente.

Exemplo com argumento database:

```
import mysql.connector
bd = mysql.connector.connect(
    host="localhost",
    user="root",
    password="<senha aqui entre aspas!>",
    database="primeirobanco"
)
```

Vamos criar uma tabela para inserir os dados:

```
import mysql.connector
bd = mysql.connector.connect(
    host="localhost",
    user="root",
    password="<senha aqui entre aspas!>",
    database="primeirobanco"
)
bd_cursor = bd.cursor()
bd_cursor.execute("CREATE TABLE alunos (id INT AUTO_INCREMENT PRIMARY KEY, nome
VARCHAR(255), endereco VARCHAR(255))")
bd_cursor.execute("SHOW TABLES")
for tabela in bd_cursor:
    print(tabela)
```

Saída:
(‘alunos’,)

Já temos o banco de dados e a tabela criados, já podemos começar a trabalhar com o banco, gravando (inserindo) dados nele. Mostraremos também comandos para leitura, alteração e exclusão.

Exemplo de gravação de registros (inclusão):

```
import mysql.connector
bd = mysql.connector.connect(
    host="localhost",
    user="root",
    password="<senha aqui entre aspas!>",
    database="primeirobanco"
)
bd_cursor = bd.cursor()
sql = "INSERT INTO alunos (nome, endereco) VALUES (%s, %s)"
```

```
valores = [  
    ('Ana', 'Rua Joao do Pulo, 76'),  
    ('Beatriz', 'Alameda Jau, 897'),  
    ('Carlos', 'Avenida Rita Lia, 472'),  
    ('Daniel', 'Rua Oculta, sem numero'),  
]  
bd_cursor.executemany(sql, valores)  
# o comando commit() é necessário para salvar os dados no banco  
bd.commit()  
print(bd_cursor.rowcount, "registros inseridos.")
```

Saída:

4 registros inseridos.

A leitura dos dados é feita através de uma consulta, que usa o comando SELECT em SQL para pesquisar na base de dados e devolver os resultados. Vamos ver uma consulta simples que retorna todos os dados de uma tabela.

Exemplo de leitura (consulta) em banco de dados:

```
import mysql.connector  
bd = mysql.connector.connect(  
    host="localhost",  
    user="root",  
    password="<senha aqui entre aspas!>",  
    database="primeirobanco"  
)  
bd_cursor = bd.cursor()  
  
bd_cursor.execute("SELECT * FROM alunos")  
alunos = bd_cursor.fetchall()  
for aluno in alunos:  
    print(aluno)
```

Saída:

```
(1, 'Ana', 'Rua Joao do Pulo, 76')  
(2, 'Beatriz', 'Alameda Jau, 897')  
(3, 'Carlos', 'Avenida Rita Lia, 472')  
(4, 'Daniel', 'Rua Oculta, sem numero')
```

A alteração nos registros dos bancos de dados é chamada de atualização (update). É utilizado o comando SQL update e após a alteração o comando commit() para efetivá-la.

Exemplo de update:

```
import mysql.connector
bd = mysql.connector.connect(
    host="localhost",
    user="root",
    password="<senha aqui entre aspas!>",
    database="primeirobanco"
)
bd_cursor = bd.cursor()

sql = "UPDATE alunos SET endereco = 'Rua das Camélias, 50' WHERE nome = 'Ana'"
bd_cursor.execute(sql)
bd.commit()
print(bd_cursor.rowcount, "registro alterado.")
```

Saída:

1 registro alterado.

A última operação tratada em nossos estudos seria a operação de exclusão ou deleção. Nessa operação excluimos um ou mais registros de nosso banco de dados.

Exemplo:

```
import mysql.connector
bd = mysql.connector.connect(
    host="localhost",
    user="root",
    password="<senha aqui entre aspas!>",
    database="primeirobanco"
)
bd_cursor = bd.cursor()

sql = "DELETE FROM alunos WHERE nome = 'Daniel'"
bd_cursor.execute(sql)

bd.commit()
print(bd_cursor.rowcount, "registro deletado.")
```

Saída:

1 registro deletado.

Para visualizar as alterações, faça uma consulta utilizando o comando SELECT.

Exemplo de aplicação

Banco de dados é uma disciplina complexa e extensa, e o conteúdo não se esgota por aqui. Como sugestão, estude a linguagem SQL e pratique com Python.



