

Contenido

1. Cubierta

2. Página de _____

título 3. Página de _____

derechos de autor 4. Sobre el autor _____

5. Acerca del revisor técnico _____

6. Contenido breve _____

7. Contenidos en Detalle _____

8. Agradecimientos _____

9. Introducción _____

1. ¿Para quién es este libro? _____

2. Acerca de este libro _____

3. Cómo usar este libro _____

1. Números de línea y sangría _____

2. Líneas de código largas _____

4. Descarga e instalación de Python 5. Inicio de _____

IDLE 6. Búsqueda de ayuda en línea _____

10. Capítulo 1: El caparazón interactivo _____

1. Algunas matemáticas simples _____

1. Números enteros y números de punto flotante 2. _____

Expresiones _____

2. Evaluación de expresiones 3. _____

Errores de sintaxis 4. _____

Almacenamiento de valores en _____

variables 5. Resumen _____

11. Capítulo 2: Escritura de programas

1. Valores de cadena
2. Concatenación de cadenas
3. Escritura de programas en el editor de archivos de IDLE

1. Crear el programa Hello World 2. Guardar su programa 3. Ejecutar su programa

4. Cómo funciona el programa Hello World

1. Comentarios para el Programador 2. Funciones: Mini-Programas Dentro de Programas 3. El Fin del Programa

5. Nombrar variables 6.

Resumen

12. Capítulo 3: Adivina el número

1. Ejecución de muestra de Guess the Number 2. Código fuente de Guess the Number

3. Importando el Módulo aleatorio 4. Generando Números Aleatorios con random.randint()

Función

5. Bienvenida al jugador 6. Declaraciones de control de flujo

1. Uso de bucles para repetir código 2. Agrupación con bloques 3. Bucle con instrucciones for

7. Obtener la conjectura del jugador

8. Convertir valores con las funciones int(), float() y str() 9. El tipo de datos booleano

1. Operadores de comparación 2. Comprobación de verdadero o falso con condiciones

3. Experimentando con Booleanos, Operadores de Comparación y
Condiciones
4. La diferencia entre = y ==

10. declaraciones if
11. Salir de los bucles temprano con la declaración de descanso
12. Verificar si el jugador ganó 13. Verificar si el jugador perdió
14. Resumen

13. Capítulo 4: Un programa para contar chistes

1. Ejemplo de ejecución de
chistes 2. Código fuente para chistes
3. Cómo funciona el código
4. Caracteres de escape 5.
Comillas simples y dobles 6. El
parámetro de palabra clave final de la función print() 7. Resumen

14. Capítulo 5: Reino del Dragón

1. Cómo jugar Dragon Realm 2. Ejemplo
de ejecución de Dragon Realm 3. Diagrama
de flujo para Dragon Realm 4. Código
fuente para Dragon Realm 5. Importación de
módulos aleatorios y de tiempo 6. Funciones en Dragon
Realm

1. declaraciones def
2. Llamar a una función 3.
Dónde colocar las definiciones de funciones

7. Cadenas multilínea 8.
Cómo hacer un bucle con instrucciones while 9.
Operadores booleanos

1. El operador and 2. El
operador or

3. El operador not 4.

Evaluación de operadores booleanos

10. Valores devueltos

11. Alcance Global y Alcance Local

12. Parámetros de función

13. Visualización de los resultados del

juego 14. Decidir qué cueva tiene el dragón amigo 15. El ciclo del
juego

1. Llamar a las funciones en el programa 2. Pedirle

al jugador que toque de nuevo

16. Resumen

15. Capítulo 6: Uso del depurador

1. Tipos de errores 2.

El depurador

1. Iniciar el depurador 2. Recorrer

paso a paso el programa con el depurador

3. Encontrar el error 4.

Establecer puntos de

interrupción 5. Usar puntos

de interrupción 6. Resumen

16. Capítulo 7: Diseño de Hangman con diagramas de flujo

1. Cómo jugar Hangman 2.

Ejemplo de ejecución de Hangman

3. Arte ASCII

4. Diseño de un programa con un diagrama de flujo

1. Creación del diagrama de flujo

2. Ramificación desde un cuadro de diagrama

de flujo 3. Finalización o reinicio del juego

4. Adivinar de nuevo 5.

Ofrecer retroalimentación al jugador

5. Resumen

17. Capítulo 8: Escribiendo el Código del Ahorcado

1. Código fuente de Hangman 2.

Importación del módulo aleatorio 3. Variables
constantes

4. El tipo de datos de listas

1. Acceso a elementos con índices

2. Concatenación de listas

3. El operador de entrada

5. Métodos de llamada

1. Los métodos de lista reverse() y append() 2. El método

de cadena split()

6. Obtener una palabra secreta de la lista de palabras 7.

Mostrar el tablero al jugador

1. Las funciones list() y range() 2. Segmentación

de listas y cadenas 3. Visualización de la

palabra secreta con espacios en blanco

8. Obtener la suposición del jugador

1. Los métodos de cadena lower() y upper() 2. Salir del

bucle while

9. declaraciones elif

10. Asegurarse de que el jugador ingresó una suposición válida 11.

Pedir al jugador que juegue de nuevo 12. Revisión de las

funciones del ahorcado 13. El ciclo del juego

1. Llamar a la función `displayBoard()` 2.
- Permitir que el jugador ingrese su suposición
3. Verificar si la letra está en la palabra secreta 4. Verificar si el jugador ganó 5. Manejar una suposición incorrecta 6.
- Verificar si el jugador perdió 7. Terminar o Restablecer el juego

14. Resumen

18. Capítulo 9: Ahorcado extendido

1. Adición de más conjeturas
2. El tipo de datos del diccionario
 1. Obtener el tamaño de los diccionarios con `len()`
 2. La diferencia entre diccionarios y listas
 3. Los métodos de diccionario `keys()` y `valores()` 4. Uso de diccionarios de palabras en Hangman
3. Selección aleatoria de una lista 4.
- Eliminación de elementos de las listas 5.
- Asignación múltiple 6. Impresión de la categoría de palabras para el jugador 7. Resumen

19. Capítulo 10: Tres en raya

1. Ejemplo de ejecución de Tic-Tac-Toe 2. Código fuente de Tic-Tac-Toe
3. Diseño del Programa
 1. Representación del tablero como datos 2.
 - Elaboración de estrategias con la IA del juego
4. Importar el módulo aleatorio 5.
- Imprimir el tablero en la pantalla 6.
- Permitir que el jugador elija X u O 7.
- Decidir quién va primero

8. Colocando una Marca en el Tablero

1. Lista de referencias

2. Usando Lista de Referencias en hacerJugada()

9. Verificar si el jugador ganó 10. Duplicar los datos

del tablero 11. Verificar si un espacio en el tablero

está libre 12. Permitir que el jugador ingrese un movimiento 13.

Evaluación de cortocircuito

14. Elegir un movimiento de una lista de movimientos 15. El

valor Ninguno

16. Creando la IA de la computadora

1. Verificar si la computadora puede ganar en un solo movimiento 2. Verificar si

el jugador puede ganar en un solo movimiento 3. Verificar los espacios de las

esquinas, el centro y los lados (en ese

Ordenar)

4. Comprobando si el tablero está lleno

17. El circuito del juego

1. Elegir la marca del jugador y quién juega primero 2. Ejecutar el turno

del jugador 3. Ejecutar el turno de la computadora 4. Pedirle al jugador

que juegue de nuevo

18. Resumen

20. Capítulo 11: El juego de deducción de bagels

1. Ejemplo de ejecución de bagels

2. Código fuente para bagels 3.

Diagrama de flujo para bagels 4.

Importación aleatoria y definición de getSecretNum()

5. Barajar un conjunto único de dígitos

1. Cambiar el orden de los elementos de la lista con random.shuffle()

Función

2. Obtener el número secreto de los dígitos barajados

6. Operadores de asignación aumentada 7.

Calcular las pistas para dar 8. El método de lista

sort() 9. El método de cadena join() 10. Verificar

si una cadena tiene solo números 11. Comenzar

el juego 12. Interpolación de cadenas 13. El juego Círculo

1. Obtener la suposición del jugador

2. Obtener las pistas para la suposición del jugador 3.

Verificar si el jugador ganó o perdió 4. Pedirle al jugador

que juegue de nuevo

14. Resumen

21. Capítulo 12: El sistema de coordenadas cartesianas

1. Grillas y Coordenadas Cartesianas

2. Números negativos 3. El

sistema de coordenadas de una pantalla de computadora

4. Trucos matemáticos

1. Truco 1: Un menos se come el signo más a su izquierda 2.

Truco 2: Dos menos se combinan en un más

3. Truco 3: dos números que se suman pueden intercambiar lugares

5. Valores absolutos y la función abs() 6. Resumen

22. Capítulo 13: Búsqueda del tesoro con sonar

1. Ejecución de muestra de Sonar Treasure Hunt

2. Código fuente de Sonar Treasure Hunt

3. Diseño del programa 4.

Importación de los módulos random, sys y math 5. Creación

de un nuevo tablero de juego

6. Dibujar el tablero de juego

1. Dibujar las coordenadas X en la parte superior del tablero 2.

Dibujar el océano 3. Imprimir una fila en el océano 4. Dibujar las coordenadas X en la parte inferior del tablero

Junta

7. Crear los cofres del tesoro aleatorios 8.

Determinar si un movimiento es válido 9. Colocar un movimiento en el tablero

1. Encontrar el cofre del tesoro más cercano

2. Eliminar valores con el método de lista remove() 3. Obtener el movimiento del jugador

10. Impresión de las instrucciones del juego para el

jugador 11. El bucle del juego

1. Mostrar el estado del juego para el jugador 2.

Manejar el movimiento del jugador 3. Encontrar un cofre del tesoro hundido 4. Verificar si el jugador ganó 5. Verificar si el jugador perdió 6. Terminar el programa con la función sys.exit()

12. Resumen

23. Capítulo 14: Cifrado César

1. Criptografía y cifrado 2. Cómo

funciona el cifrado César 3. Ejecución

de muestra del cifrado César 4. Código

fuente del cifrado César 5. Configuración

de la longitud máxima de la clave 6. Decidir

cifrar o descifrar el mensaje 7. Obtener el mensaje del

reproductor 8. Obtener la clave del reproductor 9. Cifrar

o descifrar el mensaje

1. Encontrar cadenas pasadas con el método de cadena `find()` 2.

Cifrar o descifrar cada letra

10. Iniciar el programa 11. La
técnica de fuerza bruta 12. Agregar el
modo de fuerza bruta 13. Resumen

24. Capítulo 15: El juego inverso

1. Cómo jugar Reversegam 2.

Ejemplo de ejecución de Reversegam

3. Código fuente para Reversegam 4.

Importación de módulos y configuración de constantes 5.

La estructura de datos del tablero de juego

1. Dibujar la estructura de datos del tablero en la pantalla 2.

Crear una nueva estructura de datos del tablero

6. Comprobar si un movimiento es válido

1. Comprobación de cada una de las ocho

direcciones 2. Averiguar si hay fichas para voltear

7. Comprobación de coordenadas válidas

1. Obtener una lista con todos los movimientos
válidos 2. Llamar a la función `bool()`

8. Obtener la puntuación del tablero de juego 9.

Obtener la elección de ficha del jugador 10.

Determinar quién comienza primero 11. Colocar
una ficha en el tablero 12. Copiar la estructura de
datos del tablero 13. Determinar si un espacio
está en una esquina 14. Obtener la jugada del jugador 15.

Obtener la jugada de la computadora

1. Elaborar estrategias con movimientos de
esquina 2. Obtener una lista de los movimientos de mayor puntuación

16. Impresión de las puntuaciones en la pantalla

17. Inicio del juego

1. Comprobación de un punto muerto

2. Ejecución del turno del jugador 3.

Ejecución del turno de la computadora

18. El ciclo del juego 19.

Pedirle al jugador que vuelva a jugar 20. Resumen

25. Capítulo 16: Simulación de IA de juego inverso

1. Hacer que la computadora juegue contra sí misma

1. Ejemplo de ejecución de la simulación 1

2. Código fuente de la simulación 1

3. Eliminar las indicaciones del reproductor y agregar una computadora

Jugador

2. Hacer que la computadora juegue sola varias veces

1. Ejemplo de ejecución de la simulación 2

2. Código fuente para Simulación 2

3. Realizar un seguimiento de múltiples juegos 4.

Comentar las llamadas a la función print() 5. Usar

porcentajes para calificar las IA

3. Comparación de diferentes algoritmos de IA

1. Código fuente para Simulación 3

2. Cómo funcionan las IA en Simulation 3

3. Comparando las IA

4. Resumen

26. Capítulo 17: Creación de gráficos

1. Instalación de pygame

2. Hello World en pygame 3.

Ejecución de muestra de pygame Hello World

4. Código fuente para pygame Hello World 5.

Importación del módulo pygame 6. Inicialización
de pygame 7. Configuración de la ventana de
pygame

1. Tuplas

2. Objetos de superficie

8. Configuración de variables de

color 9. Escritura de texto en la ventana de pygame

1. Uso de fuentes para aplicar estilo al

texto 2. Representación de un objeto

de fuente 3. Configuración de la ubicación del texto con atributos Rect

10. Relleno de un objeto de superficie con un

color 11. Funciones de dibujo de pygame

1. Dibujar un polígono 2.

Dibujar una línea 3. Dibujar

un círculo 4. Dibujar una

elipse 5. Dibujar un

rectángulo 6. Colorear píxeles

12. El método blit() para objetos de superficie 13.

Dibujar el objeto de superficie en la pantalla 14.

Eventos y bucle de juego

1. Obtener objetos de eventos

2. Salir del programa

15. Resumen

27. Capítulo 18: Animación de gráficos

1. Ejecución de muestra del programa de animación 2.

Código fuente para el programa de animación 3. Mover

y rebotar las cajas 4. Configuración de las variables

constantes

1. Variables constantes para la dirección

2. Variables constantes para el color

5. Configuración de las estructuras de datos de la

caja 6. El bucle del juego

1. Manejo cuando el jugador sale 2. Mover

cada caja 3. Rebotar una caja 4. Dibujar las

cajas en la ventana en su nuevo

Posiciones

5. Dibujar la ventana en la pantalla

7. Resumen

28. Capítulo 19: Detección de colisiones

1. Ejecución de muestra del programa de detección de colisiones

2. Código fuente para el programa de detección de colisiones 3.

Importación de los módulos 4. Uso de un reloj para marcar el ritmo

del programa 5. Configuración de la ventana y las estructuras de

datos 6. Configuración de variables para rastrear el movimiento 7

Gestión de eventos

1. Manejo del evento KEYDOWN 2. Manejo

del evento KEYUP

8. Teletransportar al jugador 9.

Agregar nuevos cuadrados de comida

10. Mover al jugador alrededor de la ventana

1. Dibujar el reproductor en la ventana 2.

Verificar colisiones

11. Dibujar los cuadrados de alimentos en la ventana

12. Resumen

29. Capítulo 20: Uso de sonidos e imágenes

1. Adición de imágenes con Sprites

2. Archivos de imagen y sonido 3.

Ejemplo de ejecución del programa Sprites and Sounds 4.

Código fuente del programa Sprites and Sounds 5. Configuración
de la ventana y la estructura de datos

1. Agregando un Sprite

2. Cambiando el Tamaño de un Sprite

6. Configuración de la música y los sonidos

1. Agregar archivos de

sonido 2. Activar y desactivar el sonido

7. Dibujar el jugador en la ventana 8.

Comprobar si hay colisiones 9. Dibujar las
cerezas en la ventana 10. Resumen

30. Capítulo 21: Un juego de los Dodgers con sonidos e imágenes

1. Revisión de los tipos de datos básicos de pygame

2. Ejemplo de ejecución de Dodger 3. Código fuente
de Dodger 4. Importación de módulos 5. Configuración
de variables constantes 6. Definición de funciones

1. Terminar y pausar el juego 2. Hacer

un seguimiento de las colisiones de malos

3. Dibujar texto en la ventana

7. Inicialización de pygame y configuración de la ventana

8. Configuración de objetos de fuente, sonido e imagen 9.

Visualización de la pantalla de inicio 10. Inicio del juego 11.

El bucle del juego

1. Manejo de eventos del teclado

2. Manejo del movimiento del mouse

12. Agregar nuevos malos

13. Mover el personaje del jugador y los malos 14.

Implementar los códigos de trucos 15. Eliminar los malos

16. Dibujar la ventana

1. Dibujar la puntuación del jugador

2. Dibujar el personaje y los malos del jugador

17. Comprobación de colisiones

18. La pantalla Game Over

19. Modificando el Juego de los

Dodgers 20. Resumen

31. Índice

32. Recursos

33. La Fundación Frontera Electrónica (EFF)

34. No se limite a jugar, ¡hágalo!

1. yo

2. ii

3. iii

4. iv

5to c

6. nosotros

7. viii

8. VIII

9. ix

10. x
11. xi
12. xi
13. XIII
14. xiv
15. XV
16. xvi
17. xviii
18. xviii
19. xix
20. xx
21. xxi
22. XXII
23. XXIII
24. XXIV
25. xxiv
26. xxv
27. xxvii
28. xxvii
29. 1
30. 2
31. 3
32. 4
33. 5
34. 6
35. 7
36. 8
37. 9
38. 10
39. 11
40. 12
41. 13
42. 14
43. 15
44. 16
45. 17
46. 18
47. 19
48. 20
49. 21

- 50. [22](#)
- 51. [23](#)
- 52. [24](#)
- 53. [25](#)
- 54. [26](#)
- 55. [27](#)
- 56. [28](#)
- 57. [29](#)
- 58. [30](#)
- 59. [31](#)
- 60. [32](#)
- 61. [33](#)
- 62. [34](#)
- 63. [35](#)
- 64. [36](#)
- 65. [37](#)
- 66. [38](#)
- 67. [39](#)
- 68. [40](#)
- 69. [41](#)
- 70. [42](#)
- 71. [43](#)
- 72. [44](#)
- 73. [45](#)
- 74. [46](#)
- 75. [47](#)
- 76. [48](#)
- 77. [49](#)
- 78. [50](#)
- 79. [51](#)
- 80. [52](#)
- 81. [53](#)
- 82. [54](#)
- 83. [55](#)
- 84. [56](#)
- 85. [57](#)
- 86. [58](#)
- 87. [59](#)
- 88. [60](#)
- 89. [61](#)

- 90. [62](#)
- 91. [63](#)
- 92. [64](#)
- 93. [65](#)
- 94. [66](#)
- 95. [67](#)
- 96. [68](#)
- 97. [69](#)
- 98. [70](#)
- 99. [71](#)
- 100. [72](#)
- 101. [73](#)
- 102. [74](#)
- 103. [75](#)
- 104. [76](#)
- 105. [77](#)
- 106. [78](#)
- 107. [79](#)
- 108. [80](#)
- 109. [81](#)
- 110. [82](#)
- 111. [83](#)
- 112. [84](#)
- 113. [85](#)
- 114. [86](#)
- 115. [87](#)
- 116. [88](#)
- 117. [89](#)
- 118. [90](#)
- 119. [91](#)
- 120. [92](#)
- 121. [93](#)
- 122. [94](#)
- 123. [95](#)
- 124. [96](#)
- 125. [97](#)
- 126. [98](#)
- 127. [99](#)
- 128. [100](#)
- 129. [101](#)

- 130. [102](#)
- 131. [103](#)
- 132. [104](#)
- 133. [105](#)
- 134. [106](#)
- 135. [107](#)
- 136. [108](#)
- 137. [109](#)
- 138. [110](#)
- 139. [111](#)
- 140. [112](#)
- 141. [113](#)
- 142. [114](#)
- 143. [115](#)
- 144. [116](#)
- 145. [117](#)
- 146. [118](#)
- 147. [119](#)
- 148. [120](#)
- 149. [121](#)
- 150. [122](#)
- 151. [123](#)
- 152. [124](#)
- 153. [125](#)
- 154. [126](#)
- 155. [127](#)
- 156. [128](#)
- 157. [129](#)
- 158. [130](#)
- 159. [131](#)
- 160. [132](#)
- 161. [133](#)
- 162. [134](#)
- 163. [135](#)
- 164. [136](#)
- 165. [137](#)
- 166. [138](#)
- 167. [139](#)
- 168. [140](#)
- 169. [141](#)

- 170. [142](#)
- 171. [143](#)
- 172. [144](#)
- 173. [145](#)
- 174. [146](#)
- 175. [147](#)
- 176. [148](#)
- 177. [149](#)
- 178. [150](#)
- 179. [151](#)
- 180. [152](#)
- 181. [153](#)
- 182. [154](#)
- 183. [155](#)
- 184. [156](#)
- 185. [157](#)
- 186. [158](#)
- 187. [159](#)
- 188. [160](#)
- 189. [161](#)
- 190. [162](#)
- 191. [163](#)
- 192. [164](#)
- 193. [165](#)
- 194. [166](#)
- 195. [167](#)
- 196. [168](#)
- 197. [169](#)
- 198. [170](#)
- 199. [171](#)
- 200. [172](#)
- 201. [173](#)
- 202. [174](#)
- 203. [175](#)
- 204. [176](#)
- 205. [177](#)
- 206. [178](#)
- 207. [179](#)
- 208. [180](#)
- 209. [181](#)

- 210. [182](#)
- 211. [183](#)
- 212. [184](#)
- 213. [185](#)
- 214. [186](#)
- 215. [187](#)
- 216. [188](#)
- 217. [189](#)
- 218. [190](#)
- 219. [191](#)
- 220. [192](#)
- 221. [193](#)
- 222. [194](#)
- 223. [195](#)
- 224. [196](#)
- 225. [197](#)
- 226. [198](#)
- 227. [199](#)
- 228. [200](#)
- 229. [201](#)
- 230. [202](#)
- 231. [203](#)
- 232. [204](#)
- 233. [205](#)
- 234. [206](#)
- 235. [207](#)
- 236. [208](#)
- 237. [209](#)
- 238. [210](#)
- 239. [211](#)
- 240. [212](#)
- 241. [213](#)
- 242. [214](#)
- 243. [215](#)
- 244. [216](#)
- 245. [217](#)
- 246. [218](#)
- 247. [219](#)
- 248. [220](#)
- 249. [221](#)

250. [222](#)
251. [223](#)
252. [224](#)
253. [225](#)
254. [226](#)
255. [227](#)
256. [228](#)
257. [229](#)
258. [230](#)
259. [231](#)
260. [232](#)
261. [233](#)
262. [234](#)
263. [235](#)
264. [236](#)
265. [237](#)
266. [238](#)
267. [239](#)
268. [240](#)
269. [241](#)
270. [242](#)
271. [243](#)
272. [244](#)
273. [245](#)
274. [246](#)
275. [247](#)
276. [248](#)
277. [249](#)
278. [250](#)
279. [251](#)
280. [252](#)
281. [253](#)
282. [254](#)
283. [255](#)
284. [256](#)
285. [257](#)
286. [258](#)
287. [259](#)
288. [260](#)
289. [261](#)

- 290. [262](#)
- 291. [263](#)
- 292. [264](#)
- 293. [265](#)
- 294. [266](#)
- 295. [267](#)
- 296. [268](#)
- 297. [269](#)
- 298. [270](#)
- 299. [271](#)
- 300. [272](#)
- 301. [273](#)
- 302. [274](#)
- 303. [275](#)
- 304. [276](#)
- 305. [277](#)
- 306. [278](#)
- 307. [279](#)
- 308. [280](#)
- 309. [281](#)
- 310. [282](#)
- 311. [283](#)
- 312. [284](#)
- 313. [285](#)
- 314. [286](#)
- 315. [287](#)
- 316. [288](#)
- 317. [289](#)
- 318. [290](#)
- 319. [291](#)
- 320. [292](#)
- 321. [293](#)
- 322. [294](#)
- 323. [295](#)
- 324. [296](#)
- 325. [297](#)
- 326. [298](#)
- 327. [299](#)
- 328. [300](#)
- 329. [301](#)

- 330. [302](#)
- 331. [303](#)
- 332. [304](#)
- 333. [305](#)
- 334. [306](#)
- 335. [307](#)
- 336. [308](#)
- 337. [309](#)
- 338. [310](#)
- 339. [311](#)
- 340. [312](#)
- 341. [313](#)
- 342. [314](#)
- 343. [315](#)
- 344. [316](#)
- 345. [317](#)
- 346. [318](#)
- 347. [319](#)
- 348. [320](#)
- 349. [321](#)
- 350. [322](#)
- 351. [323](#)
- 352. [324](#)
- 353. [325](#)
- 354. [326](#)
- 355. [327](#)
- 356. [328](#)
- 357. [329](#)
- 358. [330](#)
- 359. [331](#)
- 360. [332](#)
- 361. [333](#)
- 362. [334](#)
- 363. [335](#)
- 364. [336](#)
- 365. [337](#)
- 366. [338](#)
- 367. [339](#)
- 368. [340](#)
- 369. [341](#)

370. [342](#)

371. [343](#)

372. [344](#)

373. [345](#)

374. [346](#)

375. [347](#)

376. [348](#)

377. [349](#)

INVENTA LA TU PROPIA
JUEGOS DE COMPUTADORA
CON PITÓN

4^a EDICIÓN

Al Sweigart



San Francisco

INVENTA TUS PROPIOS JUEGOS DE ORDENADOR CON PYTHON, 4º
EDICIÓN.

Copyright © 2017 por Al Sweigart.

Algunos derechos reservados. Este trabajo tiene la licencia Creative Commons Attribution-NonCommercial-ShareAlike 3.0 United States License. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-nc-sa/3.0/us/> o envíe una carta a Creative Commons, PO Box 1866, Mountain View, CA 94042, EE. UU.

Impreso en EE. UU.

Primera impresión

20 19 18 17 16 1 2 3 4 5 6 7 8 9

ISBN-10: 1-59327-795-4

ISBN-13: 978-1-59327-795-6

Editor: William Pollock

Editora de producción: Laurel Chun

Ilustración de portada: Josh Ellingson

Diseño de Interiores: Octopod Studios

Editor de desarrollo: Jan Cash

Revisor técnico: Ari Lacenski

Correctora: Rachel Monaghan

Compositora: Susan Glinert Stevens

Correctora: Paula L. Fleming Indexadora:

Nancy Guenther

Las imágenes de sprite en la Figura 20-1 en la página 302, de izquierda a derecha, fueron creadas por fsvieira, przemek.sz, LordNeo y Supercut. La imagen del sprite de hierba en la Figura 20-2 en la página 302 fue creada por txturs. Estas imágenes se han dedicado al dominio público con una dedicación de dominio público CC0 1.0.

Para obtener información sobre distribución, traducciones o ventas al por mayor, comuníquese directamente con No Starch Press, Inc.: No Starch Press, Inc.

245 8th Street, San Francisco, CA 94103

Teléfono: 1.415.863.9900; info@nostarch.com

www.nostarch.com

Datos de catalogación en publicación de la Biblioteca del Congreso

Nombres: Sweigart, Al, autor.

Título: Inventá tus propios juegos de computadora con Python / por Al Sweigart.

Descripción: San Francisco: No Starch Press, Inc., [2017]

Identificadores: LCCN 2016037817 (imprimir) | LCCN 2016044807 (libro electrónico) | ISBN

9781593277956 | ISBN 1593277954 | ISBN 9781593278113 (epub) | ISBN

159327811X (epub) | ISBN 9781593278120 (móvil) | ISBN 1593278128 (móvil)

Asignaturas: LCSH: Juegos de ordenador--Programación. | Python (Programa de computadora

idioma)

Clasificación: LCC QA76.76.C672 S785 2017 (impresión) | LCC QA76.76.C672 (libro electrónico)

| DDC 794.8/1526--dc23

Registro de LC disponible en <https://lccn.loc.gov/2016037817>

No Starch Press y el logotipo de No Starch Press son marcas comerciales registradas de No Starch Press, Inc. Otros nombres de productos y empresas mencionados aquí pueden ser marcas comerciales de sus respectivos propietarios. En lugar de utilizar un símbolo de marca comercial cada vez que aparece un nombre de marca comercial, utilizamos los nombres solo de forma editorial y en beneficio del propietario de la marca comercial, sin intención de infringir la marca comercial.

La información de este libro se distribuye "tal cual", sin garantía.

Si bien se han tomado todas las precauciones en la preparación de este trabajo, ni el autor ni No Starch Press, Inc. tendrán responsabilidad alguna ante ninguna persona o entidad con respecto a cualquier pérdida o daño causado o presuntamente causado directa o indirectamente por el información contenida en el mismo.

Sobre el Autor

Al Sweigart es un desarrollador de software, autor de libros de tecnología y fanático de los juegos que realmente sabe dónde está su toalla. Ha escrito varios libros de programación para principiantes, incluidos Automatice the Boring Stuff with Python y Scratch Programming Playground, también de No Starch Press. Sus libros están disponibles gratuitamente bajo una licencia Creative Commons en su sitio web <https://inventwithpython.com/>.

Acerca del revisor técnico

Ari Lacenski es desarrollador de aplicaciones Android y software Python. Vive en el Área de la Bahía, donde escribe sobre la programación de Android en <http://gradlewhy.ghost.io/> y mentores con Women Who Code.

CONTENIDOS BREVES

[Expresiones de gratitud](#)

[Introducción](#)

[Capítulo 1: El caparazón interactivo](#)

[Capítulo 2: Programas de escritura](#)

[Capítulo 3: Adivina el número](#)

[Capítulo 4: Un programa para contar chistes](#)

[Capítulo 5: Reino del Dragón](#)

[Capítulo 6: Uso del depurador](#)

[Capítulo 7: Diseño de Hangman con diagramas de flujo](#)

[Capítulo 8: Escribiendo el Código del Ahorcado](#)

[Capítulo 9: Ahorcado extendido](#)

[Capítulo 10: Tres en raya](#)

[Capítulo 11: El juego de deducción de bagels](#)

[Capítulo 12: El sistema de coordenadas cartesianas](#)

[Capítulo 13: Búsqueda del tesoro con sonar](#)

[Capítulo 14: Cifrado César](#)

[Capítulo 15: El juego inverso](#)

[Capítulo 16: Simulación de IA de Reversegam](#)

[Capítulo 17: Creación de gráficos](#)

[Capítulo 18: Animación de gráficos](#)

[Capítulo 19: Detección de colisiones](#)

[Capítulo 20: Uso de sonidos e imágenes](#)

[Capítulo 21: Un juego de los Dodgers con sonidos e imágenes](#)

[Índice](#)

CONTENIDOS EN DETALLE

EXPRESIONES DE GRATITUD

INTRODUCCIÓN

¿Para quién es este libro?

Sobre este libro

Como usar este libro

Números de línea y sangría

Líneas de código largas

Descarga e instalación de Python

Inicio inactivo

Encontrar ayuda en línea

1

LA CONCHA INTERACTIVA

Algunas matemáticas simples

Números enteros y números de coma flotante

Expresiones

Evaluación de expresiones

Errores de sintaxis

Almacenamiento de valores en variables

Resumen

2

PROGRAMAS DE ESCRITURA

Valores de cadena

[Concatenación de cadenas](#)

[Escribir programas en el editor de archivos de IDLE](#)

[Creando el programa Hola Mundo](#)

[Guardar su programa](#)

[Ejecutando su programa](#)

[Cómo funciona el programa Hola Mundo](#)

[Comentarios para el programador](#)

[Funciones: Miniprogramas Programas internos](#)

[El fin del programa](#)

[Nombrando Variables](#)

[Resumen](#)

3

[ADIVINA EL NÚMERO](#)

[Ejemplo de ejecución de Adivina el número](#)

[Código fuente de Adivina el número](#)

[Importando el módulo aleatorio](#)

[Generando números aleatorios con random.randint\(\)](#)

[Función](#)

[Bienvenida al jugador](#)

[Declaraciones de control de flujo](#)

[Uso de bucles para repetir código](#)

[Agrupación con bloques](#)

[Bucles con sentencias for](#)

[Obtener la suposición del jugador](#)

[Conversión de valores con las funciones int\(\), float\(\) y str\(\)](#)

[El tipo de datos booleano](#)

[Operadores de comparación](#)

[Comprobación de verdadero o falso con condiciones](#)

[Experimentando con Booleanos, Comparación](#)

[Operadores y Condiciones](#)

[La diferencia entre = y ==](#)

[si declaraciones](#)

[Dejando los bucles temprano con la declaración de descanso](#)

[Comprobar si el jugador ganó](#)

[Comprobando si el jugador perdió](#)

[Resumen](#)

4

UN PROGRAMA DE BROMAS

[Ejemplo de racha de chistes](#)

[Código fuente para chistes](#)

[Cómo funciona el código](#)

[Personajes de escape](#)

[Cotizaciones simples y dobles](#)

[El parámetro de palabra clave final de la función print\(\)](#)

[Resumen](#)

5

REINO DEL DRAGÓN

[Cómo jugar Reino del Dragón](#)

[Muestra Run of Dragon Realm](#)

[Diagrama de flujo para Dragon Realm](#)

[Código fuente para Dragon Realm](#)

[Importación de los módulos aleatorios y de tiempo](#)

[Funciones en Dragon Realm](#)

[Declaraciones def](#)

[Llamar a una función](#)

Dónde poner definiciones de funciones

Cadenas multilínea

Cómo hacer bucles con instrucciones while

Operadores booleanos

El operador y

El operador o

El no operador

Evaluación de operadores booleanos

Valores devueltos

Alcance Global y Alcance Local

Parámetros de función

Visualización de los resultados del juego

Decidir qué cueva tiene el dragón amistoso

El bucle del juego

Llamar a las funciones en el programa

Pedirle al jugador que juegue de nuevo

Resumen

6

USO DEL DEPURADOR

Tipos de errores

el depurador

Iniciar el depurador

Pasos a través del programa con el depurador

Encontrar el error

Establecimiento de puntos de interrupción

Uso de puntos de interrupción

Resumen

7

DISEÑO DEL AHORCADO CON DIAGRAMAS DE FLUJO

[Cómo jugar al ahorcado](#)

[Muestra de ejecución del ahorcado](#)

[Arte ASCII](#)

[Diseño de un programa con un diagrama de flujo](#)

[Creación del diagrama de flujo](#)

[Ramificación desde un cuadro de diálogo de flujo](#)

[Terminar o reiniciar el juego](#)

[Adivinando de nuevo](#)

[Ofreciendo retroalimentación al jugador](#)

[Resumen](#)

8

ESCRIBIENDO EL CÓDIGO DEL AHORCADO

[Código fuente de Hangman](#)

[Importando el módulo aleatorio](#)

[Variables constantes](#)

[El tipo de datos de listas](#)

[Acceso a elementos con índices](#)

[Concatenación de listas](#)

[El operador en](#)

[Métodos de llamada](#)

[Los métodos de lista reverse\(\) y append\(\)](#)

[El método de cadena split\(\)](#)

[Obtener una palabra secreta de la lista de palabras](#)

[Mostrar el tablero al jugador](#)

[Las funciones list\(\) y range\(\)](#)

[Segmentación de listas y cadenas](#)

Mostrar la palabra secreta con espacios en blanco

Obtener la suposición del jugador

Los métodos de cadena lower() y upper()

Saliendo del ciclo while

declaraciones elif

Asegurarse de que el jugador ingresó una suposición válida

Pedirle al jugador que juegue de nuevo

Revisión de las funciones del ahorcado

El bucle del juego

Llamar a la función displayBoard()

Permitir que el jugador ingrese su suposición

Comprobar si la letra está en la palabra secreta

Comprobar si el jugador ganó

Manejo de una suposición incorrecta

Comprobando si el jugador perdió

Terminar o reiniciar el juego

Resumen

9

Ahorcado extensible

Agregar más conjeturas

El tipo de datos del diccionario

Obtener el tamaño de los diccionarios con len()

La diferencia entre diccionarios y listas

Los métodos de diccionario keys() y values()

Uso de diccionarios de palabras en Hangman

Elegir aleatoriamente de una lista

Eliminación de elementos de las listas

Asignación Múltiple

[Impresión de la categoría de palabras para el reproductor](#)

[Resumen](#)

[10](#)

[TIC-TAC-TOE](#)

[Muestra de ejecución de Tic-Tac-Toe](#)

[Código fuente para Tic-Tac-Toe](#)

[Diseño del programa](#)

[Representando a la Junta como Datos](#)

[Elaboración de estrategias con la IA del juego](#)

[Importando el módulo aleatorio](#)

[Impresión del tablero en la pantalla](#)

[Permitir que el jugador elija X u O](#)

[Decidir quién va primero](#)

[Poner una marca en la pizarra](#)

[Lista de referencias](#)

[Usando Lista de Referencias en hacerJugada\(\)](#)

[Comprobar si el jugador ganó](#)

[Duplicación de los datos de la placa](#)

[Comprobar si un espacio en el tablero está libre](#)

[Permitir que el jugador ingrese un movimiento](#)

[Evaluación de cortocircuito](#)

[Elegir un movimiento de una lista de movimientos](#)

[El valor de ninguno](#)

[Creando la IA de la computadora](#)

[Comprobando si la computadora puede ganar en uno](#)

[Moverse](#)

[Comprobar si el jugador puede ganar en un solo movimiento](#)

[Comprobación de los espacios de las esquinas, el centro y los lados \(en](#)

esa orden)

Comprobando si el tablero está lleno

El bucle del juego

Elegir la marca del jugador y quién va primero

Ejecutar el turno del jugador

Ejecutar el turno de la computadora

Pedirle al jugador que juegue de nuevo

Resumen

11

EL JUEGO DE DEDUCCIÓN DE BAGELS

Ejemplo de ejecución de bagels

Código fuente para bagels

Diagrama de flujo para bagels

Importación aleatoria y definición de getSecretNum()

Barajar un conjunto único de dígitos

Cambiar el orden de los elementos de la lista con random.shuffle()

Función

Obtener el número secreto de los dígitos barajados

Operadores de asignación aumentada

Calculando las pistas para dar

El método de lista sort()

El método de cadena join()

Comprobando si una cadena tiene solo números

Comenzando el juego

Interpolación de cadenas

El bucle del juego

Obtener la suposición del jugador

Obtener las pistas para la conjetura del jugador

Comprobar si el jugador ganó o perdió

Pedirle al jugador que juegue de nuevo

Resumen

12

EL SISTEMA DE COORDENADAS CARTESIANAS

Grillas y Coordenadas Cartesianas

Números negativos

El sistema de coordenadas de una pantalla de computadora

trucos matematicos

Truco 1: Un signo menos se come el signo más a su izquierda

Truco 2: dos menos se combinan en un más

Truco 3: dos números que se agregan pueden intercambiarse

Lugares

Valores absolutos y la función abs()

Resumen

13

BÚSQUEDA DEL TESORO CON SONAR

Ejemplo de ejecución de la búsqueda del tesoro con sonar

Código fuente de Sonar Treasure Hunt

Diseño del programa

Importación de los módulos random, sys y math

Creación de un nuevo tablero de juego

Dibujar el tablero de juego

Dibujar las coordenadas X a lo largo de la parte superior de la

Junta

Dibujando el océano

Imprimir una fila en el océano

Dibujar las coordenadas X a lo largo de la parte inferior de la Junta

Crear los cofres del tesoro al azar

Determinar si un movimiento es válido

Colocar un movimiento en el tablero

Encontrar el cofre del tesoro más cercano

Eliminar valores con el método de lista remove()

Obtener el movimiento del jugador

Impresión de las instrucciones del juego para el jugador

El bucle del juego

Mostrar el estado del juego para el jugador

Manejar el movimiento del jugador Encontrar un

cofre del tesoro hundido Verificar si el jugador

ganó Verificar si el jugador perdió Terminar el

programa con la función sys.exit()

Resumen

14

CIFRA CÉSAR

Criptografía y Cifrado

Cómo funciona el cifrado César

Ejemplo de ejecución del cifrado César

Código fuente del cifrado César

Configuración de la longitud máxima de la clave

Decidir cifrar o descifrar el mensaje

Obtener el mensaje del jugador

Obtener la clave del reproductor

Cifrar o descifrar el mensaje

Encontrar cadenas pasadas con el método de cadena find()

Cifrar o descifrar cada letra

Inicio del programa

La técnica de la fuerza bruta

Agregar el modo de fuerza bruta

Resumen

15

EL JUEGO REVERSEGAM

Cómo jugar juego inverso

Ejemplo de ejecución de Reversegam

Código fuente para Reversegam

Importación de módulos y configuración de constantes

La estructura de datos del tablero de juego

Dibujar la estructura de datos de la placa en la pantalla

Creación de una nueva estructura de datos del tablero

Comprobar si un movimiento es válido

Comprobación de cada una de las ocho direcciones

Averiguar si hay mosaicos para voltear

Comprobación de coordenadas válidas

Obtener una lista con todos los movimientos válidos

Llamar a la función bool()

Obtener la puntuación del tablero de juego

Obtener la elección de mosaico del jugador

Determinar quién va primero

Colocar una ficha en el tablero

Copiar la estructura de datos del tablero

Determinar si un espacio está en una esquina

Obtener el movimiento del jugador

Obtener el movimiento de la computadora

Elaboración de estrategias con movimientos de esquina

Obtener una lista de los movimientos con mayor puntuación

Impresión de las partituras en la pantalla

Comenzando el juego

Comprobación de un punto muerto

Ejecutar el turno del jugador

Ejecutar el turno de la computadora

El bucle del juego

Pedirle al jugador que juegue de nuevo

Resumen

decisón

DEVOLVER SIMULACIÓN DE IA

Hacer que la computadora juegue contra sí misma

Ejemplo de ejecución de la simulación 1

Código fuente para la simulación 1

Eliminación de las indicaciones del reproductor y adición de una
jugador de la computadora

Hacer que la computadora juegue sola varias veces

Ejemplo de ejecución de la simulación 2

Código fuente para la simulación 2

Seguimiento de varios juegos

Comentando las llamadas a la función print()

Uso de porcentajes para calificar las IA

Comparación de diferentes algoritmos de IA

Código fuente para la simulación 3

Cómo funcionan las IA en Simulation 3

Comparando las IA

Resumen

17

CREAR GRÁFICOS

Instalando pygame

hola mundo en pygame

Ejemplo de ejecución de pygame Hello World

Código fuente de pygame Hello World

Importando el módulo pygame

Inicializando pygame

Configuración de la ventana de pygame

tuplas

Objetos de superficie

Configuración de variables de color

Escribir texto en la ventana de pygame

Uso de fuentes para aplicar estilo al texto

Representación de un objeto de fuente

Configuración de la ubicación del texto con atributos Rect

Rellenar un objeto de superficie con un color

Funciones de dibujo de pygame

Dibujar un polígono

Dibujar una línea

Dibujar un círculo

Dibujar una elipse

dibujar un rectángulo

Píxeles para colorear

El método blit() para objetos de superficie

Dibujar el objeto de superficie en la pantalla

Eventos y el Game Loop

[Obtención de objetos de eventos](#)

[Salir del programa](#)

[Resumen](#)

18

GRÁFICOS ANIMADOS

[Ejemplo de ejecución del programa de animación](#)

[Código Fuente del Programa de Animación](#)

[Mover y rebotar las cajas](#)

[Configuración de las variables constantes](#)

[Variables constantes para la dirección](#)

[Variables constantes para el color](#)

[Configuración de las estructuras de datos de Box](#)

[El bucle del juego](#)

[Manejo cuando el jugador sale](#)

[Mover cada caja](#)

[rebentar una caja](#)

[Dibujar las cajas en la ventana en su nuevo](#)

[Posiciones](#)

[Dibujar la ventana en la pantalla](#)

[Resumen](#)

19

DETECCIÓN DE COLISIONES

[Ejemplo de ejecución del programa de detección de colisiones](#)

[Código fuente para el programa de detección de colisiones](#)

[Importación de los módulos](#)

[Usar un reloj para marcar el ritmo del programa](#)

[Configurar la ventana y las estructuras de datos](#)

Configuración de variables para realizar un seguimiento del movimiento

Gestión de eventos

Manejo del evento KEYDOWN

Manejo del evento KEYUP

Teletransportar al jugador

Agregar nuevos cuadrados de alimentos

Mover el jugador alrededor de la ventana

Dibujar el jugador en la ventana

Comprobación de colisiones

Dibujar los cuadrados de comida en la ventana

Resumen

20

USO DE SONIDOS E IMÁGENES

Adición de imágenes con sprites

Archivos de imagen y sonido

Ejemplo de ejecución del programa Sprites and Sounds

Código fuente del programa Sprites and Sounds

Configuración de la ventana y la estructura de datos

Agregar un objeto

Cambiar el tamaño de un sprite

Configuración de la música y los sonidos

Adición de archivos de sonido

Activar y desactivar el sonido

Dibujar el jugador en la ventana

Comprobación de colisiones

Dibujar las cerezas en la ventana

Resumen

21

UN JUEGO DE DODGER CON SONIDOS E IMÁGENES

Revisión de los tipos de datos básicos de pygame

Ejemplo de ejecución de Dodger

Código fuente de Dodger

Importación de los módulos

Configuración de las variables constantes

Definición de funciones

Terminar y pausar el juego

Hacer un seguimiento de las colisiones de Baddie

Dibujar texto en la ventana

Inicializando pygame y configurando la ventana

Configuración de objetos de fuente, sonido e imagen

Visualización de la pantalla de inicio

Comenzando el juego

El bucle del juego

Manejo de eventos de teclado

Manejo del movimiento del ratón

Agregar nuevos malos

Mover el personaje del jugador y los malos

Implementando los códigos de trucos

Quitando a los malos

Dibujar la ventana

Dibujar la puntuación del jugador

Dibujar el personaje del jugador y los malos

Comprobación de colisiones

La pantalla de Game Over

Modificando el Juego de los Dodgers

Resumen

ÍNDICE

EXPRESIONES DE GRATITUD

Este libro no hubiera sido posible sin el trabajo excepcional del equipo de No Starch Press. Gracias a mi editor, Bill Pollock; gracias a mis editores, Laurel Chun, Jan Cash y Tyler Ortman, por su increíble ayuda durante todo el proceso; gracias a mi editor técnico Ari Lacenski por su exhaustiva revisión; y gracias a Josh Ellingson por otro

gran portada

INTRODUCCIÓN



Cuando jugué videojuegos por primera vez cuando era niño, me enganché. Pero no solo quería jugar videojuegos, quería crearlos. Encontré un libro como este que me enseñó a escribir mis primeros programas y juegos. Fue divertido y fácil. Los primeros juegos que hice fueron como los de este libro. No eran tan elegantes como los juegos de Nintendo que mis padres me compraron, pero eran juegos que yo mismo había creado.

