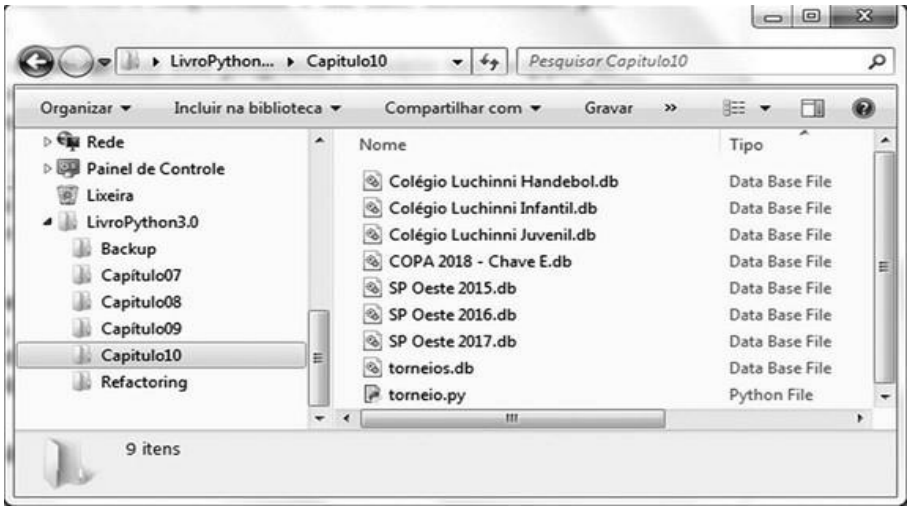
Armazenamento dos dados do programa

Esse programa manipula certa quantidade de dados que precisa ser armazenada permanentemente. Foi escolhido o SQLite para essa finalidade. Esse armazenamento pode ser pensado e estruturado de diversas maneiras distintas, e a maneira escolhida visa à simplicidade e a um fácil entendimento por parte do leitor iniciante.

O primeiro aspecto é que ao usar esse programa o usuário tem a opção de criar e gerenciar vários torneios ao mesmo tempo. Assim, os nomes dos torneios devem ser armazenados. Para isso, é mantido um banco de dados chamado “torneios.db”, dentro do qual há uma tabela que contém dois campos: o nome do torneio e a quantidade de turnos do mesmo. Quando o programa é executado pela primeira vez em um computador esse arquivo é criado. Daí para a frente, ele sempre é lido logo no início e utilizado para montar o menu de torneios da tela inicial.

Além disso, para cada torneio cadastrado é criado um arquivo de banco de dados do SQLite com o nome “<nometorneio>.db”, em que o identificador

<nometorneio> é substituído pelo nome escolhido pelo usuário. Todos os arquivos de banco de dados são criados e mantidos na mesma pasta na qual se encontra o programa. Na Figura 10.7 podem ser vistos todos os arquivos “.db” mencionados, em conjunto com o arquivo do programa “torneio.py”.

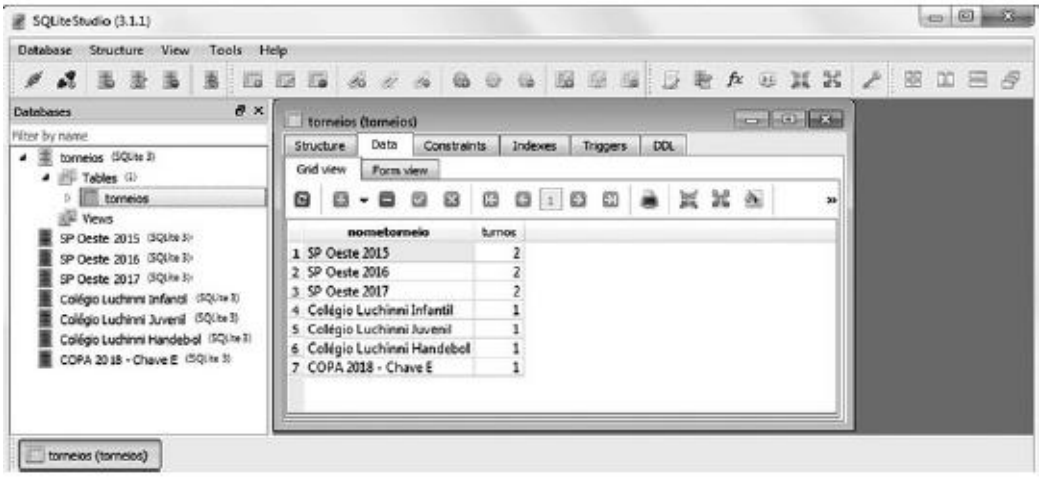


Armazenamento de dados do programa torneio.py.

O banco de dados “torneios.db” contém apenas a tabela “Torneios”, que, por sua vez, tem dois campos: um para o nome do torneio e outro para a quantidade de turnos, conforme definido

no Quadro 10.3 e ilustrado na Figura 10.8.

Quadro 10.3 Campos da tabela “Torneios”.



Banco de dados torneios.db visualizado por meio do SQLite Studio.

Por sua vez, os bancos de dados de torneios têm duas tabelas: a tabela “Times”, para armazenar os nomes dos times participantes, e a tabela “Jogos”, para armazenar as rodadas e os resultados dos jogos.