Saiba mais

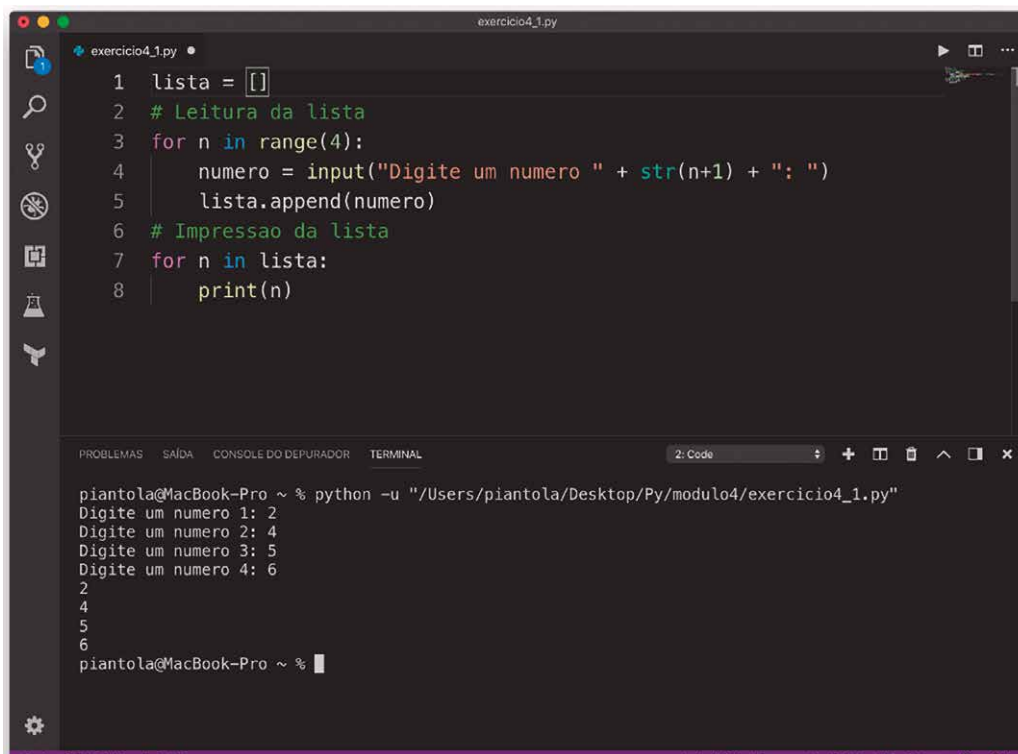
Para se aprofundar em programação Python para o SGBD MySQL, leia a documentação diretamente no site oficial do conector:

MYSQL. *Chapter 6 MySQL Connector/Python Developer Guide*. [s.d.]. Disponível em: <https://cutt.ly/SmhosuF>. Acesso em: 22 jun. 2021.

8.3 Exercícios sugeridos

1. Crie um programa que leia uma lista de quatro números inteiros e imprima-os na tela.
2. Crie um programa que leia uma lista com cinco notas, calcule a média e mostre na tela.
3. Crie um programa que leia uma lista de seis números inteiros e imprima-os na ordem inversa da leitura.
4. Construa um programa que receba do usuário uma lista com oito números inteiros, calcule e imprima a soma dos quadrados desses números.
5. Faça um programa e crie um dicionário colocando nele seus dados: nome, idade, telefone e endereço.
6. Usando o dicionário criado anteriormente, imprima na tela seu nome e telefone.
7. Construa um dicionário e coloque nele os dados de um carro digitados pelo usuário: Marca, Modelo, Placa e Ano. Imprima na tela o resultado.
8. Agora, usando o dicionário do exercício anterior, imprima todos os itens do dicionário na forma chave: valor.
9. Crie um programa que utilize o modulo SYS para ler os argumentos de linha do terminal e imprima-os na tela.
10. Crie um banco de dados usando Python e MySQL chamado 'empregados' e uma tabela com os campos: 'id', 'nome' e 'endereço'.

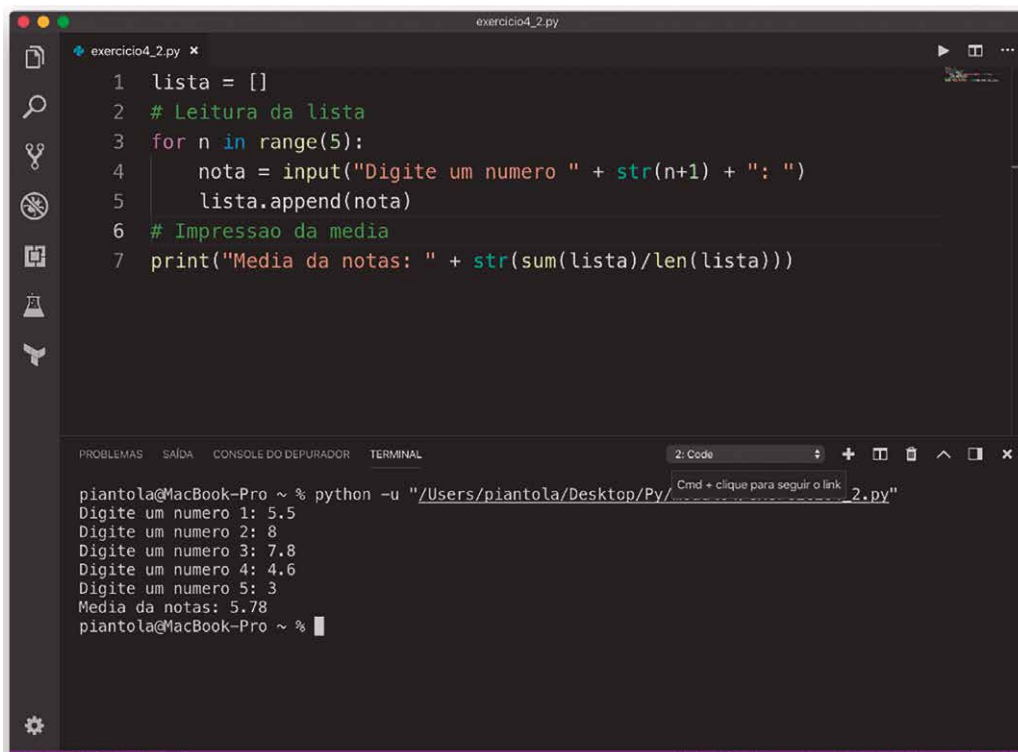
8.3.1 Respostas dos exercícios



```
exercicio4_1.py
1 lista = []
2 # Leitura da lista
3 for n in range(4):
4     numero = input("Digite um numero " + str(n+1) + ": ")
5     lista.append(numero)
6 # Impressao da lista
7 for n in lista:
8     print(n)

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL
2: Code
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo4/exercicio4_1.py"
Digite um numero 1: 2
Digite um numero 2: 4
Digite um numero 3: 5
Digite um numero 4: 6
2
4
5
6
piantola@MacBook-Pro ~ %
```