Ahora, como adulto, todavía me divierte programar y me pagan por ello. Pero incluso si no quieres convertirte en un programador de computadoras, la programación es una habilidad útil y divertida de tener. Entrena su cerebro para pensar lógicamente, hacer planes y reconsiderar sus ideas siempre que encuentre errores en su código.

Muchos libros de programación para principiantes se dividen en dos categorías. La primera categoría incluye libros que no enseñan tanto programación como “software de creación de juegos” o lenguajes que simplifican tanto que lo que se enseña ya no es programación. La otra categoría consiste en libros que enseñan programación como un libro de texto de matemáticas: todos los principios y conceptos, con pocas aplicaciones de la vida real para el

lector. Este libro tiene un enfoque diferente y te enseña cómo programar haciendo videojuegos. Muestro el código fuente de los juegos desde el principio y explico los principios de programación de los ejemplos. Este enfoque fue la clave para mí cuando estaba aprendiendo a programar. Cuanto más aprendía cómo funcionaban los programas de otras personas, más ideas tenía para mis propios programas.

Todo lo que necesita es una computadora, un software gratuito llamado intérprete de Python y este libro. Una vez que aprenda a crear los juegos de este libro, podrá desarrollar juegos por su cuenta

Las computadoras son máquinas increíbles y aprender a programarlas no es tan difícil como la gente piensa. Un programa de computadora es un montón de instrucciones que la computadora puede entender, al igual que un libro de cuentos es un montón de oraciones que el lector puede entender. Para dar instrucciones a una computadora, escribe un programa en un lenguaje que la computadora entienda. Este libro le enseñará un lenguaje de programación llamado Python. Hay muchos otros lenguajes de programación que puede aprender, como BASIC, Java, JavaScript, PHP y C++.

Cuando era niño, aprendí BASIC, pero los lenguajes de programación más nuevos como Python son aún más fáciles de aprender. Python también es utilizado por programadores profesionales en su trabajo y cuando programan por diversión. Además, su instalación y uso son totalmente gratuitos: solo necesitará una conexión a Internet para descargar eso.

Debido a que los videojuegos no son más que programas de computadora, también se componen de instrucciones. Los juegos que creará a partir de este libro parecen simples en comparación con los juegos para Xbox, PlayStation o Nintendo. Estos juegos no tienen fantasía

gráficos porque están destinados a enseñarle los conceptos básicos de codificación. Son deliberadamente simples para que pueda concentrarse en aprender a programar. ¡Los juegos no tienen que ser complicados para ser divertidos!

¿PARA QUIÉN ES ESTE LIBRO?

La programación no es difícil, pero es difícil encontrar materiales que te enseñen a hacer cosas interesantes con la programación. Otros libros de informática tratan muchos temas que la mayoría de los codificadores nuevos no necesitan. Este libro te enseñará a programar tus propios juegos; ¡Aprenderás una habilidad útil y tendrás juegos divertidos para demostrarlo! Este libro es para:

- Principiantes completos que quieren aprender por sí mismos programación, incluso si no tienen experiencia previa.
- Niños y jóvenes que quieran aprender a programar creando juegos.
- Adultos y profesores que deseen enseñar programación a otros.
- Cualquier persona, joven o mayor, que quiera aprender a programar aprendiendo un lenguaje de programación profesional.

SOBRE ESTE LIBRO

En la mayoría de los capítulos de este libro, se presenta y explica un único proyecto de juego nuevo. Algunos de los capítulos cubren temas útiles adicionales, como la depuración. Los nuevos conceptos de programación se explican a medida que los juegos los utilizan, y los capítulos deben leerse en orden. Aquí hay un breve resumen de lo que encontrará en cada capítulo:

- Capítulo 1: El shell interactivo explica cómo Python

El shell interactivo se puede usar para experimentar con el código una línea a la vez.

- Capítulo 2: Escritura de programas cubre cómo escribir programas completos en el editor de archivos de Python.
- En el Capítulo 3: Adivina el número, programarás el primer juego del libro, Adivina el número, que le pide al jugador que adivine un número secreto y luego proporciona pistas sobre si la suposición es demasiado alta o demasiado baja.
- En el Capítulo 4: Un programa para contar chistes, escribirá un programa simple que le cuente al usuario varios chistes.
- En el Capítulo 5: Dragon Realm, programarás un juego de adivinanzas en el que el jugador debe elegir entre dos cuevas: una tiene un dragón amistoso y la otra tiene un dragón hambriento.
- Capítulo 6: Uso del Depurador cubre cómo usar el depurador para solucionar problemas en su código.
- Capítulo 7: Diseño del ahorcado con diagramas de flujo explica cómo se pueden usar los diagramas de flujo para planificar programas más largos, como el juego del ahorcado.
- En el Capítulo 8: Escribiendo el Código del Ahorcado, escribirá el juego del Ahorcado, siguiendo el diagrama de flujo del Capítulo 7.
- Capítulo 9: Extendiendo Hangman extiende el juego Hangman con nuevas funciones al hacer uso del tipo de datos de diccionario de Python.
- En el Capítulo 10: Tic-Tac-Toe, aprenderá a escribir un juego de Tic-Tac-Toe de humano contra computadora que utiliza inteligencia artificial.
- En el Capítulo 11: El juego de deducción de bagels, aprenderá

cómo hacer un juego de deducción llamado Bagels en el que el jugador debe adivinar números secretos basándose en pistas.

- En el Capítulo 12: El sistema de coordenadas cartesianas explica el sistema de coordenadas cartesianas, que usará más adelante. juegos.
- En el Capítulo 13: Sonar Treasure Hunt, aprenderá a escribir un juego de búsqueda de tesoros en el que el jugador busca en el océano cofres de tesoros perdidos.
- En el Capítulo 14: Cifrado César, creará un sencillo programa de cifrado que le permite escribir y decodificar secretos mensajes
- En el Capítulo 15: The Reversegam Game, programarás un juego avanzado tipo Reversi de humano contra computadora que tiene un oponente de inteligencia artificial casi imbatible.
- En el Capítulo 16: Simulación de IA de Reversegam amplía el Juego Reversegam en el Capítulo 15 para crear múltiples IA que compiten en juegos de computadora contra computadora.
- En el Capítulo 17: Creación de gráficos presenta el módulo pygame de Python y le muestra cómo usarlo para dibujar gráficos 2D.
- En el Capítulo 18: Animación de gráficos le muestra cómo Animar gráficos con pygame.
- En el Capítulo 19: Detección de colisiones, aprenderá a detectar cuándo los objetos chocan entre sí en juegos 2D.
- En el Capítulo 20: Uso de sonidos e imágenes, mejorará sus juegos simples de pygame agregando sonidos e imágenes.
- En el Capítulo 21: Un juego de Dodger con sonidos e imágenes combina los conceptos de los capítulos 17 a 20 para crear un juego animado llamado Dodger.

COMO USAR ESTE LIBRO

La mayoría de los capítulos de este libro comenzarán con una ejecución de muestra del programa destacado del capítulo. Esta ejecución de muestra le muestra cómo se ve el programa cuando lo ejecuta. Las partes que escribe el usuario se muestran en negrita.

Le recomiendo que ingrese el código de cada programa en el editor de archivos de IDLE usted mismo en lugar de descargarlo o copiarlo y pegarlo. Recordará más si se toma el tiempo de escribir el código.

Números de línea y sangría

Al escribir el código fuente de este libro, no escriba los números de línea al comienzo de cada línea. Por ejemplo, si vio la siguiente línea de código, no necesitaría escribir el 9. en el lado izquierdo, o el espacio inmediatamente siguiente:

```
9. numero = random.randint(1, 20)
```

Ingresarías solo esto:

```
numero = aleatorio.randint(1, 20)
```

Esos números están ahí solo para que este libro pueda referirse a líneas específicas en el programa. No son parte del código fuente del programa real.

Además de los números de línea, ingrese el código exactamente como aparece en este libro. Observe que algunas de las líneas de código tienen una sangría de cuatro u ocho (o más) espacios. Los espacios al comienzo de la línea cambian la forma en que Python interpreta las instrucciones, por lo que es muy importante incluirlos.

Veamos un ejemplo. Los espacios sangrados aquí son

marcados con círculos negros (•) para que pueda verlos.

```
while adivina < 10:  
    •••si el número == 42:  
        •••••imprimir('Hola')
```

La primera línea no tiene sangría, la segunda línea tiene una sangría de cuatro espacios y la tercera línea tiene una sangría de ocho espacios. Aunque los ejemplos de este libro no tienen círculos negros para marcar los espacios, cada carácter en IDLE tiene el mismo ancho, por lo que puede contar la cantidad de espacios contando la cantidad de caracteres en la línea superior o inferior.

Líneas de código largas

Algunas instrucciones de código son demasiado largas para caber en una línea del libro y se ajustarán a la línea siguiente. Pero la línea cabrá en la pantalla de su computadora, así que escríbala en una sola línea sin presionar ENTER. Puede saber cuándo comienza una nueva instrucción mirando los números de línea a la izquierda. Este ejemplo tiene solo dos instrucciones:

```
1. print('¡Esta es la primera instrucción!xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'  
      'xxxxxxxx') 2.  
print('Esta es la segunda instrucción, no la tercera.')
```

La primera instrucción se ajusta a una segunda línea en la página, pero la segunda línea no tiene un número de línea, por lo que puede ver que sigue siendo la línea 1 del código.

DESCARGA E INSTALACIÓN PITÓN

Deberá instalar un software llamado intérprete de Python.

El programa intérprete entiende las instrucciones que escribe en Python. Me referiré al software de interpretación de Python como Python de ahora en adelante.

En esta sección, le mostraré cómo descargar e instalar Python 3, específicamente, Python 3.4, para Windows, OS X o Ubuntu. Hay versiones de Python más nuevas que la 3.4, pero el módulo pygame , que se usa en los capítulos 17 a 21, actualmente solo admite hasta la 3.4.

Es importante saber que existen algunas diferencias significativas entre Python 2 y Python 3. Los programas de este libro usan Python 3 y obtendrá errores si intenta ejecutarlos con Python 2. De hecho, esto es muy importante, que he agregado un pingüino de dibujos animados para recordárselo.



En Windows, descargue el MSI de Windows x86-64 instalador de <https://www.python.org/downloads/release/python-344/> y luego haga doble clic en él. Es posible que deba ingresar la contraseña de administrador para su computadora. Siga las instrucciones que el instalador muestra en la pantalla para instalar Python, como se indica aquí:

1. Seleccione Instalar para todos los usuarios y luego haga clic en Siguiente.
2. Instálelo en la carpeta C:\Python34 haciendo clic en Siguiente.
3. Haga clic en Siguiente para omitir la sección Personalizar Python.

En OS X, descargue el instalador de Mac OS X de 64 bits/32 bits desde <https://www.python.org/downloads/release/python-344/> y luego haga doble clic en él. Siga las instrucciones del instalador que aparece en la pantalla para instalar Python, como se indica aquí:

1. Si recibe la advertencia "'Python.mpkg' no se puede abrir porque es de un desarrollador no identificado", mantenga presionada la tecla CONTROL mientras hace clic con el botón derecho en el archivo Python.mpkg y luego seleccione Abrir en el menú que aparece. Es posible que deba ingresar la contraseña de administrador para su computadora.
2. Haga clic en Continuar a través de la sección de Bienvenida y haga clic en Acepto para aceptar la licencia.
3. Seleccione Macintosh HD (o como se llame su disco duro) y haga clic en Instalar.

Si está ejecutando Ubuntu, puede instalar Python desde el Centro de software de Ubuntu siguiendo estos pasos:

1. Abra el Centro de software de Ubuntu.
2. Ingrese Python en el cuadro de búsqueda en la esquina superior derecha de la ventana.
3. Seleccione INACTIVO (Python 3.4 GUI de 64 bits).
4. Haga clic en Instalar. Es posible que deba ingresar la contraseña de administrador para completar la instalación.

Si los pasos anteriores no funcionan, puede encontrar una instalación alternativa de Python <https://www.nostarch.com/inventwithpython/>.

instrucciones a

ARRANQUE INACTIVO

IDLE significa Entorno de desarrollo interactivo .

IDLE es como un procesador de textos para escribir programas en Python.

Iniciar IDLE es diferente en cada sistema operativo:

- En Windows, haga clic en el menú Inicio en la esquina inferior izquierda de la pantalla, escriba IDLE y seleccione IDLE (GUI de Python).
- En OS X, abra Finder y haga clic en Aplicaciones. Haga doble clic Python 3.x y luego haga doble clic en el ícono IDLE.
- En Ubuntu u otras distribuciones de Linux, abra una ventana de terminal e ingrese `inactivo3`. También puede hacer clic en Aplicaciones en la parte superior de la pantalla. Luego haga clic en Programación e IDLE 3.

La ventana que aparece cuando ejecuta IDLE por primera vez es el shell interactivo, como se muestra en [la Figura 1](#). Puede ingresar instrucciones de Python en el shell interactivo en el indicador `>>>` y Python las ejecutará. Después de que la computadora realice las instrucciones, un nuevo mensaje `>>>` esperará su siguiente instrucción.



Figura 1: El shell interactivo del programa IDLE

ENCONTRAR AYUDA EN LÍNEA

Puede encontrar los archivos de código fuente y otros recursos para esto en <https://www.nostarch.com/inventwithpython/>. Si desea hacer preguntas de programación relacionadas con este libro, visite <https://reddit.com/r/inventwithpython/>, o puede enviarle sus preguntas sobre programación por correo electrónico a al@inventwithpython.com.

Antes de hacer cualquier pregunta, asegúrese de hacer lo siguiente:

- Si está escribiendo un programa en este libro pero recibe un error, busque errores tipográficos con la herramienta de comparación en línea en <https://www.nostarch.com/inventwithpython#diff> antes de hacer su pregunta. Copie y pegue su código en la herramienta diff para encontrar cualquier diferencia entre el código del libro y

tuyo.

- Busque en la web para ver si alguien más ya hizo (y respondió) su pregunta.

Tenga en cuenta que cuanto mejor formule sus preguntas de programación, mejor podrán ayudarlo los demás. Cuando haga preguntas de programación, haga lo siguiente:

- Explique lo que intenta hacer cuando recibe el error.

Esto le permitirá a tu ayudante saber si estás en el camino equivocado por completo.

- Copie y pegue todo el mensaje de error y su código.

- Proporcione su sistema operativo y versión.

- Explique lo que ya ha intentado hacer para resolver su problema. Esto les dice a las personas que ya has trabajado un poco para tratar de resolver las cosas por tu cuenta.

- Ser cortés. No exija ayuda ni presione a sus ayudantes para que respondan rápidamente.

Ahora que sabe cómo pedir ayuda, estará aprendiendo

¡Para programar tus propios juegos de computadora en poco tiempo!

1

LA CONCHA INTERACTIVA



Antes de poder crear juegos, debe aprender algunos conceptos básicos de programación. Comenzará en este capítulo aprendiendo cómo usar el shell interactivo de Python y realizar operaciones aritméticas básicas.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Operadores
- Números enteros y de punto flotante
- Valores
- Expresiones
- Errores de sintaxis
- Almacenamiento de valores en variables

ALGUNAS MATEMÁTICAS SIMPLES

Inicie IDLE siguiendo los pasos de “Inicio de IDLE” en la página xxvi.
Primero usará Python para resolver algunos problemas matemáticos simples. El caparazón interactivo puede funcionar como una calculadora. Escriba $2 + 2$ en el shell interactivo en el indicador `>>>` y presione

INGRESAR. (En algunos teclados, esta tecla es RETURN.) La figura 1-1 muestra cómo se ve este problema matemático en el shell interactivo: observe que responde con el número 4.

The screenshot shows a window titled "Python 3.4.4 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main area displays the Python interpreter's response to the input "2 + 2". The text in the window reads:

```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2 + 2
4
>>> |
```

In the bottom right corner of the window, it says "Ln: 5 Col: 4".

Figura 1-1: Ingresando $2 + 2$ en el shell interactivo

Este problema matemático es una simple instrucción de programación. El signo más (+) le dice a la computadora que sume los números 2 y 2. La computadora hace esto y responde con el número 4 en la siguiente línea. La Tabla 1-1 enumera los otros símbolos matemáticos disponibles en Python.

Tabla 1-1: Operadores matemáticos

Operador	Operación
+	Suma
-	Sustracción
*	Multiplicación
/	División

El signo menos (-) resta números, el asterisco (*) multiplica números y la barra inclinada (/) divide números. Cuando se usan de esta manera, +, -, * y / se denominan operadores. Operadores

decirle a Python qué hacer con los números que los rodean.

Números enteros y números de punto flotante Los números enteros (o enteros para abreviar) son números enteros como 4, 99 y 0. Los números de punto flotante (o flotantes para abreviar) son fracciones o números con puntos decimales como 3,5, 42,1 y 5,0. En Python, 5 es un número entero, pero 5.0 es un flotante. Estos números se llaman valores. (Más adelante aprenderemos sobre otros tipos de valores además de los números). En el problema de matemáticas que ingresaste en el shell, 2 y 2 son valores enteros.

Expresiones El

problema matemático $2 + 2$ es un ejemplo de una expresión. Como muestra la Figura 1-2 , las expresiones se componen de valores (los números) conectados por operadores (los signos matemáticos) que producen un nuevo valor que el código puede usar. Las computadoras pueden resolver millones de expresiones en segundos.

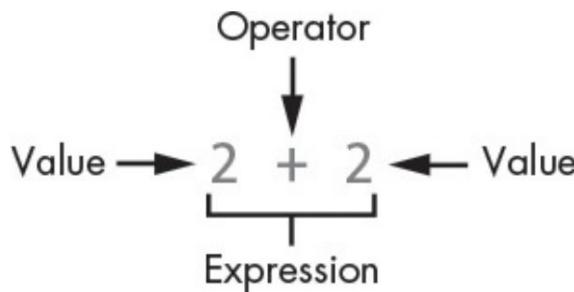


Figura 1-2: Una expresión se compone de valores y operadores.

Intente ingresar algunas de estas expresiones en el shell interactivo, presionando ENTER después de cada una:

```

>>> 2+2+2+2+2
10
>>> 8*6
48
>>> 10-5+6
  
```

```
11  
>>> 2 +      2  
4
```

Todas estas expresiones parecen ecuaciones matemáticas regulares, pero observe todos los espacios en el ejemplo 2 + 2. El Python ~~permite~~ ^{no} de espacios entre valores y operadores.

Sin embargo, siempre debe comenzar las instrucciones al principio de la línea (sin espacios) al ingresarlas en el shell interactivo.

EVALUACIÓN DE EXPRESIONES

Cuando una computadora resuelve la expresión $10 + 5$ y devuelve el valor 15, ha evaluado la expresión. Evaluar una expresión reduce la expresión a un solo valor, al igual que resolver un problema matemático reduce el problema a un solo número: la respuesta. Por ejemplo, las expresiones $10 + 5$ y $10 + 3 + 2$ ambos evalúan a 15.

Cuando Python evalúa una expresión, sigue un orden de operaciones tal como lo hace cuando hace matemáticas. Solo hay algunas reglas:

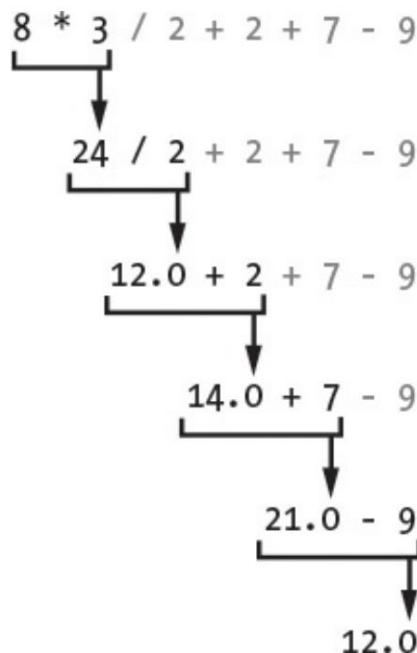
- Las partes de la expresión entre paréntesis se evalúan primero.
- La multiplicación y la división se hacen antes que la suma y la resta.
- La evaluación se realiza de izquierda a derecha.

La expresión $1 + 2 * 3 + 4$ da como resultado 11, no 13, porque $2 * 3$ se evalúa primero. Si la expresión fuera $(1 + 2) * (3 + 4)$ daría como resultado 21, porque el $(1 + 2)$ y $(3 + 4)$ dentro

los paréntesis se evalúan antes de la multiplicación.

Las expresiones pueden ser de cualquier tamaño, pero siempre se evaluarán como un único valor. Incluso los valores individuales son expresiones.

Por ejemplo, la expresión 15 se evalúa como el valor 15. La expresión $8 * 3 / 2 + 2 + 7 - 9$ se evaluará como el valor 12,0 a través de los siguientes pasos:



Aunque la computadora está realizando todos estos pasos, no los ve en el shell interactivo. El shell interactivo le muestra solo el resultado:

```
>>> 8 * 3 / 2 + 2 + 7 - 9
12.0
```

Tenga en cuenta que las expresiones con el operador de división / **siempre** se evalúan como flotantes; por ejemplo, $24 / 2$ se evalúa como 12.0. Las operaciones matemáticas con incluso un valor flotante también se evalúan como valores flotantes, por lo que $12.0 + 2$ se evalúa como 14,0.

ERRORES DE SINTAXIS

Si ingresa 5 + en el shell interactivo, recibirá el siguiente mensaje de error:

```
>>> 5 +  
Error de sintaxis: sintaxis invalida
```

Este error ocurrió porque 5 + no es una expresión.

Las expresiones tienen valores conectados por operadores, y el operador + espera un valor antes y después. Aparece un mensaje de error cuando falta un valor esperado.

SyntaxError significa que Python no entiende la instrucción porque la escribiste incorrectamente. La programación de computadoras no se trata solo de darle a la computadora instrucciones a seguir, sino también de saber cómo darle esas instrucciones correctamente.

Sin embargo, no se preocupe por cometer errores. Los errores no dañarán su computadora. Simplemente vuelva a escribir la instrucción correctamente en el shell interactivo en el siguiente mensaje >>> .

ALMACENAMIENTO DE VALORES EN VARIABLES

Cuando una expresión se evalúa como un valor, puede usar ese valor más tarde almacenándolo en una variable. Piense en una variable como un cuadro que puede contener un valor.

Una instrucción de asignación almacenará un valor dentro de una variable. Escriba un nombre para la variable, seguido del signo igual (=), que se denomina operador de asignación, y luego el valor para almacenar en la variable. Por ejemplo, ingrese lo siguiente en el shell interactivo:

```
>>> spam = 15  
>>>
```

El cuadro de la variable spam ahora almacena el valor 15, como se muestra en la Figura 1-3.

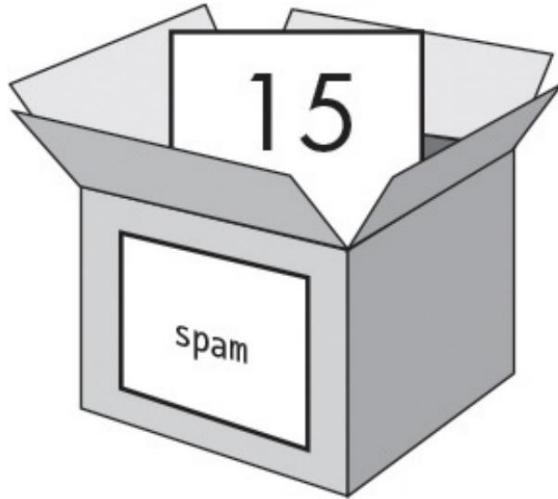


Figura 1-3: Las variables son como cajas que pueden contener valores.

Cuando presione ENTER, no verá nada en respuesta. En Python, sabe que la instrucción fue exitosa si no aparece ningún mensaje de error. Aparecerá el indicador >>> para que pueda ingresar la siguiente instrucción .

A diferencia de las expresiones, las sentencias son instrucciones que no se evalúan a ningún valor. Esta es la razón por la que no se muestra ningún valor en la línea siguiente en el shell interactivo después de `spam = 15`. Si está confundido acerca de qué instrucciones son expresiones y cuáles son declaraciones, recuerde que las expresiones se evalúan como un solo valor. Cualquier otro tipo de instrucción es una declaración.

Las variables almacenan valores, no expresiones. Por ejemplo, considere las expresiones en las sentencias `spam = 10 + 5` y `spam = 10 + 7 - 2`. Ambas se evalúan como 15. El resultado final es el mismo: ambas sentencias de asignación almacenan el valor 15 en la variable

correo no deseado.

Un buen nombre de variable describe los datos que contiene. Imagina que te mudaste a una casa nueva y etiquetaste todos tus

cosas de cajas de mudanza . ¡Nunca encontrarías nada! Los nombres de variables spam, huevos y tocino son ejemplos de nombres utilizados para variables en este libro.

La primera vez que se usa una variable en una declaración de asignación, Python creará esa variable. Para verificar qué valor hay en una variable, ingrese el nombre de la variable en el shell interactivo:

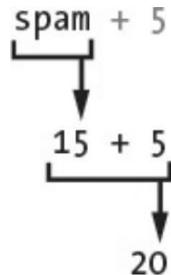
```
>>> spam = 15
>>> correo no deseado
15
```

La expresión spam evalúa el valor dentro del spam variables: 15.

También puede utilizar variables en expresiones. Intente ingresar lo siguiente en el shell interactivo:

```
>>> spam = 15
>>> spam + 5
```

Establece el valor de la variable spam en 15, por lo que escribir spam + 5 es como escribir la expresión 15 + 5. Estos son los pasos para evaluar spam + 5 :



No puede usar una variable antes de que una instrucción de asignación la cree. Si intenta hacerlo, Python le dará un NameError porque aún no existe tal variable con ese nombre. escribir mal

el nombre de la variable también provoca este error:

```
>>> correo no deseado = 15
>>> correo no deseado
Rastreo (llamadas recientes más última):
  Archivo "<pyshell#8>", línea 1, en <módulo>
    spma
NameError: el nombre 'spma' no está definido
```

El error apareció porque hay una variable de spam pero no una variable de spma .

Puede cambiar el valor almacenado en una variable ingresando otra instrucción de asignación. Por ejemplo, ingrese lo siguiente en el shell interactivo:

```
>>> correo no deseado = 15
>>> correo no deseado + 5
20
>>> correo no deseado = 3
>>> correo no deseado + 5
8
```

Cuando ingresa correo no deseado + 5 por primera vez, la expresión se evalúa como 20 porque almacenó 15 dentro del correo no deseado. Sin embargo, cuando ingresa spam = 3, el valor 15 en el cuadro de la variable se reemplaza o se sobrescribe con el valor 3 , ya que la variable solo puede contener un valor a la vez. Debido a que el valor del spam ahora es 3, cuando ingresa spam + 5, la expresión se evalúa como 8. Sobrescribir es como sacar un valor del cuadro de la variable para poner un nuevo valor, como se muestra en la Figura 1-4.

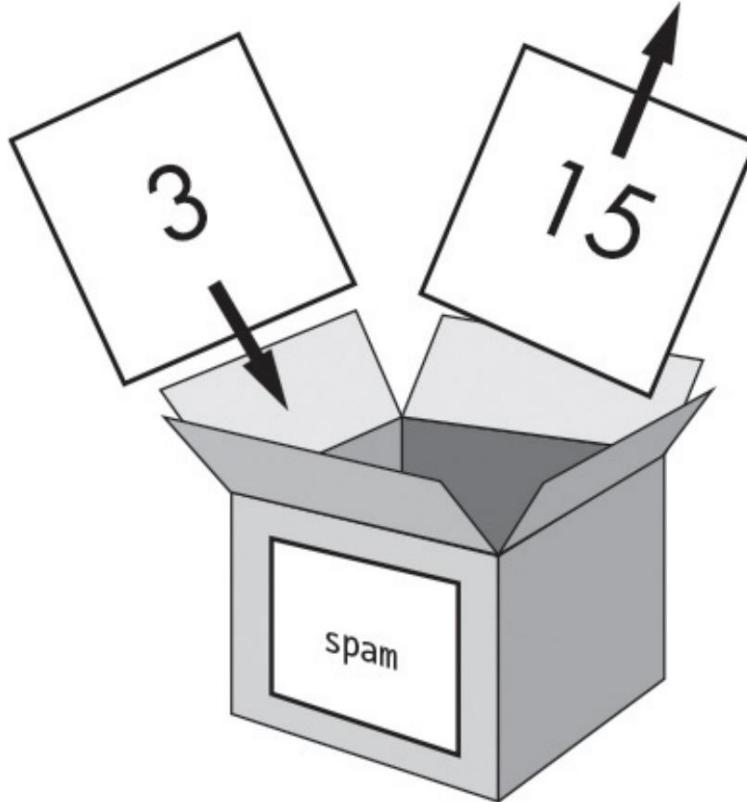


Figura 1-4: El valor 15 en spam se sobrescribe con el valor 3.

Incluso puede usar el valor en la variable spam para asignar un nuevo valor para spam:

```
>>> spam = 15
>>> spam = spam + 5
```

La instrucción de asignación `spam = spam + 5` dice: "El nuevo valor de la variable spam será el valor actual de spam más cinco". Para seguir aumentando el valor en el correo no deseado por 5 varias veces, ingrese lo siguiente en el shell interactivo:

```
>>> correo basura = 15
>>> correo basura = correo basura +
5 >>> correo basura = correo basura
+ 5 >>> correo basura = correo basura
+ 5 >>> correo basura
```

En este ejemplo, le asigna a spam un valor de 15 en la primera declaración. En la siguiente declaración, agrega 5 al valor de spam y asigna a spam el nuevo valor `spam + 5`, que se evalúa como 20.

Cuando haces esto tres veces, el spam se evalúa como 30.

Hasta ahora hemos visto una sola variable, pero puede crear tantas variables como necesite en sus programas. Por ejemplo, asignemos diferentes valores a dos variables llamadas huevos y tocino, así:

```
>>> tocino = 10  
>>> huevos = 15
```

Ahora la variable tocino tiene 10 dentro, y la variable huevos tiene 15 adentro. Cada variable es su propia caja con su propio valor, como se muestra en la Figura 1-5.

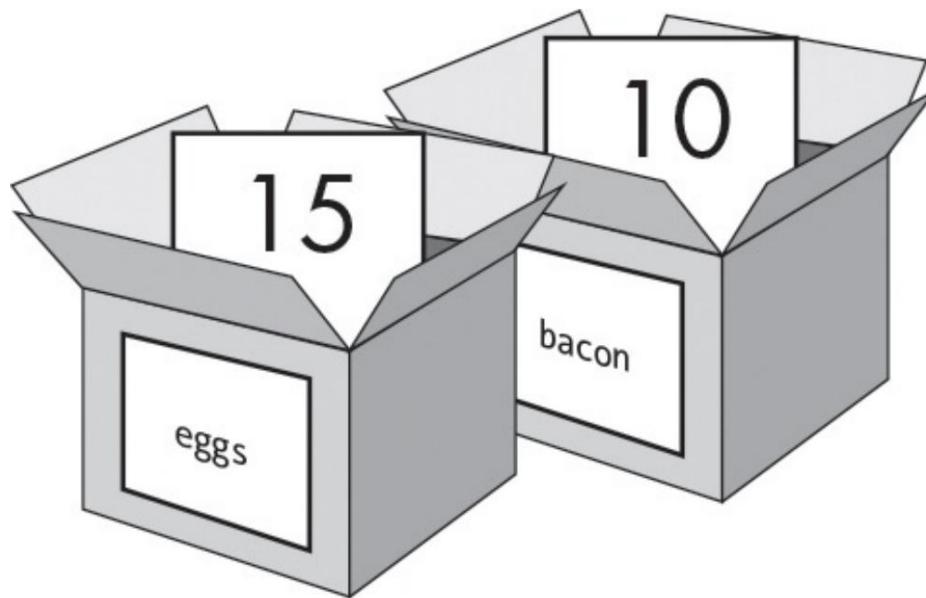


Figura 1-5: Las variables de tocino y huevos que cada tienda valora.

Ingrese `spam = tocino + huevos` en el caparazón interactivo, luego verifique el nuevo valor del spam:

```
>>> tocino = 10
>>> huevos =
15 >>> spam = tocino +
huevos >>> spam 25
```

El valor en spam ahora es 25. Cuando agrega tocino y huevos, está agregando sus valores, que son 10 y 15, respectivamente.

Las variables contienen valores, no expresiones, por lo que a la variable de spam se le asignó el valor 25, no la expresión tocino + huevos. Después de que la instrucción spam = tocino + huevos asigne el valor 25 al correo no deseado, el cambio de tocino o huevos no afectará el correo no deseado.

RESUMEN

En este capítulo, aprendió los conceptos básicos para escribir instrucciones de Python. Debido a que las computadoras no tienen sentido común y solo entienden instrucciones específicas, Python necesita que le digas exactamente qué hacer.

Las expresiones son valores (como 2 o 5) combinados con operadores (como + o -). Python puede evaluar expresiones, es decir, reducir la expresión a un solo valor. Puede almacenar valores dentro de las variables para que su programa pueda recordar esos valores y usarlos más tarde.

Hay algunos otros tipos de operadores y valores en Python. En el próximo capítulo, repasará algunos conceptos básicos más y escribirá su primer programa. Aprenderá a trabajar con texto en expresiones. Python no se limita solo a números; es más que una calculadora!

2

PROGRAMAS DE ESCRITURA



Ahora veamos qué puede hacer Python con el texto. Casi todos los programas muestran texto al usuario, y el usuario ingresa texto en los programas a través del teclado. En este capítulo, creará su primer programa, que hace ambas cosas. Aprenderá a almacenar texto en variables, combinar texto y mostrar texto en la pantalla. El programa que creará muestra el saludo ¡Hola, mundo! y pide el nombre del usuario.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Cuerdas
- Concatenación de cadenas
- Tipos de datos (como cadenas o enteros)
- Uso del editor de archivos para escribir programas
- Guardar y ejecutar programas en IDLE
- Flujo de ejecución
- Comentarios
- La función imprimir()
- La función de entrada()

• Distinción de mayúsculas y minúsculas

VALORES DE CADENA

En Python, los valores de texto se denominan cadenas. Los valores de cadena se pueden usar como valores enteros o flotantes. Puede almacenar cadenas en variables. En el código, los valores de cadena comienzan y terminan con una comilla simple, '. Ingrese este código en el shell interactivo:

```
>>> spam = 'hola'
```

Las comillas simples le dicen a Python dónde comienza y termina la cadena. No forman parte del texto del valor de cadena. Ahora, si ingresa correo no deseado en el shell interactivo, verá el contenido de la variable de correo no deseado . Recuerde, Python evalúa las variables como el valor almacenado dentro de la variable. En este caso, esta es la cadena.

'Hola'.

```
>>> spam = 'hola'  
>>> spam  
'hola'
```

Las cadenas pueden tener cualquier carácter de teclado y pueden ser tan largo como quieras. Estos son todos ejemplos de cadenas:

```
'Hola'  
'¡Hola!'  
'GATITOS'  
'7 manzanas, 14 naranjas, 3 limones'  
Todo lo que no tenga que ver con los elefantes es irrelevante.  
'Hace mucho tiempo, en una galaxia muy, muy lejana...'  
'O*%&#wY%*&OCfsdYO*&gfC%YO*&%3yc8r2'
```

CONCATENACIÓN DE CADENA

Puede combinar valores de cadena con operadores para crear expresiones, tal como lo hizo con valores enteros y flotantes.

Cuando combina dos cadenas con el operador + , se llama concatenación de cadenas. Ingrese 'Hola' + '¡Mundo!' en el interactivo caparazón:

```
>>> 'Hola' + '¡Mundo!'
'¡Hola Mundo!'
```

La expresión se evalúa como un único valor de cadena, 'HelloWorld!'. No hay espacio entre las palabras porque no había espacio en ninguna de las dos cadenas concatenadas, a diferencia de este ejemplo:

```
>>> 'Hola'      + '¡Mundo!'
'¡Hola Mundo!'
```

El operador + funciona de manera diferente en valores enteros y de cadena porque son tipos de datos diferentes. Todos los valores tienen un tipo de datos. El tipo de datos del valor 'Hola' es una cadena. El tipo de datos del valor 5 es un número entero. El tipo de datos le dice a Python qué deben hacer los operadores al evaluar expresiones. El operador + concatena valores de cadena, pero agrega valores enteros y flotantes.

PROGRAMAS DE ESCRITURA EN IDLE'S EDITOR DE ARCHIVOS

Hasta ahora, ha estado escribiendo instrucciones en el shell interactivo de IDLE una a la vez. Sin embargo, cuando escribe programas, ingresa varias instrucciones y hace que se ejecuten todas a la vez, y esto es lo que hará a continuación. ¡Es hora de escribir tu primer programa!

Además del intérprete, IDLE tiene otra parte llamada

el editor de archivos. Para abrirlo, haga clic en el menú Archivo en la parte superior del caparazón interactivo. Luego seleccione Nueva ventana si está usando Windows o Nuevo archivo si está usando OS X. Aparecerá una ventana en blanco para que escriba el código de su programa, como se muestra en la Figura 2-1.



Figura 2-1: El editor de archivos (izquierda) y el shell interactivo (derecha)

Las dos ventanas se ven similares, pero recuerde esto: el shell interactivo tendrá el indicador `>>>`, mientras que el editor de archivos no lo hará

Creación del programa Hello World Es tradicional que los programadores hagan que su primer programa muestre Hello world! en la pantalla. Ahora creará su propio programa Hello World.



Cuando ingrese su programa, recuerde no ingresar los números al comienzo de cada línea de código. Están allí para que este libro pueda hacer referencia al código por número de línea. La esquina inferior derecha del editor de archivos le indicará dónde está el cursor parpadeante para que pueda verificar en qué línea de código se encuentra.

La Figura 2-2 muestra que el cursor está en la línea 1 (subiendo y bajando por el editor) y la columna 0 (hacia la izquierda y la derecha).

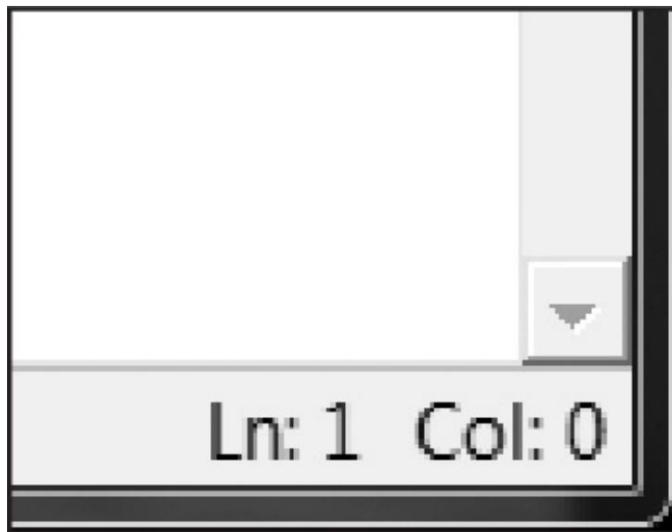


Figura 2-2: La parte inferior derecha del editor de archivos le indica en qué línea está el cursor.

Ingrese el siguiente texto en la nueva ventana del editor de archivos. Este es el código fuente del programa. Contiene las instrucciones que Python seguirá cuando se ejecute el programa.

hola.py

```
1. # Este programa me saluda y me pide mi nombre. 2.  
print('¡Hola mundo!') 3. print('¿Cuál es tu nombre?') 4.  
myName = input() 5. print('Es un placer conocerte,
```

```
' + miNombre)
```

IDLE escribirá diferentes tipos de instrucciones con diferentes colores. Una vez que haya terminado de escribir el código, la ventana debería parecerse a la Figura 2-3.

```

# This program says hello and asks for my name.
print('Hello world!')
print('What is your name?')
myName = input()
print('It is good to meet you, ' + myName)

```

Figura 2-3: El editor de archivos se verá así después de ingresar su código.

Verifique para asegurarse de que su ventana INACTIVA tenga el mismo aspecto.

Guardar su programa Una vez que

haya ingresado su código fuente, guárdelo haciendo clic en Archivo Guardar como. O

- ▶ presione CTRL-S para guardar con un atajo de teclado.

La Figura 2-4 muestra la ventana Guardar como que se abrirá. Ingrese hello.py en el campo de texto Nombre de archivo y luego haga clic en Guardar.