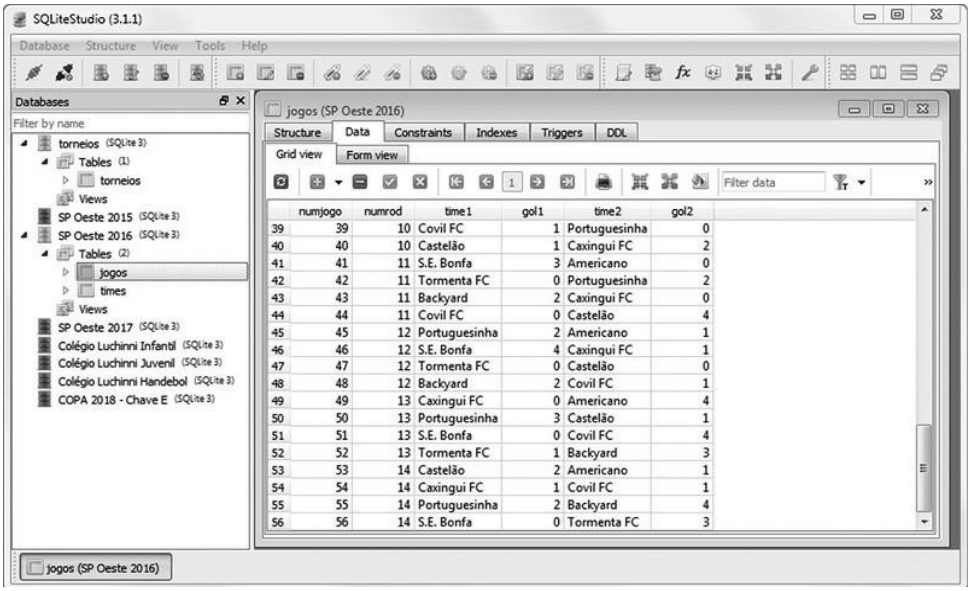
Campo	Tipo	Observação
nometime	Texto	Contém o nome do time.

Quadro 10.4 Campos da tabela “Times”

Campo	Tipo	Observações
numjogo	Número inteiro	Número do jogo.
numrod	Número inteiro	Número da rodada.
time1	Texto	Nome do primeiro time.
gol1	Número inteiro	Gols marcados pelo time1.
time2	Texto	Nome do segundo time.
gol2	Número inteiro	Gols marcados pelo time2.

Quadro 10.5 Campos da tabela “Jogos”.

A Figura 10.9 ilustra, por meio do SQLite Studio, a organização do banco do torneio, exibindo dados armazenados na tabela “Jogos”.



Banco de dados “SP Oeste 2016” visualizado por meio do SQLite Studio.

Deste ponto em diante, tem início a descrição do programa. Tudo o que será apresentado a partir daqui diz respeito ao código-fonte do programa e é necessário que esteja presente no arquivo “torneio.py” para que este funcione corretamente. O programa está organizado em 33 funções que devem estar reunidas em um único código-fonte. Tais funções serão apresentadas agrupadas segundo a finalidade a que se destinam dentro do programa e podem ser divididas em três grupos.

	Grupo	Descrição
1	Funções gerais	Pequeno número de funções que não se enquadram em nenhum dos outros dois próximos grupos.
2	Criação de torneio	Funções necessárias às funcionalidades de criação de um novo torneio.
3	Gerenciamento de torneio	Funções necessárias às funcionalidades relativas ao gerenciamento de um torneio.

Funções necessárias às funcionalidades relativas ao gerenciamento de um torneio.

Objetos globais e início do programa

Nessa implementação foram utilizados quatro objetos de escopo global para conter informações que, dada sua relevância, são necessárias em muitas funções do programa. Assim, em vez de passá-los como parâmetro a cada chamada de função, optou-se pelo escopo global, que faz que sejam visíveis e possam ser alteradas em todas as funções do programa.

Três desses objetos são empregados para gerenciamento de torneio em andamento, são criados dentro da função GerenciaTorneio e eliminados ao seu final, por não serem necessários em outras partes do programa. O quarto objeto global é utilizado na composição das telas.

Identificador	Tipo	Observações
Torneio	String	Contém o nome do torneio escolhido no menu principal para ser gerenciado.
Turnos	Número inteiro	Contém a quantidade de turnos do torneio escolhido no menu principal.
Times	Lista de listas	Contém os dados necessários para exibição da tabela de classificação do torneio. Cada sublista contida em “Times” é formada por um string, que é o nome do time e oito números inteiros que são: quantidade de pontos, jogos, vitórias, empates, derrotas, gols pró, gols contra e saldo de gols. Exemplo de uma dessas sublistas: [“Americano”, 5, 4, 1, 2, 1, 4, 4, 0]
LargTela	Número inteiro	Usado para definir a largura das saídas formatadas que compõem o visual das telas do programa.

Quadro 10.6 Objetos globais usados no programa.

No início do código-fonte deve haver a importação dos módulos indicados no Exemplo 10.1, pois algumas de suas funções são utilizadas no programa, bem como é definido o objeto LargTela.

Exemplo 10.1 Programa torneio.py – módulos importados from datetime import date import os import sqlite3 LargTela = 70

O ponto de partida do programa está no Exemplo 10.2, que contém a chamada à função MenuPrincipal e, ao término desta, faz o fechamento do programa exibindo mensagem de encerramento.

Exemplo 10.2 Programa torneio.py – ponto de partida do programa

```
# Ponto de início da execução MenuPrincipal()  print("\n"*2)
ExibeLinha("Programa Torneio encerrado", 30) print("\n")
Pausa("Pressione Enter para sair.", 30)
```

Grupo de funções gerais

A função MenuPrincipal é a primeira dessas funções gerais. Seu propósito é permanecer em laço até que o usuário decida sair do programa. Nesse laço é feita a exibição da tela inicial mostrada na Figura 10.1 e é lida e processada a opção do usuário. Para exibir a lista de torneios cadastrados no programa é carregado o objeto local "Torneios" (não confundir com o objeto global "Torneio" mencionado no item anterior) com o uso da função PreparaAmbiente. Caso haja torneios cadastrados, o comando for é usado para iterar sobre o objeto e montar a lista de torneios.

Quanto às opções disponíveis ao usuário, "N" leva à chamada da função CriaNovoTorneio descrita no Item 10.2.5 e o número do torneio mostrado na tela leva à chamada da função GerenciaTorneio descrita no Item 10.2.6. "S" encerra o laço dessa função e leva ao término do programa.