Figura 64 – Resolução do exercício 1



```
exercicio4_2.py
1 lista = []
2 # Leitura da lista
3 for n in range(5):
4     nota = input("Digite um numero " + str(n+1) + ": ")
5     lista.append(nota)
6 # Impressao da media
7 print("Media da notas: " + str(sum(lista)/len(lista)))

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL
2: Code
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/2.py"
Digite um numero 1: 5.5
Digite um numero 2: 8
Digite um numero 3: 7.8
Digite um numero 4: 4.6
Digite um numero 5: 3
Media da notas: 5.78
piantola@MacBook-Pro ~ %
```

Figura 65 – Resolução do exercício 2


```
exercico4_3.py
1 lista = []
2 # Leitura da lista
3 for n in range(6):
4     numero = input("Digite um numero " + str(n+1) + ": ")
5     lista.append(numero)
6 # Impressao da reverso
7 lista.reverse()
8 print(lista)
```

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL 2: Code

```
piantola@MacBook-Pro ~ % python -u "/Users/piantola/De...ercico4_3.py"
Digite um numero 1: 5
Digite um numero 2: 4
Digite um numero 3: 3
Digite um numero 4: 2
Digite um numero 5: 1
Digite um numero 6: 0
[0, 1, 2, 3, 4, 5]
piantola@MacBook-Pro ~ %
```

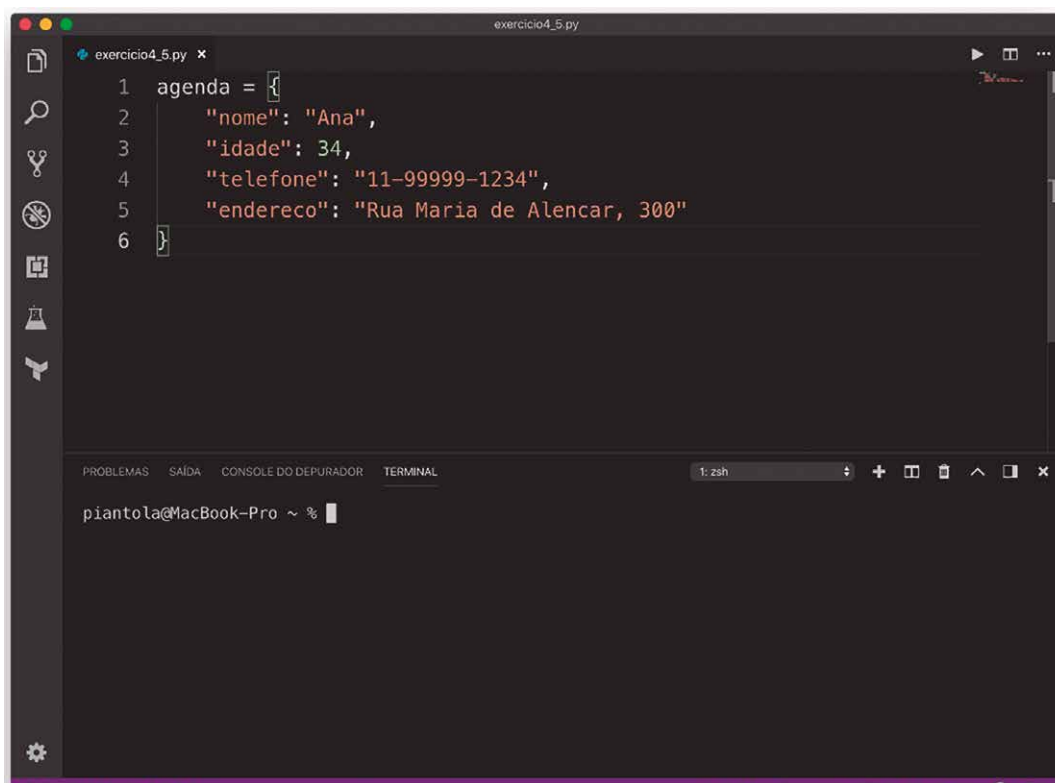
Figura 66 – Resolução do exercício 3

```
exercico4_4.py
1 lista = []
2 # Leitura da lista
3 for n in range(8):
4     numero = input("Digite um numero " + str(n+1) + ": ")
5     lista.append(numero)
6 # Calculo da soma dos quadrados
7 soma_quadrados = 0
8 for n in lista:
9     soma_quadrados += n**2
10 print(soma_quadrados)
```

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL 2: Code

```
piantola@MacBook-Pro ~ % python -u "/User...Py/modulo4/exercico4_4.py"
Digite um numero 1: 2
Digite um numero 2: 2
Digite um numero 3: 2
Digite um numero 4: 2
Digite um numero 5: 2
Digite um numero 6: 2
Digite um numero 7: 2
Digite um numero 8: 2
32
piantola@MacBook-Pro ~ %
```