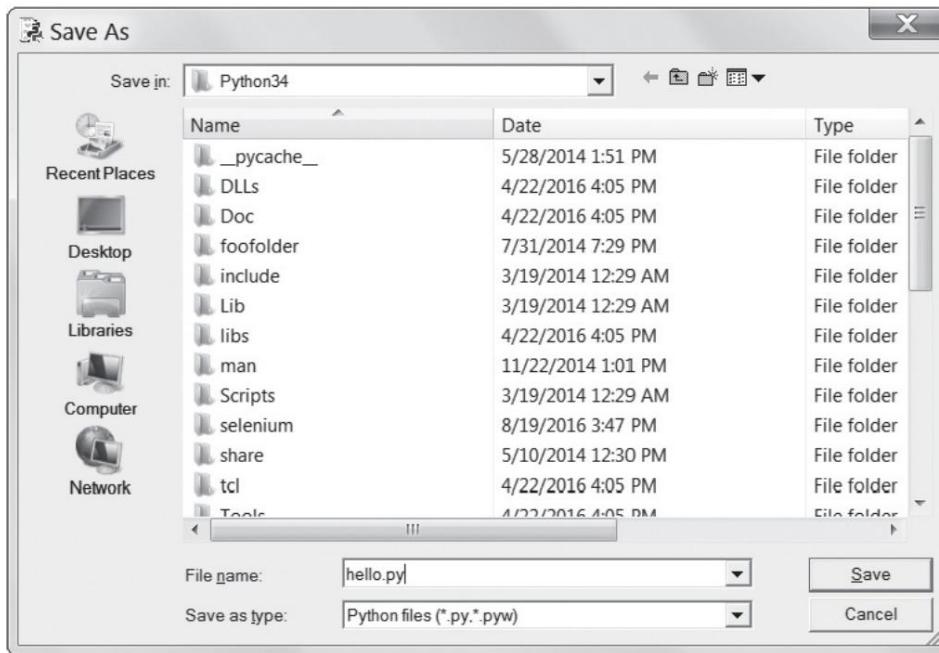


Figura 2-4: Guardar el programa

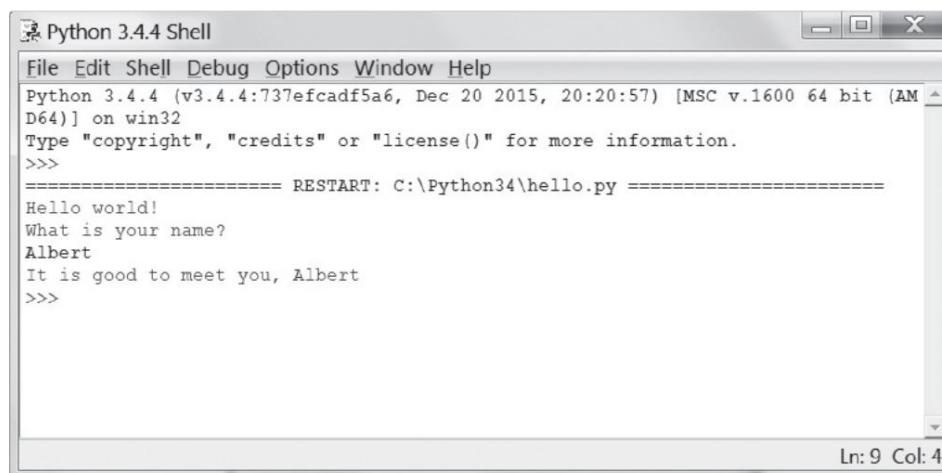
Debe guardar sus programas a menudo mientras los escribe. De esa manera, si la computadora falla o usted accidentalmente

salir de IDLE, no perderá mucho trabajo.

Para cargar su programa previamente guardado, haga clic en Archivo Abrir. Seleccione el archivo hello.py en la ventana que aparece y haga clic en el botón Abrir . Su programa hello.py guardado se abrirá en el editor de archivos.

Ejecutando su programa Ahora es el momento de ejecutar el programa. Haga clic en Archivo Ejecutar módulo. O simplemente presione F5 desde el editor de archivos (FN-5 en OS X). Su programa se ejecutará en el shell interactivo.

Introduce tu nombre cuando el programa te lo pida. Esto se parecerá a la Figura 2-5.



The screenshot shows the Python 3.4.4 Shell window. The title bar reads "Python 3.4.4 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The main window displays the following text:

```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
=====
RESTART: C:\Python34\hello.py =====
Hello world!
What is your name?
Albert
It is good to meet you, Albert
>>>
```

The status bar at the bottom right shows "Ln: 9 Col: 4".

Figura 2-5: El shell interactivo después de ejecutar hello.py

Cuando escriba su nombre y presione ENTER, el programa lo saludará por su nombre. ¡Felicidades! Has escrito tu primer programa y ahora eres programador de computadoras. Presione F5 nuevamente para ejecutar el programa por segunda vez e ingrese otro nombre.

Si tienes un error, compara tu código con el código de este libro con la en línea diferencia herramienta a <https://www.nostarch.com/inventwithpython#diff>. Copiar y

pegue su código del editor de archivos en la página web y haga clic en el botón Comparar . Esta herramienta resaltará cualquier diferencia entre su código y el código de este libro, como se muestra en la Figura 2-6.

Mientras codifica, si obtiene un NameError similar al siguiente, eso significa que está usando Python 2 en lugar de Python 3.

¡Hola Mundo!

¿Cómo te llamas?

Alberto

Rastreo (última llamada más reciente):

Archivo "C:/Python26/test1.py", línea 4, en <módulo>

miNombre = entrada()

Archivo "<cadena>", línea 1, en <módulo>

NameError: el nombre 'Albert' no está definido

Para solucionar el problema, instale Python 3.4 y vuelva a ejecutar el programa. (Consulte “[Descarga e instalación de Python](#)” en la página [xxv](#)).

Diff Tool

The diff tool can help you find typos in your code by showing you the differences between your program and the programs in the book.

Select program: Copy and paste your code here:

```
AlSim1.py
AlSim2.py
AlSim3.py
animation.py
bagels.py
cipher.py
collisionDetection.py
dodger.py
dragon.py
favorite.py
guess.py
hangman.py
hello.py
jokes.py
pygameHelloWorld.py
pygameInput.py
```

The Book's Program

```
1 # This program says hello and asks for my name.
2 print('Hello world!')
3 print('What is your name?')
4 myName = input()
5 print('It is good to meet you, ' + myName)
6
```

Your Program

```
1 # This program says hello and asks for my name.
2 print('Hello world!')
3 print('What is your name?')
4 myName = input()
5 print('It is good to meet you, ' + myName)
6
```

diff view generated by jsdifflib

Figura 2-6: Uso de la herramienta diff en <https://www.nostarch.com/inventwithpython#diff>

CÓMO HOLA MUNDO OBRAS DEL PROGRAMA

Cada línea de código es una instrucción interpretada por Python.

Estas instrucciones conforman el programa. Las instrucciones de un programa de computadora son como los pasos de una receta. Python completa cada instrucción en orden, comenzando desde la parte superior del programa y moviéndose hacia abajo.

El paso en el que Python está trabajando actualmente en el programa se llama ejecución. Cuando se inicia el programa, la ejecución es en la primera instrucción. Después de ejecutar la instrucción, Python baja a la siguiente instrucción.

Miremos cada línea de código para ver qué está haciendo.

Comenzaremos con la línea número 1.

Comentarios para el programador

La primera línea del programa Hello World es un comentario:

1. # Este programa me saluda y me pide mi nombre.

Cualquier texto que siga a una almohadilla (#) es un comentario. Los comentarios son las notas del programador sobre lo que hace el código; no están escritos para Python sino para ti, el programador. Python ignora los comentarios cuando ejecuta un programa. Los programadores suelen poner un comentario en la parte superior de su código para darle un título a su programa. El comentario en el programa Hola Mundo te dice que el programa te saluda y te pregunta tu nombre.

Funciones: Miniprogramas Programas internos

Una función es como un miniprograma dentro de su programa que contiene varias instrucciones para que Python las ejecute. Lo bueno de las funciones es que solo necesitas saber qué hacen, no cómo lo hacen. Python ya proporciona algunas funciones integradas. Usamos print() e input() en Hello World programa.

Una llamada de función es una instrucción que le dice a Python que ejecute el código dentro de una función. Por ejemplo, su programa llama a la función imprimir() para mostrar una cadena en la pantalla. La función print() toma la cadena que escribe entre paréntesis como entrada y muestra ese texto en la pantalla.

La función imprimir()

Las líneas 2 y 3 del programa Hello World son llamadas a print():

```
2. print('¡Hola mundo!') 3.  
print('¿Cuál es tu nombre?')
```

Un valor entre paréntesis en una llamada de función es un argumento. El argumento en la llamada a la función print() de la línea 2 es '¡Hola mundo!', y el argumento en la llamada a la función print() de la línea 3 es '¿Cuál es tu nombre?'. Esto se llama pasar el argumento a la función.

La función de entrada ()

La línea 4 es una declaración de asignación con una variable, miNombre, y una llamada de función, input():

```
4. miNombre = entrada()
```

Cuando se llama a input() , el programa espera a que el usuario ingrese texto. La cadena de texto que ingresa el usuario se convierte en el

valor al que se evalúa la llamada de función. Las llamadas a funciones pueden ser usadas en expresiones donde se puede usar un valor.

El valor al que se evalúa la llamada a la función se llama valor de retorno (De hecho, “el valor que devuelve una llamada de función” significa lo mismo que “el valor que devuelve una llamada de función”). En este caso, el valor de retorno de la función `input()` es la cadena que el usuario ingresó: su nombre. Si el usuario ingresa `Albert`, el

La llamada a la función `input()` se evalúa como la cadena '`Albert`'. La evaluación se ve así:

```
myName = input()  
          ↓  
myName = 'Albert'
```

Así es como el valor de cadena '`Albert`' se almacena en `myName` variable.

Expresiones en llamadas a funciones

La última línea del programa Hello World es otro `print()`

Llamada de función:

```
5. print('Encantado de conocerte, ' + miNombre)
```

La expresión '`Encantado de conocerte, ' + myName` está entre paréntesis de `print()`. Debido a que los argumentos son siempre valores únicos, Python primero evaluará esta expresión y luego pasará ese valor como argumento. Si '`Albert`' está almacenado en `myName`, la evaluación se ve así:

```
print('It is good to meet you, ' + myName)
      ↓
print('It is good to meet you, ' + 'Albert')
      ↓
print('It is good to meet you, Albert')
```

Así es como el programa saluda al usuario por su nombre.

El final del programa Una vez que

el programa ejecuta la última línea, termina o sale.

Esto significa que el programa deja de ejecutarse. Python olvida todos los valores almacenados en las variables, incluida la cadena almacenada en myName. Si vuelve a ejecutar el programa e ingresa un nombre diferente, el programa pensará que ese es su nombre:

¡Hola Mundo!

¿Cómo te llamas?

Carolyn

Es un placer conocerte, Carolyn

Recuerde, la computadora hace exactamente lo que usted programa para que haga. Las computadoras son tontas y solo siguen exactamente las instrucciones que les das. A la computadora no le importa si escribes tu nombre, el nombre de otra persona o alguna tontería. Escribe lo que quieras. La computadora lo tratará de la misma manera:

¡Hola Mundo!

¿Cómo te llamas?

caca

es bueno conocerte, caca

NOMBRAR VARIABLES

Dar nombres descriptivos a las variables hace que sea más fácil

entender lo que hace un programa. Podría haber llamado a la variable myName abrahamLincoln o nAmE, y Python habría ejecutado el programa de la misma manera. Pero esos nombres en realidad no dicen mucho sobre qué información podría contener la variable.

Como se discutió en el Capítulo 1, si se mudara a una nueva casa y etiquetara cada caja de mudanza como Cosas, ¡eso no sería útil en absoluto! Los ejemplos de conchas interactivas de este libro usan nombres de variables como spam, huevos y tocino porque la variable

los nombres en estos ejemplos no importan. Sin embargo, todos los programas de este libro usan nombres descriptivos, por lo que su programas

Los nombres de variables distinguen entre mayúsculas y minúsculas, lo que significa lo mismo el nombre de la variable en un caso diferente se considera diferente variable. Entonces spam, SPAM, Spam y sPAM son cuatro diferentes Variables en Python. Cada uno contiene sus propios valores separados. Es una mala idea tener variables en mayúsculas y minúsculas en su programa. En su lugar, utilice nombres descriptivos para sus variables.

Los nombres de las variables suelen estar en minúsculas. Si hay más de una palabra en el nombre de la variable, es una buena idea poner cada palabra en mayúscula después de la primera. Por ejemplo, el nombre de la variable lo que he desayunado **esta mañana** es mucho más fácil de leer que lo que he desayunado esta mañana. Poner en mayúsculas las variables de esta manera se llama camel case (porque se parece a las jorobas en el lomo de un camello) y hace que el código sea más legible.

Los programadores también prefieren usar nombres de variables más cortos para que el código sea más fácil de entender: desayuno o comida Esta mañana es más legible que whatIHadForBreakfastThisMorning. Estos son convenciones: formas opcionales pero estándar de hacer las cosas en la programación de Python.

RESUMEN

Una vez que comprenda cómo usar cadenas y funciones, puede comenzar a crear programas que interactúen con los usuarios. Esto es importante porque el texto es la forma principal en que el usuario y la computadora se comunicarán entre sí. El usuario ingresa texto a través del teclado con la función `input()` y la computadora muestra el texto en la pantalla con la función `print()`.

Las cadenas son solo valores de un nuevo tipo de datos. Todos los valores tienen un tipo de datos, y el tipo de datos de un valor afecta cómo funciona el operador `+`.

Las funciones se utilizan para llevar a cabo instrucciones complicadas en su programa. Python tiene muchas funciones integradas que aprenderá en este libro. Las llamadas a funciones se pueden utilizar en expresiones dondequiera que se use un valor.

La instrucción o paso en su programa donde Python está trabajando actualmente se llama ejecución. En el Capítulo 3, aprenderá más sobre cómo hacer que la ejecución se mueva de otra manera que simplemente hacia abajo en el programa. Una vez que aprendas esto, ¡estarás listo para crear juegos!

3

ADIVINA EL NÚMERO



En este capítulo, vas a hacer un juego de Adivina el Número. La computadora pensará en un número secreto del 1 al 20 y le pedirá al usuario que lo adivine. Después de cada intento, la computadora le dirá al usuario si el número es demasiado alto o demasiado bajo. El usuario gana si puede adivinar el número en seis intentos.

Este es un buen juego para codificar porque cubre muchos conceptos de programación en un programa corto. Aprenderá cómo convertir valores a diferentes tipos de datos y cuándo necesitaría hacerlo. Como este programa es un juego, de ahora en adelante llamaremos jugador al usuario.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- declaraciones de importación
- Módulos
- La función randint()
- para declaraciones
- Bloques
- Las funciones str(), int() y float()
- Booleanos

- Operadores de comparación
- Condiciones
- La diferencia entre = y ==
- declaraciones if
- romper declaraciones

EJECUCIÓN DE MUESTRA DE GUESS THE NÚMERO

Así es como se ve el programa Guess the Number para el jugador cuando se ejecuta. La entrada del jugador está marcada en negrita.

¡Hola! ¿Cómo te llamas?
Alberto

Bueno, Albert, estoy pensando en un número entre 1 y 20.

Adivina.

10

Su conjetura es demasiado alta.

Adivina.

2

Su conjetura es demasiado baja.

Adivina.

4

¡Buen trabajo, Alberto! ¡Adivinaste mi número en 3 intentos!

CÓDIGO FUENTE PARA ADIVINAR EL NÚMERO

Abra una nueva ventana del editor de archivos haciendo clic en Archivo Nueva ventana. En la ventana en blanco que aparece, ingresa el código fuente y guárdalo como guess.py. Luego ejecute el programa presionando F5.



Cuando ingrese este código en el editor de archivos, asegúrese de prestar atención al espacio al principio de las líneas. Algunas líneas necesitan una sangría de cuatro u ocho espacios.

Si obtiene errores después de ingresar este código, compare el código que escribió con el código del libro con la herramienta de comparación en línea en <https://www.nostarch.com/inventwithpython#diff>.

supongo.py

```
1. # Este es un juego de Adivina el Número. 2.  
importar al azar 3.  
  
4. conjeturas tomadas = 0  
5.  
6. print('¡Hola! ¿Cuál es tu nombre?') 7.  
myName = input()  
8.  
9. número = random.randint(1, 20) 10.  
print('Bueno, ' + miNombre + ', tu número es ' + str(número) + '.')  
11  
12. para conjeturasTomadas en el rango  
(6): 13. print('Haz una conjetura.') # Cuatro espacios delante de "imprimir"  
14. conjetura = entrada() 15. conjetura = int(conjetura)
```

```

diecisés.

17. if adivinar < número: 18.
imprimir('Tu suposición es' + 'Demasiado Baja.') # Ocho espacios delante de "imprimir"
19

20. if adivinar > número:
21     print('Tu suposición es demasiado alta.')
22

23. si adivinar == número:
24     descanso
25

26. if adivinar == número:
27. conjeturasTomadas = str(conjeturasTomadas +
1) 28. print('¡Buen trabajo, ' + Adivinaste el Número en ' conjeturasTomadas +      +
    ' ¡conjeturas!')
29

30. if adivinar != número:
31. número = str(número) 32.
print('Nop. El número en el que estaba pensando era' + ' ' + número + '.')

```

IMPORTAR EL ALEATORIO MÓDULO

Echemos un vistazo a las dos primeras líneas de este programa:

-
1. # Este es un juego de Adivina el Número.
 2. importar al azar
-

La primera línea es un comentario, que vio en el Capítulo 2.
 Recuerda que Python ignorará todo lo que esté después del carácter # . El comentario aquí solo nos recuerda lo que hace este programa.

La segunda línea es una declaración de importación . Recordar, Las sentencias son instrucciones que realizan alguna acción pero no se evalúan a un valor como lo hacen las expresiones. Ya ha visto la instrucción de asignación, que almacena un valor en una variable.

Si bien Python incluye muchas funciones integradas, algunas funciones están escritas en programas separados llamados módulos. Puede usar estas funciones importando sus módulos a su programa con una declaración de importación .

La línea 2 importa el módulo aleatorio para que el programa pueda llamar a la función randint() . Esta función generará un número aleatorio para que el jugador lo adivine.

Ahora que ha importado el módulo aleatorio , necesita configurar algunas variables para almacenar valores que su programa usará más adelante.

La línea 4 crea una nueva variable llamada conjeturas realizadas:

4. conjeturas tomadas = 0

Almacenarás el número de conjeturas que el jugador ha hecho en esta variable. Dado que el jugador no ha adivinado nada en este punto del programa, almacene el número entero 0 aquí.

6. print('¡Hola! ¿Cuál es tu nombre?') 7.
myName = input()

Las líneas 6 y 7 son las mismas que las líneas de Hello World. en el Capítulo 2. Los programadores a menudo reutilizan código de otros programas para ahorrarse trabajo.

La línea 6 es una llamada de función para imprimir(). Recuerda que una función es como un miniprograma dentro de tu programa. Cuando su programa llama a una función, ejecuta este miniprograma. El código dentro de print() muestra el argumento de cadena que le pasó en el pantalla.

La línea 7 le permite al jugador ingresar su nombre y lo almacena en la variable myName . Recuerde, es posible que la cadena no sea realmente la

nombre del jugador; es simplemente cualquier cadena que el jugador escriba.

Las computadoras son tontas y siguen sus instrucciones, pase lo que pase.

GENERANDO NÚMEROS ALEATORIOS CON RANDOM.RANDINT() FUNCIÓN

Ahora que sus otras variables están configuradas, puede usar la función del módulo aleatorio para configurar el número secreto de la computadora:

```
9. numero = random.randint(1, 20)
```

La línea 9 llama a una nueva función llamada `randint()` y almacena el valor de retorno en número. Recuerde, las llamadas a funciones pueden ser parte de expresiones porque se evalúan como un valor.

El módulo aleatorio proporciona la función `randint()`, por lo que debe llamarlo con `random.randint()` (¡no olvide el punto!) para decirle a Python que la función `randint()` está en el módulo aleatorio .

`randint()` devolverá un número entero aleatorio entre (e incluyendo) los dos argumentos enteros que le pasa. La línea 9 pasa 1 y 20, separados por comas, entre paréntesis que siguen al nombre de la función. El entero aleatorio que `randint()` devuelve se almacena en una variable llamada número: este es el secreto número que el jugador está tratando de adivinar.

Solo por un momento, regrese al shell interactivo e ingrese `import random` para importar el módulo aleatorio . Luego ingrese `random.randint(1, 20)` para ver cómo se evalúa la llamada a la función. Devolverá un número entero entre 1 y 20. Repita el código nuevamente y la llamada a la función devolverá otro número entero. El `randint()`

La función devuelve un número entero aleatorio cada vez, al igual que lanzar un dado resultará en un número aleatorio cada vez. Por ejemplo, ingrese lo siguiente en el shell interactivo. Los resultados que obtenga cuando llame a la función randint() probablemente serán diferentes (¡después de todo, es aleatorio!).

```
>>> importar aleatorio  
>>> aleatorio.randint (1, 20)  
12  
>>> aleatorio.aleatorio(1, 20)  
18  
>>> aleatorio.randint(1, 20) 3  
  
>>> aleatorio.aleatorio(1, 20)  
18  
>>> aleatorio.aleatorio(1, 20)  
7
```

También puede probar diferentes rangos de números cambiando los argumentos. Por ejemplo, ingrese random.randint(1, 4) para obtener solo números enteros entre 1 y 4 (incluidos 1 y 4). O prueba random.randint(1000, 2000) para obtener números enteros entre 1000 y 2000.

Ingrese este código en el shell interactivo y vea qué números que obtienes:

```
>>> aleatorio.randint(1, 4) 3  
  
>>> aleatorio.randint(1000, 2000)  
1294
```

Puedes cambiar ligeramente el código del juego para que el juego se comporte de manera diferente. En nuestro código original, usamos un número entero entre 1 y 20:

```
9. número = random.randint(1, 20)  
10. print('Bueno, estoy pensando en un número entre 1 y 20.')
```

Intente cambiar el rango de enteros a (1, 100) en su lugar:

```
9. número = random.randint(1, 100)
10. print('Bueno, estoy pensando en un número entre 1 y 100.')
```

Ahora la computadora pensará en un número entero entre 1 y 100 en lugar de 1 y 20. Cambiar la línea 9 cambiará el rango del número aleatorio, pero recuerda cambiar también la línea 10 para que el juego le diga al jugador el nuevo rango en lugar del antiguo uno.

Puedes usar la función randint() siempre que quieras agregar aleatoriedad a tus juegos. Usarás la aleatoriedad en muchos juegos. (Piense en cuántos juegos de mesa usan dados).

BIENVENIDA AL JUGADOR

Después de que la computadora asigna un número entero al azar, saluda al jugador:

```
10. print('Bueno, ' + miNombre + ', estoy pensando en un número entre 1 y 20.')
```

En la línea 10, print() le da la bienvenida al jugador por su nombre y le dice que la computadora está pensando en un número aleatorio.

A primera vista, puede parecer que hay más de un argumento de cadena en la línea 10, pero examine la línea con cuidado. Los operadores + entre las tres cadenas las concatenan en una sola cadena. Y esa cadena es el argumento pasado a print().

Si observa detenidamente, verá que las comas están dentro de las comillas y forman parte de las propias cadenas.

DECLARACIONES DE CONTROL DE FLUJO

En capítulos anteriores, la ejecución del programa comenzaba en la instrucción superior del programa y avanzaba hacia abajo, ejecutando cada instrucción en orden. Pero con las sentencias `for`, `if`, `else` y `break`, puede hacer el ciclo de ejecución u omitir instrucciones según las condiciones. Este tipo de declaraciones son declaraciones de control de flujo, ya que cambian el flujo de la ejecución del programa a medida que se mueve alrededor de su programa.

Uso de bucles para repetir código

La línea 12 es una instrucción `for`, que indica el comienzo de un ciclo `for`:

12. para conjeturas tomadas en el rango (6):

Los bucles te permiten ejecutar código una y otra vez. La línea 12 repetirá su código seis veces. Una declaración `for` comienza con la palabra clave `for`, seguida de un nuevo nombre de variable, la palabra clave `in`, una llamada a la función `range()` que especifica el número de bucles que debe hacer y dos puntos. Repasemos algunos conceptos adicionales para que pueda trabajar con bucles.

Agrupación con bloques Se

pueden agrupar varias líneas de código en un bloque. Cada línea de un bloque de código comienza con al menos el número de espacios que tiene la primera línea del bloque. Puede saber dónde comienza y termina un bloque mirando la cantidad de espacios al principio de las líneas. Esta es la sangría de la línea.

Los programadores de Python suelen utilizar cuatro espacios adicionales de sangría para comenzar un bloque. Cualquier línea siguiente que tenga la misma sangría es parte del bloque. El bloque termina cuando hay una línea de código con la misma sangría que

antes de que comenzara el bloqueo. También puede haber bloques dentro de otros bloques. La Figura 3-1 muestra un diagrama de código con los bloques delineados y numerados.

```

12. for guessesTaken in range(6):
① 13.     print('Take a guess.')
14.     guess = input()
15.     guess = int(guess)
16.
17.     if guess < number:
② 18.         print('Your guess is too low.')
19.
20.     if guess > number:
③ 21.         print('Your guess is too high.')
22.
23.     if guess == number:
④ 24.         break
25.

26. if guess == number:

```

Figura 3-1: Un ejemplo de bloques y su sangría. Los puntos grisos representan espacios.

En la Figura 3-1, la línea 12 no tiene sangría y no está dentro de ningún bloque. La línea 13 tiene una sangría de cuatro espacios. Dado que esta línea tiene más sangría que la línea anterior, aquí comienza un nuevo bloque. Cada línea que sigue a esta con la misma cantidad de sangría o más se considera parte del bloque .

Si Python encuentra otra línea con menos sangría que la primera línea del bloque, el bloque ha terminado. Las líneas en blanco se ignoran.

La línea 18 tiene una sangría de ocho espacios, que comienza con el bloque . Este bloque está dentro del bloque . Pero la siguiente línea, línea

20, tiene una sangría de sólo cuatro espacios. Como la sangría ha disminuido, sabes que el bloque de la línea 18 ha terminado, y como la línea 20 tiene la misma sangría que la línea 13, sabes que está en el bloque .

La línea 21 vuelve a aumentar la sangría a ocho espacios, por lo que ha comenzado otro bloque nuevo dentro de un bloque: bloque . En línea 23, salimos del bloque , y en la línea 24 entramos en el último bloque dentro de un bloque, bloque . Tanto el bloque como el bloque terminan en la línea 24.

Bucles con **sentencias for** La sentencia for marca el comienzo de un bucle. Los bucles ejecutan el mismo código repetidamente. Cuando la ejecución llega a una **sentencia for**, entra en el bloque que sigue a la **sentencia for**.

Después de ejecutar todo el código en este bloque, la ejecución vuelve a la parte superior del bloque para ejecutar el código nuevamente.

Ingrese lo siguiente en el shell interactivo:

```
>>> for i in range(3):
    print('¡Hola! i está configurado como', i)
```

```
¡Hola! me pongo a 0
¡Hola! me pongo a 1
¡Hola! estoy puesto a 2
```

Note que después de escribir para i en el rango(3): y presionar ENTER, el shell interactivo no mostró otro indicador >>> porque esperaba que escribiera un bloque de código. Presione ENTER nuevamente después de la última instrucción para decirle al shell interactivo que terminó de ingresar el bloque de código. (Esto se aplica solo cuando está trabajando en el shell interactivo. Al escribir

.py archivos en el editor de archivos, no necesita insertar una línea en blanco).

Veamos el ciclo for en la línea 12 de guess.py:

```
12. para conjeturasTomadas en el rango  
(6): 13. print('Haz una conjetura.') # Cuatro espacios delante de "imprimir"  
14. conjetura = entrada() 15. conjetura = int(conjetura)
```

deciséis.

```
17. if adivinar < número: 18.  
imprimir('Tu suposición es  
demasiado baja.') # Ocho espacios delante de "imprimir"  
19  
20. if adivinar > número:  
21     print('Tu suposición es demasiado alta.')  
22  
23. si adivinar == número:  
24     descanso  
25  
26. si adivinar == número:
```

En Guess the Number, el bloque for comienza en la declaración for en la línea 12, y la primera línea después del bloque for es la línea

26

Una declaración for siempre tiene dos puntos (:) después de la condición. Las declaraciones que terminan con dos puntos esperan un nuevo bloque en la siguiente línea. Esto se ilustra en la Figura 3-2.

```

12. for guessesTaken in range(6):
13.     print('Take a guess.') # Four spaces in front of "print" ←
14.     guess = input()
15.     guess = int(guess)
16.
17.     if guess < number:
18.         print('Your guess is too low.') # Eight spaces in front of "print"
19.
20.     if guess > number:
21.         print('Your guess is too high.')
22.
23.     if guess == number:
24.         break
25.
26. if guess == number:

```

The execution loops six times.

Figura 3-2: El flujo de ejecución del bucle for

La figura 3-2 muestra cómo fluye la ejecución. La ejecución ingresará al bloque for en la línea 13 y seguirá descendiendo. Una vez que el programa llega al final del bloque for , en lugar de bajar a la siguiente línea, la ejecución vuelve al inicio del bloque for en la línea 13. Lo hace seis veces debido a la

llamada a la función range(6) en la instrucción for . Cada vez que el la ejecución pasa por el bucle se denomina iteración.

Piense en la declaración for como si dijera: "Ejecute el código en el siguiente bloque una cierta cantidad de veces".

CONSEGUIR LA CONJURACIÓN DEL JUGADOR

Las líneas 13 y 14 le piden al jugador que adivine cuál es el número secreto y le permite ingresar su suposición:

```

13. print('Adivina.') # Cuatro espacios delante de "print" 14.
guess = input()

```

Ese número que ingresa el jugador se almacena en una variable llamada adivinar.

CONVERTIR VALORES CON EL INT(), FLOTANTE() Y STR() FUNCIONES

La línea 15 llama a una nueva función llamada int():

```
15. suposición = int(suposición)
```

La función int() toma un argumento y devuelve el valor del argumento como un entero.

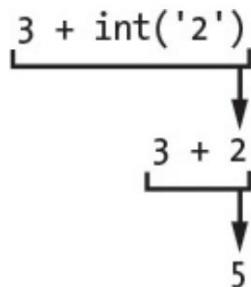
Ingrese lo siguiente en el shell interactivo para ver cómo funciona la función int() :

```
>>> int('42') 42
```

La llamada int('42') devolverá el valor entero 42.

```
>>> 3 + int('2')
5
```

La línea 3 + int('2') muestra una expresión que usa el valor de retorno de int() como parte de una expresión. Se evalúa al número entero valor 5:



Aunque puede pasar una cadena a int(), no puede pasarle cualquier cadena. Pasar 'cuarenta y dos' a int() resultará en un error:

```
>>> int('cuarenta y dos')
```

Rastreo (llamadas recientes más última):

```
Archivo "<pyshell#5>", línea 1, en <módulo>
int('cuarenta y dos')
ValueError: literal no válido para int() con base 10: 'cuarenta y dos'
```

La cadena que le pasas a int() debe estar hecha de números.

En Guess the Number, obtenemos el número del jugador usando la función input(). Recuerde, la función input() siempre devuelve una cadena de texto que ingresó el jugador. Si el jugador escribe 5, la función input() devolverá el valor de cadena '5', no el valor entero 5.

Pero necesitaremos comparar el número del jugador con un número entero más tarde, y Python no puede usar los operadores de comparación < y > para comparar una cadena y un valor entero:

```
>>> 4 < '5'
```

Rastreo (llamadas recientes más última):

```
Archivo "<pyshell#0>", línea 1, en <módulo>
4 < '5'
TypeError: tipos no ordenados: int() < str()
```

Por lo tanto, necesitamos convertir la cadena en un número entero:

```
14. adivinar = entrada()
15. adivinar = int(adivina)
```

En la línea 14, asignamos la variable adivinar al valor de cadena de cualquier número que haya escrito el jugador. La línea 15 sobrescribe el valor de cadena en adivinar con el valor entero devuelto por int(). El código int(guess) devuelve un nuevo valor entero basado en la cadena que se proporcionó, y guess = asigna ese nuevo valor para adivinar. Esto permite que el código más adelante en el programa compare si la suposición es mayor, menor o igual que el número secreto en la variable numérica .

Las funciones float() y str() devolverán de manera similar float y

versiones de cadena de los argumentos que se les pasan. Ingrese lo siguiente en el shell interactivo:

```
>>> flotar('42') 42.0
```

```
>>> flotar(42) 42.0
```

Cuando se pasa la cadena '42' o el entero 42 a float(), el se devuelve float 42.0 .

Ahora intenta usar la función str() :

```
>>> cadena(42)  
'42'  
>>> cadena(42.0)  
'42.0'
```

Cuando el entero 42 se pasa a str(), se devuelve la cadena '42' . Pero cuando el flotante 42.0 se pasa a str(), la cadena '42.0' es regresado.

Con las funciones int(), float() y str() , puede tomar un valor de un tipo de datos y devolverlo como un valor de un tipo de datos diferente.

EL TIPO DE DATOS BOOLEAN

Cada valor en Python pertenece a un tipo de datos. Los tipos de datos que se han introducido hasta ahora son enteros, flotantes, cadenas y ahora booleanos.

El tipo de datos booleano tiene solo dos valores: verdadero o falso. Los valores booleanos deben introducirse con un

T o F mayúsculas y el resto del nombre del valor en minúsculas.

Los valores booleanos se pueden almacenar en variables al igual que los otros tipos de datos:

```
>>> spam = Verdadero
>>> huevos = Falso
```

En este ejemplo, establece el correo no deseado en Verdadero y los huevos en Falso.

Recuerda poner en mayúscula la primera letra.

Utilizará valores booleanos (llamados bools para abreviar) con operadores de comparación para formar condiciones. Cubriremos primero los operadores de comparación y luego repasaremos las condiciones.

Operadores de comparación Los

operadores de comparación comparan dos valores y se evalúan como un valor booleano verdadero o falso . La Tabla 3-1 enumera todas las comparaciones operadores.

Tabla 3-1: Operadores de comparación

Operador	Operación
<	Menos que
>	Mas grande que
<=	Menos que o igual a
>=	Mayor que o igual a
==	Igual a
!=	No igual a

Ya ha leído acerca de los operadores matemáticos +, -, * y / .

Como cualquier operador, los operadores de comparación se combinan con valores para formar expresiones como conjeturas tomadas < 6.

La línea 17 del programa Guess the Number utiliza el operador de comparación menor que:

17. si adivinar < número:

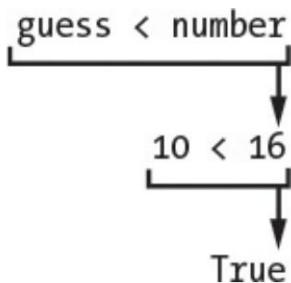
Discutiremos las declaraciones if con más detalle en breve; por ahora, veamos la expresión que sigue a la palabra clave if (la parte adivinar < número). Esta expresión contiene dos valores (los valores en las variables adivinar y número) conectados por un operador (el signo <, o menor que).

Comprobación de verdadero o falso con condiciones Una condición es una expresión que combina dos valores con un operador de comparación (como < o >) y se evalúa como un valor booleano. Una condición es simplemente otro nombre para una expresión que se evalúa como Verdadero o Falso. Un lugar que usamos

las condiciones están en las sentencias if .

Por ejemplo, la condición adivinar < número en la línea 17 pregunta: "¿Es el valor almacenado en adivinar menor que el valor almacenado en número?" Si entonces, la condición se evalúa como Verdadera. Si no, la condición evalúa a Falso.

Digamos que adivinar almacena el entero 10 y número almacena el entero 16. Debido a que 10 es menor que 16, esta condición se evalúa como el valor booleano de True. La evaluación quedaría así:



Experimentando con Booleanos, Comparación Operadores y Condiciones

Ingrese las siguientes expresiones en el shell interactivo para ver

sus resultados booleanos:

```
>>> 0 < 6
```

Verdadero

```
>>> 6 < 0
```

Falso

La condición $0 < 6$ devuelve el valor booleano True porque el número 0 es menor que el número 6. Pero como 6 no es menor que 0, la condición $6 < 0$ se evalúa como Falsa.

Observe que $10 < 10$ se evalúa como Falso porque el número 10 no es más pequeño que el número 10:

```
>>> 10 < 10
```

Falso

Los valores son los mismos. Si Alice tuviera la misma altura que Bob, no dirías que Alice es más alta que Bob o que Alice es más baja que Bob. Ambas declaraciones serían falso.

Ahora ingrese estas expresiones en el shell interactivo:

```
>>> 10 == 10
```

Verdadero

```
>>> 10 == 11
```

Falso

```
>>> 11 == 10
```

Falso

```
>>> 10 != 10
```

Falso

En este ejemplo, 10 es igual a 10, por lo que $10 == 10$ se evalúa como Verdadero. Pero 10 no es igual a 11, entonces $10 == 11$ es Falso. Incluso si se invierte el orden, 11 sigue sin ser igual a 10, por lo que $11 == 10$ es Falso. Finalmente, 10 es igual a 10, entonces $10 != 10$ es Falso.

También puede evaluar expresiones de cadena con comparación operadores:

```
>>> 'Hola' == 'Hola'
```

Verdadero

```
>>> 'Adiós' != 'Hola'
```

Verdadero

```
>>> 'Hola' == 'HOLA'
```

Falso

'Hola' es igual a 'Hola', por lo que 'Hola' == 'Hola' es Verdadero. 'Adiós' no es igual a 'Hola', por lo que 'Adiós' != 'Hola' también es Verdadero.

Observe que la última línea se evalúa como Falsa. Las letras mayúsculas y minúsculas no son iguales en Python, por lo que 'Hola' no es igual a 'HOLA'.

Los valores de cadena y enteros nunca serán iguales entre sí.

Por ejemplo, ingrese lo siguiente en el shell interactivo:

```
>>> 42 == 'Hola'
```

Falso

```
>>> 42 != '42'
```

Verdadero

En el primer ejemplo, 42 es un número entero y 'Hola' es una cadena, por lo que los valores no son iguales y la expresión se evalúa como Falsa.

En el segundo ejemplo, la cadena '42' todavía no es un número entero, por lo que la expresión "el número entero 42 no es igual a la cadena '42'" se evalúa como Verdadero.

La diferencia entre = y ==

Tenga cuidado de no confundir el operador de asignación, =, y el operador de comparación igual a, ==. El signo igual, =, se usa en instrucciones de asignación para almacenar un valor en una variable, mientras que

el signo igual doble, ==, se usa en expresiones para ver si dos valores son iguales. Es fácil usar accidentalmente uno cuando quieras usar el otro.

Puede ser útil recordar que tanto el operador de comparación igual a, ==, como el operador de comparación no igual a, !=, tienen dos caracteres.

DECLARACIONES SI

La línea 17 es una declaración if :

```
17. if adivinar < número: 18.  
imprimir('Tu suposición es
```

El bloque de código que sigue a la declaración if se ejecutará si la condición de la declaración if se evalúa como True. Si la condición es falsa, se omite el código del bloque if . Usando declaraciones if , puede hacer que el programa execute cierto código solo cuando lo deseé.

La línea 17 verifica si la suposición del jugador es menor que el número secreto de la computadora. Si es así, la ejecución se mueve dentro del bloque if en la línea 18 e imprime un mensaje que le dice al jugador que su suposición fue demasiado baja.

La línea 20 verifica si la suposición del jugador es mayor que el numero secreto:

```
20. if adivinar > número:  
21     print('Tu suposición es demasiado alta.')
```

Si esta condición es verdadera, entonces la llamada a la función print() le dice a la jugador que su conjetura es demasiado alta.

DEJAR LOS BUCLES TEMPRANO CON LA DECLARACIÓN DE DESCANSO

La sentencia if de la línea 23 comprueba si el número que jugador adivinó es igual al número secreto. Si es así, el programa ejecuta la instrucción break en la línea 24:

```
23. si adivinar == número: 24.
```

```
romper
```

Una declaración de ruptura le dice a la ejecución que salte inmediatamente fuera del bloque for a la primera línea después del final del bloque for .

La instrucción break se encuentra solo dentro de los bucles, como en un for bloquear.

COMPROBAR SI EL JUGADOR GANADO

El bloque for termina en la siguiente línea de código sin sangría, que es la línea 26:

```
26. si adivinar == número:
```

La ejecución deja el bloque for porque tiene repetido seis veces (cuando el jugador se queda sin conjeturas) o porque se ha ejecutado la instrucción break en la línea 24 (cuando el jugador adivina el número correctamente).

La línea 26 verifica si el jugador acertó. En ese caso, la ejecución ingresa al bloque if en la línea 27:

```
27. conjeturas realizadas = str(conjeturas realizadas  
+ 1) 28. print('¡Buen trabajo, Advinaste el número en ' conjeturas realizadas + '  
conjeturas!')
```

Las líneas 27 y 28 se ejecutan solo si la condición en la instrucción if de la línea 26 es verdadera (es decir, si el jugador adivinó correctamente el número de la computadora).

La línea 27 llama a la función str() , que devuelve la forma de cadena de conjeturas realizadas + 1 (ya que la función de rango va de 0 a 5 en lugar de 1 a 6). La línea 28 concatena cadenas para decirle al jugador que ganó y cuántos intentos tomó. Solo los valores de cadena pueden concatenarse con otras cadenas. Esta es la razón por la cual la línea 27 tuvo que cambiar conjeturas realizadas + 1 a la forma de cadena. De lo contrario, intentar concatenar una cadena con un número entero haría que Python mostrara un error.

COMPROBAR SI EL JUGADOR PERDIDO

Si el jugador se queda sin conjeturas, la ejecución irá a esta línea de código:

30. si adivina != número:

La línea 30 usa el operador de comparación no igual a != para verificar si la última suposición del jugador no es igual al número secreto. Si esta condición se evalúa como Verdadera, la ejecución se mueve en el bloque if de la línea 31.

Las líneas 31 y 32 están dentro del bloque if , ejecutándose solo si la condición en la línea 30 es verdadera:

31. número = str(número) 32.
print('Nop. El número en el que estaba pensando era ' + número + '.')

En este bloque, el programa le dice al jugador cuál es el secreto

número era. Esto requiere cadenas concatenadas, pero el número almacena un valor entero. La línea 31 sobrescribe el número con una cadena para que pueda concatenarse con el 'No. El número que estaba pensando de era ' y '.' cadenas en la línea 32.

En este punto, la ejecución ha llegado al final del código y el programa termina. ¡Felicidades! ¡Acabas de programar tu primer juego real!

Puede ajustar la dificultad del juego cambiando la cantidad de intentos que obtiene el jugador. Para darle al jugador solo cuatro intentos, cambie el código en la línea 12:

12. para conjeturas tomadas en el rango (4):

Al pasar 4 a range(), se asegura de que el código dentro del ciclo se ejecute solo cuatro veces en lugar de seis. Esto hace que el juego sea mucho más difícil. Para hacer el juego más fácil, pase un número entero más grande a la llamada a la función range() . Esto hará que el bucle se ejecute unas cuantas veces más y acepte más intentos por parte del jugador.

RESUMEN

La programación es simplemente la acción de escribir código para programas, es decir, crear programas que puedan ser ejecutados por una computadora.

Cuando ve a alguien usando un programa de computadora (por ejemplo, jugando su juego Guess the Number), todo lo que ve es un texto que aparece en la pantalla. El programa decide qué texto mostrar en la pantalla (la salida del programa) en función de sus instrucciones y del texto que el jugador escribió con el teclado (la entrada del programa). Un programa es solo una colección de instrucciones que actúan sobre la entrada del usuario.

Hay algunos tipos de instrucciones:

- Las expresiones son valores conectados por operadores. Todas las expresiones se evalúan hasta un solo valor. Por ejemplo, `2 + 2` evalúa a `4` o '`Hola' + "Mundo'` se evalúa como '`Hola Mundo`'.

Cuando las expresiones están junto a las palabras clave `if` y `for`, también se les puede llamar condiciones.

- Las instrucciones de asignación almacenan valores en variables para que pueda recuerde los valores más adelante en el programa.
- Las sentencias `if`, `for` y `break` son sentencias de control de flujo que puede hacer que la ejecución salte instrucciones, repita las instrucciones o salga de los bucles. Las llamadas a funciones también cambian el flujo de ejecución saltando a las instrucciones dentro de una función.
- Las funciones `print()` e `input()` muestran texto en la pantalla y obtienen texto del teclado. Las instrucciones que se ocupan de la entrada y salida del programa se denominan E/S (pronunciado eye oh).

Eso es todo, sólo esas cuatro cosas. Por supuesto, hay muchos detalles que aprender acerca de esos cuatro tipos de instrucciones. En capítulos posteriores, leerá sobre más tipos de datos y operadores, más declaraciones de control de flujo y muchas otras funciones que vienen con Python. También hay diferentes tipos de E/S más allá del texto, como la entrada del mouse y la salida de sonido y gráficos.

4

UN PROGRAMA DE BROMAS



El programa de este capítulo cuenta algunos chistes al usuario y demuestra formas más avanzadas de usar cadenas con la función `print()`. La mayoría de los juegos en este libro tendrán texto simple para entrada y salida. El usuario escribe la entrada en el teclado y la salida es el texto que se muestra en la pantalla.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Caracteres de escape
- Uso de comillas simples y comillas dobles para cadenas
- Usar el parámetro de palabra clave final de `print()` para saltar líneas nuevas

Ya aprendió cómo mostrar una salida de texto simple con la función `print()`. Ahora echemos un vistazo más profundo a cómo funcionan las cadenas y `print()` en Python.