Exemplo 10.3 Função MenuPrincipal

```
def MenuPrincipal():
while True:
Torneios = PreparaAmbiente() TopoTela("Menu Principal") print("Opções: ")
print(" (N) Criar Novo Torneio")  print(" (.) Gerenciar Torneio Existente") if len(Torneios) ==
0:
print(" { não há torneios cadastrados }") else:
for i, t in Torneios.items():

print(" ({}) {}".format(i, t["nome"])) print(" (S) Sair do programa")
print("\npara escolher digite o que está entre parênteses") opc = input("sua opção? >>> ")
```

```
opc = opc.upper() if opc == “N”: CriaNovoTorneio()
elif opc.isnumeric():
n = int(opc)

if n in Torneios: GerenciaTorneio(Torneios[n]) elif opc == “S”:
break
```

A função PreparaAmbiente verifica se o banco de dados “torneios.db” existe. Em caso afirmativo, lê os torneios cadastrados, carrega e retorna um dicionário. Caso contrário, cria o banco de dados e a tabela de torneios dentro deste e retorna um dicionário vazio. Essa criação ocorrerá apenas na primeira vez em que o programa for executado em um computador.

Exemplo 10.4 Função PreparaAmbiente def PreparaAmbiente():

```
“”” Se o B.D. torneios.db existe, então, lê os torneios. Caso contrário, cria o B.D. Necessário no primeiro uso do programa,”””
conector = sqlite3.connect(“torneios.db”) cursor = conector.cursor()
sql = “””

select name from sqlite_master
where type=’table’ and name = ‘torneios’ “””
cursor.execute(sql) R = {}
N = 1

if cursor.fetchone() == None:

sql = “create table torneios (nometorneio text, turnos int)” cursor.execute(sql)
else:

sql = “select * from torneios” cursor.execute(sql)
for x in sorted(cursor.fetchall()): item = {}
item[“nome”] = x[0]
```

item[“turnos”] = x[1] R[N] = item

N+=1

cursor.close() conector.close() return R

O Exemplo 10.5 exibe três funções criadas para auxiliar na exibição das telas e para pausar o programa ao exibir alguma mensagem para o usuário. São funções auxiliares que ajudam a organizar o código eliminando redundância desnecessária.

Exemplo 10.5 Funções auxiliares de exibição em tela def ExibeLinha(msg, tam = 0, alinha = “^”):

borda = (LargTela - tam - 2) // 2 sfmt = “{:” + alinha + str(tam) + “}”

print(“-”*borda, sfmt.format(msg), “-”*borda) def Pausa(msg, tam=64):

if msg != “”: ExibeLinha(msg, tam) input()

def TopoTela(msg = “”):

print(“\n”*2, “-” * LargTela, sep = “”) ExibeLinha(“Programa Torneio”, 40, “^”)

if msg != “”:

ExibeLinha(msg, 40, “^”) print(“-” * LargTela)

Grupo de funções de criação de torneio

Quando o usuário deseja criar um novo torneio, ocorre a chamada da função CriaNovoTorneio, mostrada no Exemplo 10.6. Essa é uma função concentradora, ou seja, ela existe para organizar a sequência lógica das tarefas que serão efetivamente realizadas por outras funções. Em linhas gerais, por meio dela é feita a leitura do nome do novo torneio e de sua quantidade de turnos, dos times participantes, e é gerado o banco de dados do torneio.

Exemplo 10.6 Função CriaNovoTorneio def CriaNovoTorneio():

NomeTorneio = input(“\nNome do Novo Torneio: “) if not ValidaTorneio(NomeTorneio):

return None

QtdeTurnos = ObtemQtdeTurnos() if QtdeTurnos == 0:

return None

TopoTela(“Criação de Novo Torneio”)

ExibeLinha(“Novo Torneio: “ + NomeTorneio, 64) ExibeLinha(“(a qualquer momento digite ‘sair’ para desistir)”, 64)

print(“-” * LargTela)

ExibeLinha(“Digite os nomes dos times participantes”, 70)

ExibeLinha(“Digite ‘fim’ para concluir e salvar os nomes dos times”, 70)

ListaTimes = ObtemNomesTimes() if ListaTimes:

CriaBDTorneio(NomeTorneio, ListaTimes) GravaNomeTorneio(NomeTorneio, QtdeTurnos) GeraGravaJogos(NomeTorneio, QtdeTurnos, ListaTimes)

A seguir são apresentadas as funções chamadas por CriaNovoTorneio, na ordem em que ocorrem.

A função ValidaTorneio (Exemplo 10.7) recebe o nome digitado e retorna *False* caso este seja nulo ou já esteja cadastrado. Neste último caso, faz uma pausa para mostrar uma mensagem. Caso essas duas situações não ocorram, então, retorna *True*.

A função ObtemQtdeTurnos (Exemplo 10.7) foi criada para ler e tratar a quantidade de turnos. É possível que o usuário digite qualquer coisa, inclusive texto, porém, o que se espera é que digite apenas os algarismos 1 ou 2, então, foram implementados um tratamento de exceção e um laço que se repetirá até que um valor apropriado seja fornecido. Como o usuário pode desistir de cadastrar o novo torneio, também é permitida a digitação do algarismo 0. No retorno da chamada, se a quantidade de turnos for zero, sua criação é interrompida (ver Exemplo 10.6).

Exemplo 10.7 Funções ValidaTorneio e ObtemQtdeTurnos def ValidaTorneio(NomeTorneio):

if NomeTorneio == “”:

return False

elif ExisteTorneio(NomeTorneio): Pausa(NomeTorneio + “ já existe (pressione Enter)”) return False

else:

return True

def ObtemQtdeTurnos():

while True:

print(“Digite a quantidade de turnos (1 ou 2): “) print(“Digite 0 para cancelar.”)
Qtde = input(“quantos turnos? >>> “) try:
Qtde = int(Qtde) except:
print(“Entrada inválida {}”.format(Qtde)) print(“Digite qtde 1 ou 2. Digite 0 para desistir.”) else:
if 0 <= Qtde <= 2:

return Qtde

A próxima função, ObtemNomesTimes, permanece em laço fazendo a leitura dos nomes dos times participantes. Esse laço termina com a digitação de uma destas palavras: “sair” e “fim”. No caso de “fim”, o laço termina e a função retorna uma lista com os nomes digitados. No caso de “sair”, a função retorna *None*, indicando que o usuário desistiu do processo.