Figura 67 – Resolução do exercício 4

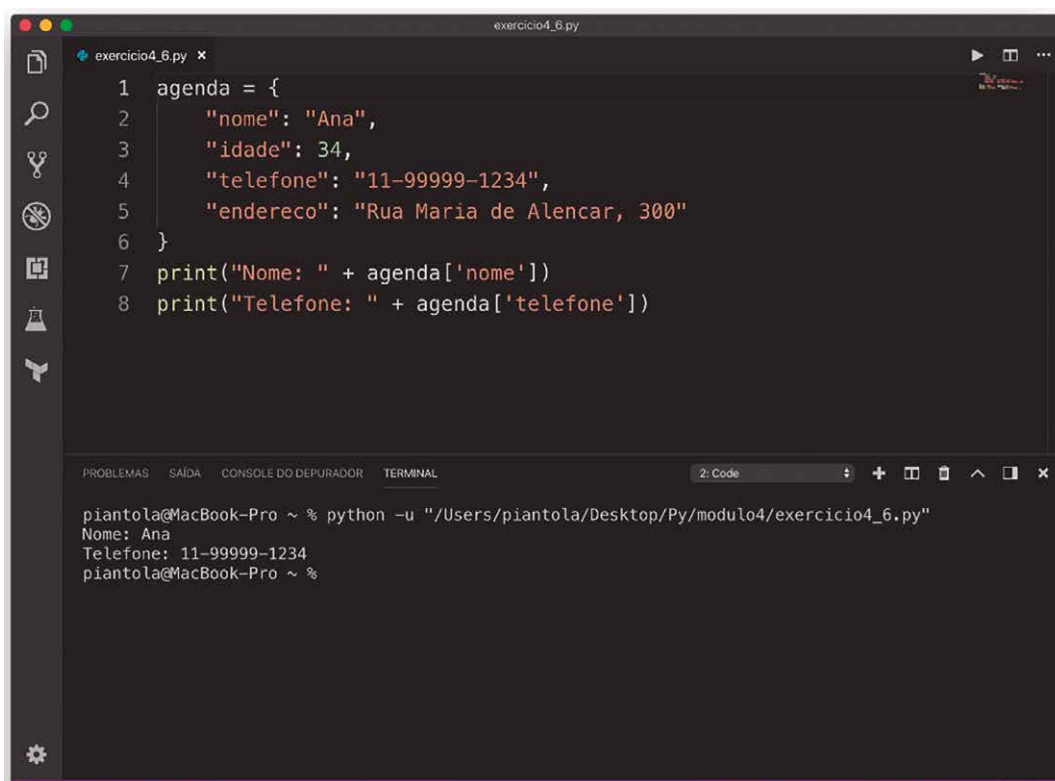


```
exercicio4_5.py
1 agenda = {}
2     "nome": "Ana",
3     "idade": 34,
4     "telefone": "11-99999-1234",
5     "endereço": "Rua Maria de Alencar, 300"
6 }
```

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL 1: zsh

piantola@MacBook-Pro ~ %

Figura 68 – Resolução do exercício 5



```
exercicio4_6.py
1 agenda = {
2     "nome": "Ana",
3     "idade": 34,
4     "telefone": "11-99999-1234",
5     "endereço": "Rua Maria de Alencar, 300"
6 }
7 print("Nome: " + agenda['nome'])
8 print("Telefone: " + agenda['telefone'])
```

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL 2: Code

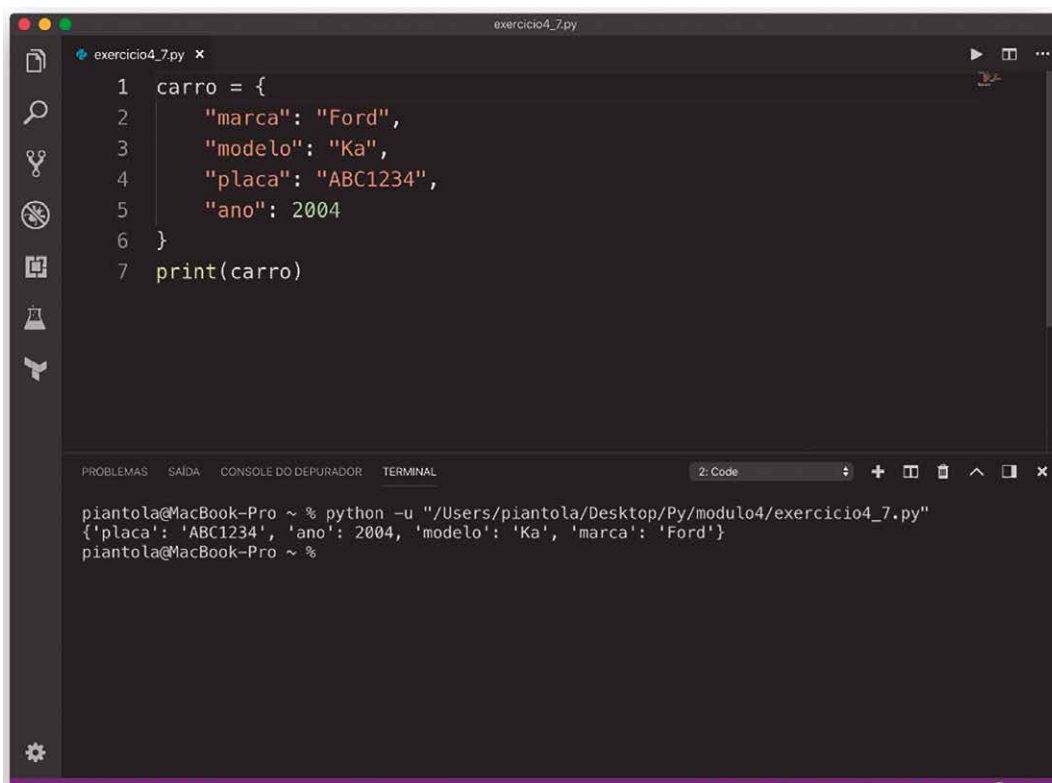
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo4/exercicio4_6.py"