MUESTRA DE EJECUCIÓN DE CHISTES

Esto es lo que ve el usuario cuando ejecuta el programa Chistes:

¿Qué obtienes cuando cruzas un muñeco de nieve con un vampiro?

¡Congelación!

¿Cómo llaman los dentistas a la cavidad de un astronauta?

¡Un agujero negro!

TOC Toc.

¿Quién está ahí?

Vaca interrumpiendo.

Vaca interrumpiendo qu-MOO!

CÓDIGO FUENTE PARA CHISTES

Abra una nueva ventana del editor de archivos haciendo clic en Archivo Nueva ventana. En la ventana en blanco que aparece, ingresa el código fuente y guárdalo como chistes.py. Luego ejecute el programa presionando F5.



Si obtiene errores después de ingresar este código, compare el código que escribió con el código del libro con la herramienta de comparación en línea en <https://www.nostarch.com/inventwithpython#diff>.

```
1. print('¿Qué obtienes cuando cruzas un muñeco de nieve con un vampiro?')
2. input() 3. print('¡Congelación!') 4. print() 5. print('¿Cómo llaman los dentistas
a un astronauta? ¡s cavity?') 6. input() 7. print('¡Un agujero negro!') 8. print() 9.
print('Toc, toc.') 10. input() 11. print("¿Quién está ahí ?") 12. input() 13.
print('Vaca interrumpiendo.') 14. input() 15. print('Vaca interrumpiendo wh',
end="") 16. print('-MOO!')
```

CÓMO FUNCIONA EL CÓDIGO

Comencemos mirando las primeras cuatro líneas de código:

```
1. print('¿Qué obtienes cuando cruzas un muñeco de nieve con un vampiro?')
2. input() 3. print('¡Congelación!') 4. print()
```

Las líneas 1 y 3 usan la llamada a la función `print()` para preguntar y dar la respuesta al primer chiste. No desea que el usuario lea inmediatamente el remate del chiste, por lo que hay una llamada a la función `input()` después de la primera instancia de `print()`. El usuario leerá el chiste, presionará ENTER y luego leerá el remate.

El usuario aún puede escribir una cadena y presionar ENTRAR, pero esta cadena devuelta no se almacena en ninguna variable. El programa simplemente lo olvidará y pasará a la siguiente línea de código.

La última llamada a la función `print()` no tiene un argumento de cadena.

Esto le dice al programa que solo imprima una línea en blanco. Las líneas en blanco son útiles para evitar que el texto se vea abarrotado.

PERSONAJES DE ESCAPE

Las líneas 5 a 8 imprimen la pregunta y la respuesta del siguiente chiste:

```
5. print('¿Cómo llaman los dentistas a la cavidad de un  
astronauta?') 6. input() 7. print('¡Un agujero negro!') 8. print()
```

En la línea 5, hay una barra invertida justo antes de la comilla simple: \'. (Tenga en cuenta que \ es una barra invertida y / es una barra inclinada). Esta barra invertida le indica que la letra que le sigue es un carácter de escape. Un carácter de escape le permite imprimir caracteres especiales que son difíciles o imposibles de ingresar en el código fuente, como la comilla simple en un valor de cadena que comienza y termina con comillas simples.

En este caso, si no incluyéramos la barra invertida, la comilla simple en astronaut\'s se interpretaría como el final de la cadena.

Pero esta cita debe ser parte de la cadena. La comilla simple escapada le dice a Python que debe incluir la comilla simple en la cadena.

Pero, ¿qué sucede si realmente desea mostrar una barra invertida?

Cambia de tu programa jokes.py al shell interactivo e ingrese esta instrucción print() :

```
>>> print('Se fueron volando en un helicóptero verde\verde azulado.')  
Se fueron volando en un helicóptero green eal.
```

Esta instrucción no imprimió una barra invertida porque la t en verde azulado se interpretó como un carácter de escape ya que venía después de una

barra invertida El \t simula presionar TAB en su teclado.

Esta línea le dará la salida correcta:

```
>>> print('Se fueron volando en un helicóptero verde\'verde azulado.')  
Se fueron volando en un helicóptero verde/verde azulado.
```

De esta manera, \\ es un carácter de barra invertida, y no hay \t para interpretar como TAB.

La tabla 4-1 es una lista de algunos caracteres de escape en Python, incluido \n, que es el carácter de escape de nueva línea que haber usado antes.

Tabla 4-1: Caracteres de escape

Personaje de escape	Lo que realmente se imprime
\\\	barra invertida (\)
\'	Una frase ('')
\\"	Comillas dobles ("")
\n	Nueva línea
\t	Pestaña

Hay algunos caracteres de escape más en Python, pero estos son los caracteres que probablemente necesitará para crear juegos.

COTIZACIONES SIMPLES Y DOBLES

Mientras todavía estamos en el caparazón interactivo, echemos un vistazo más de cerca a las comillas. Las cadenas no siempre tienen que estar entre comillas simples en Python. También puedes ponerlos entre doble

cotizaciones. Estas dos líneas imprimen lo mismo:

```
>>> imprimir('Hola mundo')
Hola Mundo
>>> imprimir("Hola mundo")
Hola Mundo
```

Pero no puedes mezclar comillas. Esta línea te dará un error.
porque usa ambos tipos de comillas a la vez:

```
>>> imprimir('Hola mundo')
SyntaxError: EOL al escanear una cadena entre comillas simples
```

Me gusta usar comillas simples para no tener que mantener presionada la tecla MAYÚS para escribirlas. Son más fáciles de escribir y a Python no le importa de ninguna manera.

Además, tenga en cuenta que así como necesita \' para tener una comilla simple en una cadena rodeada de comillas simples, necesita \" para tener una comilla doble en una cadena rodeada de comillas dobles. Mire este ejemplo:

```
>>> print("Pedí prestado el auto de Abe por una semana. Dijo, "Claro".")
Pedí prestado el auto de Abe por una semana. Él dijo: "Claro".
```

Utiliza comillas simples para rodear la cadena, por lo que debe agregar una barra invertida antes de la comilla simple en Abe\'s. Pero las comillas dobles en "Seguro". no necesita barras invertidas. El intérprete de Python es lo suficientemente inteligente como para saber que si una cadena comienza con un tipo de comillas, el otro tipo de comillas no significa que la cadena termina.

Ahora mira otro ejemplo:

```
>>> print("Ella dijo, \"No puedo creer que les hayas prestado tu auto.\\"")
Ella dijo: "No puedo creer que les hayas prestado tu auto".
```

La cadena está entre comillas dobles, por lo que debe agregar barras invertidas para todas las comillas dobles dentro de la cadena.
No es necesario escapar de la comilla simple en can't.

Para resumir, en las cadenas de comillas simples, no necesita escapar de las comillas dobles, pero sí necesita escapar de las comillas simples, y en las cadenas de comillas dobles, no necesita escapar de las comillas simples, pero sí necesita escapar de las comillas dobles. cotizaciones.

EL FIN DE LA FUNCIÓN PRINT() PARÁMETRO DE PALABRA CLAVE

Ahora volvamos a chistes.py y echemos un vistazo a las líneas 9 a 16:

```
9. print('Toc toc.') 10.  
input() 11. print("¿Quién  
está ahí?") 12. input() 13.  
print('Vaca interrumpiendo.')  
14. input() 15. print(' Vaca  
interrumpiendo wh', end=")  
16. print('-MOO!')
```

¿Notaste el segundo argumento en la función print() de la línea 15 ?
Normalmente, print() agrega un carácter de nueva línea al final de la cadena que imprime. Esta es la razón por la que una función de impresión () en blanco solo imprimirá una nueva línea. Pero print() puede tener opcionalmente un segundo parámetro: fin.

Recuerde que un argumento es el valor pasado en una llamada de función. La cadena en blanco que se pasa a print() se denomina argumento de palabra clave. El final en end=" se denomina parámetro de palabra clave. Para pasar un argumento de palabra clave a este parámetro de palabra clave, debe escribir end= antes.

Cuando ejecutamos esta sección de código, la salida es

TOC Toc.

¿Quién está ahí?

Vaca interrumpiendo.

Vaca interrumpiendo qu-MOO!

Debido a que pasamos una cadena en blanco al parámetro final , la función print() agregará una cadena en blanco en lugar de agregar una nueva línea. Por eso '-¡MUU!' aparece junto a la línea anterior, en lugar de en su propia línea. No hubo nueva línea después de la Se imprimió la cadena 'Interrupting cow wh' .

RESUMEN

Este capítulo explora las diferentes formas en que puede usar la función print() .

Los caracteres de escape se utilizan para caracteres que son difíciles de escribir en el código con el teclado. Si desea utilizar caracteres especiales en una cadena, debe utilizar un carácter de escape de barra invertida, \, seguido de otra letra para el carácter especial. Por ejemplo, \n sería una nueva línea. Si su carácter especial es una barra invertida, use \\.

La función print() agrega automáticamente un carácter de nueva línea al final de una cadena. La mayoría de las veces, este es un atajo útil. Pero a veces no quieres un carácter de nueva línea. Para cambiar esto, puede pasar una cadena en blanco como argumento de palabra clave para el parámetro de palabra clave final de print() . Por ejemplo, para imprimir spam en la pantalla sin un carácter de nueva línea, llamaría a print('spam', end="").

5

REINO DEL DRAGÓN



El juego que crearás en este capítulo se llama Dragon Realm. El jugador decide entre dos cuevas, que contienen un tesoro o una perdición segura.

CÓMO JUGAR EL REINO DEL DRAGÓN

En este juego, el jugador se encuentra en una tierra llena de dragones. Todos los dragones viven en cuevas con sus grandes montones de tesoros recogidos. Algunos dragones son amigables y comparten su tesoro. Otros dragones tienen hambre y se comen a cualquiera que entre en su cueva. El jugador se acerca a dos cuevas, una con un dragón amistoso y la otra con un dragón hambriento, pero no sabe qué dragón está en qué cueva. El jugador debe elegir entre los dos.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Diagramas de flujo
- Crear sus propias funciones con la palabra clave def
- Cadenas multilínea

- declaraciones while
- Los operadores **y**, o, y no booleanos
- Tablas de verdad
- La palabra clave de retorno
- Alcance variable global y local
- Parámetros y argumentos
- La función dormir()

MUESTRA DE EJECUCIÓN DEL REINO DEL DRAGÓN

Así es como se ve el juego Dragon Realm cuando se ejecuta.

La entrada del jugador está en negrita.

Estás en una tierra llena de dragones. Frente a ti,
ves dos cuevas. En una cueva, el dragón es amigable
y compartirá su tesoro contigo. El otro dragón es
codicioso y hambriento, y te comerá en cuanto te vea.

¿A qué cueva entrarás? (1 o 2)

1

Te acercas a la cueva...

Es oscuro y espeluznante...

¡Un gran dragón salta frente a ti! Abre sus fauces y...

¡Te engulle de un bocado!

¿Quieres jugar de nuevo? (sí o no)

no

DIAGRAMA DE FLUJO PARA DRAGÓN REINO

A menudo ayuda anotar todo lo que desea que haga su juego o programa antes de comenzar a escribir el código. Cuando haces esto, estás diseñando el programa.

Por ejemplo, puede ser útil dibujar un diagrama de flujo. Un diagrama de flujo es un diagrama que muestra todas las acciones posibles que pueden ocurrir en el juego y qué acciones están conectadas. La figura 5-1 es una

diagrama de flujo para Dragon Realm.

Para ver lo que sucede en el juego, coloca tu dedo en el cuadro INICIO. Luego sigue una flecha de ese cuadro a otro caja. Su dedo es como la ejecución del programa. El programa termina cuando su dedo aterriza en el cuadro FIN.

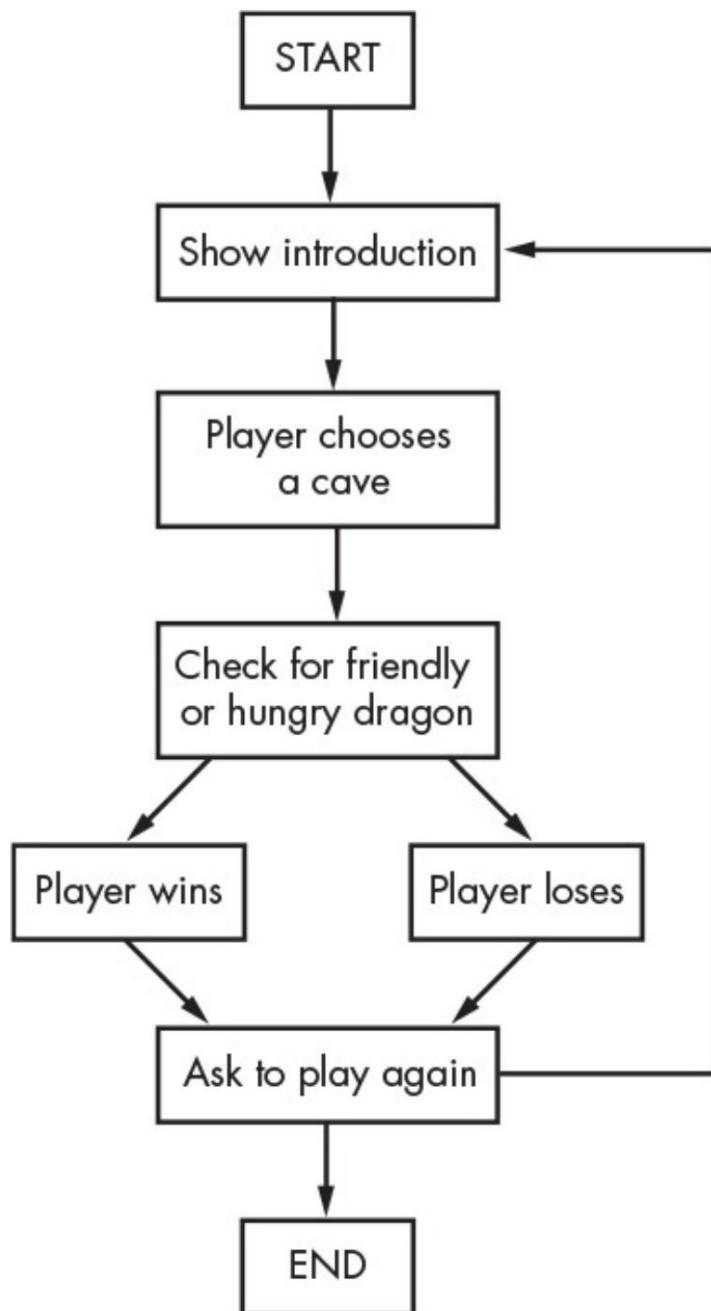


Figura 5-1: Diagrama de flujo para el juego Dragon Realm

Cuando llegue al cuadro "Comprobar si hay un dragón amistoso o hambriento", puede ir al cuadro "El jugador gana" o al cuadro "El jugador pierde". En este punto de bifurcación, el programa puede ir en diferentes direcciones. De cualquier manera, ambos caminos eventualmente terminarán en el cuadro "Solicitar jugar de nuevo".

CÓDIGO FUENTE PARA DRAGÓN REINO

Abra una nueva ventana del editor de archivos haciendo clic en Archivo Nueva ventana. Ingrese el código fuente y guárdelo como dragon.py. Luego ejecute el programa presionando F5. Si obtiene errores, compare el código que escribió con el código del libro con la herramienta de comparación en línea en <https://www.nostarch.com/inventwithpython#diff>.



dragón.py

-
1. importación aleatoria 2.
 - tiempo de importación
 - 3.
 4. def displayIntro():
 5. print("Estás en una tierra llena de dragones. Frente a ti,

6. ves dos cuevas. En una cueva, el dragón es amigable 7. y compartirá su tesoro contigo. El otro dragón 8. es codicioso y hambriento, y te comerá nada más verlo.") 9. print() 10.

```
11. def elegirCueva():
    "
12. cueva =
13. while cueva != '1' y cueva != '2':
14     print('¿A qué cueva entrarás? (1 o 2)') cave = input()
15.

decisión.

17. cueva de regreso
18

19. def comprobarCueva(cuevaelegida):
20. print('Te acercas a la cueva...') 21.
time.sleep(2) 22. print('Está oscuro y
tenebroso...') 23. time.sleep (2) 24. print('¡Un
gran dragón salta frente a ti! Abre sus fauces

y...')
25. print() 26.
time.sleep(2) 27.

28. cuevaamistosa = random.randint(1, 2)
29

30. if cuevaelegida == str(cuevaamigable):
31     print('¡Te da su tesoro!')
32. más:
33.     print('¡Te engulle de un bocado!')
34.

35. volver a jugar = 'sí'
36. while volver a jugar == 'sí' o volver a jugar == 'y': 37.
mostrarIntro() 38. NúmeroCueva = elegirCueva() 39.
VerificarCueva(NúmeroCueva) 40.

41. print('¿Quieres volver a jugar? (sí o no)') 42. volver a jugar
= entrada()
```

Veamos el código fuente con más detalle.

IMPORTAR EL ALEATORIO Y MÓDULOS DE TIEMPO

Este programa importa dos módulos:

1. importación aleatoria 2.

tiempo de importación

El módulo aleatorio proporciona la función randint() , que usamos en el juego

Adivina el número del Capítulo 3. La línea 2 importa el módulo de tiempo , que contiene funciones relacionadas con el tiempo.

FUNCIONES EN DRAGON REALM

Las funciones le permiten ejecutar el mismo código varias veces sin tener que copiar y pegar ese código una y otra vez.

En cambio, coloca ese código dentro de una función y llama a la función cuando lo necesite. Debido a que escribe el código solo una vez en la función, si el código de la función tiene un error, solo tiene que cambiarlo en un lugar del programa.

Ya ha utilizado algunas funciones, como print(), input(), randint(), str() e int().

Sus programas han llamado a estas funciones para ejecutar el código dentro de ellas. En el juego Dragon Realm, escribirás tus propias funciones usando def declaraciones.

Declaraciones def

La línea 4 es una declaración de definición :

-
4. def displayIntro():
 5. print("Estás en una tierra llena de dragones. Frente a ti,
 6. ves dos cuevas. En una cueva, el dragón es amigable
 7. y compartirá su tesoro contigo El otro dragón

8. es codicioso y hambriento, y te comerá nada más verlo."") 9. print()

La declaración def define una nueva función (en este caso, el función displayIntro()), a la que puede llamar más adelante en el programa.

La figura 5-2 muestra las partes de una sentencia def . Tiene la palabra clave def seguida de un nombre de función entre paréntesis y luego dos puntos (:). El bloque después de la instrucción def se llama bloque de definición .

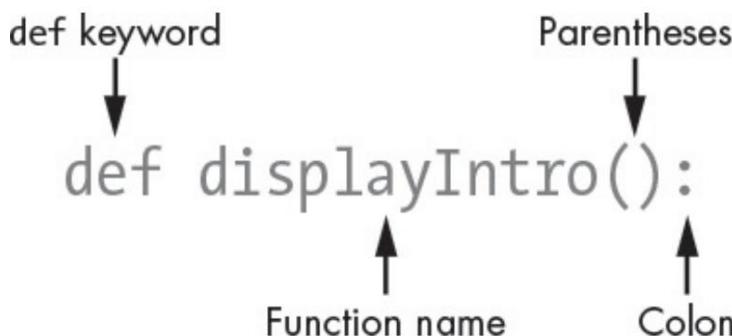


Figura 5-2: Partes de una sentencia def

Llamar a una función

Cuando define una función, especifica las instrucciones para que se ejecute en el siguiente bloque. Cuando llamas a una función, se ejecuta el código dentro del bloque de definición . A menos que llame a la función, las instrucciones en el bloque de definición no se ejecutarán.

En otras palabras, cuando la ejecución llega a una sentencia def , salta a la primera línea después del bloque def . Pero cuando se llama a una función, la ejecución se mueve dentro de la función. a la primera línea del bloque de definición .

Por ejemplo, mire la llamada a la función displayIntro() en línea 37:

37. mostrarIntro()

Llamar a esta función ejecuta la llamada `print()` , que muestra el
“Estás en una tierra llena de dragones...” introducción.

Dónde poner definiciones de funciones

La instrucción `def` y el bloque `def` de una función deben aparecer antes de llamar a la función, al igual que debe asignar un valor a una variable antes de usar esa variable. Si coloca la llamada a la función antes de la definición de la función, obtendrá un error. Veamos un programa corto como ejemplo. Abra una nueva ventana del editor de archivos, ingrese este código, guárdelo como `ejemplo.py` y ejecútelo:

```
decir adiós()
```

```
def decir Adiós():
    print('¡Adiós!')
```

Si intenta ejecutar este programa, Python le dará un mensaje de error similar a este:

Rastreo (última llamada más reciente):

Archivo "C:/Users/AI/AppData/Local/Programs/Python/Python36/example.py", línea 1,
en <módulo> sayGoodbye()

NameError: el nombre 'sayGoodbye' no está definido

Para corregir este error, mueva la definición de la función antes de la llamada a la función:

```
def decir Adiós():
    print('¡Adiós!')
```

```
decir adiós()
```

Ahora, la función se define antes de llamarla, por lo que Python sabrá qué debe hacer `sayGoodbye()` . De lo contrario, Python

no tendrá las instrucciones para sayGoodbye() cuando se llame y por lo que no será capaz de ejecutarlo.

CUERDAS MULTILÍNEA

Hasta ahora, todas las cadenas en nuestras llamadas a la función print() han estado en una línea y han tenido un carácter de comillas al principio y al final. Sin embargo, si usa tres comillas al principio y al final de una cadena, puede ocupar varias líneas. Estas son cadenas de varias líneas.

Ingrese lo siguiente en el shell interactivo para ver cómo las cadenas multilínea funcionan:

```
>>> fizz = "Querida Alice,  
regresaré a la casa de Carol a fin de mes.  
Tu amigo,  
Beto"  
  
>>> imprimir(efervescencia)  
Querida Alice,  
Regresará a la casa de Carol a fin de mes.  
Tu amigo,  
Beto
```

Tenga en cuenta los saltos de línea en la cadena impresa. En una cadena de varias líneas, los caracteres de nueva línea se incluyen como parte de la cadena. No tiene que usar el carácter de escape \n ni las comillas de escape siempre que no use tres comillas juntas. Estos saltos de línea facilitan la lectura del código cuando se trata de grandes cantidades de texto.

CÓMO HACER UN BUCLE CON WHILE DECLARACIONES

La línea 11 define otra función llamada elegirCueva():

11. def elegirCueva():

El código de esta función le pregunta al jugador a qué cueva quiere entrar, ya sea 1 o 2. Necesitaremos usar una instrucción while para pedirle al jugador que elija una cueva, lo que marca el comienzo de un nuevo tipo de ciclo: un rato círculo.

A diferencia de un bucle for , que se repite una cantidad específica de veces, un bucle while se repite siempre que cierta condición sea verdadera. Cuando la ejecución llega a una instrucción while , evalúa la condición junto a la palabra clave while . Si la condición se evalúa como True, la ejecución se mueve dentro del siguiente bloque, llamado bloque while . Si la condición se evalúa como Falsa, la ejecución se mueve más allá del bloque while .

Puede pensar en una declaración while como casi lo mismo que una declaración if . La ejecución del programa entra en los bloques de ambas declaraciones si su condición es Verdadera. Pero cuando el condición llega al final del bloque en un ciclo while , se mueve Vuelva a la instrucción while para volver a verificar la condición.

Mire el bloque de definición de chooseCave() para ver un ciclo while en acción:

12. cueva = ""
13. while cueva != '1' y cueva != '2':

La línea 12 crea una nueva variable llamada cueva y almacena una cadena en blanco en él. Luego comienza un ciclo while en la línea 13. La función elegirCueva() debe asegurarse de que el jugador haya ingresado 1 o 2 y no otra cosa. Un bucle aquí sigue preguntando al jugador qué cueva elige hasta que ingrese una de estas dos válidas.

respuestas Esto se llama validación de entrada.

La condición también contiene un nuevo operador que no ha visto antes: `y`. Así como `-` y `*` son operadores matemáticos, `y ==` y `!=` son operadores de comparación, el operador `and` es un operador booleano. Echemos un vistazo más de cerca a Boolean operadores.

OPERADORES BOOLEANOS

La lógica booleana trata con cosas que son verdaderas o falsas.

Los operadores booleanos comparan valores y evalúan a un solo valor booleano.

Piense en la oración "Los gatos tienen bigotes y los perros tienen cola". "Los gatos tienen bigotes" es verdadero, y "los perros tienen colas" también lo es, por lo que toda la oración "Los gatos tienen bigotes y los perros tienen colas" es verdadera.

Pero la oración "Los gatos tienen bigotes y los perros tienen alas" sería falsa. Aunque "los gatos tienen bigotes" es cierto, los perros no tienen alas, por lo que "los perros tienen alas" es falso. En la lógica booleana, las cosas solo pueden ser completamente verdaderas o completamente falsas. Debido a la palabra `y`, la oración completa es verdadera solo si ambas partes son verdaderas. Si una o ambas partes son falsas, entonces toda la oración es falsa.

El operador `and` El operador

and en Python también requiere que toda la expresión booleana sea verdadera o falsa. Si los valores booleanos a ambos lados de la palabra clave `y` son verdaderos, la expresión se evalúa como verdadera. Si uno o ambos valores booleanos son falsos, la expresión se evalúa como falsa.

Introduzca las siguientes expresiones con el operador y en la concha interactiva:

```
>>> Verdadero y Verdadero
```

```
Verdadero
```

```
>>> Verdadero y Falso
```

```
Falso
```

```
>>> Falso y Verdadero
```

```
Falso
```

```
>>> Falso y Falso
```

```
Falso
```

```
>>> spam = 'Hola' >>>
```

```
10 < 20 y spam == 'Hola'
```

```
Verdadero
```

El operador and se puede utilizar para evaluar dos expresiones booleanas cualesquiera. En el último ejemplo, `10 < 20` se evalúa como Verdadero y `spam == 'Hola'` también se evalúa como Verdadero, por lo que las dos expresiones booleanas unidas por el operador y se evalúan como Verdadero.

Si alguna vez olvida cómo funciona un operador booleano, puede consultar su tabla de verdad, que muestra cómo se evalúa cada combinación de valores booleanos. La tabla 5-1 muestra todas las combinaciones para el operador y .

Tabla 5-1: La tabla de verdad del operador y

A y B	Evalúa a
verdadero y verdadero	Verdadero
Verdadero y falso	Falso
falso y verdadero	Falso
falso y falso	Falso

El operador or El

operador or es similar al operador and , excepto que se evalúa como True si cualquiera de los dos valores booleanos es True. La única vez que el operador or se evalúa como Falso es si ambos valores booleanos son falsos.

Ahora ingrese lo siguiente en el shell interactivo:

```
>>> Verdadero o Verdadero
```

Verdadero

```
>>> Verdadero o Falso
```

Verdadero

```
>>> Falso o Verdadero
```

Verdadero

```
>>> Falso o Falso
```

Falso

```
>>> 10 > 20 o 20 > 10
```

Verdadero

En el último ejemplo, 10 no es mayor que 20 pero 20 es mayor que 10, por lo que la primera expresión se evalúa como Falsa y la segunda expresión se evalúa como Verdadera. Debido a que la segunda expresión es True, toda esta expresión se evalúa como True.

La tabla de verdad del operador or se muestra en la tabla 5-2.

Tabla 5-2: La tabla de verdad del operador or

A o B	Evalúa a
verdadero o verdadero	Verdadero
Verdadero o falso	Verdadero
Falso o Verdadero	Verdadero
Falso o Falso	Falso

El operador not En lugar de combinar dos valores, el operador not trabaja en un solo valor. El operador no se evalúa como el valor booleano opuesto: las expresiones verdaderas se evalúan como falsas y las expresiones falsas se evalúan como verdaderas.

Ingrese lo siguiente en el shell interactivo:

```
>>> no es cierto
Falso
>>> no es falso
Verdadero
>>> no ('negro' == 'blanco')
Verdadero
```

El operador not también se puede utilizar en cualquier expresión booleana. En el último ejemplo, la expresión 'negro' == 'blanco' se evalúa como Falso. Es por eso que no ('negro' == 'blanco') es Verdadero.

La tabla de verdad del operador not se muestra en la tabla 5-3.

Tabla 5-3: La tabla de verdad del operador no

No un	Evalúa a
no es verdad	Falso
No falso	Verdadero

Evaluación de operadores booleanos

Mire la línea 13 del juego Dragon Realm nuevamente:

13. while cueva != '1' y cueva != '2':

La condición tiene dos partes conectadas por el booleano y operador. La condición es verdadera solo si ambas partes son verdaderas.

La primera vez que se verifica la condición de la declaración while , cave se establece en la cadena en blanco, ". La cadena en blanco no es igual a la cadena '1', por lo que el lado izquierdo se evalúa como Verdadero. La cadena en blanco tampoco es igual a la cadena '2', por lo que el lado derecho se evalúa como Verdadero.

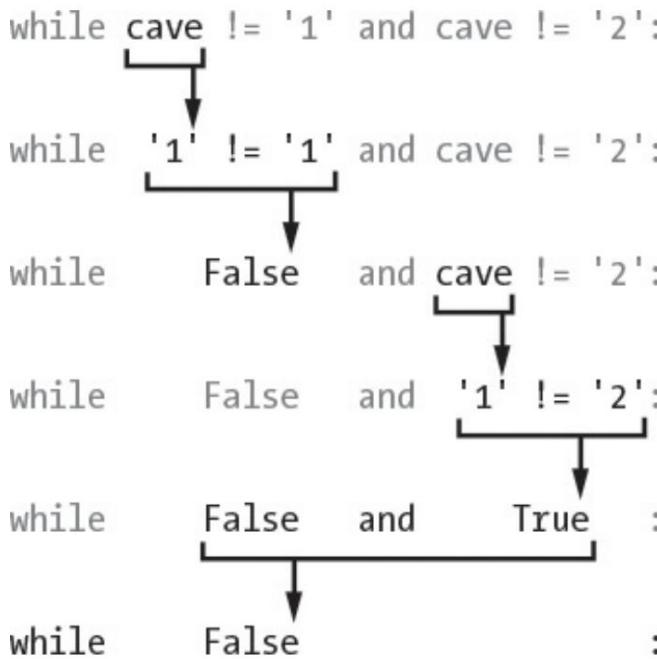
Entonces la condición se convierte en Verdadera y Verdadera. porque ambos los valores son True, toda la condición se evalúa como True, por lo que la ejecución del programa ingresa al bloque while , donde el programa intentará asignar un valor que no esté en blanco para cave.

La línea 14 le pide al jugador que elija una cueva:

```
13. while cueva != '1' y cueva != '2':  
14     print('¿A qué cueva entrarás? (1 o 2)') cueva = input()  
15.
```

La línea 15 le permite al jugador escribir su respuesta y presionar ENTER. Esta respuesta se almacena en la cueva. Después de ejecutar este código, la ejecución regresa al principio de la instrucción while y vuelve a verificar la condición en la línea 13.

Si el jugador ingresó 1 o 2, cueva será '1' o '2' (ya que input() siempre devuelve cadenas). Esto hace que la condición sea Falsa y la ejecución del programa continuará más allá del ciclo while . Por ejemplo, si el usuario ingresó '1', entonces la evaluación se vería así:



Pero si el jugador ingresó 3 o 4 o HOLA, esa respuesta sería inválida. La condición sería entonces Verdadera y

ingrese el bloque while para preguntarle al jugador nuevamente. El programa sigue preguntando al jugador qué cueva elige hasta que ingresa 1 o 2. Esto garantiza que una vez que la ejecución continúa, la variable cueva contiene una respuesta válida.

VALORES DEVUELTOS

La línea 17 tiene algo nuevo llamado declaración de retorno :

17. cueva de regreso

Una declaración de devolución aparece solo dentro de los bloques de definición donde se define una función, en este caso, elegirCueva() . recuerda como la función input () devuelve un valor de cadena que ingresó el jugador? La función elegirCueva() también devolverá un valor. La línea 17 devuelve la cadena que está almacenada en cueva, ya sea '1' o '2'.

Una vez que se ejecuta la declaración de retorno , la ejecución del programa

salta inmediatamente fuera del bloque def (al igual que la instrucción break hace que la ejecución salte fuera de un bloque while). La ejecución del programa vuelve a la línea con la llamada de función. La llamada de función en sí misma se evaluará como la de la función.
valor de retorno

Pase a la línea 38 por un momento donde el chooseCave() función se llama:

38. numeroCueva = elegirCueva()

En la línea 38, cuando el programa llama a la función elegirCueva(), que se definió en la línea 11, la llamada a la función evalúa la cadena dentro de la cueva, que luego se almacena en númeroCueva. El bucle while dentro de chooseCave() garantiza que chooseCave() devolverá solo '1' o '2' como valor de retorno. Entonces caveNumber solo puede tener uno de estos dos valores.

ALCANCE GLOBAL Y LOCAL ALCANCE

Hay algo especial en las variables que se crean dentro de las funciones, como la variable `cave` en la función chooseCave().
función en la línea 12:

12. cueva = "

Se crea un ámbito local cada vez que se llama a una función. Cualquier variable asignada en esta función existe dentro del ámbito local.
Piense en un ámbito como un contenedor de variables. Lo que hace que las variables en los ámbitos locales sean especiales es que se olvidan cuando la función regresa y se volverán a crear si se vuelve a llamar a la función. El valor de una variable local no se recuerda

entre llamadas de función.

Las variables que se asignan fuera de las funciones existen en el ámbito global. Solo hay un alcance global y se crea cuando comienza su programa. Cuando su programa finaliza, el alcance global se destruye y todas sus variables se olvidan. De lo contrario, la próxima vez que ejecute su programa, las variables recordarán sus valores de la última vez que lo ejecutó.

Una variable que existe en un ámbito local se denomina variable local, mientras que una variable que existe en el ámbito global se denomina variable global. Una variable debe ser una u otra; no puede ser a la vez local y global.

La cueva variable se crea dentro de la función elegirCueva() . Esto significa que se crea en el local de la función chooseCave() . alcance. Se olvidará cuando elijaCueva() regrese y se volverá a crear si se llama a elegirCueva() por segunda vez.

Las variables locales y globales pueden tener el mismo nombre, pero son variables diferentes porque están en diferentes ámbitos. Escribamos un nuevo programa para ilustrar estos conceptos:

```
def tocino():
    spam = 99 # Crea una variable local llamada spam
    print(spam) # Imprime 99

    spam = 42 # Crea una variable global llamada spam
    print(spam) # Imprime 42    bacon()    print(spam) #
    Imprime 42    # Llama a la función bacon() e imprime 99
```

Primero creamos una función llamada tocino(). En tocino(), creamos una variable llamada spam y asígnele 99 . En , llamamos a print() para imprimir esta variable de spam local, que es 99. En , una variable global

variable llamada spam también se declara y se establece en 42. Esta variable es global porque está fuera de todas las funciones. Cuando la variable spam global se pasa a print() en , imprime 42. Cuando se llama a la función bacon() en , se ejecutan y , y el

La variable local de spam se crea, establece y luego se imprime. Entonces, llamar a bacon() imprime el valor 99. Después de que regresa la llamada a la función bacon() , la variable local de spam se olvida. Si imprimimos spam en , estamos imprimiendo la variable global, por lo que la salida allí es 42.

Cuando se ejecuta, este código generará lo siguiente:

```
42
99
42
```

El lugar donde se crea una variable determina en qué ámbito se encuentra. Tenga esto en cuenta al escribir sus programas.

PARÁMETROS DE FUNCIÓN

La siguiente función definida en el programa Dragon Realm se llama checkCave().

```
19. def comprobarCueva(cuevaellegida):
```

Observe el texto cueva elegida entre paréntesis. Este es un parámetro: una variable local que utiliza el código de la función. Cuando se llama a la función, los argumentos de la llamada son los valores asignados a los parámetros.

Volvamos al shell interactivo por un momento.

Recuerda que para algunas llamadas a funciones, como str() o randint(), pasarías uno o más argumentos entre paréntesis:

```
>>> cadena(5)
'5'

>>> aleatorio.aleatorio(1, 20)
14

>>> len('Hola') 5
```

Este ejemplo incluye una función de Python que aún no has visto: `len()`. La función `len()` devuelve un número entero que indica cuántos caracteres hay en la cadena que se le pasa. En este caso nos dice que la cadena 'Hola' tiene 5 caracteres.

También pasará un argumento cuando llame a `checkCave()`. Este argumento se almacena en una nueva variable llamada `cuevaElegida`, que es el parámetro de `comprobarCueva()`.

Aquí hay un programa corto que demuestra la definición de un función (`sayHello`) con un parámetro (`nombre`):

```
def decirHola(nombre):
    print('Hola, ' + nombre + '. Tu nombre tiene ' + str(len(nombre)) + ' letras.')
    decirHola('Alice')
    decirHola('Bob')
spam = 'Carol dice hola (' + spam + ')'
```

Cuando llama a `sayHello()` con un argumento entre paréntesis, el argumento se asigna al parámetro de `nombre` y se ejecuta el código de la función. Solo hay una línea de código en la función `sayHello()`, que es una llamada a la función `print()`.

Dentro de la llamada a la función `print()` hay algunas cadenas y la variable de `nombre`, junto con una llamada a la función `len()`. Aquí, `len()` se usa para contar la cantidad de caracteres en el nombre. Si ejecuta el programa, la salida se ve así:

Hola, Alicia. Tu nombre tiene 5 letras.
Hola Bob. Tu nombre tiene 3 letras.
Hola Carol. Tu nombre tiene 5 letras.

Para cada llamada a sayHello(), se imprime un saludo y la longitud del argumento del nombre . Tenga en cuenta que debido a que la cadena 'Carol' está asignada a la variable spam , sayHello(spam) es equivalente a decirHola('Carol').

MOSTRAR LOS RESULTADOS DEL JUEGO

Volvamos al código fuente del juego Dragon Realm:

```
20. print('Te acercas a la cueva...') 21.  
time.sleep(2)
```

El módulo de tiempo tiene una función llamada sleep() que pausa el programa. La línea 21 pasa el valor entero 2 para que time.sleep() haga una pausa en el programa durante 2 segundos.

```
22. print('Es oscuro y tenebroso...')  
23. time.sleep(2)
```

Aquí el código imprime algo más de texto y espera otros 2 segundos. Estas breves pausas añaden suspense al juego en lugar de mostrar todo el texto a la vez. En el programa Chistes del Capítulo 4, llamaste a la función input() para hacer una pausa hasta que el jugador presionara ENTER. Aquí, el jugador no tiene que hacer nada excepto esperar un par de segundos.

```
24. print('¡Un gran dragón salta frente a ti! Abre sus fauces  
y...')  
25. imprimir()  
26. tiempo.dormir(2)
```

Con la creación de suspense, nuestro programa determinará a continuación qué cueva tiene el dragón amistoso.

DECIDIR QUÉ CUEVA TIENE LA DRAGÓN AMIGABLE

La línea 28 llama a la función `randint()` , que devolverá aleatoriamente 1 o 2 .

```
28. cuevaamistosa = random.randint(1, 2)
```

Este valor entero se almacena en `friendlyCave` e indica el cueva con el dragón amistoso.

```
30. if cuevaellegida == str(cuevaamigable):  
31.     print('¡Te da su tesoro!')
```

La línea 30 comprueba si la cueva elegida por el jugador en la variable `cueva elegida` ('1' o '2') es igual a la cueva del dragón amigo.

Pero el valor en `friendlyCave` es un número entero porque `randint()` devuelve números enteros. No puede comparar cadenas y enteros con el signo `==` , porque nunca serán iguales entre sí: '1' no es igual a 1 y '2' no es igual a 2.

Así que `friendlyCave` se pasa a la función `str()` , que devuelve el valor de cadena de `friendlyCave`. Ahora los valores tendrán el mismo tipo de datos y se pueden comparar significativamente entre sí. También podríamos haber convertido la cueva elegida en un número entero . valor en su lugar. Entonces la línea 30 se vería así:

```
if int(cuevaellegida) == cuevaamigable:
```

Si la cueva elegida es igual a la cueva amiga, la condición se evalúa

a Verdadero, y la línea 31 le dice al jugador que ha ganado el tesoro.

Ahora tenemos que agregar algo de código para ejecutar si la condición es falso. La línea 32 es una declaración else :

32. más:

33. print('¡Te engulle de un bocado!')

Una sentencia else solo puede venir después de un bloque if . el otro El bloque se ejecuta si la condición de la instrucción if es falsa. Pensar en esto como la forma en que el programa dice: "Si esta condición es verdadera, entonces ejecute el bloque if o, de lo contrario, ejecute el bloque else ".

En este caso, la declaración else se ejecuta cuando la cueva elegida no está igual a cuevaamigable. Luego, se ejecuta la llamada a la función print() en la línea 33, diciéndole al jugador que ha sido devorado por el dragón.

EL BUCLE DEL JUEGO

La primera parte del programa define varias funciones pero no ejecuta el código dentro de ellas. La línea 35 es donde la principal

parte del programa comienza porque es la primera línea que se ejecuta:

35. volver a jugar = 'sí'

36. while volver a jugar == 'sí' or volver a jugar == 'y':

Esta línea es donde comienza la parte principal del programa.

Las declaraciones def anteriores simplemente definían las funciones. Ellos no ejecutó el código dentro de esas funciones.

Las líneas 35 y 36 configuran un bucle que contiene el resto del código del juego. Al final del juego, el jugador puede decirle al programa si quiere volver a jugar. Si lo hacen, la ejecución entra en el bucle while para ejecutar todo el juego de nuevo. Si no lo hacen, la condición de la instrucción while será Falsa,

y la ejecución se moverá al final del programa y terminará.

La primera vez que la ejecución llega a esta instrucción `while` , la línea 35 acabará de asignar 'sí' a la variable `jugar` de nuevo . Eso significa que la condición será Verdadera al comienzo de la programa. Esto garantiza que la ejecución entre en el bucle al menos una vez.

Llamar a las funciones en el programa

La línea 37 llama a la función `mostrarIntro()` :

37. `mostrarIntro()`

Esta no es una función de Python, es la función que definió anteriormente en la línea 4. Cuando se llama a esta función, la ejecución del programa salta a la primera línea en `displayIntro()` función en la línea 5. Cuando todas las líneas en la función han sido ejecutada, la ejecución vuelve a la línea 37 y continúa descendiendo.

La línea 38 también llama a una función que definiste:

38. `numeroCueva = elegirCueva()`

Recuerda que la función `elegirCueva()` le permite al jugador elegir la cueva a la que quiere entrar. Cuando se ejecuta la cueva de retorno de la línea 17 , la ejecución del programa vuelve a la línea 38. La llamada a `elegirCueva()` evalúa el valor de retorno, que será un valor entero que representa la cueva en la que el jugador eligió entrar. Este valor de retorno se almacena en una nueva variable llamada número de cueva.

Luego, la ejecución del programa pasa a la línea 39:

39. comprobarCueva(numeroCueva)

La línea 39 llama a la función checkCave() , pasando el valor en caveNumber como argumento. La ejecución no solo salta a la línea 20, sino que el valor en caveNumber se copia en el parámetro elegidoCueva dentro de la función comprobarCueva() . esta es la función que mostrará '¡Te da su tesoro!' o '¡Te engulle de un bocado!' dependiendo de la cueva que el jugador elija para entrar.

Pedirle al jugador que vuelva a jugar Ya sea que el jugador gane o pierda, se le pregunta si quiere volver a jugar.

41. print('¿Quieres volver a jugar? (sí o no)') 42.
volver a jugar = entrada()

La variable playAgain almacena lo que escribe el jugador. La línea 42 es la última línea del bloque while , por lo que el programa vuelve a la línea 36 para verificar la condición del ciclo while : reproducir de nuevo == 'sí' o jugar de nuevo == 'y'.

Si el jugador ingresa la cadena 'sí' o 'y', entonces la ejecución vuelve a entrar en el bucle en la línea 37.