Exemplo 10.8 Função ObtemNomesTimes def ObtemNomesTimes():
L = []

Cont = 1 while True:
s = input(“Time {:d}: “.format(Cont)) if s.upper() == “SAIR”: return None if s.upper() == “FIM”: break L.append(s)
Cont += 1 return L

Por fim, nas últimas linhas da função CriaNovoTorneio é processada a criação do torneio. A função CriaBDTorneio gera a estrutura do banco de dados necessária ao armazenamento dos dados, criando o BD fisicamente, e nele, as tabelas “Times” e “Jogos”. A tabela “Times” recebe todos os nomes digitados pelo usuário contidos no parâmetro “ListaTimes”.

A função GravaNomeTorneio insere o nome do novo torneio na tabela do banco de dados

“torneios.db”, e a partir daí esse novo torneio aparecerá no menu da tela principal.

Exemplo 10.9 Funções CriaBDTorneio e GravaNomeTorneio def CriaBDTorneio(NomeTorneio, ListaTimes):

Cria o BD do torneio

conector = sqlite3.connect(NomeTorneio + “.db”) cursor = conector.cursor()

Cria a tabela times

sql = “create table times (nometime text)” cursor.execute(sql)

Insere os times na tabela

sql = “insert into times (nometime) values (?)” for nome in ListaTimes:

cursor.execute(sql, (nome,)) conector.commit()

Cria a tabela de Jogos sql = “””

create table jogos (

numjogo int NOT NULL PRIMARY KEY ASC, numrod int,

time1 text, gol1 int, time2 text, gol2 int)””” cursor.execute(sql) cursor.close() conector.close()

def GravaNomeTorneio(NomeTorneio, QtdeTurnos): # Insere o nome do torneio no BD de torneios conector = sqlite3.connect(“torneios.db”)

cursor = conector.cursor()

sql = “insert into torneios (nometorneio, turnos) values (?, ?)” cursor.execute(sql, (NomeTorneio, QtdeTurnos)) conector.commit()

cursor.close() conector.close()

Algoritmo de criação dos jogos

Por fim, a função GeraGravaJogos é a responsável pela criação dos jogos e rodadas, fazendo o emparelhamento dos times.

Em termos de lógica e algoritmos, essa é a rotina mais interessante e a grande novidade desse projeto. A grande questão que se coloca neste ponto é: dados os nomes dos times, como escrever um algoritmo que produza todos os jogos de um turno sem que falte algum jogo ou haja

repetição? A resposta é utilizar um algoritmo conhecido com **Round-Robin Tournament**. Mas cuidado, não o confunda com o algoritmo Round-Robin sem a palavra “Tournament”. Este último é um algoritmo muito conhecido empregado em sistemas operacionais e não guarda qualquer relação com o Round-Robin Tournament, que será utilizado aqui.

Esse algoritmo é exemplificado na Figura 10.10. Ele consiste em fazer o emparelhamento dos times o primeiro com o último, o segundo com o penúltimo, e assim por diante, de modo que isso produz todos os jogos da primeira rodada. Caso o número de times seja ímpar, acrescenta-se um time fictício (no algoritmo chamado de FOLGA) e procede-se da mesma maneira.

Para produzir a segunda rodada, deve-se mover o time B para a última posição da lista, adiantando os times de C até H uma posição para a esquerda. Para a terceira rodada, transfere-se o time C para a última posição e adiantam-se os demais. Durante todo o processo, o time A não deve ser movido, e quando o time B chegar à sua posição original, todos os jogos de um turno estarão montados. Para dois turnos, é só usar o mesmo conjunto de jogos já gerados e, se for o caso, trocá-los de posição, fazendo que o mandante de campo apareça do lado esquerdo, e o visitante, do lado direito na tabela de jogos.

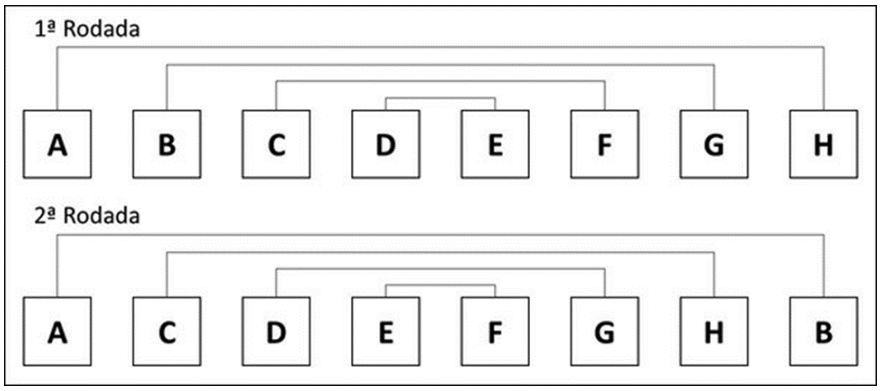


Ilustração do algoritmo Roun-Robin Tournament.

A função GeraGravaJogos recebe como parâmetros de entrada o nome do torneio, a quantidade de turnos e a lista de times. As partidas são geradas no dicionário **Jogos**, cuja chave é o número do jogo e o valor associado é um dicionário aninhado contendo o número da rodada, o time 1 e o time 2.

Após a geração do dicionário **Jogos**, todas as partidas são gravadas no banco de dados, para 1 ou 2 turnos, conforme o caso.

Exemplo 10.10 Função GeraGravaJogos

```
def GeraGravaJogos(NomeTorneio, QtdeTurnos, ListaTimes): # Gera os Jogos
Jogos = {} NJogo = 1
Qtde = len(ListaTimes) if Qtde % 2 == 1:
ListaTimes.append("FOLGA")

Qtde += 1

# Implementa o algoritmo Round-Robin Tournament for r in range(Qtde-1):
for i in range(Qtde//2):

if ListaTimes[i] == "FOLGA" or \

ListaTimes[Qtde-1-i] == "FOLGA": continue
umJogo = {} umJogo["rodada"] = r+1
umJogo["time1"] = ListaTimes[i] umJogo["time2"] = ListaTimes[Qtde-1-i] Jogos[NJogo] =
umJogo
NJogo += 1

aux = ListaTimes[1] del(ListaTimes[1]) ListaTimes.append(aux)
# Insere no banco de dados os jogos gerados conector = sqlite3.connect(NomeTorneio +
".db") cursor = conector.cursor()
sql = """

insert into jogos (numjogo, numrod, time1, time2) values (?, ?, ?, ?)
"""

for NJogo, Jogo in Jogos.items():

cursor.execute(sql, (NJogo, Jogo["rodada"], Jogo["time1"], Jogo["time2"]))
if QtdeTurnos == 2:
```

for NJogo, Jogo in Jogos.items():

cursor.execute(sql, (NJogo+(Qtde-1)*Qtde/2, Jogo[“rodada”]+Qtde-1,
Jogo[“time2”], Jogo[“time1”])) conector.commit() cursor.close()
conector.close()