Nome: Ana

Telefone: 11-99999-1234

piantola@MacBook-Pro ~ %

Figura 69 – Resolução do exercício 6

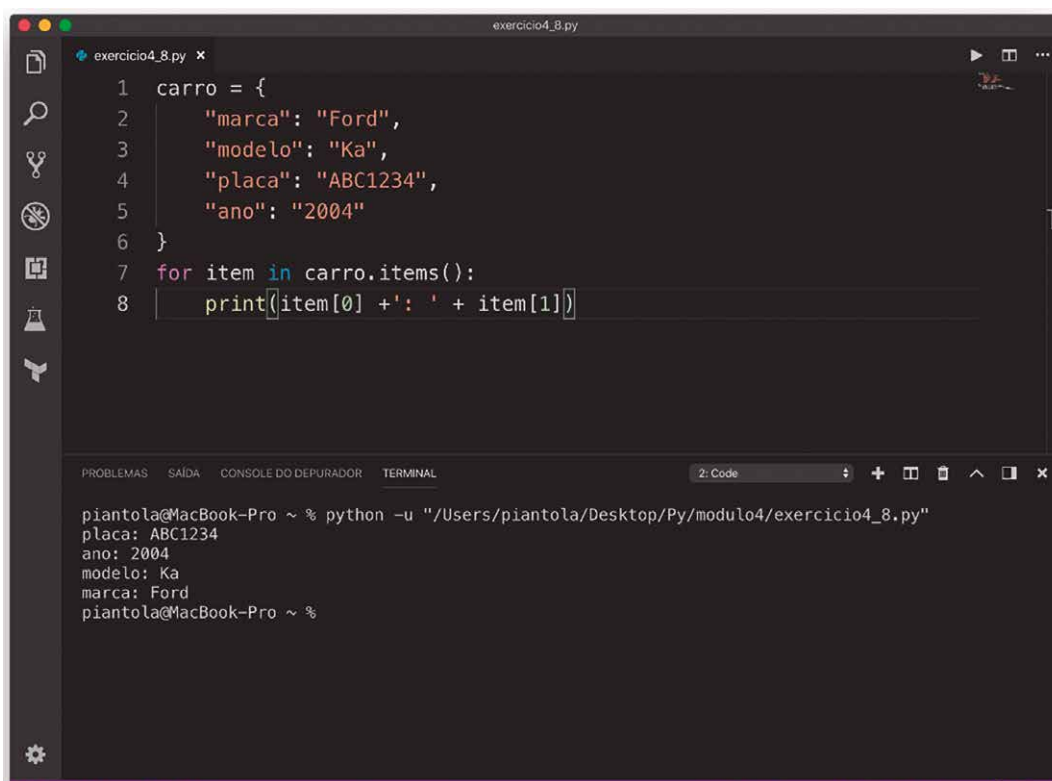


```
exercico4_7.py
1 carro = {
2     "marca": "Ford",
3     "modelo": "Ka",
4     "placa": "ABC1234",
5     "ano": 2004
6 }
7 print(carro)
```

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL 2: Code

```
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo4/exercico4_7.py"
{'placa': 'ABC1234', 'ano': 2004, 'modelo': 'Ka', 'marca': 'Ford'}
piantola@MacBook-Pro ~ %
```

Figura 70 – Resolução do exercício 7

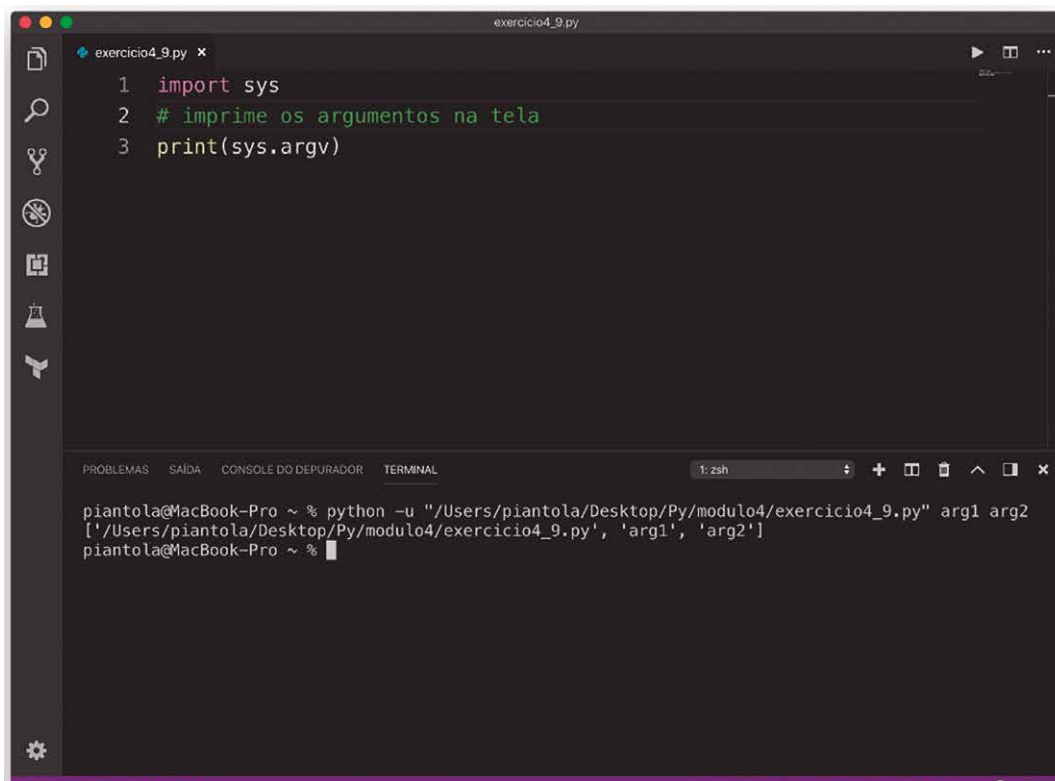


```
exercico4_8.py
1 carro = {
2     "marca": "Ford",
3     "modelo": "Ka",
4     "placa": "ABC1234",
5     "ano": "2004"
6 }
7 for item in carro.items():
8     print(item[0] + ': ' + item[1])
```