Si el jugador ingresa 'no' o 'n' o algo tonto como 'Abraham Lincoln', entonces la condición es Falsa y la ejecución del programa continúa hasta la línea después del bloque while . Pero como no hay cualquier línea después del bloque while , el programa termina.

Una cosa a tener en cuenta: la cadena 'YES' no es igual a la cadena 'yes' ya que la computadora no considera las letras mayúsculas y minúsculas como iguales.

Si el jugador ingresó la cadena 'SÍ', entonces el mientras que la condición de la declaración se evaluaría como Falsa y la

el programa aún terminaría. Posteriormente, el programa Hangman le mostrará cómo evitar este problema. (Consulte “Los métodos de cadena lower() y upper() ” en la página 101).

¡Acabas de completar tu segundo juego! En Dragon Realm, usaste mucho de lo que aprendiste en el juego Guess the Number y aprendiste algunos trucos nuevos. Si no entendió algunos de los conceptos de este programa, revise cada línea del código fuente nuevamente e intente cambiar el código para ver cómo cambia el programa.

En el Capítulo 6, no crearás otro juego. En cambio, tú aprenderá a usar una característica de IDLE llamada depurador.

RESUMEN

En el juego Dragon Realm, creaste tus propias funciones. Una función es un miniprograma dentro de su programa. El código dentro de la función se ejecuta cuando se llama a la función. Al dividir su código en funciones, puede organizar su código en secciones más cortas y fáciles de entender.

Los argumentos son valores que se copian en los parámetros de la función cuando se llama a la función. La llamada de función en sí misma evalúa al valor de retorno.

También aprendió sobre los ámbitos de las variables. Las variables creadas dentro de una función existen en el ámbito local y las variables creadas fuera de todas las funciones existen en el ámbito global. El código en el ámbito global no puede hacer uso de variables locales. Si una variable local tiene el mismo nombre que una variable global, Python la considera una variable separada. Asignar nuevos valores a la variable local no cambiará el valor de la variable global.

Los ámbitos variables pueden parecer complicados, pero son

útil para organizar funciones como piezas de código separadas del resto del programa. Debido a que cada función tiene su propio alcance local, puede estar seguro de que el código de una función no causará errores en otras funciones.

Casi todos los programas tienen funciones porque son muy útiles. Al comprender cómo funcionan las funciones, puede ahorrarse mucho escribir y hacer que los errores sean más fáciles de corregir.

6

USO DEL DEPURADOR



Si ingresa el código incorrecto, la computadora no le dará el programa correcto.

Un programa de computadora siempre hará lo que le digas, pero lo que le digas que haga puede no ser lo que realmente quieras que haga. Estos errores son errores en un programa de computadora. Los errores ocurren cuando el programador no ha pensado detenidamente qué está haciendo exactamente el programa.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Tres tipos de errores
- Depurador de IDLE
- Los botones Ir y Salir
- Entrar, pasar y salir
- Puntos de ruptura

TIPOS DE ERRORES

Hay tres tipos de errores que pueden ocurrir en su programa:

Errores de sintaxis Este tipo de error proviene de errores tipográficos. Cuándo

el intérprete de Python ve un error de sintaxis, se debe a que su código no está escrito en el lenguaje de Python adecuado. Un programa de Python con un solo error de sintaxis no se ejecutará.

Errores de tiempo de ejecución Estos son errores que ocurren mientras se ejecuta el programa. El programa funcionará hasta que llegue a la línea de código con el error, y luego el programa terminará con un mensaje de error (esto se llama bloqueo). El intérprete de Python mostrará un rastreo: un mensaje de error que muestra la línea que contiene el problema.

Errores semánticos Estos errores, que son los más difíciles de corregir, no bloquean el programa, pero evitan que el programa haga lo que el programador pretendía que hiciera. Por ejemplo, si el programador desea que la variable total sea la suma de los valores de las variables a, b y c pero escribe c, entonces el valor `total = a * b * total` será incorrecto. Esto podría bloquear el programa más adelante, pero no será inmediatamente obvio dónde ocurrió el error semántico.

Encontrar errores en un programa puede ser difícil, ¡si es que los nota! Al ejecutar su programa, es posible que descubra que a veces las funciones no se llaman cuando se supone que deben hacerlo, o tal vez se llaman demasiadas veces. Puede codificar la condición para un ciclo while incorrectamente, de modo que se reproduzca la cantidad incorrecta de veces. Puede escribir un bucle que nunca salga, un error semántico conocido como bucle infinito. Para detener un programa atascado en un bucle infinito, puede presionar CTRL-C en la concha interactiva.

De hecho, cree un ciclo infinito ingresando este código en el shell interactivo (recuerde presionar ENTER dos veces para dejar que el

shell interactivo sabe que ha terminado de escribir en el bloque while):

```
>>> mientras es Verdadero:  
     print('Presiona Ctrl-C para detener este bucle infinito!!!')
```

Ahora presione y mantenga presionado CTRL-C para detener el programa. El shell interactivo se verá así:

```
¡Presiona Ctrl-C para detener este ciclo infinito!  
Rastreo (llamadas recientes más última):  
Archivo "<pyshell#1>", línea 2, en <módulo>  
print('¡Presiona Ctrl-C para detener este bucle infinito!')  
Archivo "C:\Program Files\Python 3.5\lib\idlelib\PyShell.py", línea 1347, en  
escribe  
    devolver self.shell.write(s, self.tags)  
Interrupción del teclado
```

El ciclo while siempre es verdadero, por lo que el programa continuará imprimiendo la misma línea para siempre hasta que el usuario lo detenga. En este ejemplo, presionamos CTRL-C para detener el bucle infinito después de que el bucle while se hubiera ejecutado cinco veces.

EL DEPURADOR

Puede ser difícil averiguar el origen de un error porque las líneas de código se ejecutan rápidamente y los valores de las variables cambian muy a menudo. Un depurador es un programa que le permite recorrer su código una línea a la vez en el mismo orden en que Python ejecuta cada instrucción. El depurador también le muestra qué valores se almacenan en las variables en cada paso.

Inicio del depurador En IDLE,
abra el juego Dragon Realm que creó en el Capítulo 5. Después de abrir
el archivo dragon.py , haga clic en el shell interactivo y luego haga clic en
Depurar depurador para most~~ar~~ la ventana de control de depuración
(Figura 6-1).

Cuando se ejecuta el depurador, la ventana de control de depuración
se verá como la Figura 6-2. Asegúrese de seleccionar las casillas de
verificación Stack, Locals, Source y Globals .

Ahora, cuando ejecute el juego Dragon Realm presionando F5, el
depurador de IDLE se activará. A esto se le llama ejecutar un programa
bajo un depurador. Cuando ejecuta un programa Python bajo el depurador,
el programa se detendrá antes de ejecutar la primera instrucción. Si hace
clic en la barra de título del editor de archivos (y ha seleccionado la casilla
de verificación Fuente en la ventana Control de depuración), la primera
instrucción se resalta en gris. La ventana de control de depuración muestra
que la ejecución está en la línea 1, que es la línea aleatoria de importación .

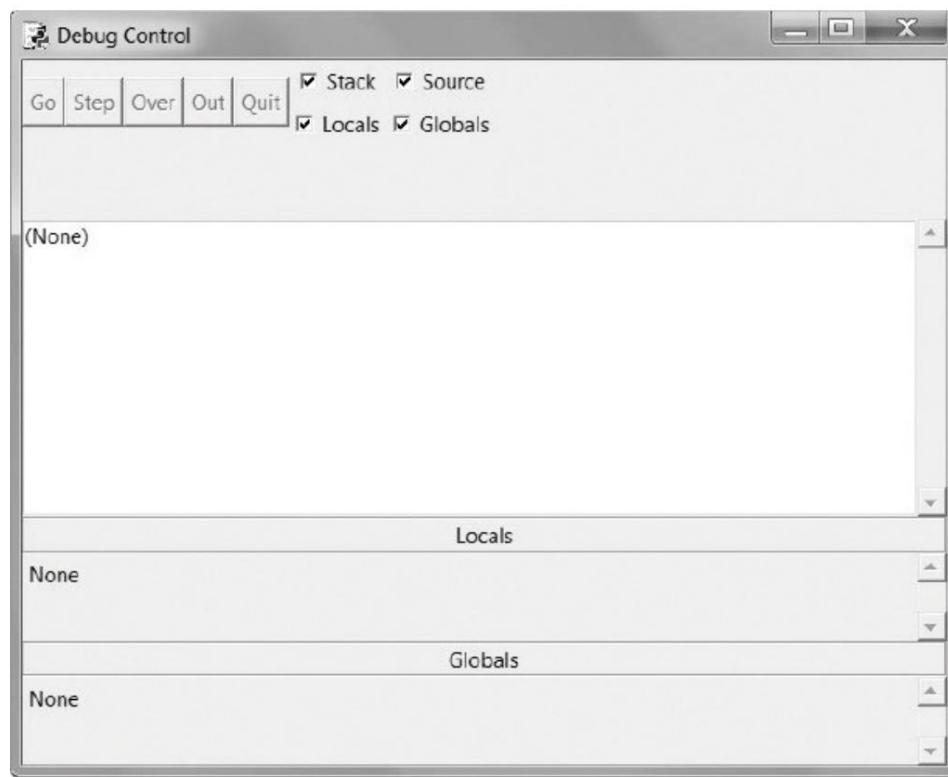


Figura 6-1: La ventana de control de depuración

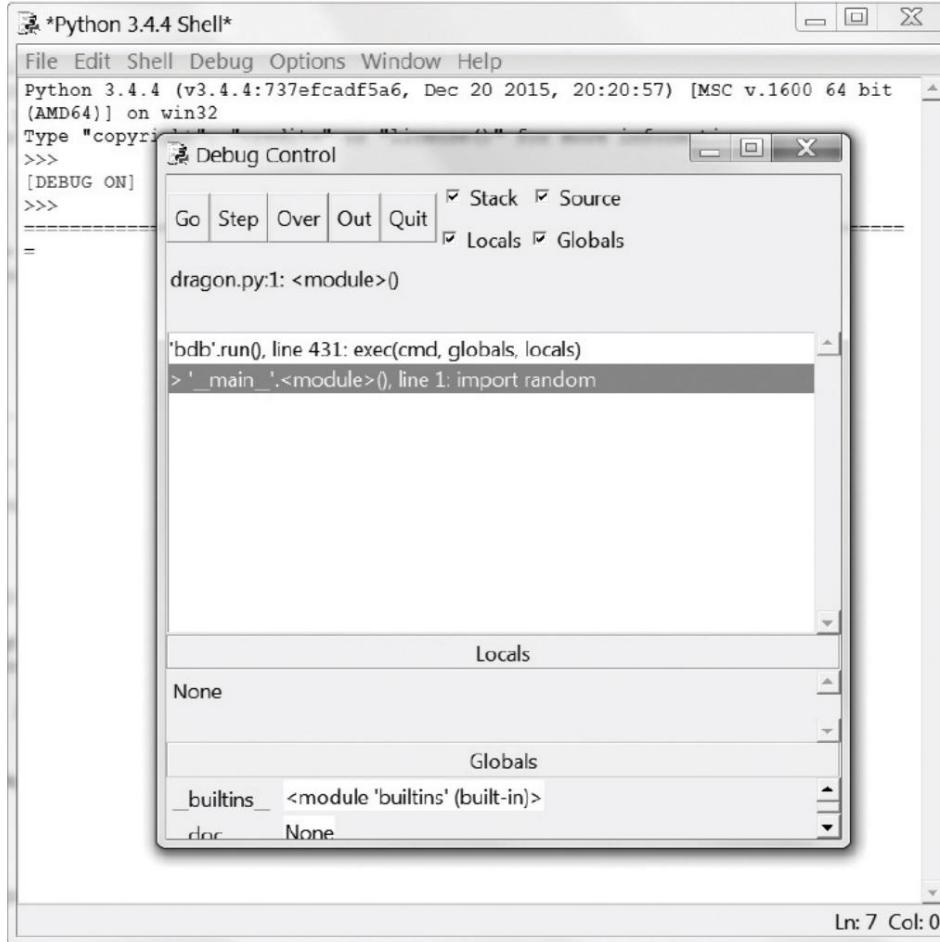


Figura 6-2: Ejecutando el juego Dragon Realm bajo el depurador

Pasos por el programa con el depurador El depurador le permite ejecutar una instrucción a la vez; este proceso se llama paso a paso. Ejecute una sola instrucción ahora haciendo clic en el botón Paso en la ventana Control de depuración. Python ejecutará la instrucción aleatoria de importación y luego se detendrá antes de ejecutar la siguiente instrucción. La ventana Control de depuración le muestra qué línea está a punto de ejecutarse cuando hace clic en el botón Paso, por lo que la ejecución ahora debería estar en la línea 2, la línea de tiempo de importación . Haga clic en el botón Salir para terminar el programa por ahora.

Este es un resumen de lo que sucede cuando haces clic en el

Botón de paso mientras ejecuta Dragon Realm bajo un depurador. Presione F5 para comenzar a ejecutar Dragon Realm nuevamente y luego siga estos instrucciones:

1. Haga clic en el botón Paso dos veces para ejecutar las dos líneas de importación.
2. Haga clic en el botón Paso tres veces más para ejecutar las tres declaraciones de definición.
3. Vuelva a hacer clic en el botón Paso para definir la variable reproducir de nuevo.
4. Haga clic en Ir para ejecutar el resto del programa o haga clic en Salir para terminar el programa.

programa.

El depurador omitió la línea 3 porque es una línea en blanco.

Tenga en cuenta que solo puede avanzar con el depurador; no puedes retroceder.

Área de Globales

El área Globales en la ventana Control de depuración es donde se muestran todas las variables globales. Recuerde, las variables globales son variables creadas fuera de cualquier función (es decir, en el ámbito global).

El texto junto a los nombres de las funciones en el área Globales parece "<función checkCave en 0x012859B0>". Los nombres de los módulos también tienen texto confuso junto a ellos, como "<módulo 'aleatorio' de 'C:\\Python31\\lib\\random.pyc'>". No necesita saber qué significa este texto para depurar sus programas. Con solo ver si las funciones y los módulos están en el área Globals decirle si se ha definido una función o si se ha importado un módulo.

También puede ignorar __builtins__, __doc__, __name__ y otras líneas similares en el área Globals.
(Esas son variables que aparecen en todos los programas de Python).

En el programa Dragon Realm, las tres declaraciones de definición ,

que ejecutan y definen funciones, aparecerán en el área Globales de la ventana Control de depuración. Cuando se crea la variable jugar de nuevo, también aparecerá en el área Globales. Junto al nombre de la variable estará la cadena 'sí'. El depurador le permite ver los valores de todas las variables en el programa mientras se ejecuta el programa. Esto es útil para corregir errores.

Área de locales

Además del área Globales, hay un área Locales, que le muestra las variables de ámbito local y sus valores. El área Locales contendrá variables solo cuando la ejecución del programa esté dentro de una función. Cuando la ejecución está en el alcance global, esta área está en blanco.

Los botones Ir y Salir

Si se cansa de hacer clic en el botón Paso repetidamente y solo desea que el programa se ejecute normalmente, haga clic en el botón Ir en la parte superior de la ventana Control de depuración. Esto le indicará al programa que se ejecute normalmente en lugar de paso a paso.

Para finalizar el programa por completo, haga clic en el botón Salir en la parte superior de la ventana Control de depuración. El programa saldrá inmediatamente. Esto es útil si debe comenzar a depurar nuevamente desde el principio del programa.

Entrar, pasar y salir

Inicie el programa Dragon Realm con el depurador. Siga avanzando hasta que el depurador esté en la línea 37. Como se muestra en la Figura 6-3, esta es la línea con `displayIntro()`. Cuando haga clic en Paso nuevamente, el depurador saltará a esta llamada de función y aparecerá en

línea 5, la primera línea en la función `displayIntro()` . Este tipo de paso, que es lo que has estado haciendo hasta ahora, se llama entrar.

Cuando la ejecución se detenga en la línea 5, querrá dejar de dar pasos. Si hizo clic en Paso una vez más, el depurador entraría en la función `imprimir()` . La función `print()` es una de las funciones integradas de Python, por lo que no es útil seguir paso a paso con el depurador. Las propias funciones de Python, como `print()`, `input()`, `str()` y `randint()`, se han revisado cuidadosamente para detectar errores.

Puede asumir que no son las partes que causan errores en su programa.

No querrás perder el tiempo repasando los aspectos internos de la función `print()` . Entonces, en lugar de hacer clic en Paso a paso en el código de la función `print()` , haga clic en Sobre. Esto pasará por encima del código dentro de la función `print()` . El código dentro de `print()` se ejecutará a velocidad normal, y luego el depurador se detendrá una vez que la ejecución regrese de `print()`.

Pasar por alto es una forma conveniente de omitir paso a paso por el código dentro de una función. El depurador ahora se detendrá en la línea 38, número de cueva = `elegirCueva()` .

Vuelva a hacer clic en Paso para acceder a la función `elegirCueva()` . Siga recorriendo el código hasta la línea 15, la llamada `input()` . El programa esperará hasta que escriba una respuesta en el shell interactivo, como cuando ejecuta el programa normalmente. Si intenta hacer clic en el botón Paso ahora, no pasará nada porque el programa está esperando una respuesta del teclado.

The screenshot shows a Windows-style application window titled "dragon.py - C:\nostarchinvent\dragon.py (3.4.4)". The main pane displays the Python code for a game. The code defines several functions: `displayIntro()`, `chooseCave()`, `checkCave()`, and a loop that calls `displayIntro()`, `chooseCave()`, and `checkCave()` repeatedly until the user types 'no'. The line `displayIntro()` at line 37 is highlighted with a red rectangle.

The "Debug Control" window is open, showing the current stack trace:

```

dragon.py:37: <module>()
'bdb'.run(), line 431: exec(cmd, globals, locals)
> '_main_'.<module>(), line 37: displayIntro()

```

The "Locals" and "Globals" panes are visible, both showing "None".

Figura 6-3: Siga pisando hasta llegar a la línea 37.

Vuelva a hacer clic en el caparazón interactivo y escriba a qué cueva desea ingresar. El cursor parpadeante debe estar en la línea inferior del shell interactivo antes de poder escribir. De lo contrario, el texto que escriba no aparecerá.

Una vez que presione ENTER, el depurador continuará con el paso a través de líneas de código de nuevo.

A continuación, haga clic en el botón Salir de la ventana Control de depuración. Esto se denomina paso a paso porque hará que el depurador pase por encima de tantas líneas como sea necesario hasta que la ejecución haya regresado de la función en la que se encuentra. Después de que salte, la ejecución estará en la línea posterior a la que llamó. la

función.

Si no está dentro de una función, al hacer clic en Salir, el depurador ejecutará todas las líneas restantes del programa.

Este es el mismo comportamiento que ocurre cuando hace clic en el botón Ir.

Aquí hay un resumen de lo que hace cada botón:

Ir Ejecuta el resto del código con normalidad o hasta que alcanza un punto de interrupción (consulte “Configuración de puntos de interrupción” en la página 73).

Paso Ejecuta una instrucción o un paso. Si la línea es una llamada de función, el depurador entrará en la función.

Over Ejecuta una instrucción o un paso. Si la línea es una llamada de función, el depurador no ingresará a la función, sino que la ignorará .

Out Sigue saltando líneas de código hasta que el depurador abandona la función en la que estaba cuando se hizo clic en Out. Esto sale de la función.

Salir Finaliza inmediatamente el programa.

Ahora que sabemos cómo usar el depurador, intentemos encontrar errores en algunos programas.

ENCONTRAR EL ERROR

El depurador puede ayudarlo a encontrar la causa de los errores en su programa. Como ejemplo, aquí hay un pequeño programa con un error. El programa presenta un problema de suma aleatoria para que el usuario lo resuelva. En el shell interactivo, haga clic en Archivo Nueva ventana para abrir una nueva ventana del editor de archivos. Ingresa a este programa

en esa ventana y guárdelo como `buggy.py`.

`buggy.py`

```
1. importar aleatorio
2. numero1 = aleatorio.randint(1, 10) 3.
numero2 = aleatorio.randint(1, 10) 4.
print('Que es ' + str(numero1) + '+ ' + str(numero2) + '?')

6. si respuesta == número1 + número2:
7. print('¡Correcto!') 8.
else:
9. print('¡No! La respuesta es ' + str(numero1 + número2))
```

Escriba el programa exactamente como se muestra, incluso si ya sabe cuál es el error. Luego ejecute el programa presionando F5. Así es como podría verse cuando ejecuta el programa:

¿Cuánto es 5 + 1?

6

¡No! la respuesta es 6

¡Eso es un error! El programa no falla, pero no funciona correctamente. El programa dice que el usuario está equivocado incluso si ingresa la respuesta correcta.

Ejecutar el programa bajo un depurador ayudará a encontrar el causa del error. En la parte superior del shell interactivo, haga clic en Depurar depurador▶ para mostrar la ventana Control de depuración. (Asegúrese de haber marcado las casillas de verificación Stack, Source, Locals y Globals). Luego presione F5 en el editor de archivos para ejecutar el programa. Esta vez se ejecutará bajo el depurador.

El depurador comienza en la línea aleatoria de importación :

1. importar al azar

No sucede nada especial aquí, así que simplemente haga clic en Paso para ejecutar

eso. Verá el módulo aleatorio agregado al área Globals.

Haga clic en Paso nuevamente para ejecutar la línea 2:

2. numero1 = aleatorio.randint(1, 10)

Aparecerá una nueva ventana del editor de archivos con el archivo random.py . Ha entrado en la función randint() dentro del módulo aleatorio . Sabe que las funciones integradas de Python no serán la fuente de sus errores, así que haga clic en Salir para salir de la función randint() y volver a su programa. Luego cierre la ventana del archivo random.py . La próxima vez, puede hacer clic en Sobre para pasar por encima de la función randint() en lugar de entrar en ella.

La línea 3 también es una llamada a la función randint() :

3. numero2 = aleatorio.randint(1, 10)

Omita entrar en este código haciendo clic en Sobre.

La línea 4 es una llamada print() para mostrarle al jugador el azar números:

4. print('¿Qué es ' + str(número1) + ' + ' + str(número2) + '?')

¡Usted sabe qué números imprimirá el programa incluso antes de que los imprima! Solo mire el área Globales de la ventana Control de depuración. Puede ver las variables número1 y número2 , y junto a ellas están los valores enteros almacenados en esas variables.

La variable número1 tiene el valor 4 y la variable número2 tiene el valor 8. (Sus números aleatorios probablemente serán diferentes). Cuando haga clic en Paso, la función str() concatenará la versión de cadena de estos enteros, y el programa mostrará la cadena en la llamada print() con estos

valores. Cuando ejecuté el depurador, se parecía a la Figura 6-4.

Haga clic en Paso desde la línea 5 para ejecutar input().

5. respuesta = entrada()

El depurador espera hasta que el jugador ingresa una respuesta en el programa. Ingrese la respuesta correcta (en mi caso, 12) en el shell interactivo. El depurador se reanudará y bajará a la línea 6:

6. si respuesta == número1 + número2:

7. imprimir('¡Correcto!')

La línea 6 es una declaración if . La condición es que el valor en respuesta debe coincidir con la suma de número1 y número2. Si el Si la condición es verdadera, el depurador se moverá a la línea 7. Si la condición es falsa, el depurador se moverá a la línea 9. Haga clic en Paso una vez más para averiguar a dónde va.

8. más:

9. print('¡No! La respuesta es ' + str(número1 + número2))

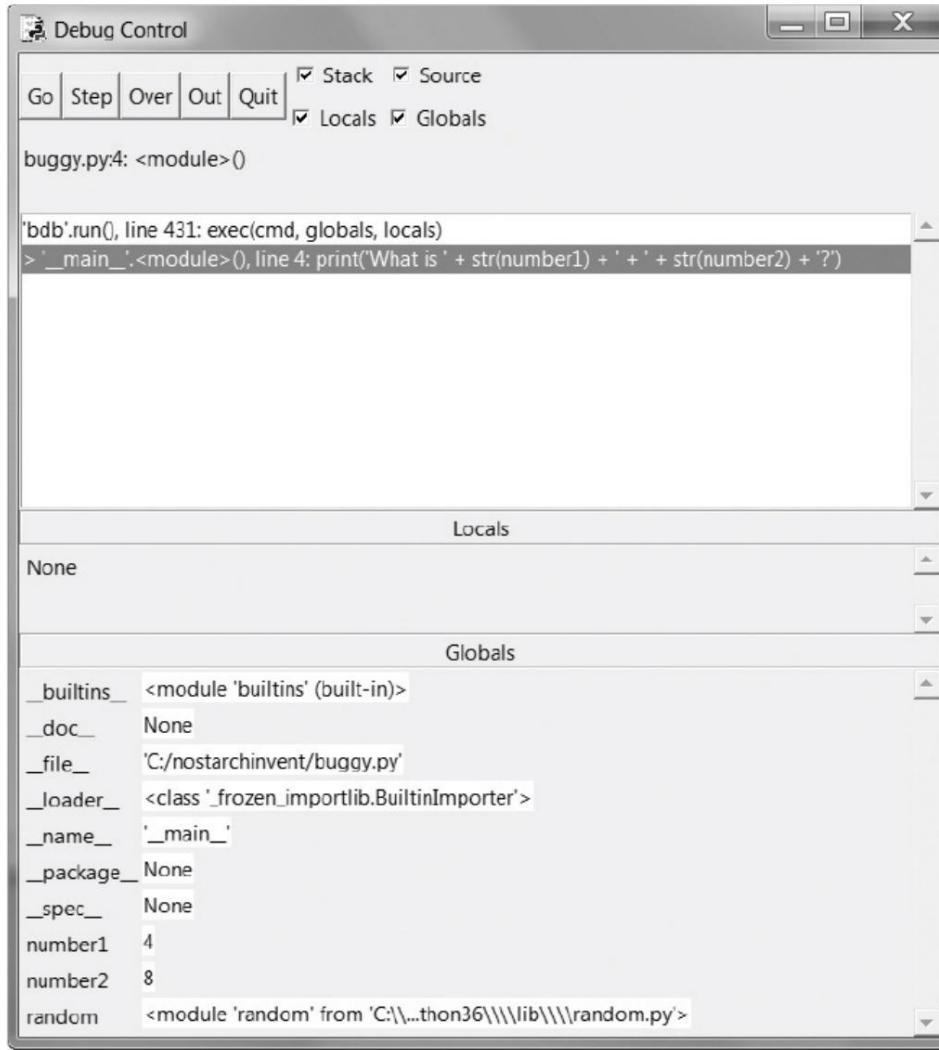


Figura 6-4: el número 1 se establece en 4 y el número 2 se establece en 8.

¡El depurador ahora está en la línea 9! ¿Qué sucedió? La condición en la instrucción if debe haber sido Falsa. mira el valores para número1 , número2 y respuesta. Observe que el número 1 y el número 2 son números enteros, por lo que su suma también habría sido un entero. Pero la respuesta es una cadena.

Eso significa que la respuesta == número1 + número2 tendría evaluado a '12' == 12. Un valor de cadena y un valor entero nunca serán iguales entre sí, por lo que la condición evaluada a Falso.

Ese es el error del programa: el código usa answer cuando debería haber usado int(answer). Cambie la línea 6 a int(respuesta) == número1 + número2 y vuelva a ejecutar el programa.

¿Cuánto es 2 + 3?

5

¡Correcto!

Ahora el programa funciona correctamente. Ejecútelo una vez más e ingrese una respuesta incorrecta a propósito. ¡Ya ha depurado este programa! Recuerde, la computadora ejecutará sus programas exactamente como los escribe, incluso si lo que escribe no es lo que pretende.

CONFIGURACIÓN DE PUNTOS DE RUPTURA

Recorrer el código una línea a la vez puede resultar demasiado lento. A menudo querrá que el programa se ejecute a velocidad normal hasta que llegue a cierta línea. Puede establecer un punto de interrupción en una línea cuando desee que el depurador tome el control una vez que la ejecución llegue a esa línea. Si cree que hay un problema con su código en, digamos, la línea 17, simplemente establezca un punto de interrupción en esa línea (o tal vez unas pocas líneas antes).

Cuando la ejecución llega a esa línea, el programa entrará en el depurador. Luego puede pasar por las líneas para ver lo que está sucediendo. Al hacer clic en Ir, el programa se ejecutará normalmente hasta que llegue a otro punto de interrupción o al final del programa.

Para establecer un punto de interrupción en Windows, haga clic con el botón derecho en la línea en el editor de archivos y seleccione Establecer punto de interrupción en el menú que aparece. En OS X, pulse CTRL y haga clic para acceder a un menú y seleccione Establecer

Punto de ruptura. Puede establecer puntos de interrupción en tantas líneas como desee. El editor de archivos resalta cada línea de punto de interrupción en amarillo. La figura 6-5 muestra un ejemplo de cómo se ve un punto de interrupción.

```
dragon.py - C:\nostarchinvent\dragon.py (3.4.4)
File Edit Format Run Options Window Help
import random
import time

def displayIntro():
    print('''You are in a land full of dragons. In front of you,
you see two caves. In one cave, the dragon is friendly
and will share his treasure with you. The other dragon
is greedy and hungry, and will eat you on sight.'''')
    print()

def chooseCave():
    cave = ''
    while cave != '1' and cave != '2':
        print('Which cave will you go into? (1 or 2)')
        cave = input()

    return cave

Ln: 15 Col: 17
```

Figura 6-5: El editor de archivos con dos puntos de interrupción establecidos

Para eliminar el punto de interrupción, haga clic en la línea y seleccione Borrar Breakpoint en el menú que aparece.

USO DE PUNTOS DE INTERRUPCIÓN

A continuación, veremos un programa que llama a `random.randint(0, 1)` para simular lanzamientos de monedas. Si la función devuelve el entero 1, será cara, y si devuelve el entero 0, será cruz.

La variable de lanzamientos rastrearía cuántos lanzamientos de monedas se han hecho. La variable cabezas rastrearía cuántas caras salieron.

El programa hará lanzamientos de monedas 1,000 veces. Esto le tomaría a una persona más de una hora, ¡pero la computadora puede hacerlo en un segundo! No hay ningún error en este programa, pero el depurador nos permitirá ver el estado del programa mientras se ejecuta.

Ingrese el siguiente código en el editor de archivos y guárdelo como `coinFlips.py`. Si obtiene errores después de ingresar este código,

compare el código que escribió con el código del libro con la herramienta de diferencias en línea en <https://www.nostarch.com/inventwithpython#diff>.

CoinFlips.py

```

1. import random
print('Lanzaré una moneda 1000 veces. Adivina cuántas veces saldrá
cabezas (Presione enter para
comenzar)')
3. input()
4. volteas = 0

5. cabezas = 0
6. while volteas < 1000:
    if random.randint(0, 1) == 1:
        caras =
        caras + 1
    9.     volteas = volteas + 1
10
11    if volteas == 900:
12        print('900 volteas y ha habido ' + str(caras) + ' cara.')
13. si volteas == 100:
14        print('En 100 lanzamientos, ha salido cara hasta ' + str(caras) + ' veces
ahora.')
15. if volteas == 500:
    print('Hasta la mitad, y ha salido cara ' + str(caras) + ' veces.')
    print('Hasta la mitad, y ha salido cara ' + str(cabezas) + ' veces.')
17
18. print()
19. print('De 1000 lanzamientos de monedas, salió cara ' + str(caras) + ' veces!')
print('¿Estuvo cerca?')

```

El programa corre bastante rápido. Pasa más tiempo esperando que el usuario presione ENTER que lanzando la moneda. Digamos que querías verlo hacer lanzamientos de monedas uno por uno. En la ventana del shell interactivo, haga clic en Depurar depurador para ~~abrir~~ la ventana Control de depuración. Luego presione F5 para ejecutar el programa.

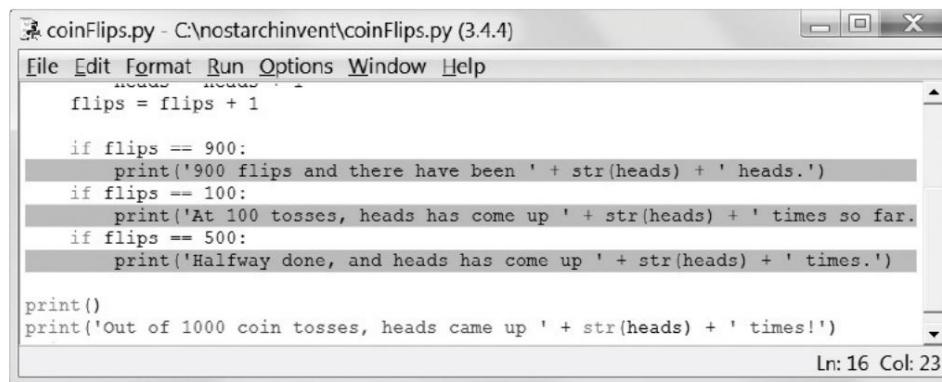
El programa se inicia en el depurador en la línea 1. Presione Paso tres veces en la ventana Control de depuración para ejecutar el primero.

tres líneas (es decir, las líneas 1, 2 y 3). Notará que los botones se deshabilitan porque se llamó a `input()` y el shell interactivo está esperando que el usuario escriba algo. Haga clic en el caparazón interactivo y presione ENTRAR. (Asegúrese de hacer clic debajo del texto en el shell interactivo; de lo contrario, es posible que IDLE no reciba sus pulsaciones de teclas).

Puede hacer clic en Paso unas cuantas veces más, pero se dará cuenta de que le llevará bastante tiempo completar todo el programa.

En su lugar, establezca un punto de interrupción en las líneas 12, 14 y 16 para que el depurador se interrumpa cuando los cambios sean iguales a 900, 100 y 500, respectivamente. El editor de archivos resaltará estas líneas como se muestra en la

Figura 6-6.



```
coinFlips.py - C:\nostarchinvent\coinFlips.py (3.4.4)
File Edit Format Run Options Window Help
heads heads
flips = flips + 1

if flips == 900:
    print('900 flips and there have been ' + str(heads) + ' heads.')
if flips == 100:
    print('At 100 tosses, heads has come up ' + str(heads) + ' times so far.')
if flips == 500:
    print('Halfway done, and heads has come up ' + str(heads) + ' times.')

print()
print('Out of 1000 coin tosses, heads came up ' + str(heads) + ' times!')
Ln: 16 Col: 23
```

Figura 6-6: Tres puntos de interrupción establecidos en coinflips.py

Después de configurar los puntos de interrupción, haga clic en Ir en la ventana Control de depuración. El programa se ejecutará a velocidad normal hasta que alcance el siguiente punto de interrupción. Cuando flip se establece en 100, el la condición para la instrucción if en la línea 13 es verdadera. Esto hace que la línea 14 (donde hay un punto de interrupción establecido) para ejecutar, lo que le dice al depurador que detenga el programa y se haga cargo. Mire el área Globales de la ventana de control de depuración para ver cuáles son los valores de `flips` y `heads`.

Haga clic en Ir de nuevo y el programa continuará hasta que

alcanza el siguiente punto de interrupción en la línea 16. Nuevamente, observe cómo han cambiado los valores en volteretas y caras .

Haga clic en Ir de nuevo para continuar la ejecución hasta el próximo se alcanza el punto de interrupción, que está en la línea 12.

RESUMEN

Escribir programas es solo la primera parte de la programación. La siguiente parte es asegurarse de que el código que escribió realmente funcione.

Los depuradores le permiten recorrer el código una línea a la vez.

Puede examinar qué líneas se ejecutan en qué orden y qué valores que contienen las variables. Cuando recorrer paso a paso línea por línea es demasiado lento, puede establecer puntos de interrupción para detener el depurador solo en las líneas que desee.

Usar el depurador es una excelente manera de comprender lo que está haciendo un programa. Si bien este libro brinda explicaciones de todo el código del juego que usamos, el depurador puede ayudarlo a descubrir más por su cuenta.

7

DISEÑO DEL AHORCADO CON DIAGRAMAS DE FLUJO



En este capítulo, diseñará un juego del ahorcado. Este juego es más complicado que nuestros juegos anteriores pero también más divertido. Debido a que el juego es avanzado, primero lo planearemos cuidadosamente creando un diagrama de flujo en este capítulo. En el Capítulo 8, escribiremos el código para Hangman.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Arte ASCII
- Diseño de un programa con diagramas de flujo

CÓMO JUGAR EL AHORCADO

Hangman es un juego para dos personas en el que un jugador piensa en una palabra y luego dibuja una línea en blanco en la página para cada letra de la palabra. Luego, el segundo jugador trata de adivinar las letras que podrían estar en la palabra.

Si el segundo jugador adivina la letra correctamente, el primer jugador escribe la letra en el espacio en blanco adecuado. Pero si el segundo jugador adivina incorrectamente, el primer jugador dibuja una sola parte del cuerpo de un hombre colgado. El segundo jugador tiene que adivinar todas las letras de la palabra antes de que el hombre colgado quede completamente dibujado para ganar el juego.

MUESTRA DE EJECUCIÓN DEL AHORCADO

Este es un ejemplo de lo que el jugador podría ver cuando ejecuta el programa Hangman que escribirá en el Capítulo 8. El texto que ingresa el jugador está en negrita.

VERDUGO

+---+

====

Letras perdidas:

Adivina una letra.

a

+---+

====

Letras perdidas:

_ a _
Adivina una letra.

Él

+---+

el ||

|

====

Letras perdidas: o

_ a _

Adivina una letra.

r

+---+

el | ||

|

====

Letras perdidas: o

_ a _

Adivina una letra.

t

+---+

el | ||

|

====

Letras perdidas: o

_ a

Adivina una letra.

a

Ya has adivinado esa letra. Elige de nuevo.

Adivina una letra.

c

¡Sí! ¡La palabra secreta es "gato"! ¡Usted ha ganado!

¿Quieres jugar de nuevo? (sí o no)

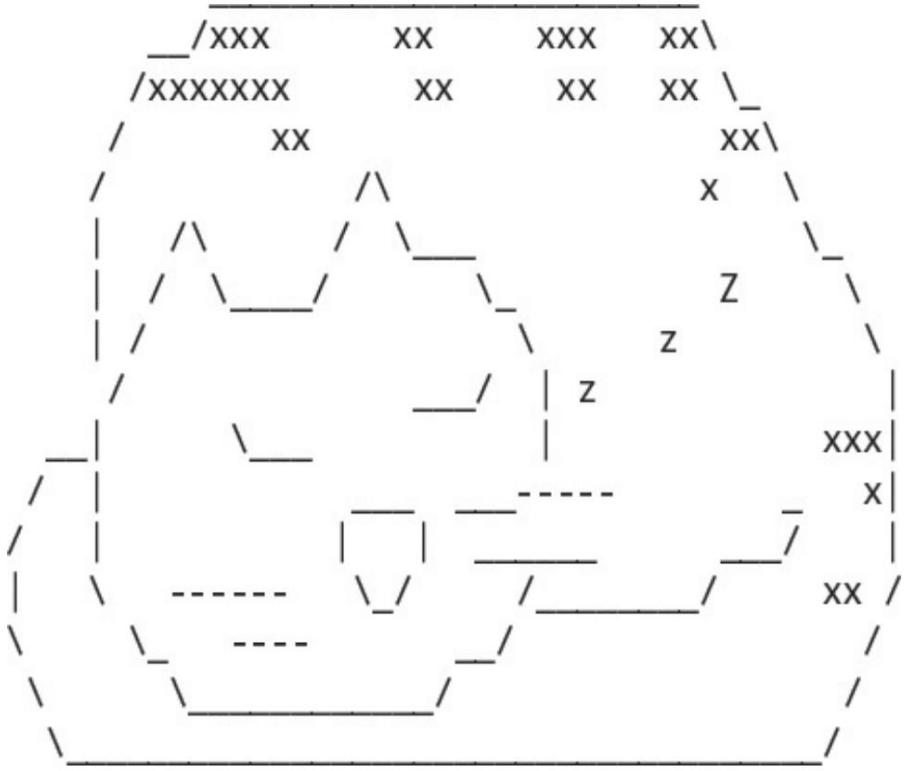
no

ARTE ASCII

Los gráficos de Hangman son caracteres del teclado impresos en la pantalla.

Este tipo de gráfico se llama arte ASCII (pronunciado ask-ee), que fue una especie de precursor de emoji.

Aquí hay un gato dibujado en arte ASCII:



Las imágenes del juego Hangman se verán así

Arte ASCII:

```
+---+ +---+ +---+ +---+ +---+ +---+
| el | el | el | el | el | | | | / | \ | \ | \ | \ |
|/\|  
||  
=====  
=====
```

DISEÑO DE UN PROGRAMA CON UNA DIAGRAMA DE FLUJO

Este juego es un poco más complicado que los que has visto hasta ahora, así que tomemos un momento para pensar en cómo se arma. Primero, creará un diagrama de flujo (como el de la Figura 5-1 en la página 47 para el juego Dragon Realm) para ayudar a visualizar lo que hará este programa.

Como se discutió en el Capítulo 5, un diagrama de flujo es un diagrama que

muestra una serie de pasos como cuadros conectados con flechas. Cada cuadro representa un paso, y las flechas muestran los posibles próximos pasos. Coloque su dedo en el cuadro de INICIO del diagrama de flujo y recorra el programa siguiendo las flechas a otros cuadros hasta que llegue al cuadro de FIN. Solo puedes moverte de un cuadro a otro en la dirección de la flecha. Tu nunca puedes

ir hacia atrás a menos que haya una flecha que vaya hacia atrás, como en el cuadro "El jugador ya adivinó esta letra".

La Figura 7-1 es un diagrama de flujo completo para Hangman.

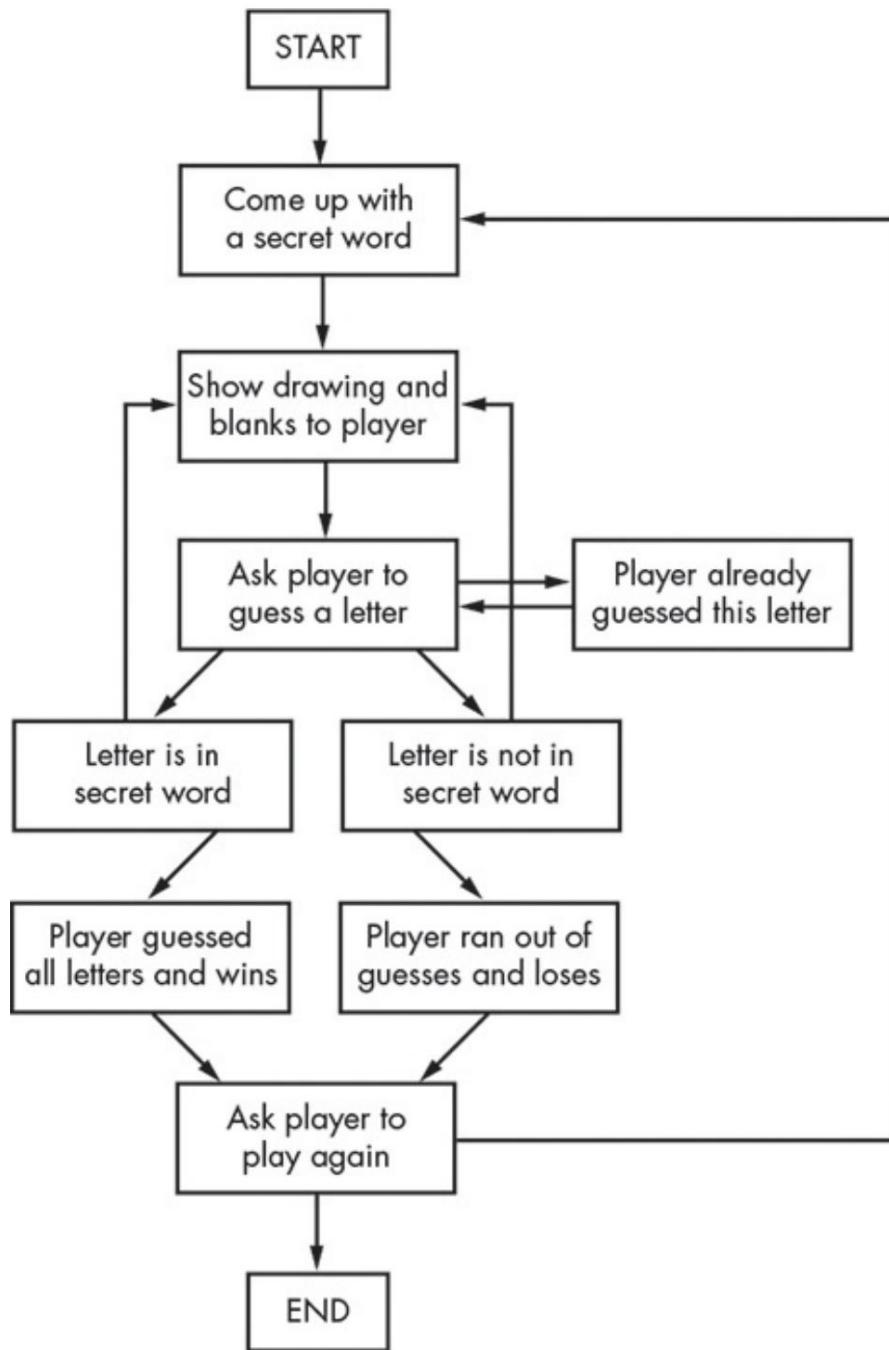


Figura 7-1: El diagrama de flujo completo para el juego Hangman

Por supuesto, no tienes que hacer un diagrama de flujo; podrías empezar a escribir código. Pero a menudo, una vez que comienza a programar, pensará en cosas que deben agregarse o cambiarse. Puede terminar teniendo que eliminar una gran parte de su código, lo que sería una pérdida de esfuerzo. Para evitar esto, es mejor planificar cómo el programa

funcionará antes de empezar a escribirlo.