Grupo de funções de gerenciamento de torneio

Neste grupo está concentrada a maior parte das funções do programa, e isso ocorre porque aqui se concentra a maior parte de suas funcionalidades. O Exemplo 10.11 exibe a função GerenciaTorneio, que é a responsável pelo desenho da tela mostrada na Figura 10.2 e concentra a chamada às demais funções de gerenciamento. Nessa função são criados os objetos globais mencionados no Item 10.2.3. Os nomes de times são carregados e é gerada a lista com os campos necessários à montagem da tabela de classificação, são exibidos os nomes de times e a classificação. Em seguida, é exibido o menu de opções e é lida e processada a opção do usuário.

Exemplo 10.11 Função GerenciaTorneio def GerenciaTorneio(t):
global Times, Torneio, Turnos Torneio = t[“nome”]
Turnos = t[“turnos”] Times = CarregaTimes()
Qtde, NRod, NJog = CalcPramsTorneio() while True:
TopoTela(“Gerenciamento de Torneio”) ExibeTimes()
ExibeClassificacao() print(“Opções: “)
print(“ (.) Para ver uma rodada digite seu número.”) print(“ Rodadas Válidas de 1 a {}”.format(NRod)) print(“ (G) Grava o Torneio em HTML”)
print(“ (E) Exclui o Torneio”)
print(“ (S) Voltar ao Menu Principal”) opc = input(“sua opção? >>> “)
opc = opc.upper() if opc == “N”: NovoTorneio()
elif opc.isnumeric(): n = int(opc)
if 1 <= n <= NRod: GerenciaRodada(n) elif opc == “G”: GravaHTML()
elif opc == “E”:

if ExcluiTorneio(Torneio): break

elif opc == “S”:

break

del(Times, Torneio, Turnos)

A função CarregaTimes é a responsável pela geração do objeto global Times a partir da leitura do banco de dados.

Exemplo 10.12 Função CarregaTimes def CarregaTimes():

global Torneio T = []

conector = sqlite3.connect(Torneio + “.db”) cursor = conector.cursor()

sql = “select nometime from times order by nometime” cursor.execute(sql)

for time in cursor.fetchall():

t = [time[0]] + [0]*8 T.append(t) cursor.close() conector.close() return T

A função ExibeTimes é responsável pela exibição dos nomes dos times participantes do torneio escolhido e também dos números de rodadas e jogos, sendo que estes são calculados em outra função, CalaParamsTorneio.

Exemplo 10.13 Funções CalcParamsTorneio e ExibeTimes def CalcPramsTorneio():

global Times, Turnos Qtde = len(Times)

if Qtde % 2 == 0:

NRod = (Qtde - 1) * Turnos else:

NRod = Qtde * Turnos

NJog = (Qtde - 1) * Qtde // 2 * Turnos return Qtde, NRod, NJog

def ExibeTimes():

global Times

Qtde, NRod, NJog = CalcPramsTorneio() ExibeLinha(“Times deste Torneio “ + Torneio, 64) cont = 1

s = “”

for t in Times:
s = s + “{:<15}”.format(t[0]) if cont % 4 == 0:
ExibeLinha(s, 64) s = “”
cont += 1 if s != “”:
ExibeLinha(s, 64)

ExibeLinha(“”, 64)

s = “Nº de Rodadas: {} - Nº de Jogos: {}” ExibeLinha(s.format(NRod, NJog), 64) print(“-” * LargTela)

Na função ExibeClassificacao é feita a exibição da tabela de classificação. Porém, para que isso seja possível, primeiro é preciso que sejam apurados os dados de classificação do torneio a partir dos resultados dos jogos, bem como seja definida a ordem do melhor para o pior colocado. Todas essas tarefas são executadas pela função chamada MontaClassificacao, e o restante de ExibeClassificacao apenas gera as saídas em tela.

Exemplo 10.14 Função ExibeClassificacao def ExibeClassificacao():
global Times, Torneio MontaClassificacao()
sfmt = “{:>3} {:<16}” + “{:>6}”*8

print(“-” * LargTela)

s = “Pos;Time;PG;J;V;E;D;GP;GC;SG”

print(sfmt.format(*s.split(“;”))) print(“-” * LargTela)
Pos = 1

for time in Times:

dados = (Pos,) + tuple(time) print(sfmt.format(*dados))

Pos += 1

print(“-” * LargTela)

Por sua vez, a função MontaClassificacao tem duas partes distintas e bem definidas. Na primeira, é feita a apuração dos resultados como pontos, número de jogos, vitórias, empates, derrotas e totais de gols marcados e sofridos. A função ZeraDadosTimes é chamada no início para zerar todos esses campos. Em seguida, a função LeJogos é chamada e retorna uma lista com todos os jogos que já tenham placar registrado (os jogos em que os campos gosl1 e gols2 sejam nulos ficam de fora). Lidos os jogos, é iniciado um laço que itera pelos Jogos comparando os gols marcados pelos times e computando os resultados de maneira apropriada por meio de chamadas à função ComputaResultado. Essas quatro funções citadas estão no Exemplo 10.15.