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL 2: Code

```
piantola@MacBook-Pro ~ % python -u "/Users/piantola/Desktop/Py/modulo4/exercico4_8.py"
placa: ABC1234
ano: 2004
modelo: Ka
marca: Ford
piantola@MacBook-Pro ~ %
```

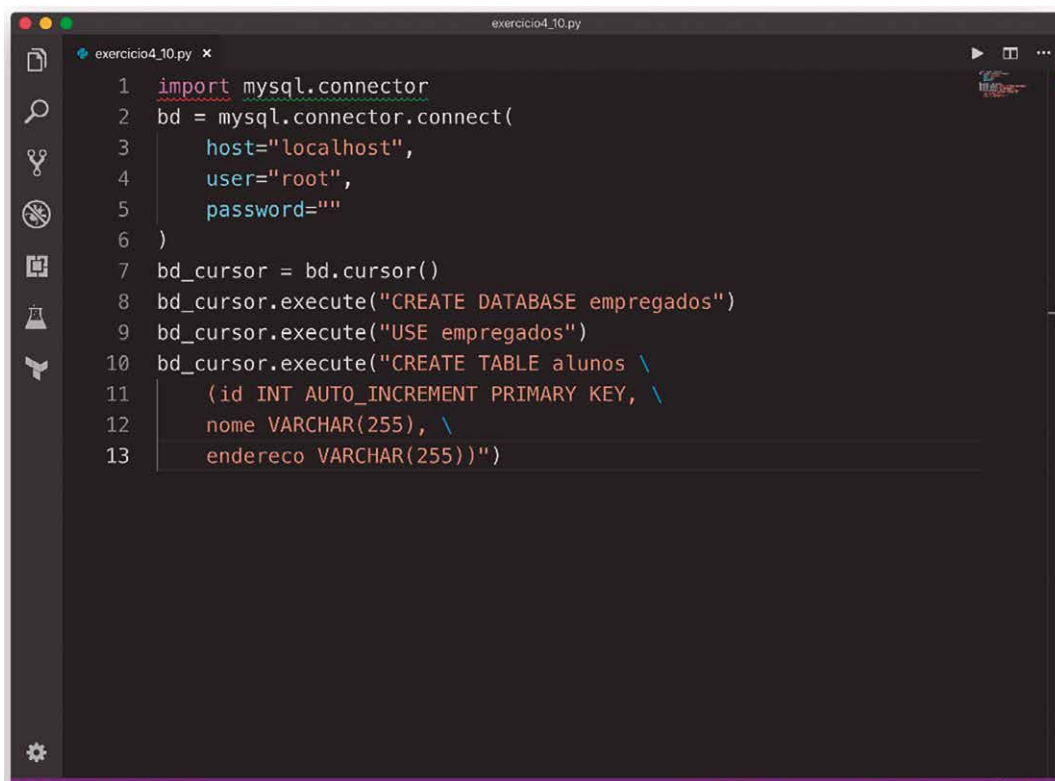
Figura 71 – Resolução do exercício 8



```
exercicio4_9.py x
1 import sys
2 # imprime os argumentos na tela
3 print(sys.argv)

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL 1: zsh
pianola@MacBook-Pro ~ % python -u "/Users/pianola/Desktop/Py/modulo4/exercicio4_9.py" arg1 arg2
['/Users/pianola/Desktop/Py/modulo4/exercicio4_9.py', 'arg1', 'arg2']
pianola@MacBook-Pro ~ %
```

Figura 72 – Resolução do exercício 9



```
exercicio4_10.py x
1 import mysql.connector
2 bd = mysql.connector.connect(
3     host="localhost",
4     user="root",
5     password=""
6 )
7 bd_cursor = bd.cursor()
8 bd_cursor.execute("CREATE DATABASE empregados")
9 bd_cursor.execute("USE empregados")
10 bd_cursor.execute("CREATE TABLE alunos \
11     (id INT AUTO_INCREMENT PRIMARY KEY, \
12     nome VARCHAR(255), \
13     endereco VARCHAR(255))")
```

Figura 73 – Resolução do exercício 10



Resumo

Na unidade IV, vimos que as listas e dicionários são estruturas cada vez mais utilizadas para resolver problemas em computação, com uma implementação pronta de listas em Python, para facilitar o trabalho do programador e reduzir a quantidade necessária de código. São mais de dez métodos; os principais para a manipulação são: (a) `append()`, que adiciona um elemento no final da lista; (b) `pop()`, que remove o elemento do final da fila; (c) `remove()`, que remove o elemento da posição informada; e (d), que insere o elemento na posição informada.

Os dicionários são estruturas de armazenamento de dados por chave e valor. A chave é um índice que não pode ter valor duplicado e o valor é o conteúdo apontado pela chave. É uma estrutura ordenada e mutável, que não permite itens duplicados. Em outras linguagens e na computação, também são conhecidos como estrutura de hash. Como no caso das listas, o Python tem essa estrutura pronta para uso com vários métodos associados para sua manipulação; os principais são: (a) `keys()`, que retorna uma lista contendo as chaves dos dicionários; (b) `values()`, que retorna uma lista contendo os valores; e (c) `items()`, que retorna uma lista contendo tuplas de chaves e valores.