Creación del diagrama de flujo Sus

diagramas de flujo no tienen que parecerse al de la Figura 7-1.

Siempre que comprenda su diagrama de flujo, será útil cuando comience a codificar.

Puede comenzar a hacer un diagrama de flujo con solo un cuadro de INICIO y FIN,

como se muestra en la Figura 7-2.

Ahora piensa en lo que sucede cuando juegas al Ahorcado.

Primero, la computadora piensa en una palabra secreta. Entonces el jugador

adivina las letras. Agregue cuadros para estos eventos, como se muestra en la

Figura 7-3. Los nuevos cuadros en cada diagrama de flujo tienen un contorno discontinuo.

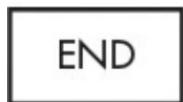


Figura 7-2: Comience su diagrama de flujo con un cuadro de INICIO y FIN.

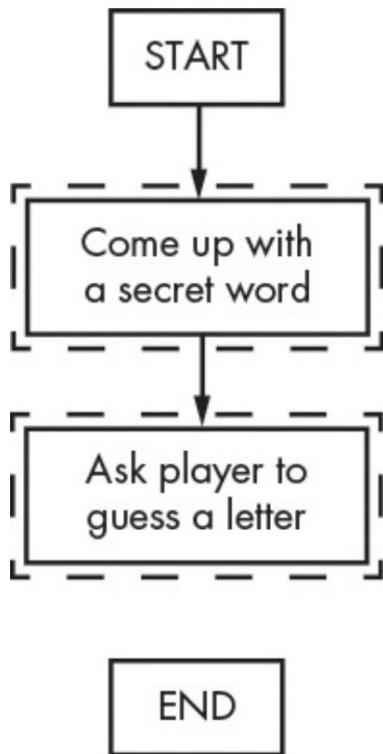


Figura 7-3: Dibuje los dos primeros pasos de Hangman como cuadros con descripciones.

Pero el juego no termina después de que el jugador adivina una letra. El programa necesita verificar si esa letra está en la palabra secreta.

Ramificación desde un cuadro de diagrama de flujo Hay dos posibilidades: la letra está en la palabra o no. Agregará dos cuadros nuevos al diagrama de flujo, uno para cada caso. Esto crea una rama en el diagrama de flujo, como se muestra en la Figura 7-4.

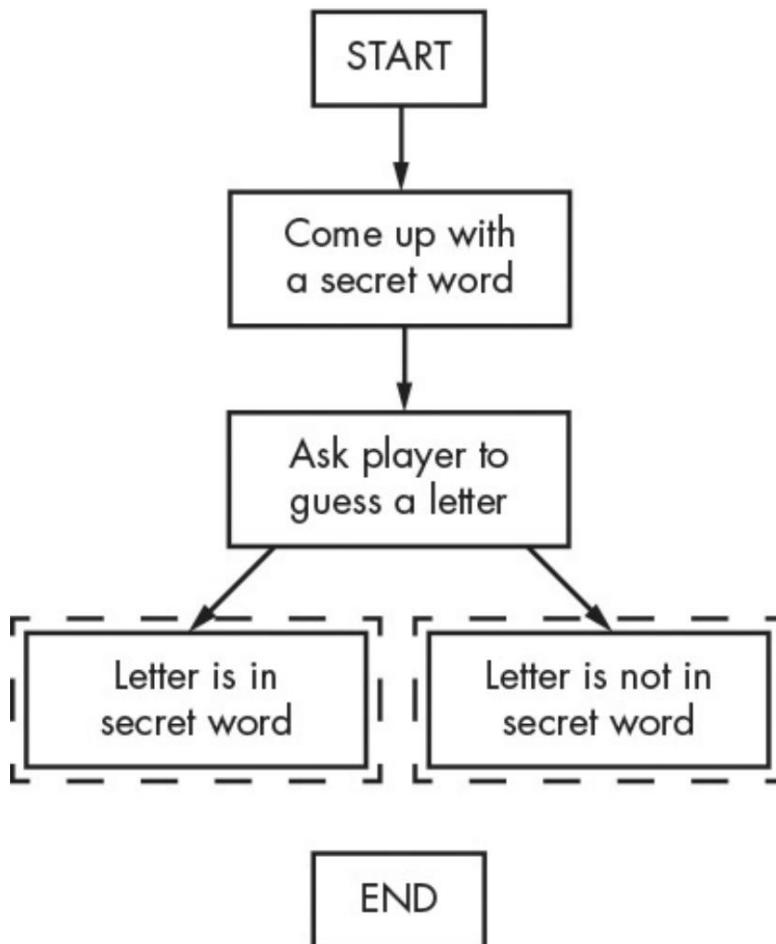


Figura 7-4: La rama tiene dos flechas que van a cajas separadas.

Si la letra está en la palabra secreta, comprueba si el jugador ha adivinado todas las letras y ha ganado el juego. Si la letra no está en la palabra secreta, comprueba si el hombre colgado está completo y si el jugador ha perdido. Agregue cajas para esos casos también.

El diagrama de flujo ahora se parece a la Figura 7-5.

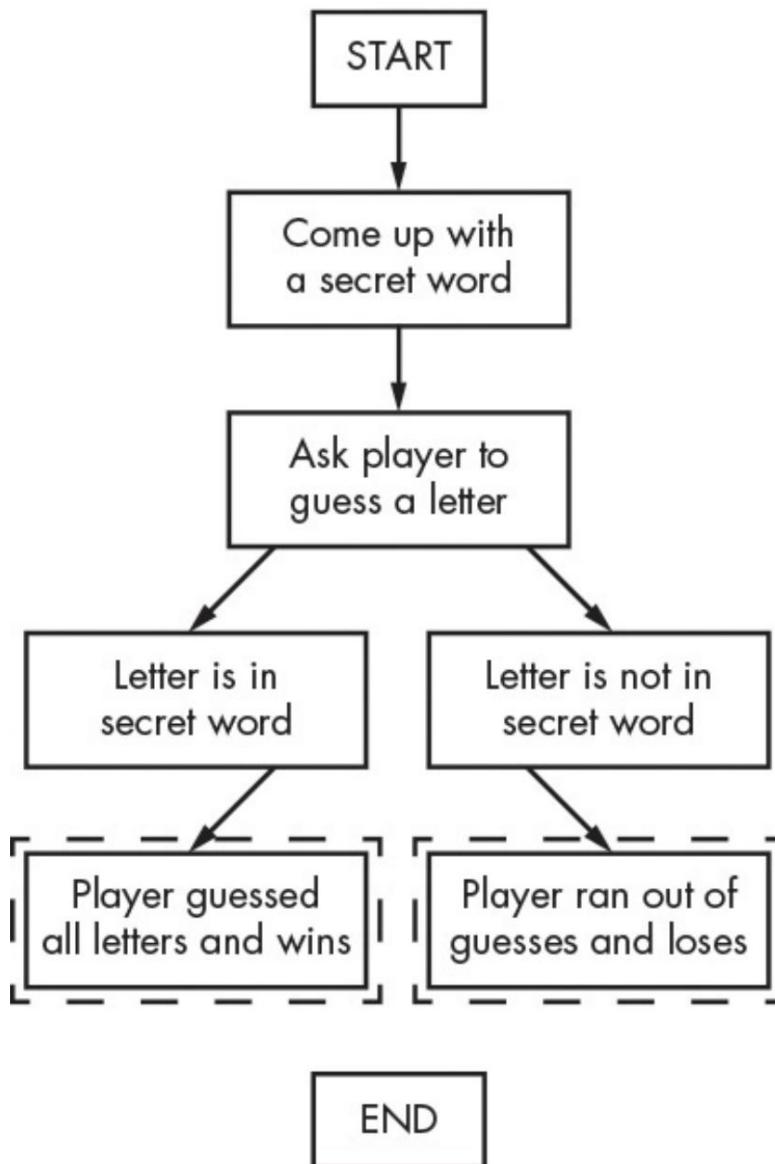


Figura 7-5: Después de la bifurcación, los pasos continúan en sus caminos separados.

No necesitas una flecha de la "Carta es en secreto palabra" al cuadro "El jugador se quedó sin conjeturas y pierde", porque es imposible que el jugador pierda si acaba de adivinar correctamente. También es imposible que el jugador gane si acaba de adivinar incorrectamente, por lo que tampoco es necesario dibujar una flecha para eso.

Terminar o reiniciar el juego

Una vez que el jugador haya ganado o perdido, pregúntale si quiere volver a jugar con una nueva palabra secreta. Si el jugador no quiere volver a jugar, el programa finaliza; de lo contrario, el programa continúa y piensa en una nueva palabra secreta. Esto se muestra en la Figura 7-6.

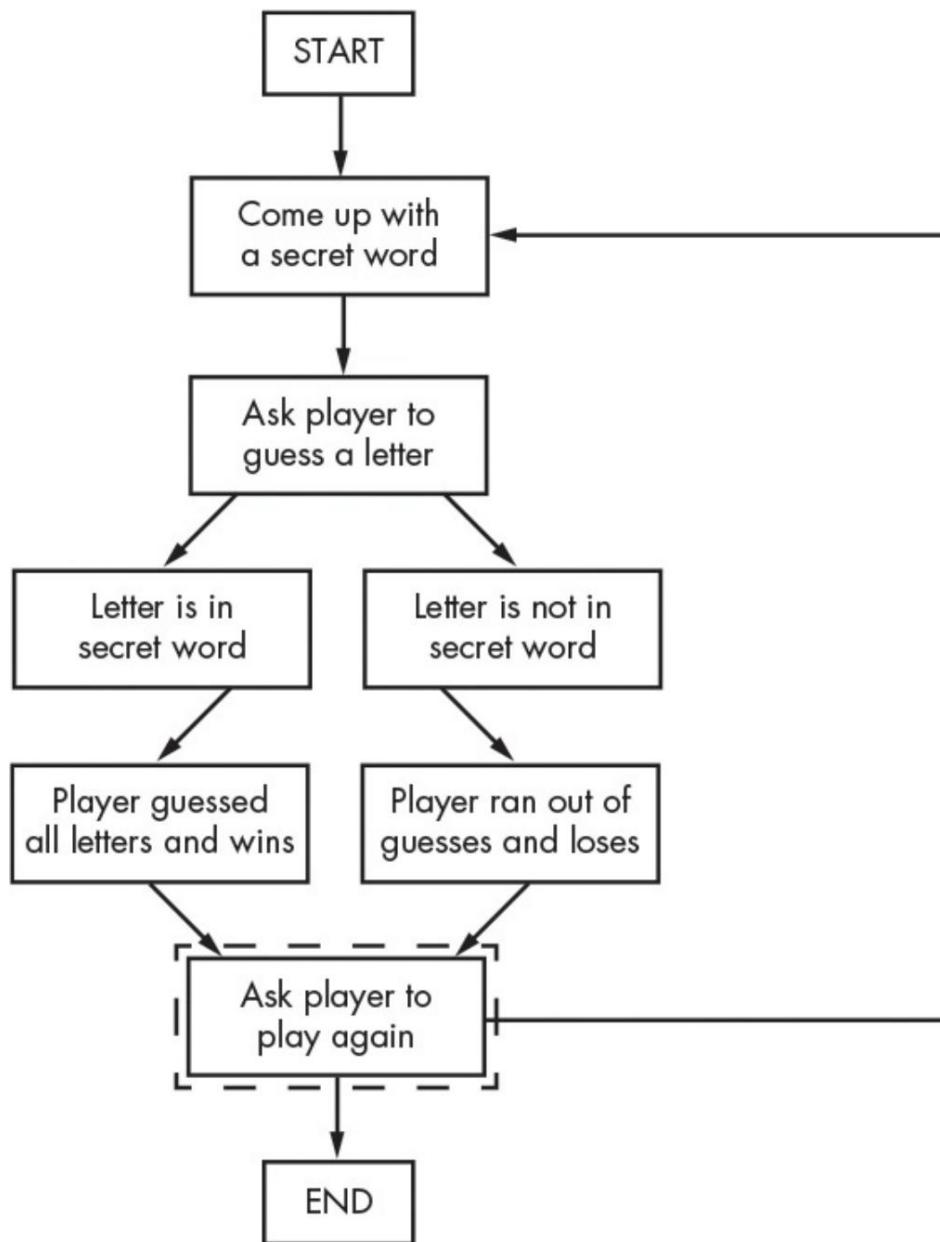


Figura 7-6: El diagrama de flujo se bifurca después de pedirle al jugador que vuelva a jugar.

Adivinando de nuevo

El diagrama de flujo parece casi completo ahora, pero todavía nos faltan algunas cosas. Por un lado, el jugador no adivina una letra solo una vez; siguen adivinando letras hasta que ganan o pierden.

Dibuja dos flechas nuevas, como se muestra en la Figura 7-7.

¿Qué pasa si el jugador vuelve a adivinar la misma letra? En lugar de contar esta letra nuevamente, permítale adivinar una letra diferente. Este nuevo cuadro se muestra en la Figura 7-8.

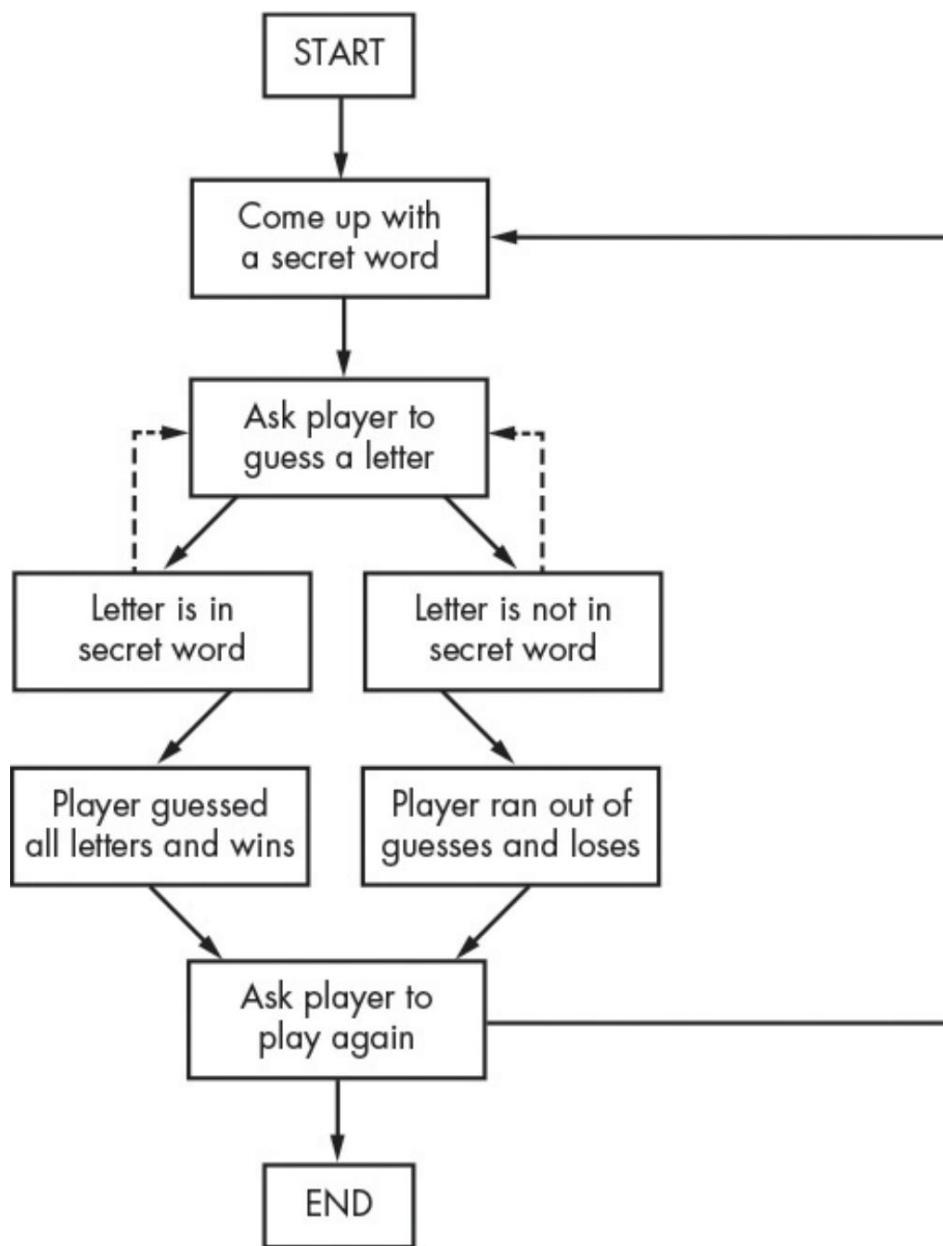


Figura 7-7: Las flechas discontinuas muestran que el jugador puede adivinar nuevamente.

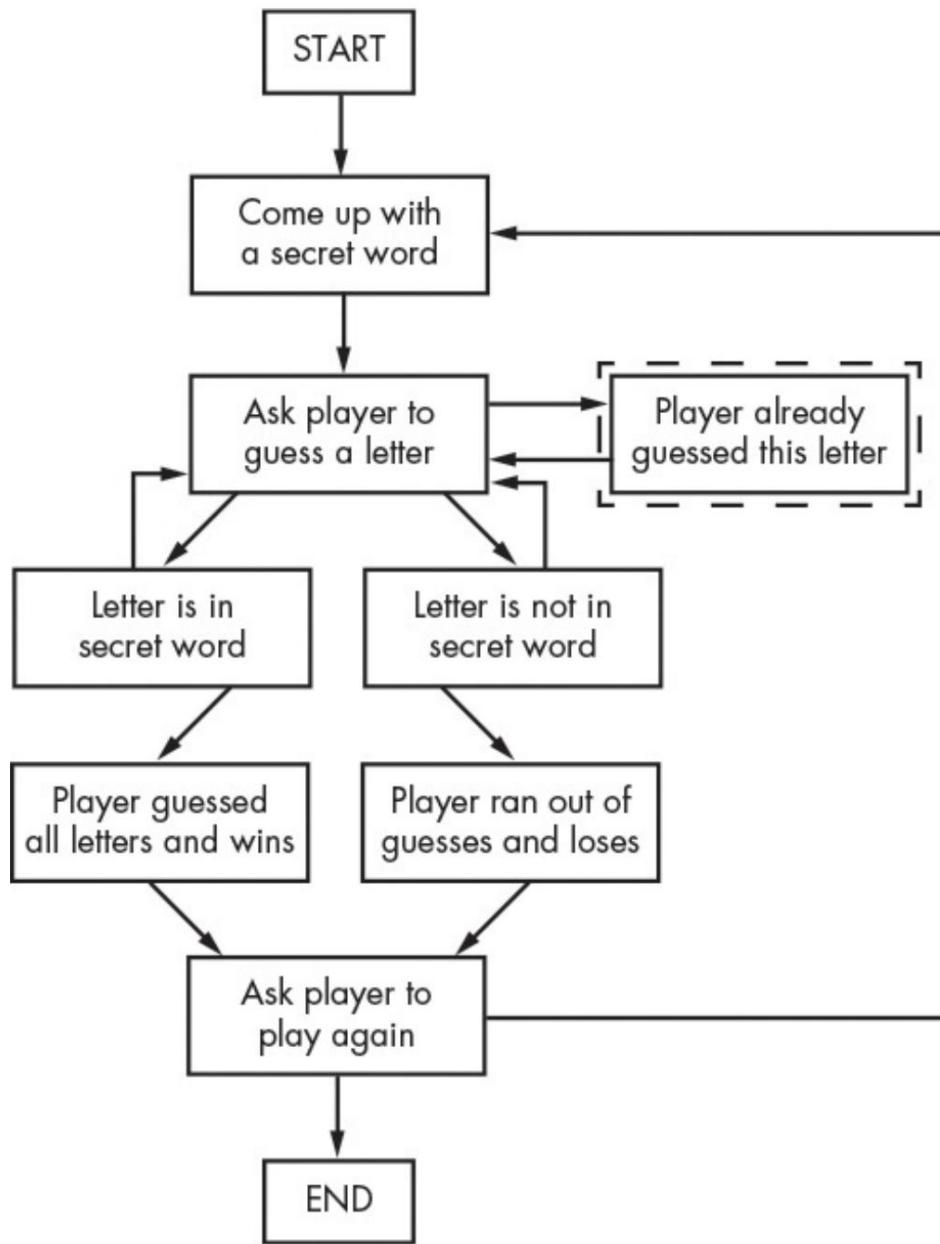


Figura 7-8: Agregue un paso en caso de que el jugador adivine una letra que ya adivinó.

Si el jugador adivina la misma letra dos veces, el diagrama de flujo lleva de vuelta al cuadro "Pedir al jugador que adivine una letra". Ofreciendo retroalimentación al jugador El jugador necesita saber cómo le está yendo en el juego. El programa debería mostrarles el dibujo del ahorcado y la palabra secreta (con espacios en blanco para las letras que no han adivinado)

todavía). Estas imágenes les permitirán ver qué tan cerca están de ganar o perder el juego.

Esta información se actualiza cada vez que el jugador adivina una letra. Agregue un cuadro "Mostrar dibujo y espacios en blanco al jugador" al diagrama de flujo entre el cuadro "Inventar una palabra secreta" y el cuadro "Pedir al jugador que adivine una letra", como se muestra en la Figura 7-9.

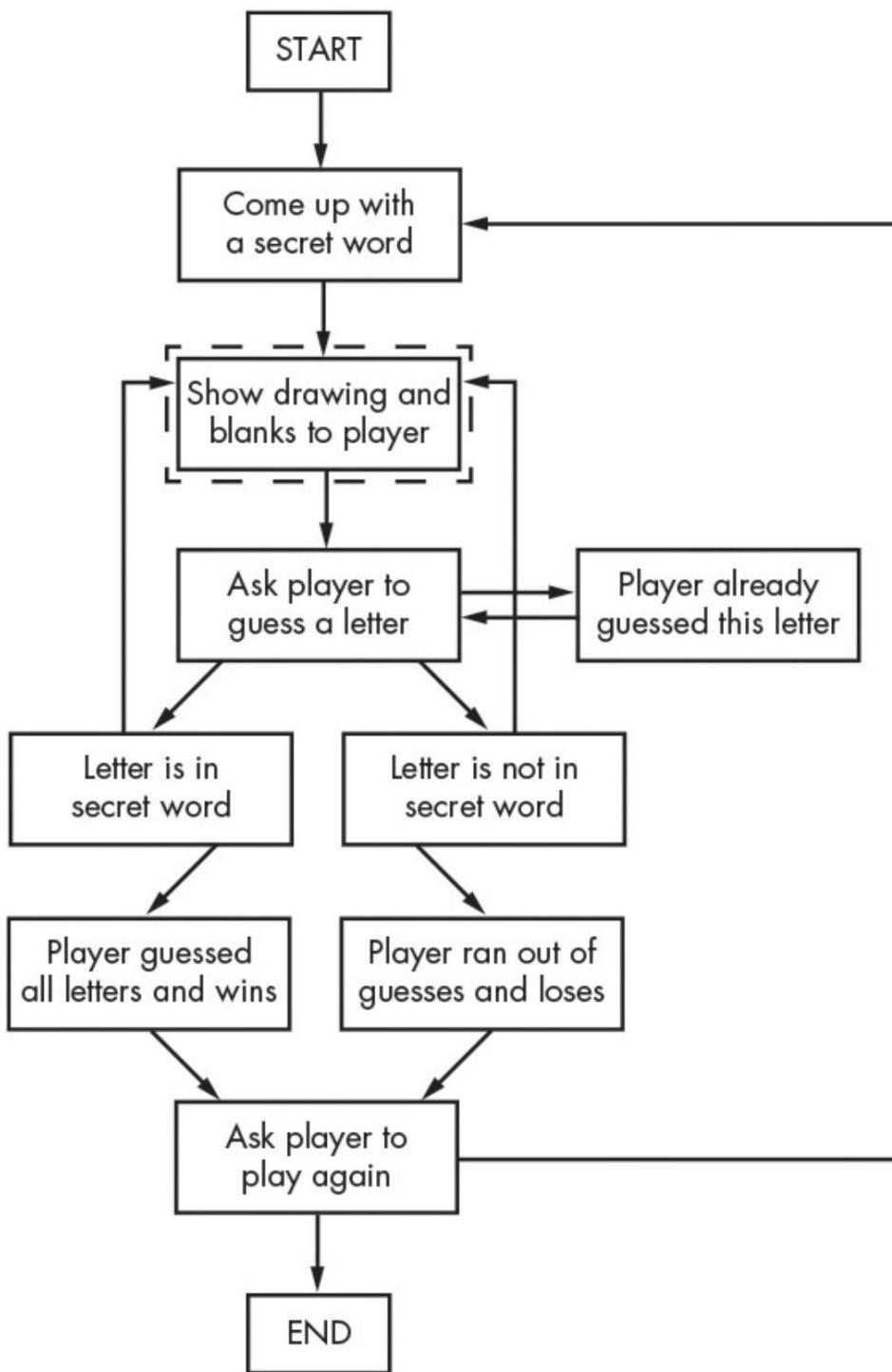


Figura 7-9: agregue un cuadro "Mostrar dibujo y espacios en blanco al jugador" para dar retroalimentación al jugador.

¡Eso se ve bien! Este diagrama de flujo muestra completamente el orden de todo lo que puede suceder en el juego Hangman.

Cuando diseñas tus propios juegos, un diagrama de flujo puede ayudarte a recordar todo lo que necesitas para codificar.

RESUMEN

Puede parecer mucho trabajo esbozar primero un diagrama de flujo sobre el programa. ¡Después de todo, la gente quiere jugar, no mirar diagramas de flujo! Pero es mucho más fácil hacer cambios e identificar problemas pensando en cómo funciona el programa antes de escribir el código para él.

Si se lanza a escribir el código primero, es posible que descubra problemas que requieran que cambie el código que ya ha escrito, perdiendo tiempo y esfuerzo. Y cada vez que cambia su código, corre el riesgo de crear nuevos errores al cambiar muy poco o demasiado. Es mucho más eficiente saber lo que quiere construir antes de construirlo. Ahora que tenemos un diagrama de flujo, ¡creemos el programa Hangman en el Capítulo 8!

8

ESCRIBIENDO EL VERDUGO CÓDIGO



El juego de este capítulo presenta muchos conceptos nuevos, pero no te preocunes: experimentarás con ellos en el shell interactivo antes de programar el juego. Aprenderá acerca de los métodos, que son funciones adjuntas a valores. También aprenderá sobre un nuevo tipo de datos llamado lista. Una vez que comprenda estos conceptos, será mucho más fácil programar Hangman.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Listas
- El operador de entrada
- Métodos
- Los métodos de cadena dividir(), inferior(), superior(), empieza con() y termina con()
- declaraciones elif

CÓDIGO FUENTE PARA EL AHORCADO

El juego de este capítulo es un poco más largo que los juegos anteriores,

pero gran parte es el arte ASCII para las imágenes del hombre colgado.

Ingrrese lo siguiente en el editor de archivos y guárdelo como hangman.py.

Si obtiene errores después de ingresar el siguiente código, compare el código que escribió con el código del libro con el código en línea

diferencia

herramienta

a

<https://www.nostarch.com/inventwithpython#diff>.



ahorcado.py

```
1. importar al azar
2. FOTOS_AHORCADO = []
3. +---+
4.
5.
6.
7.     ||| == "", ""
8. +---+
9. El ||
10
11    |
12    == "", ""
13. +---+
14. El |
15. |||
```

deciséis.

```
17     ==="", ""
18. +---+
19. El |
20. /| | 21.
===== "", ""
22
23. +---+
24. El |
25. /\ |
26. |
27     ==="", ""
28. +---+
29. El |
30. /\ |
31. / | 32.
===== "", ""
33. +---+
34. El |
35. /\ |
36. /\ |
37.     ===""]
38. palabras = 'hormiga babuino tejón murciélagos oso castor camello gato almeja cobra
    puma coyote cuervo ciervo perro burro pato águila hurón zorro rana cabra ganso
    halcón león lagarto llama topo mono alce ratón mula tritón nutria búho panda loro
    paloma pitón conejo carnero rata cuervo rinoceronte salmón foca tiburón oveja
    mofeta perezoso serpiente araña cigüeña cisne tigre sapo trucha pavo tortuga
    comadreja ballena lobo wombat cebra'.split()
39.
40. def getRandomWord(wordList): 41.
# Esta función devuelve una cadena aleatoria de la lista pasada de

instrumentos de cuerda. 42. índicePalabras = random.randint(0,
len(ListaPalabras) - 1) 43. return ListaPalabras[índicePalabras]
44.
45. def mostrarTablero(letrasperdidas, letrascorrectas, palabrasecreta):
46. imprimir(FOTOS_AHORCADO[len(letrasperdidas)]) 47. imprimir()

48.
49. print('Letras perdidas:', end=' ') 50.
for letra en letrasperdidas:
```

```
51.     imprimir(letra, fin=' ')
52. imprimir() 53.

54. espacios_en_blanco = ' ' * len(palabrasecreta)
55.

56. for i in range(len(palabrasecreta)): # Reemplaza los espacios en blanco con las letras
    adivinadas_correctamente. si palabraSecreta[i] en letrasCorrectas:
57.
58.     espacios_en_blanco = espacios_en_blanco [:i] + palabrasecreta [i] + espacios_en_blanco [i + 1:]
59.

60. para letra en espacios_en_blanco: # Muestra la palabra secreta con espacios en medio
    cada letra.
61.     imprimir(letra, fin=' ')
62. imprimir()
63.

64. def getGuess(ya_adivinado): 65. #
Devuelve la letra que ingresó el jugador. Esta función se asegura de que el jugador ingrese
una sola letra y nada más.
66. mientras sea cierto:
67.     print('Adivina una letra.')
68.     adivinar = entrada()
69.     adivinar = adivinar.lower()
70.     if len(adivinar) != 1:
71.         print('Ingrese una sola letra.') elif adivinar
72.     en yaAdivinado:
73.         print('Ya has adivinado esa letra. Elige de nuevo.')
74.     elif supongo que no en 'abcdefghijklmnopqrstuvwxyz':
75.         print('Ingrese una LETRA.') else:
76.
77.     volver a adivinar
78.

79. def playAgain(): 80.
# Esta función devuelve True si el jugador quiere volver a jugar;
    de lo contrario, devuelve False.
81. print('¿Quieres volver a jugar? (sí o no)') 82. return
    input().lower().startswith('y')
83.
84.

85. print('HANGMA N') 86.
    "letras_perdidas =
```

```

87. letrascorrectas = ""

88. palabrasecreta = obtenerPalabraAleatoria(palabras)

89. juegoTerminado = Falso 90.

91. mientras sea cierto:

92. tablero de visualización (letras perdidas, letras correctas, palabra secreta)
93.

94. # Deja que el jugador ingrese una letra.

95. adivinar = obteneradivina(letrasperdidas + letrascorrectas) 96.

97. if adivinar en palabrasecreta:
98.     letrascorrectas = letrascorrectas + adivinar
99.

100.    # Comprobar si el jugador ha ganado.
101.    encontradoTodasLetras = Verdadero
102.    for i in range(len(palabrasecreta)):
103.        si palabraSecretaj[i] no está en letrasCorrectas:
104.            encontradoTodasLetras = Falso
105.            descanso
106.        si se encuentran todas las letras:
107.            print('¡Sí! La palabra secreta es "' + palabrasecreta + "'! ¡Has
108.            ganado!") juegolsDone = True
109. más:
110.     letrasperdidas = letrasperdidas + adivinar
111.
112.     # Compruebe si el jugador ha adivinado demasiadas veces y ha
113.     perdido. if len(letrasperdidas) == len(FOTOS DEL AHORCADO) - 1:
114.         displayBoard(missedLetters, correctLetters, secretWord) print('¡Te has
115.         quedado sin conjeturas!\nDespués de ' str(len(missedLetters)) + '
116.         conjeturas perdidas y ' str(len(correctLetters)) + la palabra era +
117.             ,'
118.             conjeturas correctas,
119.             ""
120.             + palabrasecreta + "")"
121.             missLetters =

```

```

122.     letrascorrectas = ""
123.     gameIsDone = False
124.     secretWord = getRandomWord(palabras)
125. else:
126.     descanso

```

IMPORTAR EL ALEATORIO MÓDULO

El programa Hangman selecciona aleatoriamente una palabra secreta para que el jugador la adivine de una lista de palabras. El módulo aleatorio proporcionará esta capacidad, por lo que la línea 1 la importa.

1. importar al azar

Pero la variable HANGMAN_PICS en la línea 2 parece un poco diferente de las variables que hemos visto hasta ahora. Con el fin de entender lo que significa este código, necesitamos aprender acerca de algunos más conceptos.

VARIABLES CONSTANTES

Las líneas 2 a 37 son una declaración de asignación larga para la variable HANGMAN_PICS .

```

2. FOTOS_AHORCADO = [""
3. +---+
4.
5.
6.
7.     ||| ==="",
"" --recorte-- 37.
===="]

```

El nombre de la variable HANGMAN_PICS está en mayúsculas

letras. Esta es la convención de programación para variables constantes. Las constantes son variables destinadas a tener valores que nunca cambie desde su primera declaración de asignación. Aunque puede cambiar el valor en HANGMAN_PICS tal como puede hacerlo con cualquier otra variable, el nombre en mayúsculas le recuerda que no debe hacerlo.

Como con todas las convenciones, no tienes que seguir esta. Pero hacerlo hace que sea más fácil para otros programadores leer tu código. Sabrán que FOTOS_AHORCADO siempre tendrá el valor que se le asignó desde las líneas 2 a la 37.

EL TIPO DE DATO LISTAS

HANGMAN_PICS contiene varias cadenas de varias líneas. puede hacer esto porque es una lista. Las listas tienen un valor de lista que puede contener varios otros valores. Ingrese esto en el shell interactivo:

```
>>> animales = ['oso hormiguero', 'oso hormiguero', 'antílope', 'albert'] >>> animales  
['oso hormiguero', 'oso hormiguero', 'antílope', 'albert']
```

El valor de lista en animales contiene cuatro valores. Lista de valores comienzan con un corchete izquierdo, [, y terminan con un corchete derecho,]. Así es como las cadenas comienzan y terminan entre comillas marcas.

Las comas separan los valores individuales dentro de una lista. Estos valores también se denominan elementos. Cada artículo en HANGMAN_PICS es una cadena multilínea.

Las listas le permiten almacenar varios valores sin usar una variable para cada uno. Sin listas, el código se vería así:

```
>>> animales1 = 'oso hormiguero'
```

```
>>> animales2 = 'oso hormiguero'  
>>> animales3 = 'antílope' >>>  
animales4 = 'alberto '
```

Este código sería difícil de administrar si tuviera cientos o miles de cadenas. Pero una lista puede contener fácilmente cualquier número de valores.

Acceso a elementos con índices Puede acceder a un elemento dentro de una lista agregando corchetes al final de la variable de lista con un número entre ellos.

El número entre corchetes es el índice. En Python, el índice del primer elemento de una lista es 0. El segundo elemento está en el índice 1, el tercero está en el índice 2, y así sucesivamente.

Debido a que los índices comienzan en 0 y no en 1, decimos que Python las listas están indexadas a cero.

Mientras todavía estamos en el caparazón interactivo y trabajando con la lista de animales , ingrese animales[0], animales[1], animales[2] y animales[3] para ver cómo se evalúan:

```
>>> animales[0]  
'cerdo hormiguero'  
>>> animales[1]  
'oso hormiguero'  
>>> animales[2]  
'antílope' >>>  
animales[3]  
'alberto'
```

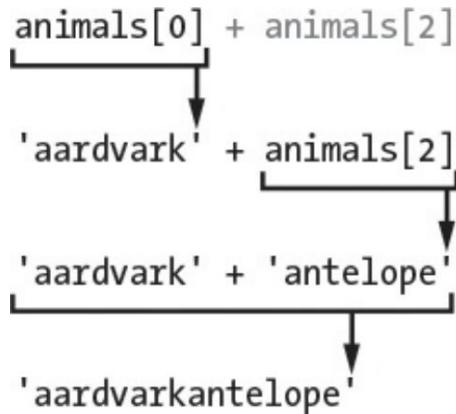
Observe que el primer valor de la lista, 'aardvark', se almacena en índice 0 y no índice 1. Cada elemento de la lista está numerado en ordenar a partir de 0.

Usando los corchetes, puede tratar elementos en la lista

como cualquier otro valor. Por ejemplo, ingrese `animales[0] + animales[2]` en el shell interactivo:

```
>>> animales[0] + animales[2] 'oso
hormiguero'
```

Ambas variables en los índices 0 y 2 de `animales` son cadenas, por lo que los valores se concatenan. La evaluación se ve así:



Índices fuera de rango e IndexError

Si intenta acceder a un índice que es demasiado alto para estar en la lista, obtendrá un `IndexError` que bloqueará su programa. Para ver un ejemplo de este error, ingrese lo siguiente en el shell interactivo:

```
>>> animales = ['oso hormiguero', 'oso hormiguero', 'antílope', 'albert'] >>>
animales[9999]
Rastreo (llamadas recientes más última):
  Archivo "", línea 1,
  en animales[9999]
IndexError: índice de lista fuera de rango
```

Como no hay ningún valor en el índice 9999, obtiene un error.

Cambio de elementos de lista con asignación de índice

También puede cambiar el valor de un elemento en una lista mediante la asignación de índice. Ingrese lo siguiente en el shell interactivo:

```
>>> animales = ['oso hormiguero', 'oso hormiguero', 'antílope',
''albert'] >>> animales[1] = 'ANTEATER'
>>> animales
['oso hormiguero', 'hormiguero', 'antílope', 'albert']
```

La nueva cadena 'ANTEATER' sobrescribe el segundo elemento en la lista de animales . Así que escribir animales[1] por sí solo se evalúa como el segundo elemento actual de la lista, pero usarlo en el lado izquierdo de un operador de asignación asigna un nuevo valor al segundo elemento de la lista.

Concatenación de listas

Puede unir varias listas en una lista usando el operador + , tal como puede hacerlo con las cadenas. Hacerlo se llama concatenación de listas.

Para ver esto en acción, ingrese lo siguiente en el shell interactivo:

```
>>> [1, 2, 3, 4] + ['manzanas', 'naranjas'] + ['Alice', 'Bob'] [1, 2,
3, 4, 'manzanas', 'naranjas', 'Alicia', 'Bob']
```

['manzanas'] + ['naranjas'] se evaluará como ['manzanas', 'naranjas']. Pero ['manzanas'] + 'naranjas' dará como resultado un error. No puedes agregar una lista valor y un valor de cadena con el operador + . Si desea agregar valores al final de una lista sin utilizar la concatenación de listas, utilice el método append() (descrito en “ Métodos de lista reverse() y append() ” en la página 95).

El operador in El operador

in puede decirle si un valor está en una lista o no.

Las expresiones que usan el operador in devuelven un valor booleano:

True si el valor está en la lista y False si no lo está. Introducir el siguiente en el shell interactivo:

```
>>> animales = ['oso hormiguero', 'oso hormiguero', 'antílope', 'albert'] >>>
'antílope' in animales
Verdadero
>>> 'hormiga' in animales
Falso
```

La expresión 'antílope' en animales devuelve True porque la cadena 'antílope' es uno de los valores en la lista de animales . Se encuentra en el índice 2. Pero cuando ingresa la expresión 'hormiga' en animales, devuelve False porque la cadena 'ant' no existe en la lista.

El operador in también funciona para cadenas, verificando si una cadena existe en otra. Ingrese lo siguiente en el shell interactivo:

```
>>> 'hola' in 'Alice le dijo hola a Bob.'
```

Almacenar una lista de cadenas de varias líneas en la variable HANGMAN_PICS cubrió muchos conceptos. Por ejemplo, vio que las listas son útiles para almacenar múltiples valores en una sola variable. También aprendió algunas técnicas para trabajar con listas, como la asignación de índices y la concatenación de listas. Los métodos son otro concepto nuevo que aprenderá a usar en el juego Hangman; los exploraremos a continuación.

MÉTODOS DE LLAMADA

Un método es una función adjunta a un valor. Para llamar a un método, debe adjuntarlo a un valor específico usando un punto. Python tiene muchos métodos útiles, y usaremos algunos de ellos en el

Programa del ahorcado.

Pero primero, veamos algunos métodos de lista y cadena.

Los métodos de lista reverse() y append() El tipo de datos list tiene un par de métodos que probablemente usará mucho: reverse() y append(). El método reverse() invertirá el orden de los elementos de la lista. Intente ingresar spam = [1, 2, 3, 4, 5, 6, 'miau', 'guau'], y luego spam.reverse() para invertir la lista. Luego ingresa spam para ver el contenido de la variable.

```
>>> spam = [1, 2, 3, 4, 5, 6, 'miau', 'guau'] >>> spam.reverse()
```

```
>>> correo no deseado  
['guau', 'miau', 6, 5, 4, 3, 2, 1]
```

El método de lista más común que usará es append(). Este método agregará el valor que pase como argumento al final de la lista. Intente ingresar lo siguiente en el shell interactivo:

```
>>> huevos = []  
>>> huevos.append('aerodeslizador')  
>>> huevos  
['aerodeslizador']  
>>> huevos.append('anguilas')  
>>> huevos ['aerodeslizador',  
'anguilas']
```

Estos métodos cambian las listas en las que son llamados. No devuelven una nueva lista. Decimos que estos métodos cambian la lista en su lugar.

El método de cadena split() El tipo de datos de cadena tiene un método split(), que devuelve una lista de cadenas creadas a partir de una cadena que se ha dividido. Intenta usar el

método split() ingresando lo siguiente en el interactivivo

caparazón:

```
>>> sentencia = entrada()
Mi madre muy enérgica acaba de servirnos nachos. >>>
oracion.dividir()
['Mi', 'muy', 'enérgico', 'madre', 'solo', 'servido', 'nosotros', 'nachos'].
```

El resultado es una lista de ocho cadenas, una cadena por cada palabra de la cadena original. La división se produce siempre que haya un espacio en la cadena. Los espacios no están incluidos en ninguno de los elementos de la lista.