Exemplo 10.15 Função MontaClassificacao e suas associadas def MontaClassificacao():

global Times, Torneio

Primeira parte – Computa os resultados ZeraDadosTimes()

Jogos = LeJogos() for jogo in Jogos:

if jogo[3] == jogo[5]:

ComputaResultado(jogo[2], “E”, jogo[3], jogo[5]) ComputaResultado(jogo[4], “E”, jogo[5], jogo[3]) elif jogo[3] < jogo[5]:

ComputaResultado(jogo[2], “D”, jogo[3], jogo[5]) ComputaResultado(jogo[4], “V”, jogo[5], jogo[3]) elif jogo[3] > jogo[5]:

ComputaResultado(jogo[2], “V”, jogo[3], jogo[5])

ComputaResultado(jogo[4], “D”, jogo[5], jogo[3]) # Segunda parte – Ordena a tabela de classificação OrdenaTimes()

def ZeraDadosTimes():

global Times

for time in Times:

```
for i in range(1, 9):

time[i] = 0  def LeJogos():
global Torneio

conector = sqlite3.connect(Torneio + ".db") cursor = conector.cursor()
sql = """

select * from jogos

where gol1 is not null order by numjogo """
cursor.execute(sql)
J = cursor.fetchall() cursor.close() conector.close() return J
def ComputaResultado(QualTime, Res, GP, GC): global Times
for time in Times:

if time[0] == QualTime: time[2] += 1 # qtde jogos time[6] += GP # gols pro time[7] += GC
# gols contra time[8] += GP-GC # saldo gols if Res == "V":
time[1] += 3 # ptos ganhos time[3] += 1 # qtde vitorias elif Res == "E":
time[1] += 1 # ptos ganhos
time[4] += 1 # qtde empates elif Res == "D":
time[5] += 1 # qtde derrotas
```

Após apurados os resultados dos jogos, é necessário ordenar a lista Times. Isso é feito na função OrdenaTimes, chamada na última linha de MontaClassificacao e na qual é implementado um algoritmo de ordenação *Bubble Sort*. Esse é um algoritmo simples e já foi explicado. Trata-se de um algoritmo de simples entendimento, porém, lento para grandes quantidades de dados. Por grandes quantidades entenda-se algo acima de 10^5 elementos. No caso de um torneio esportivo, tem-se no máximo duas dezenas de elementos, de modo que o *Bubble Sort* será suficientemente rápido.

Por outro lado, não será utilizada nenhuma função de ordenação disponível em Python,

porque o critério de comparação de dois times é muito complexo, uma vez que envolve cinco diferentes parâmetros de comparação e desempate, conforme listado no Quadro 10.2. Como pode ser visto no Exemplo 10.16, a função OrdenaTimes propriamente dita é simples. O laço externo depende de haver ocorrido troca de posição de elementos da lista, e o laço interno percorre toda a lista, comparando um elemento a seu vizinho subsequente, e caso o elemento i seja menor – porque a ordenação é decrescente – que o elemento i+1, ocorre a troca de posições. A complexidade desse processo fica por conta da função Compara, que deve atender a todos os critérios. Essa função recebe os parâmetros a e b, que são sublistas da lista de times. Ela retorna −1 caso a seja menor que b; 0 caso a seja igual a b e 1 caso a seja maior que b.

Exemplo 10.16 Função OrdenaTimes e suas associadas

```
def OrdenaTimes():

# usa BubbleSort para ordenar os times global Times
Trocou = True while Trocou:
Trocou = False i = 0
while i < len(Times)-1:

if Compara(Times[i], Times[i+1]) < 0: Times[i], Times[i+1] = Times[i+1], Times[i]
Trocou = True i += 1
def Compara(a, b): # 1º Crit. Pontos
if a[1] < b[1]: return -1
elif a[1] > b[1]:
return 1

# 2º Crit. Vitórias if a[3] < b[3]:
return -1

elif a[3] > b[3]:

return 1
```

```
# 3º Crit. Saldo Gols if a[8] < b[8]:
return -1

elif a[8] > b[8]:

return 1

# 4º Crit. Gols Pró if a[6] < b[6]:
return -1

elif a[6] > b[6]:

return 1

return ConfrontoDireto(a, b)

def ConfrontoDireto(a, b):

global Torneio

d = {“timeA”:a[0], “timeB”:b[0]} ptoA = ptoB = 0
conector = sqlite3.connect(Torneio + “.db”) cursor = conector.cursor()
sql = “””

select * from jogos where gol1 is not null and time1 = :timeA and time2 = :timeB
“””
cursor.execute(sql, d) J = cursor.fetchone() if J:
if J[3] == J[5]:

ptoA += 1
ptoB += 1
```

```
elif J[3] > J[5]:
ptoA += 3

elif J[5] < J[3]:

ptoB += 3 sql = “””
select * from jogos where gol1 is not null and time1 = :timeB and time2 = :timeA
“””

cursor.execute(sql, d) J = cursor.fetchone() if J:
if J[3] == J[5]:

ptoA += 1

ptoB += 1

elif J[3] > J[5]:

ptoB += 3

elif J[5] < J[3]:

ptoA += 3
cursor.close() conector.close() if ptoA > ptoB:
return -1

elif ptoA < ptoB:

return 1 else:
return 0
```

Por fim, há o confronto direto, que exige a recuperação dos jogos com placar já registrado e a verificação dos resultados para definir entre os times a e b qual deverá ficar na frente caso todos os demais critérios estejam empatados. Isso é executado na função ConfrontoDireto. Devem ser verificadas duas possibilidades: o time a como primeiro time e b como segundo, e o inverso. Para isso, executa-se o comando sql, que busca o jogo em cada situação, e comparam-se os gols (que estarão em J[3] e J[5]) para a atribuição de pontos.

Na tela de gerenciamento do torneio, já exibida, estão disponíveis ao usuário três funcionalidades. Ao digitar o número da rodada, o usuário tem acesso ao lançamento de resultados de jogos, conforme apresentado na tela da Figura 10.3. No Exemplo 10.17 tem-se a função GerenciaRodada, responsável pela implementação da tela exibida na Figura 10.3. Essa função permanece em laço até que o usuário opte por sair, e dentro dele é feita a exibição dos jogos da rodada por meio da função ExibeJogos, bem como se pode lançar ou apagar o resultado de um jogo.