Os módulos são como uma biblioteca de códigos prontos em um arquivo, contendo várias funções que podemos utilizar em nossa aplicação. Eles podem ser feitos pelo próprio programador ou por uma comunidade e baixados para uso. Na programação estrutura, o conceito de modularização tem dois objetivos: o primeiro e principal é a melhor organização do código-fonte. A segunda é sobre o reuso do código em outros programas (objetivo que foi mais difundido na orientação a objeto).

Os sistemas de gerenciamento de banco de dados (SGBD) são sistemas que gerenciam vários bancos de dados, e bancos de dados são coleções de tabelas, índices e procedimentos sobre um assunto. A linguagem utilizada para manipular os dados dentro das tabelas dos bancos de dados é a linguagem SQL. É possível conectar através do Python em SGBDs e de forma programática utilizar banco de dados para armazenar, processar e deletar dados. O MySQL é um SGBD bem popular hoje em dia, por possuir uma versão gratuita e por ser robusto.



Exercícios

Questão 1. Leia o texto a seguir, a respeito de dicionários em Python:

As estruturas de dados **lista** e **tupla** permitem acessar seus elementos a partir de um índice numérico inteiro que se inicia em 0. Porém, Python também provê uma estrutura de dados chamada **dicionário**, que permite que outros tipos de valores, além de inteiros, sejam usados como índices. Dicionários são representados entre chaves. Assim, um dicionário vazio é representado por `{}`. Em uma lista, o primeiro elemento adicionado à lista vazia vai sempre para a posição 0. Mas, em um dicionário, vamos sempre associar explicitamente um índice para cada valor. Esse índice é usualmente denominado **chave**.

Podemos utilizar o comando "for" para fazer iteração sobre todos os valores de um dicionário. Porém, como cada posição do dicionário contém um par chave/valor, podemos aplicar o comando "for" de diferentes maneiras, como demonstrado a seguir.

Código 1:

```
dic = {'Pedro': 1, 'Maria': 4, 'João': 9, 'Rui': 6}
for elemento in dic:
    print(elemento)
```

Código 2:

```
dic = {'Pedro': 1, 'Maria': 4, 'João': 9, 'Rui': 6}
for elemento in dic:
    print(dic[elemento])
```

Adaptado de: WAZLAWICK, R. S. *Introdução aos algoritmos e programação com Python: uma abordagem dirigida por testes*. Rio de Janeiro: Elsevier, 2018. p.131-134.

A respeito dos códigos apresentados, avalie as afirmativas.

I – Ao executar o código 1, apenas as chaves dos itens do dicionário são impressas. Dessa forma, visualizamos apenas a lista de nomes.

II – Ao executar o código 2, apenas os valores dos itens do dicionário são impressos. Dessa forma, visualizamos apenas as quantidades associadas a cada item.

III – Ambos os códigos, ao serem executados, resultam em erro devido a problemas de sintaxe.

É correto o que se afirma em:

A) I, apenas.

B) II, apenas.

C) III, apenas.

D) I e II, apenas.

E) I, II e III.

Resposta correta: alternativa D.

Análise das afirmativas

I – Afirmativa correta.

Justificativa: a variável elemento recebe, a cada execução do laço for, a chave de um item do dicionário dic. Portanto, ao imprimir essa variável, visualizamos apenas a chave associada a cada item. Assim, a execução do código resulta na impressão das strings:

```
Pedro
Maria
João
Rui
```

II – Afirmativa correta.

Justificativa: conforme mencionado, a variável elemento recebe, a cada execução do laço for, a chave de um item do dicionário dic. Ao imprimir dic[elemento], estamos buscando no dicionário os valores correspondentes a cada chave atribuída à variável elemento. Dessa forma, visualizamos apenas o valor de cada item. A execução do código resulta na impressão dos inteiros:

```
1
4
9
6
```

III – Afirmativa incorreta.

Justificativa: conforme já averiguado nas justificativas das afirmativas anteriores, não há qualquer problema de sintaxe nos códigos.

Questão 2. Leia o texto a seguir, a respeito de bancos de dados:

Os dados processados por um programa Python existem somente enquanto o programa é executado. Para que eles persistam além da execução do programa – para que possam ser processados mais tarde por algum outro programa, por exemplo –, os dados precisam ser armazenados em um arquivo.

Pode-se utilizar arquivos de texto padrão para armazenar dados de forma persistente. A vantagem dos arquivos de texto é que eles são de uso geral e fáceis de se trabalhar. Sua desvantagem é que eles não têm uma estrutura que permita que os dados sejam acessados e processados eficientemente.

Existe um tipo de arquivo especial, denominado **arquivo de bancos de dados**, ou simplesmente **bancos de dados**, que é capaz de armazenar informações de forma estruturada. A estrutura torna o conteúdo de um arquivo de bancos de dados receptivo ao processamento eficiente, incluindo a inserção, atualização, exclusão e, especialmente, acesso.

O termo "forma estruturada", neste contexto, significa que as informações contidas em um arquivo de banco de dados são armazenadas em uma ou mais tabelas. Cada tabela é identificada por um nome, como Clientes ou Produtos, e consiste em colunas e linhas. Cada coluna tem um nome e contém dados de um tipo específico: string, inteiro, real (float) e assim por diante. Cada linha da tabela contém dados correspondentes a um registro do banco de dados. Com dados armazenados em tabelas de banco de dados, podemos fazer consultas de informações usando uma linguagem de programação de banco de dados especial.