La línea 38 del programa Hangman también usa el método split() , como se muestra a continuación. El código es largo, pero en realidad es solo una declaración de asignación simple que tiene una larga cadena de palabras separadas por espacios, con una llamada al método split() al final. El método split() se evalúa como una lista con cada palabra de la cadena como un solo elemento de la lista.

```
38. palabras = 'hormiga babuino tejón murciélago oso castor camello gato almeja cobra
puma coyote cuervo ciervo perro burro pato águila hurón zorro rana cabra ganso
halcón león lagarto llama topo mono alce ratón mula tritón nutria búho panda loro
paloma pitón conejo carnero rata cuervo rinoceronte salmón foca tiburón oveja
mofeta perezoso serpiente araña cigüeña cisne tigre sapo trucha pavo tortuga
comadreja ballena lobo wombat cebra'.split()
```

Es más fácil escribir este programa usando split(). Si creó una lista para empezar, tendría que escribir ['hormiga', 'babuino', 'tejón', etc., con comillas y comas para cada palabra.

También puede agregar sus propias palabras a la cadena en la línea 38 o eliminar cualquiera que no quiera que esté en el juego. Solo asegúrate de que los espacios separen las palabras.

OBTENER UNA PALABRA SECRETA DE LA LISTA DE PALABRAS

La línea 40 define la función `getRandomWord()` . Se pasará un argumento de lista para su parámetro `wordList` . Esta función devolverá una sola palabra secreta de la lista en `wordList`.

```
40. def getRandomWord(wordList): 41. # Esta
función devuelve una cadena aleatoria de la lista pasada de

instrumentos de cuerda. 42. índicePalabras = random.randint(0,
len(ListaPalabras) - 1) 43. return ListaPalabras[índicePalabras]
```

En la línea 42, almacenamos un índice aleatorio para esta lista en la variable `wordIndex` llamando a `randint()` con dos argumentos. El primer argumento es 0 (para el primer índice posible) y el segundo es el valor que la expresión `len(wordList) - 1` evalúa (para el último índice posible en una lista de palabras).

Recuerde que los índices de lista comienzan en 0, no en 1. Si tiene una lista de tres elementos, el índice del primer elemento es 0, el índice de el segundo elemento es 1, y el índice del tercer elemento es 2. El la longitud de esta lista es 3, pero el índice 3 estaría después del último índice. Esta es la razón por la cual la línea 42 resta 1 de la longitud de `listaPalabras`. Él el código en la línea 42 funcionará sin importar el tamaño de `wordList` .

Ahora puede agregar o eliminar cadenas en `wordList` si lo desea.

La variable `wordIndex` se establecerá en un índice aleatorio para el lista pasada como el parámetro `wordList` . La línea 43 devolverá el elemento en `wordList` en el número entero almacenado en `wordIndex`.

Supongamos que `['manzana', 'naranja', 'uva']` se pasó como argumento a `getRandomWord()` y que `randint(0, 2)` devolvió el entero 2. Eso significaría que la línea 43 se evaluaría para devolver

wordList[2], y luego evalúe para devolver 'uva'. Así es como getRandomWord() devuelve una cadena aleatoria en wordList.

Entonces, la entrada de getRandomWord() es una lista de cadenas, y la salida del valor de retorno es una cadena seleccionada al azar de esa lista. En el juego Hangman, así es como se selecciona una palabra secreta para que el jugador la adivine.

MOSTRAR EL TABLERO AL JUGADOR

A continuación, necesita una función para imprimir el tablero del Ahorcado en la pantalla. También debería mostrar cuántas letras ha adivinado correctamente (o incorrectamente) el jugador.

```
45. def mostrarTablero(letrasperdidas, letrascorrectas, palabrasecreta):
46.     imprimir(FOTOS_AHORCADO[len(letrasperdidas)]) 47. imprimir()
```

Este código define una nueva función llamada displayBoard(). Esto La función tiene tres parámetros:

missLetters Una cadena de letras que el jugador ha adivinado que no están en la palabra secreta

letrascorrectas Una cadena de letras que el jugador ha adivinado que están en la palabra secreta

palabrasecreta Una cadena de la palabra secreta que el jugador está tratando de adivinar

La primera llamada a la función print() mostrará el tablero. La variable global HANGMAN_PICS tiene una lista de cadenas para cada tablero posible. (Recuerde que las variables globales se pueden leer

desde dentro de una función). FOTOS_AHORCADO[0] muestra una horca vacía, FOTOS_AHORCADO[1] muestra la cabeza (cuando el jugador pierde una letra), FOTOS_AHORCADO[2] muestra la cabeza y el cuerpo (cuando el jugador pierde dos letras), y así sucesivamente hasta HANGMAN_PICS[6], que muestra al ahorcado completo.

El número de letras en letrasperdidas reflejará cuántas conjeturas incorrectas ha hecho el jugador. Llame a len(missedLetters) para averiguar este número. Por lo tanto, si missLetters es 'aetr', entonces len('aetr') devolverá 4. Al imprimir FOTOS_AHORCADO[4] , se mostrará la imagen del hombre colgado apropiada para cuatro fallas. Esto es lo que evalúa HANGMAN_PICS[len(missedLetters)] en la línea 46.

La línea 49 imprime la cadena 'Letras perdidas:' con un carácter de espacio al final en lugar de una nueva línea:

```
49. print('Letras perdidas:', end=' ') 50.  
for letra en letrasperdidas:  
    51.     imprimir(letra, fin=' ')  
    52. imprimir()
```

El ciclo for en la línea 50 iterará sobre cada carácter en la cadena letrasperdidas y lo imprimirá en la pantalla. Recuerde que end=' ' reemplazará el carácter de nueva línea que se imprime después de la cadena con un solo carácter de espacio. Por ejemplo, si letrasperdidas fuera 'ajtw', este ciclo for mostraría ajt w.

El resto de la función displayBoard() (líneas 54 a 62) muestra las letras perdidas y crea la cadena de la palabra secreta con todas las letras aún no adivinadas como espacios en blanco. Lo hace usando la función range() y la segmentación de listas.

Las funciones list() y range()

Cuando se llama con un argumento, range() devolverá un rango

objeto de enteros desde 0 hasta (pero sin incluir) el argumento. Este objeto de rango se usa en bucles pero también se puede convertir al tipo de datos de lista más familiar con la función list() . Ingrese list(range(10)) en el shell interactivo:

```
>>> lista(rango(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> lista('Hola')
['Hola']
```

La función list() es similar a las funciones str() o int() . Toma el valor que se le pasa y devuelve una lista. Es fácil generar listas enormes con la función range() . Por ejemplo, ingrese list(range(10000)) en el shell interactivo:

```
>>> lista(rango(10000))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,...
--recorte--
...9989, 9990, 9991, 9992, 9993, 9994, 9995, 9996, 9997, 9998, 9999]
```

La lista es tan grande que ni siquiera cabe en la pantalla. Pero puedes almacenar la lista en una variable:

```
>>> spam = lista(rango(10000))
```

Si pasa dos argumentos enteros a range(), el objeto de rango que devuelve es desde el primer argumento entero hasta (pero sin incluir) el segundo argumento entero. Luego ingrese list(range(10, 20)) en el shell interactivo de la siguiente manera:

```
>>> lista(rango(10, 20))
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

Como puede ver, nuestra lista solo llega hasta 19 y no incluye 20.

Segmentación de listas y

cadenas La división de listas crea un nuevo valor de lista utilizando un subconjunto de los elementos de otra lista. Para segmentar una lista, especifique dos índices (el principio y el final) con dos puntos entre corchetes después del nombre de la lista. Por ejemplo, ingrese lo siguiente en el shell interactivo:

```
>>> spam = ['manzanas', 'bananas', 'zanahorias', 'dátiles']
>>> spam[1:3] ['bananas', 'zanahorias']
```

La expresión spam[1:3] se evalúa como una lista con elementos en spam desde el índice 1 hasta (pero sin incluir) el índice 3.

Si omite el primer índice, Python automáticamente cree que quiere el índice 0 para el primer índice:

```
>>> spam = ['manzanas', 'bananas', 'zanahorias', 'dátiles']
>>> spam[:2] ['manzanas', 'bananas']
```

Si omite el segundo índice, Python pensará automáticamente que desea el resto de la lista:

```
>>> spam = ['manzanas', 'plátanos', 'zanahorias', 'dátiles']
>>> spam[2:] ['zanahorias', 'dátiles']
```

También puede usar sectores con cadenas de la misma manera que los usa con listas. Cada carácter de la cadena es como un elemento de la lista. Ingrese lo siguiente en el shell interactivo:

```
>>> miNombre = 'Zophie la gata gorda' >>>
miNombre[4:12] 'es decir, la F'
```

```
>>> miNombre[:10]
'Zophie la'
```

```
>>> miNombre[7:] 'el  
Gato Gordo'
```

La siguiente parte del código Hangman usa el corte.

Visualización de la palabra secreta con espacios en blanco

Ahora desea imprimir la palabra secreta, pero con líneas en blanco para las letras que no se han adivinado. Puede usar el carácter de subrayado (_) para esto. Primero cree una cadena con nada más que un guión bajo para cada letra de la palabra secreta.

Luego reemplace los espacios en blanco para cada letra en letrascorrectas.

Entonces, si la palabra secreta fuera 'nutria', entonces la cadena en blanco sería '_____ ' (cinco guiones bajos). Si letrasCorrectas fuera la cadena 'rt', cambiaría la cadena a '_tt_r'. Las líneas 54 a 58 son la parte del código que hace eso:

```
54. espacios en blanco = '_' * len(palabrasecreta)  
55.  
56. for i in range(len(palabrasecreta)): # Reemplaza los espacios en blanco con las letras  
    adivinadas correctamente. si palabraSecreta[i] en letrasCorrectas:  
57.  
58.     espacios en blanco = espacios en blanco [:i] + palabrasecreta [i] + espacios en blanco [i + 1:]
```

La línea 54 crea la variable de espacios en blanco llena de guiones bajos mediante la replicación de cadenas. Recuerde que el operador * se puede usar en una cadena y un número entero, por lo que la expresión '_' * 5 se evalúa como '_____. Esto asegurará que los espacios en blanco tengan el mismo número de guiones bajos como palabrasecreta tiene letras.

La línea 56 tiene un ciclo for que pasa por cada letra en palabrasecreta y reemplaza el guión bajo con la letra real si corresponde. existe en letrascorrectas.

Echemos otro vistazo al ejemplo anterior, donde el

el valor de palabraSecreta es 'nutria' y el valor de letrasCorrectas es 'tr'. Tú querría que la cadena '_tt_r' se mostrara al jugador. Averigüemos cómo crear esta cadena.

La llamada len(palabrasecreta) de la línea 56 devolvería 5. El range(len(palabrasecreta)) se convierte en range(5), lo que hace que el ciclo for itere sobre 0, 1, 2, 3 y 4.

Porque el valor de i tomará cada valor en [0, 1, 2, 3, 4], el código en el bucle for se ve así:

si palabraSecreta[0] en letrasCorrectas:

espacios en blanco = espacios en blanco [: 0] + palabra secreta [0] + espacios en blanco [1:]

si palabraSecreta[1] en letrasCorrectas:

espacios en blanco = espacios en blanco [: 1] + palabra secreta [1] + espacios en blanco [2:]

--recorte--

Estamos mostrando solo las dos primeras iteraciones del ciclo for , pero comenzando con 0, tomaré el valor de cada número en el rango. En la primera iteración, i toma el valor 0, por lo que si instrucción comprueba si la letra en palabrasecreta en el índice 0 está en letras correctas. El ciclo hace esto para cada letra en la palabra secreta, una letra a la vez.

Si está confundido acerca del valor de algo como palabrasecreta[0] o espacios en blanco[3:], mire la figura 8-1. Muestra el valor de las variables palabrasecreta y espacios en blanco y el índice de cada letra en la cuerda.

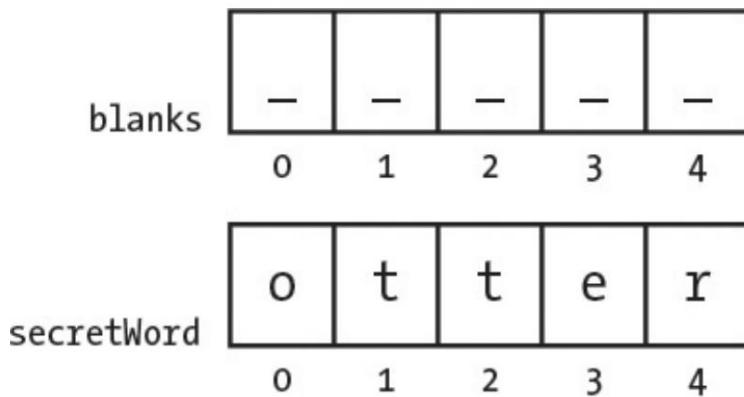


Figura 8-1: Los índices de los espacios en blanco y las cadenas de palabras secretas

Si reemplaza los segmentos de la lista y los índices de la lista con el valores que representan, el código de bucle se ve así:

```

si 'o' en 'tr': # Falso
    "
    espacios en blanco =      + 'o' + '___' # Esta línea se salta.

--snip-- si
'r' en 'tr': # Verdadero

    espacios en blanco = '_tt_' + 'r' + " # Esta línea se ejecuta.

# espacios en blanco ahora tiene el valor '_tt_r'.

```

Todos los ejemplos de código anteriores hacen lo mismo cuando palabra secreta es 'nutria' y letras correctas es 'tr'. Las próximas líneas de código imprime el nuevo valor de los espacios en blanco con espacios entre cada uno carta:

```

60. para letra en espacios en blanco: # Muestra la palabra secreta con espacios en medio
    cada letra.

61.     imprimir(letra, fin=' ') 62.

imprimir()

```

Observe que el ciclo for en la línea 60 no llama a la función range(). En lugar de iterar sobre el objeto de rango, esta llamada de función devolvería, itera sobre el valor de cadena en la variable de espacios en blanco. En cada iteración, la variable letra adquiere un nuevo

carácter de la cadena 'nutria' en espacios en blanco.

La salida impresa después de agregar los espacios sería '_ tt
_ r'.

CONSEGUIR LA CONJURACIÓN DEL JUGADOR

Se llamará a la función getGuess() para que el jugador pueda ingresar una letra para adivinar. La función devuelve la letra que el jugador adivinó como una cadena. Además, getGuess() se asegurará de que el jugador escriba una letra válida antes de regresar de la función.

```
64. def getGuess(ya adivinado):
65. # Devuelve la letra que ingresó el jugador. Esta función se asegura de
    que el jugador ingrese una sola letra y nada más.
```

Una cadena de letras que el jugador ha adivinado se pasa como argumento para el parámetro ya adivinado . Luego, la función getGuess() le pide al jugador que adivine una sola letra. Esta única letra será el valor de retorno de getGuess() . Ahora, debido a que Python distingue entre mayúsculas y minúsculas, debemos asegurarnos de que la suposición del jugador sea una letra minúscula para que podamos compararla con la palabra secreta. Ahí es donde entra en juego el método lower() .

Los métodos de cadena lower() y upper() Ingrese '!Hola mundo!'.lower() en el shell interactivo para ver un ejemplo del método lower() :

```
>>> '!Hola mundo!'.lower()
'¡Hola mundo!'
```

El método lower() devuelve una cadena con todos los caracteres en minúsculas. También hay un método upper() para cadenas, que

devuelve una cadena con todos los caracteres en mayúsculas. Pruébelo ingresando '*Hola mundo!*'.upper() en el shell interactivo:

```
>>> 'Hola mundo!'.upper()  
'HOLA MUNDO!'
```

Debido a que el método upper() devuelve una cadena, también puede llamar a un método en esa cadena.

Ahora ingrese esto en el shell interactivo:

```
>>> 'Hola mundo!'.upper().lower()  
'hola Mundo!'
```

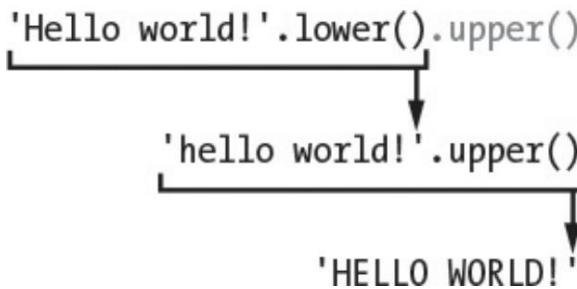
'*Hola mundo!*'.upper() se evalúa como la cadena '*HOLA MUNDO!*', y luego se llama al método lower() de la cadena. Esto devuelve la cadena '*hola mundo!*', que es el valor final en la evaluación:



El orden es importante. '*Hola mundo!*.lower().upper() no es el igual que '*Hola mundo!*.superior().inferior():

```
>>> 'Hola mundo!'.inferior().superior()  
'HOLA MUNDO!'
```

Esa evaluación se ve así:



Si una cadena se almacena en una variable, también puede llamar a una cadena método en esa variable:

```
>>> spam = '¡Hola mundo!' >>>
spam.superior()
'¡HOLA MUNDO!'
```

Este código no cambia el valor en spam. el correo no deseado La variable seguirá conteniendo '¡Hola mundo!'.

Volviendo al programa Hangman, usamos lower() cuando preguntamos por la conjetura del jugador:

```
66. mientras sea cierto:
67.     print('Adivina una letra.')
68.     adivinar = entrada()
69.     adivinar = adivinar.inferior()
```

Ahora, incluso si el jugador ingresa una letra mayúscula como adivinar, la función getGuess() devolverá una letra minúscula.

Salir del bucle while El bucle while de la línea 66 seguirá pidiendo al jugador una letra hasta que introduzca una única letra que no haya adivinado previamente.

La condición para el bucle while es simplemente el valor booleano True. Eso significa que la única forma en que la ejecución dejará este ciclo es ejecutando una declaración de ruptura , que deja el ciclo, o una declaración de retorno , que no solo deja el ciclo .

sino toda la función.

El código dentro del ciclo le pide al jugador que ingrese una letra, que se almacena en la variable adivinar. Si el jugador ingresó una letra mayúscula, se sobrescribiría con una letra minúscula en la línea 69.

DECLARACIONES ELIF

La siguiente parte del programa Hangman usa sentencias elif .

Puede pensar en las declaraciones elif o "else-if" como si dijeran: "Si esto es cierto, haz esto". O bien, si la siguiente condición es verdadera, haz eso. O si ninguno de ellos es verdadero, haz esto último. Echa un vistazo al siguiente código:

```
if nombreGato == 'Pelota':
    print('Tu gato es borroso.')
elif catName == 'Puntos':
    print('Tu gato está manchado.')
else:
    print('Tu gato no está borroso ni manchado.)
```

Si la variable catName es igual a la cadena 'Fuzzball', entonces el si la condición de la declaración es verdadera y el bloque si le dice al usuario que su gato es borroso. Sin embargo, si esta condición es falsa, Python prueba la condición de la declaración elif a continuación. Si catName es 'Spots', entonces la cadena 'Your cat is spotted'. se imprime en la pantalla. Si ambos son falsos, entonces el código le dice al usuario que su gato no está peludo ni manchado.

Puede tener tantas declaraciones elif como desee:

```
if nombreGato == 'Pelota':
    print('Tu gato es borroso.')
elif catName == 'Puntos':
```

```

print('Tu gato está manchado.')
elif catName == 'Gorditos':
    print('Tu gato es gordito.') elif
catName == 'Puff':
    print('Tu gato está hinchado.')
más:
print('Tu gato no es peludo ni manchado ni gordito ni hinchado.')

```

Cuando una de las condiciones elif es verdadera, su código se ejecuta y luego la ejecución salta a la primera línea más allá del bloque else . Entonces , uno, y solo uno, de los bloques en if-elif-else se ejecutarán las sentencias. También puedes omitir el resto bloquee si no necesita uno y solo tenga declaraciones if-elif .

ASEGURANDO QUE EL JUGADOR INTRODUCIÓ UNA SUPOSICIÓN VÁLIDA

La variable de conjetura contiene la conjetura de letras del jugador. El programa debe asegurarse de que ingresaron una suposición válida: una, y solo una, letra que aún no se haya adivinado. Si no lo hicieron, la ejecución volverá y les pedirá una carta nuevamente.

```

70.     si len(adivinar) != 1:
71.         print('Ingrese una sola letra.')
72.     elif adivinar en yaAdivinado:
73.         print('Ya has adivinado esa letra. Elige de nuevo.')
74.     elif supongo que no en 'abcdefghijklmnopqrstuvwxyz':
75.         print('Ingrese una LETRA.') else:
76.
77.     volver a adivinar

```

La condición de la línea 70 verifica si la conjetura no es una carácter largo, la condición de la línea 72 verifica si ya se ha adivinado

existe dentro de la variable ya adivinada y la condición de la línea 74 comprueba si adivinar no es una letra del alfabeto inglés estándar. Si alguna de estas condiciones es verdadera, el juego solicita al jugador que ingrese una nueva suposición.

Si todas estas condiciones son Falsas, entonces la sentencia else el bloque se ejecuta y getGuess() devuelve el valor en adivinar en línea 77.

Recuerde, solo uno de los bloques en una instrucción if-elif-else será ejecutado.

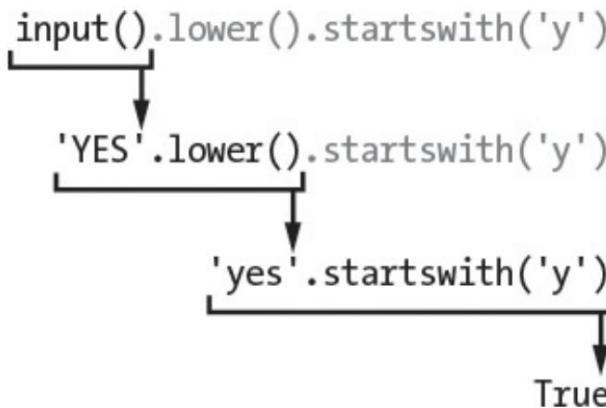
PEDIR AL JUGADOR QUE JUGUE OTRA VEZ

La función jugar de nuevo() tiene solo una llamada a la función imprimir() y un retorno declaración:

```
79. def playAgain():
80. # Esta función devuelve True si el jugador quiere volver a jugar;
     de lo contrario, devuelve
False. 81. print('¿Quieres volver a jugar? (sí o
no)')
82. return input().lower().startswith('y')
```

La declaración de retorno tiene una expresión que parece complicada, pero se puede desglosar. Aquí hay un vistazo paso a paso de cómo Python evalúa esta expresión si el usuario ingresa

SI:



El objetivo de la función jugar de nuevo () es permitir que el jugador ingrese sí o no para decirle al programa si quiere jugar otra ronda de Hangman. El jugador debe poder escribir Sí, sí, Y o cualquier otra cosa que comience con una y para que signifique "sí". Si el jugador ingresa Sí, entonces el valor de retorno de input() es la cadena 'Sí'. Y 'YES'.lower() devuelve la versión en minúsculas del

cadena adjunta. Entonces, el valor de retorno de 'YES'.lower() es 'yes'.

Pero está la segunda llamada al método, comienza con ('y'). Esto La función devuelve True si la cadena asociada comienza con el parámetro de cadena entre paréntesis y False si no lo hace.

El valor de retorno de 'yes'.startswith('y') es True.

Eso es todo, ¡usted evaluó esta expresión! Permite que el jugador ingrese una respuesta, establece la respuesta en minúsculas, verifica si comienza con la letra y y luego devuelve True si es así. hace y False si no lo hace.

En una nota al margen, también hay un método de cadena de extremos con (someString) que devolverá True si la cadena termina con la cadena en someString y False si no es así. termina con () es algo así como lo contrario de comienza con () .

RESEÑA DE EL VERDUGO

FUNCIONES

¡Esas son todas las funciones que estamos creando para este juego!

Vamos a repasarlos:

getRandomWord(wordList) Toma una lista de cadenas que se le pasan y devuelve una de ellas. Así es como se elige una palabra para que el jugador la adivine.

displayBoard(missedLetters, correctLetters, secretWord) Muestra el el estado actual del tablero, incluida la cantidad de la palabra secreta que el jugador ha adivinado hasta ahora y las letras incorrectas que ha adivinado. Esta función necesita que se le pasen tres parámetros para funcionar correctamente. Las letras correctas y las letras perdidas son cadenas formadas por las letras que el jugador ha adivinado que están y no en la palabra secreta, respectivamente. Y secretWord es la palabra secreta que el jugador está tratando de adivinar. Esta función no tiene valor de retorno.

getGuess(ya adivinado) Toma una cadena de letras que el jugador ya ha adivinado y seguirá pidiéndole al jugador una letra que no esté ya adivinada . Esta función devuelve la cadena de la letra válida que el jugador adivinó.

playAgain() Pregunta si el jugador quiere jugar otra ronda de Hangman. Esta función devuelve True si el jugador lo hace, False si no lo hace.

Después de las funciones, el código de la parte principal del programa comienza en la línea 85. Todo hasta este punto ha sido solo definiciones de funciones y una declaración de asignación grande para AHORCADO_FOTOS.

EL BUCLE DEL JUEGO

La parte principal del programa Hangman muestra el nombre del juego, configura algunas variables y ejecuta un ciclo while .

Esta sección recorre el resto del programa paso a paso.

```
85. print('HANGMAN') 86.
      "
letrasperdidas =
      "
87. letrascorrectas =
      "
88. palabrasecreta = obtenerPalabraAleatoria(palabras)
89. juegoTerminado = Falso
```

La línea 85 es la primera llamada print() que se ejecuta cuando se ejecuta el juego. Muestra el título del juego. A continuación, se asignan cadenas en blanco a las variables letrasperdidas y letrascorrectas ya que el jugador aún no ha adivinado ninguna letra perdida o correcta.

La llamada getRandomWord(words) en la línea 88 se evaluará como un palabra seleccionada al azar de la lista de palabras .

La línea 89 establece gameIsDone en False. El código establecerá gameIsDone en True cuando quiera indicar que el juego ha terminado y le preguntará al jugador si quiere volver a jugar.

Llamar a la función displayBoard() El resto del programa consiste en un bucle while . La condición del ciclo siempre es Verdadera, lo que significa que se repetirá para siempre hasta que encuentre una declaración de ruptura . (Esto sucede más adelante en la línea 126).

```
91. mientras sea cierto:
      "
92. tablero de visualización (letras perdidas, letras correctas, palabra secreta)
```

La línea 92 llama a la función displayBoard() , pasándole los tres

variables establecidas en las líneas 86, 87 y 88. Según la cantidad de letras que el jugador haya adivinado correctamente y se haya perdido, esta función muestra el tablero del ahorcado apropiado para el jugador.

Permitir que el jugador introduzca su suposición A continuación , se llama a la función getGuess() para que el jugador pueda introducir su suposición.

94. # Deja que el jugador ingrese una letra. 95. adivinar = obteneradivina(letrasperdidas + letrascorrectas)

La función getGuess () requiere un parámetro ya adivinado para que pueda verificar si el jugador ingresa una letra que ya ha adivinado. La línea 95 concatena las cadenas en las variables letrasperdidas y letrascorrectas y pasa el resultado como argumento para el parámetro ya adivinado .

Comprobar si la letra está en la palabra secreta

Si la cadena de conjetura existe en palabraSecreta, entonces este código concatena la conjetura hasta el final de la cadena de letras correctas :

97. if adivinar en palabrasecreta:
98. letrascorrectas = ~~adivinar~~ letrascorrectas

Esta cadena será el nuevo valor de letrascorrectas.

Comprobar si el jugador ganó ¿Cómo sabe el programa si el jugador ha adivinado todas las letras de la palabra secreta? Bueno, letrasCorrectas tiene cada letra que el jugador adivinó correctamente, y PalabraSecreta es la palabra secreta en sí. Pero no puede simplemente verificar si letrasCorrectas == PalabraSecreta. Si palabraSecreta fuera la cadena 'nutria' y letrasCorrectas fueran

la cadena 'orte', entonces letrasCorrectas == PalabraSecreta sería Falso aunque el jugador haya adivinado cada letra de la palabra secreta.

La única manera de estar seguro de que el jugador ha ganado es repetir cada letra en palabrasecreta y ver si existe en letras correctas. Si, y solo si, todas las letras de palabrasecreta existen en letrascorrectas , el jugador ha ganado.

```

100.      # Comprobar si el jugador ha ganado.
101.     encontradoTodasLetras = Verdadero
102.     for i in range(len(palabrasecreta)): si
103.         palabrasecreta[i] no está en letrascorrectas:
104.             encontradoTodasLetras = Falso
105.             descanso

```

Si encuentra una letra en palabrasecreta que no existe en letrascorrectas, sabrá que el jugador no ha adivinado todas las letras. La nueva variable foundAllLetters se establece en True en la línea 101 antes de que comience el bucle. El ciclo comienza suponiendo que se han encontrado todas las letras de la palabra secreta. Pero el código del ciclo en la línea 104 cambiará foundAllLetters a False la primera vez encuentra una letra en palabraSecreta que no está en letrasCorrectas.

Si se han encontrado todas las letras de la palabra secreta, el se le dice al jugador que ha ganado, y gameIsDone se establece en True:

```

106.     si se encuentran todas las letras:
107.         print('¡Sí! La palabra secreta es "' + palabrasecreta + '"! ¡Has
108.             ganado!') juegoIsDone = True

```

Manejo de una suposición incorrecta

La línea 109 es el comienzo del bloque else .

```

109. más:
110.     letrasperdidas = letrasperdidas + adivinar

```

Recuerde, el código de este bloque se ejecutará si la condición es Falsa.

¿Pero qué condición? Para averiguarlo, señale con el dedo el comienzo de la palabra clave else y muévalo hacia arriba. Verá que la sangría de la palabra clave else es la misma que la sangría de la palabra clave if en la línea 97:

97. if adivinar en palabrasecreta: --

snip-- 109. else:

110. letrasperdidas = letrasperdidas + adivinar

Entonces, si la condición en la línea 97 (adivina en palabrasecreta) fuera falsa, entonces la ejecución se movería a este bloque else .

Las letras adivinadas incorrectamente se concatenan con la cadena de letras perdidas en la línea 110. Esto es como lo que hizo la línea 98 con las letras que el jugador adivinó correctamente.

Verificando si el jugador perdió Cada vez que el jugador adivina incorrectamente, el código concatena la letra incorrecta con la cadena en letrasperdidas. Entonces, la longitud de las letras perdidas, o, en código, len (letras perdidas) , también es la cantidad de conjeturas incorrectas.

112. # Compruebe si el jugador ha adivinado demasiadas veces y ha
113. perdido. if len(letrasperdidas) == len(FOTOS DEL AHORCADO) - 1:
114. displayBoard(missedLetters, correctLetters, secretWord) print('Se te
115. acabaron las conjeturas!\nDespués de ' str(len(missedLetters)) + '
 conjeturas perdidas y ' str(len(correctLetters)) + '+
 conjeturas correctas,
 "
 la palabra era + palabrasecreta + "")')
116. juegosDone = Verdadero

La lista HANGMAN_PICS tiene siete cadenas de arte ASCII. Así que

cuando la longitud de la cadena de letras perdidas es igual a len(FOTOS DEL AHORCADO) - 1 (es decir, 6), el jugador se ha quedado sin intentos. Sabes que el jugador ha perdido porque la imagen del hombre colgado estará terminada. Recuerde, FOTOS_AHORCADO[0] es el primer elemento de la lista, y FOTOS_AHORCADO[6] es el último.

La línea 115 imprime la palabra secreta y la línea 116 establece la variable `gamelsDone` en True.

118. # Preguntar al jugador si quiere volver a jugar (pero solo si el juego está hecho).

```
119. si juegolsDone: 120.  
si jugarDeNuevo():  
121.     letrasperdidas = ""  
122.     letrascorrectas = ""  
123.     gamelsDone = False  
124.     secretWord = getRandomWord(palabras)
```

Terminar o restablecer el juego Ya sea que el jugador gane o pierda después de adivinar su letra, el juego debe preguntarle al jugador si quiere volver a jugar. La función `playAgain()` se encarga de obtener un sí o un no del jugador, por lo que se llama en la línea 120.

Si el jugador quiere volver a jugar, los valores de `letrasperdidas` y `letrascorrectas` deben restablecerse a cadenas en blanco, `gamelsDone` debe restablecerse a False y una nueva palabra secreta almacenada en `secretWord`. De esta manera, cuando la ejecución vuelva al comienzo del ciclo `while` en la línea 91, el tablero se restablecerá a un juego nuevo.

Si el jugador no ingresó algo que comenzara con y cuando se le preguntó si quería volver a jugar, entonces la condición de la línea 120 sería Falsa y el bloque `else` ejecutaría:

125. más:

126. descanso

La instrucción break hace que la ejecución salte a la primera instrucción después del bucle. Pero debido a que no hay más instrucciones después del ciclo, el programa termina.

RESUMEN

Hangman ha sido nuestro juego más avanzado hasta la fecha y has aprendido varios conceptos nuevos mientras lo creabas. A medida que sus juegos se vuelven más y más complejos, es una buena idea dibujar un diagrama de flujo de lo que debería suceder en su programa.

Las listas son valores que pueden contener otros valores. Los métodos son funciones asociadas a un valor. Las listas tienen un método append() .

Las cadenas tienen los métodos lower(), upper(), split(), beginwith(), y Endswith() .

Aprenderá sobre muchos más tipos de datos y métodos en el resto de este libro.

La instrucción elif le permite agregar una cláusula "o else-if" en medio de sus declaraciones if-else .

9

Ahorcado extensible



Ahora que ha creado un juego del ahorcado básico, veamos algunas formas de ampliarlo con nuevas funciones. En este capítulo, agregará varios conjuntos de palabras para que la computadora los utilice y la capacidad de cambiar el nivel de dificultad del juego.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- El tipo de datos del diccionario
- Pares clave-valor
- Los métodos de diccionario `keys()` y `values()`
- Asignación de múltiples variables

AÑADIR MÁS CONJUROS

Después de haber jugado Hangman varias veces, podrías pensar que seis intentos no son suficientes para que el jugador entienda muchas de las palabras. Puede darles más conjecturas fácilmente agregando más cadenas de varias líneas a la lista `HANGMAN_PICS`.

Guarde su programa `hangman.py` como `hangman2.py`. Después

agregue las siguientes instrucciones en la línea 37 y después para ampliar la lista que contiene el arte ASCII del hombre colgado:

```
37.     ==="", ""
38. +---+
39. [EI |
40. /\ |
41. /\ | 42.
      ==="", ""
43. +---+
44. [EI] |
45. /\ |
46. /\ |
47.     ==="]
```

Este código agrega dos nuevas cadenas de varias líneas a la lista HANGMAN_PICS , una con la oreja izquierda del ahorcado dibujada y la otra con ambas orejas dibujadas. Debido a que el programa le dirá al jugador que ha perdido basado en len(missedLetters) == len(HANGMAN_PICS) - 1, este es el único cambio que necesita hacer. El resto del programa funciona bien con la nueva lista HANGMAN_PICS .

EL TIPO DE DATOS DEL DICCIONARIO

En la primera versión del programa Hangman, usamos una lista de palabras de animales, pero podría cambiar la lista de palabras en la línea 48. En lugar de animales, podría tener colores:

```
48. palabras = 'rojo naranja amarillo verde azul índigo violeta blanco negro marrón'
    .separar()
```

O formas:

```
48. palabras = 'cuadrado triángulo rectángulo círculo elipse rombo trapezoide'
```

```
chevron pentágono hexágono septágono octágono'.split()
```

O frutas:

```
48. palabras = 'manzana naranja limón lima pera sandía toronja cereza  
plátano melón mango fresa tomate'.split()
```

Con alguna modificación, incluso puedes cambiar el código para que el juego Hangman use conjuntos de palabras, como animales, colores, formas o frutas. El programa puede decirle al jugador de qué conjunto es la palabra secreta.

Para realizar este cambio, necesitará un nuevo tipo de datos llamado diccionario. Un diccionario es una colección de valores como una lista. Pero en lugar de acceder a los elementos del diccionario con un índice de enteros, puede acceder a ellos con un índice de cualquier tipo de datos. Para los diccionarios, estos índices se denominan claves.

Los diccionarios usan { y } (corchetes) en lugar de [y] (corchetes).

Ingresé lo siguiente en el shell interactivo:

```
>>> spam = {'hola':'Hola, ¿cómo estás?', 4:'tocino', 'huevos':9999}
```

Los valores entre llaves son pares clave-valor. Las claves están a la izquierda de los dos puntos y los valores de la clave están a la derecha. Puede acceder a los valores como elementos en listas usando la tecla. Para ver un ejemplo, ingrese lo siguiente en el shell interactivo:

```
>>> spam = {'hola':'Hola, ¿cómo estás?', 4:'tocino', 'huevos':9999} >>> spam['hola']
```

```
'¿Hola como estás?' >>>  
spam[4] 'tocino'
```

```
>>> spam['huevos']  
9999
```

En lugar de poner un número entero entre corchetes, puede usar, por ejemplo, una clave de cadena. En el diccionario de spam , utilicé tanto el número entero 4 como la cadena 'eggs' como claves.

Obtener el tamaño de los diccionarios con len()

Puede obtener el número de pares clave-valor en un diccionario con la función len() . Por ejemplo, ingrese lo siguiente en el concha interactiva:

```
>>> cosas = {'hola':'Hola, ¿cómo estás?', 4:'tocino', 'spam':9999} >>> len(cosas)
3
```

La función len() devolverá un valor entero para el número de pares clave-valor, que en este caso es 3.

La diferencia entre diccionarios y listas

Una diferencia entre los diccionarios y las listas es que los diccionarios pueden tener claves de cualquier tipo de datos, como ha visto. Pero recuerde, debido a que 0 y '0' son valores diferentes, serán claves diferentes. Ingrese esto en el shell interactivo:

```
>>> spam = {'0':'una cadena', 0:'un entero'}
>>> spam[0] 'un entero' >>> spam['0'] 'una
cadena'
```

También puede recorrer ambas listas y las claves en los diccionarios usando un bucle for . Para ver cómo funciona esto, ingrese lo siguiente en el shell interactivo:

```
>>> favoritos = {'fruta':'manzanas', 'animal':'gatos', 'numero':42} >>>
for k en favoritos:
```

```

imprimir (k)
Fruta
número
animal
>>> para k en favoritos:
    imprimir(favoritos[k])
manzanas 42

```

gatos

Es posible que las claves y los valores se hayan impreso en un orden diferente porque, a diferencia de las listas, los diccionarios están desordenados. El primer elemento de una lista denominada listaCosas sería listaCosas[0]. Pero hay no hay primer elemento en un diccionario, porque los diccionarios no tienen ningún tipo de orden. En este código, Python simplemente elige un orden en función de cómo almacena el diccionario en la memoria, que no garantiza que sea siempre el mismo.

Ingrese lo siguiente en el shell interactivo:

```

>>> favoritos1 = {'fruta':'manzanas', 'numero':42, 'animal':'gatos'} >>> favoritos2 =
{'animal':'gatos', 'numero':42, 'fruta ':'manzanas'} >>> favoritos1 == favoritos2

```

Verdadero

La expresión favoritos1 == favoritos2 se evalúa como verdadera porque los diccionarios no están ordenados y se consideran iguales si tienen los mismos pares clave-valor. Mientras tanto, las listas están ordenadas, por lo que dos listas con los mismos valores en diferente orden no son iguales entre sí. Para ver la diferencia, ingrese esto en el shell interactivo:

```

>>> listFavs1 = ['manzanas', 'gatos', 42] >>>
listFavs2 = ['gatos', 42, 'manzanas']
>>> listaFavoritos1 == listaFavoritos2
Falso

```

La expresión `listFavs1 == listFavs2` se evalúa como Falsa porque los contenidos de las listas están ordenados de manera diferente.

Los diccionarios de métodos de diccionario de claves() y valores() tienen dos métodos útiles, claves() y valores(). Estos devolverán valores de un tipo llamado `dict_keys` y `dict_values`, respectivamente. Al igual que los objetos de rango, los formularios de lista de esos tipos de datos son devueltos por `list()`.

Ingrese lo siguiente en el shell interactivo:

```
>>> favoritos = {'fruta':'manzanas', 'animal':'gatos', 'numero':42} >>>
lista(favoritos.claves()) ['fruta', 'numero', 'animal '] >>>
lista(favoritos.valores()) ['manzanas', 42, 'gatos']
```

Usando `list()` con los métodos `keys()` o `valores()` , puede obtener una lista de solo las claves o solo los valores de un diccionario.

Uso de diccionarios de palabras en Hangman Vamos a cambiar el código en el nuevo juego Hangman para admitir diferentes conjuntos de palabras secretas. Primero, reemplace el valor asignado a las palabras con un diccionario cuyas claves sean cadenas y los valores sean listas de cadenas. El método de cadena `split()` devolverá una lista de cadenas con una palabra cada una.