Exemplo 10.17 Função GerenciaRodada def GerenciaRodada(NRod):
global Times, Torneio, Turnos Jogos = ObtemJogosRodada(NRod) while True:
TopoTela(“Gerenciamento de Torneio”) ExibeTimes()
JogosRodada = ExibeJogos(Jogos) print(“Opções: “)
print(“ (.) Para atualizar o placar de um jogo digite:”) print(“ NºJogo,GolsA,GolsB exemplo: 12,2,1”) print(“ (.) Para limpar o placar de um jogo digite:”) print(“ NºJogo,limpa exemplo: 12,limpa”)
print(“ (S) Voltar ao Menu do Torneio”) opc = input(“sua opção? >>> “)
opc = opc.upper() if opc == “S”:
break else:
msg = TrataEntrRodada(opc, JogosRodada) if msg != None:
Pausa(msg + “ (pressione Enter)”) else:
Jogos = ObtemJogosRodada(NRod)

A função ObtemJogosRodada é executada uma vez antes do início do laço e, depois, é executada novamente dentro deste. Essa função busca no banco de dados os jogos da rodada desejada, independentemente de o jogo ter sido jogado ou não, e retorna uma lista de listas.

Cada sublista é um jogo e seus elementos são: [nº jogo, nº rodada, time1, gols time1, time2, gols time2]. Na função ExibeJogos, caso os elementos 3 e 5 (os gols) sejam nulos, são omitidos da exibição; caso contrário, são exibidos. Isso evita que a palavra *None* seja mostrada na tela quando um jogo ainda não tem o resultado registrado.

Essa função ExibeJogos também gera uma lista JRod que é retornada ao seu final. Tal lista contém os números dos jogos da rodada. No retorno dessa função em GerenciaRodada a lista é atribuída ao objeto JogosRodada, que será utilizado para validar o lançamento dos resultados de um jogo. O motivo disso é explicado na descrição da função TrataEntrRodada.

Exemplo 10.18 Funções ObtemJogosRodada e ExibeJogos def ObtemJogosRodada(NRod):

global Torneio

conector = sqlite3.connect(Torneio + ".db") cursor = conector.cursor()
sql = "select * from jogos where numrod = ? order by numjogo" cursor.execute(sql, (NRod,))
J = cursor.fetchall() cursor.close() conector.close() return J
def ExibeJogos(Jogos):

ExibeLinha("*** Rodada {} ***".format(Jogos[0][1]), 64) s = "{:<6}{:<16}{:<5}x{:>5}{:>16}"
JRod = []

for J in Jogos: JRod.append(J[0])
if J[3] == J[5] == None:

ExibeLinha(s.format(J[0], J[2], " ", " ", J[4]), 64)
else:

ExibeLinha(s.format(J[0], J[2], J[3], J[5], J[4]), 64) print("-" * LargTela)
return JRod

Quando o usuário digita o resultado de um jogo, a função TrataEntrRodada é chamada. Nessa função são feitas a interpretação e a validação do que foi digitado pelo usuário e, em

seguida, a ação apropriada é tomada. É necessário lembrar que o usuário pode digitar qualquer coisa que queira, então, a validação deve ser criteriosa e a entrada só será aceita caso siga o padrão esperado. Caso não siga, a função retornará uma mensagem “Entrada Inválida”. Essa função retorna um string com uma mensagem caso a entrada seja inválida. Caso seja válida, retorna *None*.

A primeira providência dentro dessa função é remover eventuais espaços em branco. Como o padrão da entrada requer dados separados por vírgulas, é empregado o método split() do string para separação das partes. O split() deve gerar uma lista com dois (no caso de limpar o resultado) ou três (no caso de registrar resultado) elementos. Qualquer quantidade diferente invalida a entrada.

Se essa quantidade for dois, o primeiro elemento deve ser numérico (o nº do jogo) e o segundo elemento deve ser a palavra “LIMPA”. Caso isso ocorra, é chamada a função LimpaJogo; caso contrário, a entrada é inválida.

Se essa quantidade for 3, os três elementos devem ser numéricos (o nº do jogo, gols do time 1, gols do time 2). Nesse caso, optou-se por usar um bloco try-except-finally para tratar as conversões de string para inteiro. Caso algumas das conversões falhe, no bloco except retorna a mensagem de entrada inválida. Setudo correr bem, é feita a chamada à função AtualizaJogo, e o bloco finally garante o retorno de *None*.

Exemplo 10.19 Função TrataEntrRodada def TrataEntrRodada(opc, JogosRodada):
opc = opc.replace(“ “, “”) # remove espaços em branco L = opc.split(“,”)
if len(L) != 2 and len(L) != 3:

return “Entrada Invalida” if len(L) == 2:
if L[0].isnumeric() and L[1] == “LIMPA”: jogo = int(L[0])
if jogo in JogosRodada: LimpaJogo(jogo) return None
else:

return “Jogo de outra rodada” else:
return “Entrada Invalida”

```
if len(L) == 3:

try:

jogo = int(L[0]) gols1 = int(L[1]) gols2 = int(L[2])
if jogo in JogosRodada:

AtualizaJogo(jogo, gols1, gols2) else:
return “Jogo de outra rodada” except:
return “Entrada Inválida” finally:
return None
```

Por fim, as funções AtualizaJogo e LimpaJogo são responsáveis por executar *updates* na tabela de jogos, de modo a registrar ou limpar as quantidades de gols marcados pelos times do jogo. A existência da função LimpaJogo justifica-se pelo fato de que não é raro o erro de lançamento por parte do usuário, e o programa deve contar com algum recurso que permita a correção.

É possível verificar na função supracitada que essas duas funções só são chamadas se o nº do jogo digitado estiver contido na lista JogosRodada criada por ExibeJogos. Se isso não for feito, o usuário poderá alterar jogos (talvez acidentalmente) de rodadas diferentes da que está sendo visualizada, causando uma situação confusa e levando a erro de resultado.

Exemplo 10.20 Funções LimpaJogo e AtualizaJogo

```
def LimpaJogo(numjogo):
global Torneio

conector = sqlite3.connect(Torneio + “.db”) cursor = conector.cursor()
sql = “””

update jogos set gol1 = NULL, gol2 = NULL where numjogo = ?

“””

cursor.execute(sql, (numjogo,)) conector.commit() cursor.close()
```

```
conector.close()

def AtualizaJogo(numjogo, gol1, gol2): global Torneio
conector = sqlite3.connect(Torneio + ".db") cursor = conector.cursor()
sql = """

update jogos set gol1 = ?, gol2 = ? where numjogo = ?
""""


cursor.execute(sql, (gol1, gol2, numjogo)) conector.commit()
cursor.close() conector.close()
```

De volta à tela de gerenciamento do torneio, o Exemplo 10.21 exibe a função ExcluiTorneio, que é chamada quando o usuário deseja descartar um torneio cadastrado. Essa função pede que a ação seja confirmada e, caso seja, elimina o nome do torneio do banco de dados central "torneios.db" e usa a função os.remove (a biblioteca os contém funções que permitem a interação com o sistema operacional) para excluir o arquivo de banco de dados do torneio.