Arquivos de banco de dados não são lidos ou gravados por um programa de aplicação usando a interface comum de entrada/saída de arquivo. Eles normalmente também não são acessados diretamente. Em vez disso, o programa de aplicação normalmente envia comandos a um tipo especial de programa servidor, chamado **mecanismo de banco de dados** ou **sistema de gerenciamento de banco de dados**, que atua como um administrador do banco de dados. Esse programa acessará o arquivo de banco de dados em favor da aplicação. Os comandos aceitos pelos mecanismos de banco de dados são instruções escritas em uma **linguagem de consulta**.

Adaptado de: PERKOVIC, L. *Introdução à computação usando Python*: um foco no desenvolvimento de aplicações. Rio de Janeiro: LTC, 2016. p.433-435.

A respeito desse contexto, avalie as afirmativas.

I – A linguagem de consulta de banco de dados mais popular é denominada **mySQL**.

II – Em SQL, o comando **SELECT** é utilizado para fazer consultas em um banco de dados.

III – Ao programar em Python, a melhor forma de interagir com dados armazenados de forma persistente é sempre utilizar arquivos de texto padrão.

É correto o que se afirma em:

- A) I, apenas.
- B) II, apenas.
- C) III, apenas.
- D) I e II, apenas.
- E) I, II e III.

Resposta correta: alternativa B.

Análise das afirmativas.

I – Afirmativa incorreta.

Justificativa: os comandos aceitos pelos mecanismos de banco de dados são instruções escritas em uma linguagem de consulta, sendo que a mais popular é denominada Structured Query Language, ou apenas SQL. O MySQL, por sua vez, é um Sistema de Gerenciamento de Banco de Dados (SGBD), ou seja, é um próprio mecanismo de banco de dados, que utiliza a linguagem SQL. SGBDs são executados como programas independentes, fora do Python. Para acessá-los, você precisa usar uma interface de programação de aplicação (API) que ofereça classes e funções que permitam a aplicações Python enviar comandos SQL ao mecanismo de banco. O módulo sqlite3, da Biblioteca Padrão Python, oferece uma API que permite aos programas Python acessarem arquivos de banco de dados e executarem comandos SQL.

II – Afirmativa correta.

Justificativa: em SQL, o comando SELECT realiza buscas em um banco de dados. Em sua forma mais simples, esse comando é usado para recuperar uma coluna de uma tabela do banco de dados. Por exemplo, para recuperar a coluna Nome da tabela Cadastro, pode-se utilizar:

```
SELECT Nome FROM Cadastro
```

O resultado da execução desse comando é armazenado em uma tabela de resultados.

III – Afirmativa incorreta.

Justificativa: a utilização de arquivos de texto padrão (como os de extensão txt) é uma das formas de se trabalhar com dados armazenados de forma persistente. No entanto, um banco de dados é uma técnica de armazenamento muito mais apropriada do que um arquivo de texto padrão em muitas aplicações. Para dados empresariais, por exemplo, um banco de dados oferece mais segurança e praticidade no acesso e na manipulação das informações.

REFERÊNCIAS

Textuais

BARRY, P. *Use a Cabeça! Python*. 2. ed. Rio de Janeiro: Alta Books, 2018.

DOCUMENTAÇÃO PYTHON 3.9.4. *Partes da documentação*. [s.d.]. Disponível em: <https://cutt.ly/zmhvrl0>. Acesso em 20 fev. 2021.

GITHUB. *Introdução à programação estruturada*. [s.d.]. Disponível em: <https://cutt.ly/WmwhVvJ>. Acesso em: 25 jun. 2021.

GITHUB. *NumPy*. [s.d.]. Disponível em: <https://cutt.ly/GmdK3Cy>. Acesso em: 25 jun. 2021.

GOOGLE. *O que é o Colaboratory?* [s.d.]. Disponível em: <https://cutt.ly/An6gEMF>. Acesso em: 24 jun. 2021.

KINSLEY, H.; MCGUGAN, W. *Introdução ao desenvolvimento de jogos em Python com PyGame*. São Paulo: Novatec, 2015.

LOPES, A.; GARCIA, G. *Introdução a programação: 500 algoritmos*. Rio de Janeiro: Campus, 2002.

MENEZES, N.N.C. *Introdução à programação com Python: algoritmos e lógica de programação para iniciantes*. São Paulo: Novatec, 2019.

MICROSOFT. *Versões de 32 bits e 64 bits do Windows: perguntas frequentes*. [s.d.]. Disponível em: <https://cutt.ly/yn6hisx>. Acesso em: 24 jun. 2021.

MYSQL. *Chapter 6 MySQL Connector/Python Developer Guide*. [s.d.]. Disponível em: <https://cutt.ly/SmhosuF>. Acesso em: 22 jun. 2021.

PYTHON. *Python Module Index*. [s.d.]. Disponível em: <https://cutt.ly/nmg6iZf>. Acesso em: 12 fev. 2021.

PYTHON BRASIL. *Documentação e exercícios*. Disponível em: <https://cutt.ly/XmhvEQQ>. Acesso em: 15 jul. 2021.

PYTHON BRASIL. *História do Python*. [s.d.]. Disponível em: <https://cutt.ly/An6gum0>. Acesso em: 24 jun. 2021.

RAMALHO, L. *Python fluente: programação clara, concisa e eficaz*. São Paulo: Novatec, 2015.

ZELLE, J. M. *Python programming: an introduction to computer science*. 3. ed. Nova York: Franklin, Beedle & Associates Inc, 2016.



Handwriting practice lines consisting of 30 horizontal lines. Each line is preceded by a small blue dot on the left margin, serving as a starting point for letter formation. The lines are evenly spaced and extend across the width of the page.



A series of horizontal lines for writing, consisting of 30 evenly spaced lines across the page.



Interativa

Informações:
www.sepi.unip.br ou 0800 010 9000