Exemplo 10.21 Função ExcluiTorneio

```
def ExcluiTorneio(Torneio): print("\n"*3)
ExibeLinha("Confirma Exclusão do Torneio " + Torneio, 40) print("Opções: ")
print(" (C) para Confirmar")

print(" qualquer outra tecla para retornar") opc = input("sua opção? >>> ")
opc = opc.upper() if opc == "C":
conector = sqlite3.connect("torneios.db") cursor = conector.cursor()
sql = "delete from torneios where nometorneio = '" + \ Torneio + "'"
print(Torneio) print(sql) Pausa("-" * 58)
cursor.execute(sql) conector.commit() cursor.close()
conector.close() os.remove(Torneio + ".db") return True
else:

return False
```

E a última funcionalidade é a gravação do torneio em arquivo HTML. Para realizar essa tarefa foi implementada a função GravaHTML. Nessa tarefa serão utilizados HTML5 para exibição do conteúdo e CSS3 para formatação visual da página. O arquivo CSS3 está pronto e disponível (veja o Exemplo 10.24).

Referências

Chamberlin, D. et al. A history and evaluation of System R.Communications ACM, v. 24, n. 10, 1981, 632-646. Disponível em:<http://doi.acm org/10.1145/ 358769.358784>. Acesso em: 23 out 2017. DB-API 2.0 interface for SQLite database. Python Software Foundation, 2017. Disponível em: <https://docs.python.org/3.6/library/sqlite3.html>.Acesso em: 15 out. 2017.

DODIS, Y. et al. Security analysis of pseudo-random number generators with input: /dev/random is not robust. In: CCS '13 – ACM SIGSAC CONFERENCE ON COMPUTER & COMMUNICATIONS SECURITY,

2013, Berlin. Proceedings… Berlin: Association for Computing Machinery, 2013. p. 647-658.

GENERATE pseudo-random numbers. Python Software Foundation, 2017. Disponível em: <https://docs.python.org/3/library/random.html>. Acesso em: 14 set. 2017.

GOODRICH, M. T.; TAMASSIA, R.; GOLDWASSER, M. H. Data structures and algorithms in Python. Hoboken: Wiley, 2013.

GUO, P. Python is now the most popular introductory teaching language at top U.S. universities. Communications of the ACM, 7 jul. 2014. Disponível em: <https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the- most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext>. Acesso em: 7 set. 2017.

LICENSE agreement for Python 3.6.3. Python Software Foundation, 2017. Disponível em: <https://docs.python.org/3/license.html>. Acesso em: 16 set. 2017.

LUTZ, M. Learning Python. 4. ed. Sebastopol: O'Reilly Media, 2009. NEUMANN, J. Various techniques used in connection with random digits. Applied Mathematics Series, Washington, D.C., v. 12, p. 36-38, 1951.

PAYNE, J. Beggining Python: using Python 2.6 and Python 3.1. Indianapolis: John Wiley & Sons, 2010.

PYTHON DATA MODEL. Python Software Foundation, 2017. Disponível em: <https://docs.python.org/3/reference/datamodel.html>. Acesso em: 14 set. 2017.

PYTHON FORMATTED Output. Python Course. Disponível em:<www.python-course.eu/python3_formatted_output.php>. Acesso em: 16 ago. 2017.

Python Software Foundation, 2017. Disponível em:<https://docs.python.org/3/glossary.html>.

Acesso em: 16 set. 2017.

PYTHON SOFTWARE FOUNDATION. Disponível em:<www.python.org>. Acesso em: 26 set. 2017.

RAMALHO, L. Fluent Python. Sebastopol: O'Reilly Media, 2015.

ROSSUM, G. Computer programming for everybody, a funding proposal sent to DARPA, 1999. Disponível em:<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.6836&rep=rep1&type=pdf>. Acesso em: 4 set. 2017.

Python 3000 status update (long!). Artima, 29 jun. 2007. Disponível em: <http://www.artima.com/weblogs/viewpost.jsp?thread=208549>. Acesso em: 27 set. 2017.

ROSSUM, G. PEP315, 2003. Disponível em: <www.python.org/dev/peps/pep-0315/>. Acesso em: 18 set. 2017.

SQLite. Disponível em: <https://www.sqlite.org/>. Acesso em: 25 set. 2017.

SQLite STUDIO. Disponível em: <https://sqlitestudio.pl/index.rvt>. Acesso em: 25 set. 2017.

TERMINOLOGIA Python em português: um guia para a tradução de termos específicos da linguagem Python para português. Disponível em:<http://turing.com.br/pydoc/2.7/tutorial/TERMINOLOGIA.html>. Acesso em: 16 set. 2017.

THE PYTHON LANGUAGE Reference. Python Software Foundation, 2017. Disponível em: <https://docs.python.org/3/reference/index.html>. Acesso em: 19 out. 2017.

THE PYTHON STANDARD LIBRARY – BUILT-IN Types. Python Software Foundation, 2017. Disponível em:<https://docs.python.org/3/library/functions.html>. Acesso em: 19 out. 2017.

THE PYTHON STANDARD LIBRARY – GENERAL Index. Python Software Foundation, 2017. Disponível em:<https://docs.python.org/3/library/index.html>. Acesso em: 19 out. 2017.

THE UNICODE CONSORTIUM. 2017. Disponível em:<www.unicode.org/>. Acesso em: 15 out. 2017.

UNICODE HOWTO. 2017. Disponível em:<https://docs.python.org/3/howto/unicode.html>. Acesso em: 14 out. 2017.

VAZIRANI, U. V. Efficient and secure pseudo-random number generation. In: 25TH Annual Sympsium on Foundations of Computer Science, 25, 1984, Singer Island. Proceedings… Singer Island, 1984. p. 458-463.