```
48. palabras = {'Colores':'rojo naranja amarillo verde azul índigo violeta blanco negro  
marrón'.split(),  
49. 'Formas':'cuadrado triángulo rectángulo círculo elipse rombo trapecio  
cheurón pentágono hexágono septágono octágono'.split(),  
50. 'Frutas': 'manzana naranja limón lima pera sandía uva toronja cereza  
plátano melón mango fresa tomate'.split(), 51.  
'Animales':'murciélagos oso castor gato puma cangrejo ciervo perro burro pato  
águila pez rana cabra sanguijuela león lagarto mono alce ratón nutria búho panda
```

```
pitón conejo rata tiburón oveja mofeta calamar tigre pavo tortuga  
comadreja ballena lobo wombat cebra'.split()}
```

Las líneas 48 a 51 siguen siendo solo una instrucción de asignación.
La instrucción no termina hasta el último corchete en la línea 51.

ELEGIR ALEATORIAMENTE DE UNA LISTA

La función choice() en el módulo aleatorio toma un argumento de lista y devuelve un valor aleatorio de él. Esto es similar a lo que la función getRandomWord() anterior lo hizo. Utilizará choice() en la nueva versión de la función getRandomWord() .

Para ver cómo funciona la función choice() , ingrese lo siguiente en el caparazón interactivo:

```
>>> import random  
>>> random.choice(['gato', 'perro', 'ratón'])  
'ratón'  
>>> random.choice(['gato', 'perro', 'ratón'])  
'gato'
```

Así como la función randint() devuelve un entero aleatorio cada tiempo, la función choice() devuelve un valor aleatorio de la lista.

Cambie la función getRandomWord() para que su parámetro sea un diccionario de listas de cadenas, en lugar de solo una lista de cadenas. Así es como se veía originalmente la función:

```
40. def getRandomWord(wordList): 41.  
# Esta función devuelve una cadena aleatoria de la lista pasada de  
  
instrumentos de cuerda. 42. índicePalabras = random.randint(0,  
len(ListaPalabras) - 1) 43. return ListaPalabras[ÍndicePalabras]
```

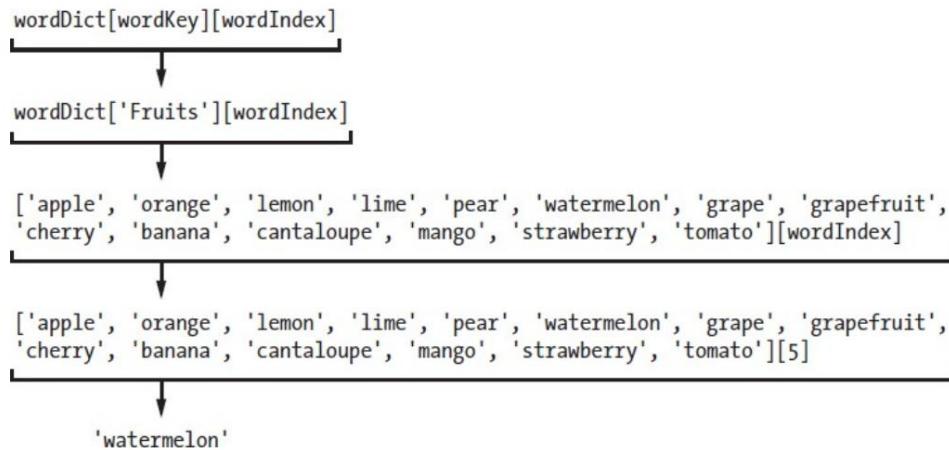
Cambie el código en esta función para que se vea así:

```
53. def getRandomWord(wordDict): 54.  
# Esta función devuelve una cadena aleatoria del diccionario pasado de listas de  
cadenas y su clave.  
55. # Primero, seleccione aleatoriamente una clave del  
diccionario: 56. wordKey = random.choice(list(wordDict.keys()))  
57.  
58. # Segundo, seleccione aleatoriamente una palabra de la lista de  
claves en el diccionario: 59. wordIndex = random.randint(0,  
len(wordDict[wordKey]) - 1)  
60  
61. return [DiccionarioDePalabras[ClaveDePalabras][ÍndiceDePalabras], ClaveDePalabras]
```

Hemos cambiado el nombre del parámetro `wordList` a `wordDict` para que sea más descriptivo. Ahora, en lugar de elegir una palabra aleatoria de una lista de cadenas, primero la función elige una clave aleatoria en el diccionario `wordDict` llamando a `random.choice()`.

Y en lugar de devolver la cadena `wordList[wordIndex]`, el
La función devuelve una lista con dos elementos. El primer elemento es
`wordDict[wordKey][wordIndex]`. El segundo elemento es `wordKey`.

La expresión `wordDict[wordKey][wordIndex]` en la línea 61 puede parecer complicada, pero es solo una expresión que puede evaluar un paso a la vez como cualquier otra cosa. Primero, imagina esa palabra Clave tiene el valor 'Frutas' y `wordIndex` tiene el valor 5. Así es como `wordDict[wordKey][wordIndex]` evaluaría:



En este caso, el elemento de la lista que devuelve esta función sería la cadena 'sandía'. (Recuerde que los índices comienzan en 0, por lo que [5] se refiere al sexto elemento de la lista, no al quinto).

Debido a que la función getRandomWord() ahora devuelve una lista de dos elementos en lugar de una cadena, a `palabrasecreta` se le asignará una lista, no una cadena. Puede asignar estos dos elementos a dos variables separadas mediante la asignación múltiple, que trataremos en ["Asignación múltiple"](#) en la página 118.

ELIMINACIÓN DE ELEMENTOS DE LISTAS

Una declaración del eliminará un elemento en un cierto índice de un lista. Debido a que `del` es una declaración, no una función ni un operador, no tiene paréntesis ni se evalúa como un valor devuelto. Para probarlo, ingrese lo siguiente en el shell interactivo:

```

>>> animales = ['oso hormiguero', 'oso hormiguero', 'antílope', 'albert']
>>> del animales[1]
>>> animales
['oso hormiguero', 'antílope', 'albert']

```

Observe que cuando eliminó el elemento en el índice 1, el elemento que solía estar en el índice 2 se convirtió en el nuevo valor en el índice 1; la

el elemento que solía estar en el índice 3 se convirtió en el nuevo valor en el índice 2; y así. Todo lo que estaba encima del elemento eliminado se movió hacia abajo un índice

Puedes escribir del `animales[1]` una y otra vez para seguir eliminando elementos de la lista:

```
>>> animales = ['oso hormiguero', 'oso hormiguero', 'antilope', 'albert']
>>> del animales[1] >>> animales

['oso hormiguero', 'antilope',
'albert'] >>> del animales[1]
>>> animales
['aardvark', 'albert']
>>> del animales[1]
>>> animales

['cerdo hormiguero']
```

La longitud de la lista `HANGMAN_PICS` también es el número de intentos que obtiene el jugador. Al eliminar cadenas de esta lista, puede reducir la cantidad de intentos y hacer que el juego sea más difícil.

Agregue las siguientes líneas de código a su programa entre las líneas `print('HANGMA N')` y `missLetters = ":"`:

```
103. print('HANGMA N')
104.
        "
105. dificultad =
106. mientras que la dificultad no está
en 'EMH': 107. print("Ingrese la dificultad: E - Fácil, M - Media, H - Difícil")
108. dificultad = entrada().superior() 109. si la dificultad == 'M': 110. del
FOTOS_AHORCADO[8] 111. del FOTOS_AHORCADO[7] 112. if
dificultad == 'H': 113. del FOTOS_AHORCADO[8] 114. del
FOTOS_AHORCADO[7] 115. del FOTOS_AHORCADO[5 ] 116. del
AHORCADO_FOTOS[3]
```

117.

118. letrasperdidas = "

Este código elimina elementos de la lista HANGMAN_PICS , haciéndola más corta según el nivel de dificultad seleccionado. A medida que aumenta el nivel de dificultad, se eliminan más elementos de la lista HANGMAN_PICS , lo que genera menos conjeturas. El resto del código en el juego Hangman usa la longitud de esta lista para saber cuándo el jugador se ha quedado sin conjeturas.

ASIGNACIÓN MÚLTIPLE

La asignación múltiple es un atajo para asignar múltiples variables en una línea de código. Para usar asignación múltiple, separe sus variables con comas y asígnelas a una lista de valores. Por ejemplo, ingrese lo siguiente en el shell interactivo:

```
>>> spam, huevos, jamón = ['manzanas', 'gatos', 42]
>>> spam
'manzanas'
>>> huevos
'gatos'
>>> también
42
```

El ejemplo anterior es equivalente al siguiente declaraciones de asignación:

```
>>> spam = ['manzanas', 'gatos', 42][0] >>>
huevos = ['manzanas', 'gatos', 42][1] >>> jamón
= ['manzanas', 'gatos ', 42][2]
```

Debe colocar el mismo número de variables en el lado izquierdo del operador de asignación = que elementos en la lista del lado derecho. Python asignará automáticamente el valor de la

primer elemento de la lista a la primera variable, el valor del segundo elemento a la segunda variable, y así sucesivamente. Si no tiene la misma cantidad de variables y elementos, el intérprete de Python le dará un error, así:

```
>>> spam, huevos, jamón, tocino = ['manzanas', 'gatos', 42, 10, 'hola']
```

Rastreo (llamadas recientes más última):

Archivo "<pyshell#8>", línea 1, en <módulo> spam,

```
huevos, jamón, tocino = ['manzanas', 'gatos', 42, 10, 'hola']
```

ValueError: demasiados valores para desempaquetar

```
>>> spam, huevos, jamón, tocino = ['manzanas', 'gatos']
```

Rastreo (llamadas recientes más última):

Archivo "<pyshell#9>", línea 1, en <módulo> spam,

```
huevos, jamón, tocino = ['manzanas', 'gatos']
```

ValueError: necesita más de 2 valores para desempaquetar

Cambie las líneas 120 y 157 del código Hangman para usar asignación múltiple con el valor de retorno de getRandomWord():

```
119. letrascorrectas = ""
```

```
120. palabrasecreta, conjuntosecreto = obtenerPalabraAleatoria (palabras)
```

```
121. juegosDone = Falso --recorte-- 156.
```

```
157.     gamelsDone = False
158.     secretWord, secretSet = getRandomWord(palabras)
159.     más:
             descanso
```

La línea 120 asigna los dos valores devueltos de getRandomWord(words) a secretWord y secretSet. La línea 157 vuelve a hacer esto si el jugador elige jugar otro juego.

IMPRESIÓN DE LA CATEGORÍA DE PALABRAS PARA EL JUGADOR

El último cambio que harás es decirle al jugador qué conjunto de palabras está tratando de adivinar. De esta forma, el jugador sabrá si la palabra secreta es un animal, un color, una forma o una fruta. Aquí está el código original:

91. mientras sea cierto:

92. tablero de visualización (letras perdidas, letras correctas, palabra secreta)

En su nueva versión de Hangman, agregue la línea 124 para que su programa se vea así:

123. mientras sea cierto:

124. print('La palabra secreta está en el conjunto: 125. ' + conjunto secreto)

displayBoard(letrasperdidas, letrascorrectas, palabrasecreta)

Ahora ha terminado con los cambios en el programa Hangman. En lugar de una sola lista de cadenas, la palabra secreta se elige de muchas listas diferentes de cadenas. El programa también le dice al jugador de qué conjunto de palabras proviene la palabra secreta. Intenta jugar esta nueva versión. Puede cambiar fácilmente el diccionario de palabras a partir de la línea 48 para incluir más conjuntos de

palabras.

RESUMEN

¡Terminamos con Hangman! Aprendió algunos conceptos nuevos cuando agregó las características adicionales en este capítulo. Incluso después de que haya terminado de escribir un juego, siempre puede agregar más funciones a medida que aprende más sobre la programación de Python.

Los diccionarios son similares a las listas, excepto que pueden usar cualquier tipo de valor para un índice, no solo números enteros. Los índices en los diccionarios se llaman claves. La asignación múltiple es un atajo para asignar múltiples variables a los valores de una lista.

Hangman era bastante avanzado en comparación con los juegos anteriores de este libro. Pero en este punto, conoce la mayoría de los conceptos básicos para escribir programas: variables, bucles, funciones y tipos de datos como listas y diccionarios. Los últimos programas de este libro seguirán siendo un desafío para dominar, ¡pero has terminado la parte más empinada de la subida!

10

TIC-TAC-TOE



Este capítulo presenta un juego de Tic-Tac-Toe. Tic-Tac-Toe normalmente se juega con dos personas. Un jugador es X y el otro jugador es O. Los jugadores se turnan para colocar su X u O. Si un jugador obtiene tres de sus marcas en el tablero en una fila, columna o diagonal, gana. Cuando el tablero se llena sin que ningún jugador gane, el juego termina en empate.

Este capítulo no introduce muchos conceptos nuevos de programación. El usuario jugará contra una inteligencia artificial simple, que escribiremos utilizando nuestros conocimientos de programación existentes. Una inteligencia artificial (AI) es un programa de computadora que puede responder inteligentemente a los movimientos del jugador. La IA que juega Tic-Tac-Toe no es complicada; en realidad son solo unas pocas líneas de código.

Comencemos mirando una ejecución de muestra del programa. El jugador hace su movimiento ingresando el número del espacio que desea tomar. Para ayudarnos a recordar qué índice de la lista corresponde a qué espacio, numeraremos el tablero como un

el teclado numérico del teclado, como se muestra en la Figura 10-1.

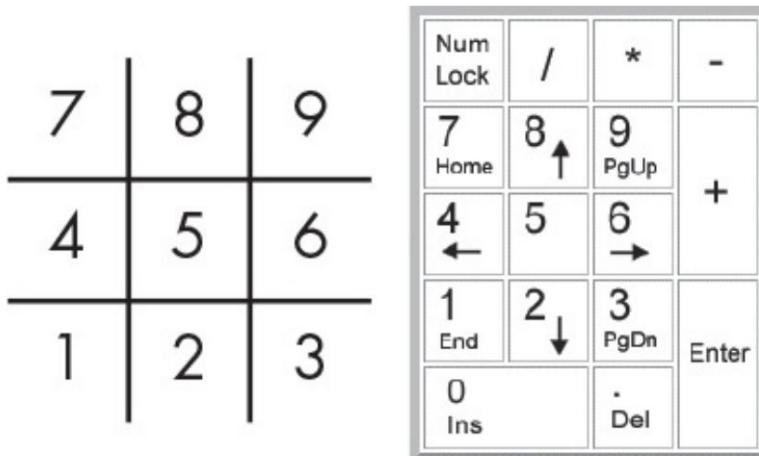


Figura 10-1: El tablero está numerado como el teclado numérico del teclado.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Inteligencia artificial
- Lista de referencias
- Evaluación de cortocircuito
- El valor Ninguno

MUESTRA DE EJECUCIÓN DE TIC-TAC-TOE

Esto es lo que ve el usuario cuando ejecuta el programa Tic-Tac-Toe. El texto que ingresa el jugador está en negrita.

```

¡Bienvenido a Tic-Tac-Toe!
¿Quieres ser X u O?
X
La computadora irá primero.
el| |
-+-+
|||
-+-+
|
| ¿Cuál es tu próximo movimiento? (1-9)
3
el| |

```

-++-

||

-++-

O| X

¿Cuál es tu próximo movimiento? (1-9)

4

O| |La

-++-

X| |

-++-

O| X

¿Cuál es tu próximo movimiento? (1-9)

5

O|O|O

-++-

X|X|

-++-

O| X

¡La computadora te ha ganado! Tú pierdes.

¿Quieres jugar de nuevo? (sí o no)

no

CÓDIGO FUENTE PARA TIC-TAC-TOE

En un archivo nuevo, ingrese el siguiente código fuente y guárdelo como tictactoe.py. Luego ejecuta el juego presionando F5. Si obtiene errores, compare el código que escribió con el código del libro con el código en línea

diferencia

herramienta

a

<https://www.nostarch.com/inventwithpython#diff>.



tictactoe.py

```
1. # tres en raya
2.
3. importar al azar
4.
5. def dibujarTablero(tablero):
6. # Esta función imprime el tablero que se pasó.
7.
8. # "tablero" es una lista de 10 cadenas que representan el tablero (ignorar
índice 0).
9. print(tablero[7] + ' | ' + tablero[8] + ' | ' + tablero[9]) 10.
print('---') 11. print(tablero[4] + ' | ' + tablero[5] + ' | ' +
tablero[6]) 12. print('---') 13. print(tablero[1] + ' | ' +
tablero[2] + ' | ' + tablero[3])
```

14

```
15. def inputPlayerLetter(): 16.
# Permite al jugador escribir qué letra quiere ser.
17. # Devuelve una lista con la letra del jugador como primer elemento y el
    la letra de la computadora como la segunda.
18     letra =
19. while not (letra == 'X' o letra == 'O'):
20     print('¿Quieres ser X u O?') letra =
21     entrada().superior()
22
```

```
23. # El primer elemento de la lista es la carta del jugador; la segunda es la letra  
    de la computadora.  
24. si letra == 'X':  
25     devolver ['X', 'O']  
26. más:  
27     devolver ['O', 'X']  
28  
29. def quién va primero():  
30. # Elige al azar qué jugador va primero. 31. if  
    random.randint(0, 1) == 0: 32. return 'computadora'  
  
33. más:  
34.     devolver 'jugador'  
35.  
36. def hacerJugada(tablero, letra, jugada):  
37. tablero[jugada] = letra  
38.  
39. def isWinner(bo, le): 40.  
# Dado un tablero y la carta de un jugador, esta función devuelve True si  
    ese jugador ha ganado.  
41. # Usamos "bo" en lugar de "tablero" y "le" en lugar de "letra" por lo que  
    no tienes que escribir tanto. 42.  
return ((bo[7] == le and bo[8] == le and bo[9] == le) o # A través del  
    cima  
43. (bo[4] == le y bo[5] == le y bo[6] == le) o # En el medio 44. (bo[1] == le y bo[2] == le  
    y bo[3] == le) o # En la parte inferior 45. (bo[7] == le y bo[4] == le y bo[1] == le) o # Por  
    el lado izquierdo 46. (bo[8] == le y bo[5] == le y bo[2] == le) o # Por el medio 47. (bo[9]  
    == le y bo[6] == le y bo[3] == le) o # Abajo a la derecha  
  
    lado  
48. (bo[7] == y y bo[5] == y y bo[3] == y) o # Diagonal 49. (bo[9] == y y bo[5]  
    == y y bo[1] == y) # Diagonal  
50  
51. def getBoardCopy(tablero): 52.  
# Hacer una copia de la lista del tablero y devolverla. 53.  
copia del tablero = []  
54. para i en tablero:  
55.     boardCopy.append(i)  
56. return boardCopy  
57.
```

```
58. def isSpaceFree(board, move): 59. #
Devuelve True si el movimiento pasado es libre en el tablero pasado. 60. regresar
tablero[mover] ==
61.
62. def obtenerJugadaJugador(tablero):
63. # Permitir que el jugador ingrese su jugada.
64. mover =
65. while move no está en '1 2 3 4 5 6 7 8 9'.split() or not
    isSpaceFree(board, int(move)): print('¿Cuál es tu siguiente
66.     movimiento? (1-9)') move = entrada() 68. return int(mover)
67.
68.
69.
70. def elegirMovidaAleatoriaDeLista(tablero, listadmovimientos): 71. #
Devuelve un movimiento válido de la lista de pasadas en el tablero de pasadas.
72. # Devuelve Ninguno si no hay un movimiento válido.
73. MovimientosPosibles = []
74. para i en listaMovimientos:
75.     si esEspacioLibre(tablero, i):
76.         jugadasposibles.append(i)
77.
78. if len(movimientosposibles) != 0: 79.
    return random.choice(movimientosposibles)
79. más:
80.     volver Ninguno
81.
82.
83. def getComputerMove(board, computerLetter): 84. # Dado un
tablero y la letra de la computadora, determina dónde mover
y devolver ese movimiento.
85. if computadoraLetra == 'X':
86.     jugadorLetra = 'O'
87. más:
88.     jugadorLetra = 'X'
89.
90. # Aquí está el algoritmo para nuestra IA de Tic-Tac-Toe: 91. #
Primero, verifica si podemos ganar en el siguiente movimiento. 92.
for i in range(1, 10): boardCopy = getBoardCopy(board) if
93.     isSpaceFree(boardCopy, i):
94.
95.         hacerMovimiento(tableroCopia, computadoraLetra, i)
```

```

96.         if isWinner(boardCopy, computerLetter):
97.             vuelvo yo
98.

99. # Comprueba si el jugador podría ganar en su próximo movimiento y bloquéalo.
100. for i in range(1, 10): copiaTablero = obtenerCopiaTablero(tablero) if
101.     hayEspacioLibre(copiatablero, i): hacerJugada(copiatablero, letraJugador, i)
102.     if esGanador(copiaTablero, letraJugador): return i
103.
104.
105.
106.

107. # Intenta tomar una de las esquinas, si están libres. 108.
mover = elegirJugadaAleatoriaDeLista(tablero, [1, 3, 7, 9]) 109. si mover != Ninguno:
110.     movimiento de regreso
111.

112. # Intenta tomar el centro, si está libre. 113. si
hay espacio libre (tablero, 5):
114.     volver 5
115.

116. # Muévete por uno de los lados.
117. return elegirMovidaAleatoriaDeLista(tablero, [2, 4, 6, 8]) 118.

119. def isBoardFull(board): 120.
# Devuelve True si se han ocupado todos los espacios del tablero. De lo contrario,
devuelve Falso.
121. for i in range(1, 10): 122. if
isSpaceFree(board, i):
123.     falso retorno
124. volver Verdadero
125.
126.

127. print('¡Bienvenido a Tic-Tac-Toe!') 128.

129. mientras sea cierto:
130. # Restablecer el tablero.
131. elTablero = [' '] * 10 132.
letraJugador, letraEquipo = entradaLetraJugador() 133. turno =
quienVaPrimero() 134. print('El '
+ girar + ' Irá primero.')

```

```
135. gamelsPlaying = Verdadero
136.
137. while gamelsPlaying: 138. if
turno == 'jugador': # Turno del
jugador
139.     dibujarTablero(elTablero)
140.     mover =
141.     obtenerJugadorMovimiento(elTablero)
142.     hacerMovimiento(elTablero, letraJugador, movimiento)
143.
144.     if esGanador(elTablero, letraJugador):
145.         dibujarTablero(elTablero) print('¡Hurra!
146.         ¡Has ganado el juego!') juegoEstáJugando = False
147.     else:
148.
149.     if elTableroLleno(elTablero):
150.         dibujarTablero(elTablero)
151.         print('¡El juego es un empate!')
152.         descanso
153.     más:
154.         turno = 'computadora'
155.
156.     más:
157.         # Turno de la
158.         computadora mover = obtenerJugadaComputadora(elTablero,
159.             letracomputadora) hacerJugada(elTablero, letracomputadora, jugada)
160.
161.         if esGanador(elTablero, computadoraLetra):
162.             dibujarTablero(elTablero) print('¡La computadora
163.             te ganó! Tú pierdes.') juegoEstáJugando = False
164.
165.     más:
166.         if elTableroLleno(elTablero):
167.             dibujarTablero(elTablero)
168.             print('¡El juego es un empate!')
169.             break
170.     más:
171.         turno = 'jugador'
172.
173. print('¿Quieres volver a jugar? (sí o no)') 174. if not
input().lower().startswith('y'):
```

DISEÑO DEL PROGRAMA

La figura 10-2 muestra un diagrama de flujo del programa Tic-Tac-Toe.

El programa comienza pidiéndole al jugador que elija su letra, X u O.

Se elige al azar quién toma el primer turno. Luego, el jugador y la computadora se turnan para hacer movimientos.

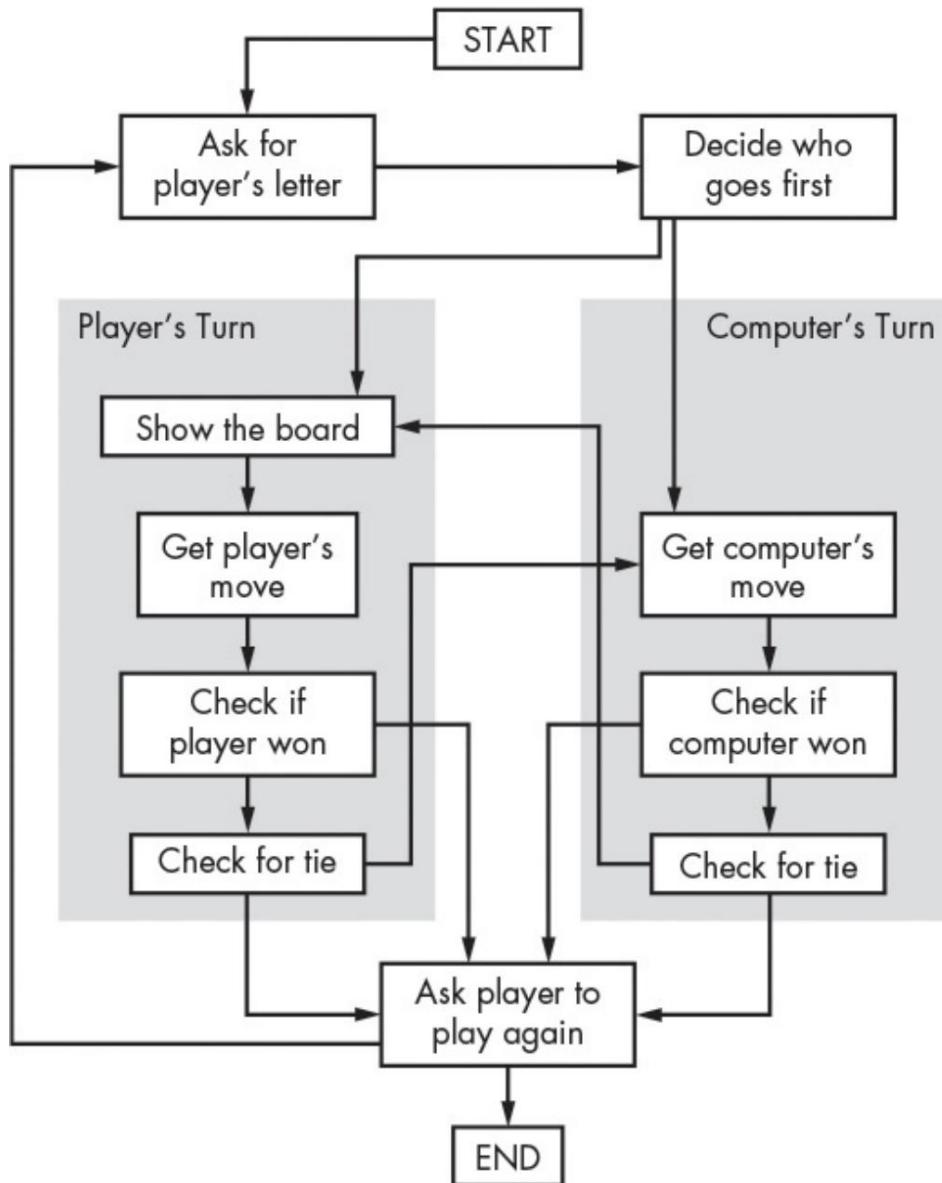


Figura 10-2: Diagrama de flujo para Tic-Tac-Toe

Los cuadros en el lado izquierdo del diagrama de flujo muestran lo que sucede durante el turno del jugador, y los del lado derecho muestran lo que sucede durante el turno de la computadora. Después de que el jugador o la computadora hace un movimiento, el programa verifica si ganó o provocó un empate, y luego el juego cambia de turno. Una vez que termina el juego, el programa le pregunta al jugador si quiere volver a jugar.

Representar el tablero como datos Primero, debe descubrir cómo representar el tablero como datos en una variable. En papel, el tablero de Tic-Tac-Toe se dibuja como un par de líneas horizontales y un par de líneas verticales, con una X, O o un espacio vacío en cada uno de los nueve espacios.

En el programa, el tablero de Tic-Tac-Toe se representa como una lista de cadenas como el arte ASCII de Hangman. Cada cuerda representa uno de los nueve espacios en el tablero. Las cadenas son 'X' para el jugador X , 'O' para el jugador O o un solo espacio ' ' para un espacio en blanco.

Recuerde que estamos diseñando nuestro tablero como un teclado numérico en un teclado. Entonces, si una lista con 10 cadenas se almacenó en una variable llamada tablero, tablero [7] sería el espacio superior izquierdo del tablero, tablero[8] sería el espacio superior central, tablero[9] sería el espacio espacio superior derecho, y así sucesivamente. El programa ignora la cadena en el índice 0 en la lista. El jugador ingresará un número del 1 al 9 para decirle al juego en qué espacio desea moverse.

Elaboración de estrategias con la IA del juego
La IA debe poder mirar el tablero y decidir qué tipos de espacios en los que se moverá. Para que quede claro, etiquetaremos tres tipos de espacios en el tablero de Tic-Tac-Toe: esquinas, lados y

el centro El gráfico de la figura 10-3 muestra qué es cada espacio.

La estrategia de la IA para jugar Tic-Tac-Toe seguirá un algoritmo simple: una serie finita de instrucciones para calcular un resultado. Un solo programa puede hacer uso de varios algoritmos diferentes. Un algoritmo se puede representar con un diagrama de flujo.

El algoritmo de la IA de Tic-Tac-Toe calculará el mejor movimiento a realizar, como se muestra en la Figura 10-4.

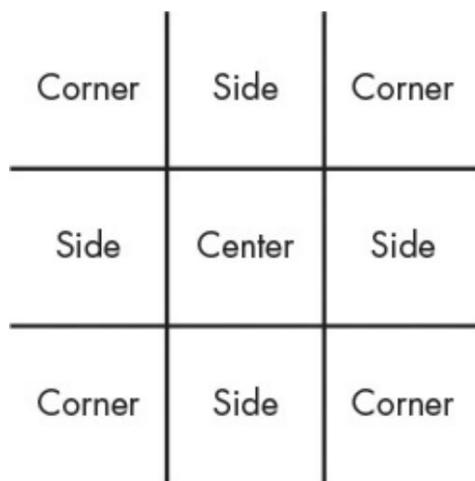


Figura 10-3: Ubicaciones de los espacios laterales, de esquina y centrales

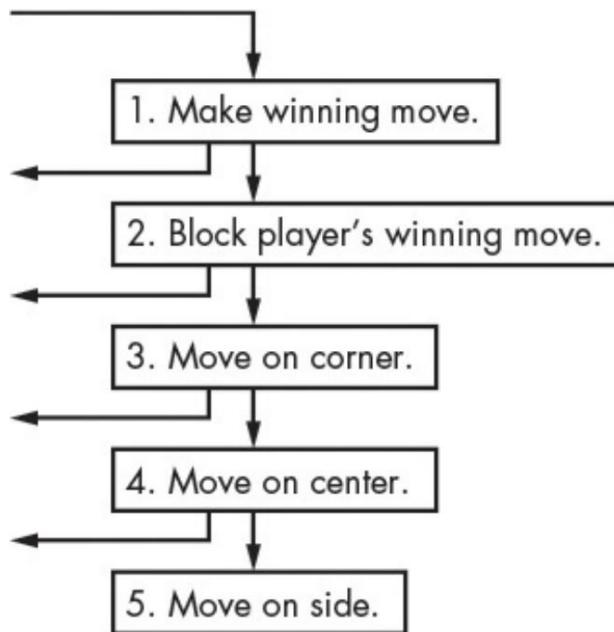


Figura 10-4: Los cuadros representan los cinco pasos del algoritmo “Obtener el movimiento de la computadora”. Las flechas que apuntan hacia la izquierda van a la casilla “Comprobar si la computadora ganó”.

El algoritmo de la IA tiene los siguientes pasos:

1. Vea si hay un movimiento que la computadora puede hacer para ganar el juego. Si lo hay, haz ese movimiento. De lo contrario, vaya al paso 2.
2. Vea si hay un movimiento que el jugador pueda hacer que haga que la computadora pierda el juego. Si lo hay, muévete allí para bloquear al jugador. De lo contrario, vaya al paso 3.
3. Compruebe si alguno de los espacios de las esquinas (espacios 1, 3, 7 o 9) está libre. Si es así, muévete allá. Si no hay espacio libre en la esquina, vaya al paso 4.
4. Comprobar si el centro está libre. Si es así, muévete allí. Si no es así, vaya al paso 5.
5. Muévase en cualquiera de los espacios laterales (espacios 2, 4, 6 u 8). No hay más pasos porque los espacios laterales son todo lo que queda si la ejecución llega al paso 5.

Todo esto tiene lugar en el cuadro de movimiento Get computer's en el diagrama de flujo de la Figura 10-2. Podría agregar esta información al diagrama de flujo con los cuadros en la Figura 10-4.

Este algoritmo se implementa en getComputerMove() y el otras funciones a las que llama getComputerMove() .

IMPORTAR EL ALEATORIO MÓDULO

El primer par de líneas se componen de un comentario y una línea que importa el módulo aleatorio para que pueda llamar al randint() función más adelante:

-
1. # tres en raya
 - 2.
 3. importar al azar
-

Has visto estos dos conceptos antes, así que sigamos adelante.
a la siguiente parte del programa.

IMPRESIÓN DEL TABLERO EN EL PANTALLA

En la siguiente parte del código, definimos una función para dibujar el tablero:

```
5. def dibujarTablero(tablero):
6. # Esta función imprime el tablero que se pasó.
7.
8. # "tablero" es una lista de 10 cadenas que representan el tablero (ignorar
índice 0).
9. print(tablero[7] + '|' + tablero[8] + '|' + tablero[9]) 10.
print('---') 11. print(tablero[4] + ' ' + tablero[5] + ' ' +
tablero[6]) 12. print('---') 13. print(tablero[1] + ' ' +
tablero[2] + ' ' + tablero[3])
```

La función `dibujarTablero()` imprime el tablero de juego representado por el parámetro `tablero`. Recuerde que el tablero se representa como una lista de 10 cadenas, donde la cadena en el índice 1 es la marca en el espacio 1 en el tablero de Tic-Tac-Toe, y así sucesivamente. La cadena en el índice 0 se ignora. Muchas de las funciones del juego funcionan pasando una lista de 10 cadenas como tablero.

Asegúrese de obtener el espacio correcto en las cuerdas; de lo contrario, el tablero se verá divertido cuando se imprima en la pantalla. Aquí hay algunos ejemplos de llamadas (con un argumento para `tablero`) a `dibujarTablero()` y lo que imprimiría la función.

```
>>> dibujarTablero([' ', ' ', ' ', ' ', 'X', 'O', ' ', 'X', ' ', 'O'])
X| |
-+-+
X|O|
-+-+
|
| >>> dibujarTablero([' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '])
```

```
||  
-+-+
```

```
||  
-+-+
```

```
||
```

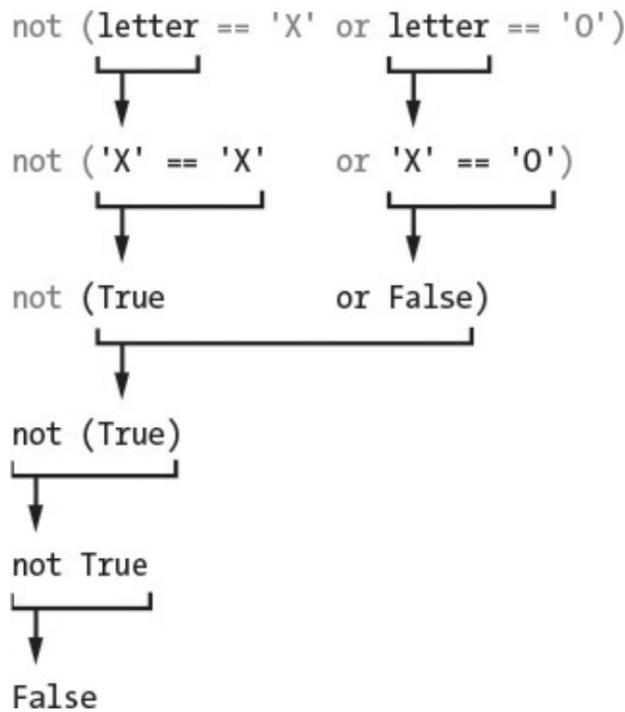
El programa toma cada cadena y la coloca en el tablero en orden numérico de acuerdo con el teclado numérico del teclado de la Figura 10-1, por lo que las primeras tres cadenas son la fila inferior del tablero, las siguientes tres cadenas son la del medio y la última. tres cuerdas son la parte superior.

DEJAR QUE EL JUGADOR ELIJA X O O

A continuación, definiremos una función para asignar X u O al jugador:

```
15. def inputPlayerLetter():  
16. # Permite al jugador ingresar qué letra quiere ser.  
17. # Devuelve una lista con la letra del jugador como primer elemento y el  
    la letra de la computadora como la segunda.  
    "  
18. letra =  
19. while not (letra == 'X' o letra == 'O'):  
20     print('¿Quieres ser X u O?') letra =  
21     entrada().superior()
```

La función `inputPlayerLetter()` pregunta si el jugador quiere ser X u O. La condición del ciclo `while` contiene paréntesis, lo que significa que la expresión dentro de los paréntesis se evalúa primero. Si la variable de letra se estableció en 'X', la expresión evaluaría así:



Si la letra tiene el valor 'X' o 'O', entonces la condición del ciclo es Falso y permite que la ejecución del programa continúe más allá del bloque while . Si la condición es verdadera, el programa seguirá pidiéndole al jugador que elija una letra hasta que el jugador ingrese una X o una O. La línea 21 cambia automáticamente la cadena devuelta por la llamada a input() a letras mayúsculas con el método de cadena upper() .

La siguiente función devuelve una lista con dos elementos:

```

23. # El primer elemento de la lista es la carta del jugador; la segunda es la
     letra de la computadora.
24. si letra == 'X':
25     devolver ['X', 'O']
26. más:
27     devolver ['O', 'X']
  
```

El primer elemento (la cadena en el índice 0) es la letra del jugador, y el segundo elemento (la cadena en el índice 1) es la letra de la computadora. Las sentencias if y else eligen la lista apropiada para retorno.

DECIDIR QUIÉN VA PRIMERO

Luego creamos una función que usa randint() para elegir si el jugador o la computadora juega primero:

```
29. def quién va
primero(): 30. # Elige al azar qué jugador va primero.
31. if random.randint(0, 1) == 0: devuelve
32.     'computadora'
33. más:
34.     devolver 'jugador'
```

La función whoGoesFirst() lanza una moneda virtual para determinar si la computadora o el jugador van primero. El lanzamiento de la moneda se realiza con una llamada a random.randint(0, 1). Hay un 50 por ciento de posibilidades de que la función devuelva 0 y un 50 por ciento de posibilidades de que la función devuelva 1. Si esta llamada a la función devuelve un 0, la función whoGoesFirst() devuelve la cadena 'computadora'. De lo contrario, la función devuelve la cadena 'jugador'. El código que llama a esto La función usará el valor devuelto para determinar quién hará el primer movimiento del juego.

COLOCAR UNA MARCA EN EL TABLERO

La función hacerJugada() es simple:

```
36. def hacerJugada(tablero, letra,
jugada): 37. tablero[jugada] = letra
```

Los parámetros son tablero, letra y movimiento. La variable tablero es la lista de 10 cadenas que representa el estado del tablero. La letra variable es la letra del jugador (ya sea 'X' o 'O'). El movimiento variable es el lugar en el tablero donde ese jugador quiere

to go (que es un número entero del 1 al 9).

Pero espera, en la línea 37, este código parece cambiar uno de los artículos en la lista del tablero al valor en letra. Porque este código es en una función, sin embargo, el parámetro del tablero se olvidará cuando la función regrese. Entonces, ¿no debería olvidarse también el cambio de tablero ?

En realidad, este no es el caso, porque las listas son especiales cuando las pasas como argumentos a las funciones. En realidad, está pasando una referencia a la lista, no a la lista en sí. Aprendamos sobre la diferencia entre listas y referencias a listas.

Listas de referencias

Ingrrese lo siguiente en el shell interactivo:

```
>>> spam = 42
>>> queso = spam
>>> spam = 100
>>> spam
100
>>> queso
42
```

Estos resultados tienen sentido a partir de lo que sabe hasta ahora. Asigne 42 a la variable spam , luego asigne el valor en spam a la variable queso. Cuando más tarde sobrescribe el correo no deseado a 100, esto no afecta el valor en queso. Esto se debe a que el spam y el queso son variables diferentes que almacenan valores diferentes.

Pero las listas no funcionan de esta manera. Cuando asigna una lista a una variable, en realidad está asignando una referencia de lista a la variable. Una referencia es un valor que apunta a la ubicación donde se almacena algún bit de datos. Veamos un código que hará que esto sea más fácil de entender. Ingrrese esto en el

concha interactiva:

```
>>> spam = [0, 1, 2, 3, 4, 5]
>>> queso = spam    >>>
queso[1] = '¡Hola!'
>>> spam
[0, '¡Hola!', 2, 3, 4, 5]
>>> queso
[0, '¡Hola!', 2, 3, 4, 5]
```

El código solo cambió la lista de quesos , pero parece que tanto la lista de quesos como la de spam han cambiado. Esto se debe a que el spam variable no contiene el valor de la lista en sí, sino más bien un referencia a la lista, como se muestra en la Figura 10-5. La lista en sí no está contenida en ninguna variable sino que existe fuera de ellas.

❶ `spam = [0, 1, 2, 3, 4, 5]`

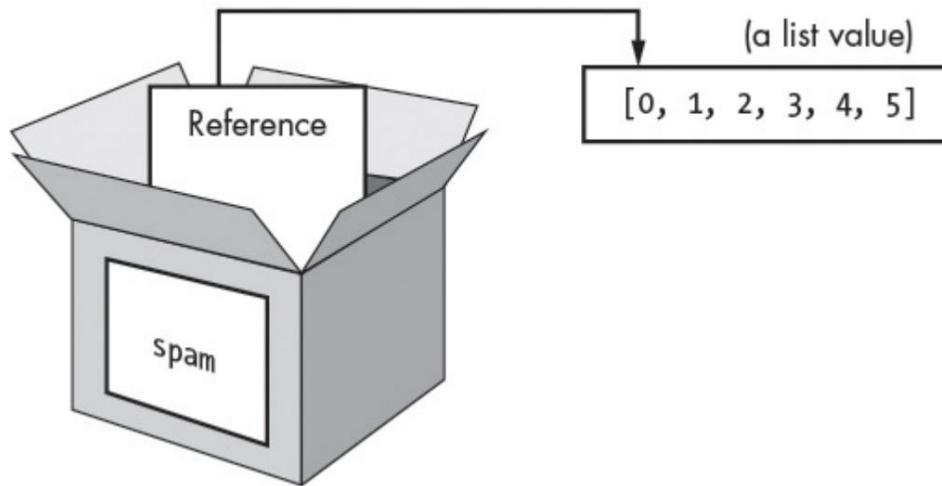


Figura 10-5: La lista de spam creada en . Las variables no almacenan listas sino referencias a listas.

Tenga en cuenta que `cheese = spam` copia la referencia de la lista en `spam` a `cheese` , en lugar de copiar el valor de la lista en sí. Ahora ambos `spam` y `cheese` almacenan una referencia que se refiere al mismo valor de lista.

Pero solo hay una lista porque la lista en sí no se copió.

La figura 10-6 muestra esta copia.

❷ `cheese = spam`

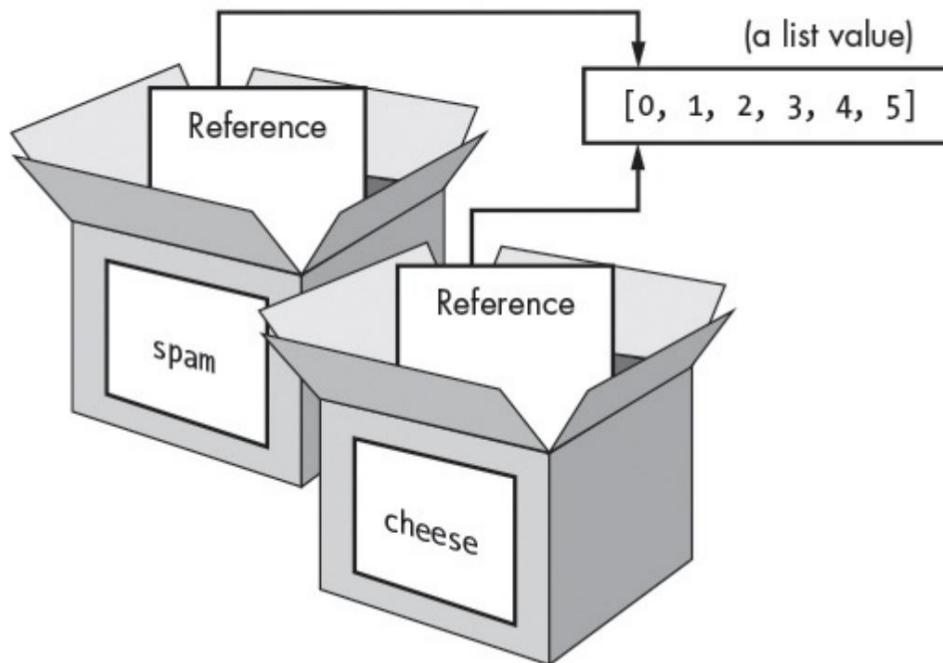


Figura 10-6: Las variables `spam` y `cheese` almacenan dos referencias a la misma lista.

Entonces el `queso[1] = '¡Hola!'` la línea en cambia la misma lista a la que se refiere el `spam`. Esta es la razón por la que el `spam` devuelve el mismo valor de lista que el `queso`. Ambos tienen referencias que se refieren a la misma lista, como se muestra en la Figura 10-7.

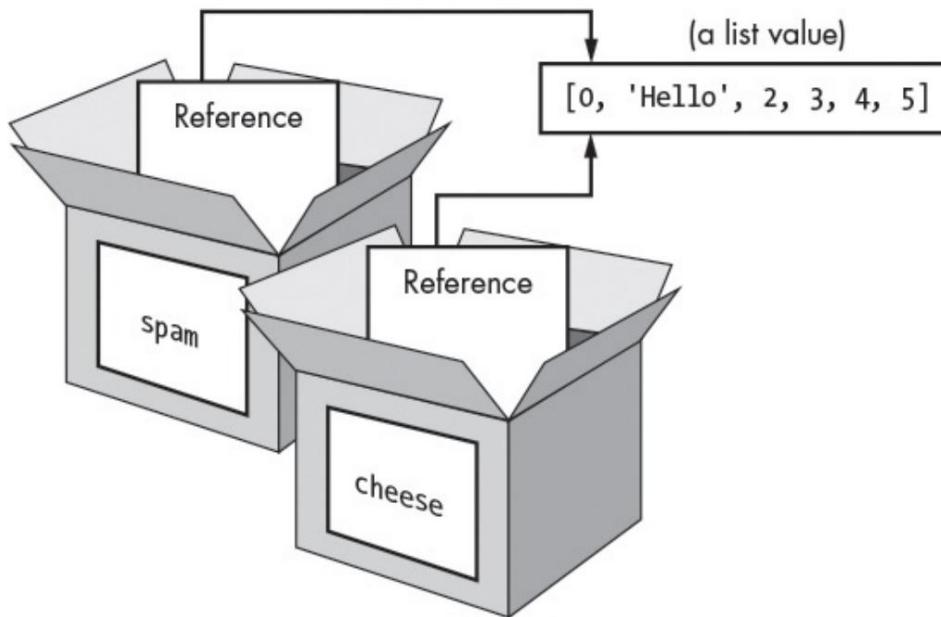


Figura 10-7: Cambiar la lista cambia todas las variables con referencias a esa lista.

Si desea que el spam y el queso almacenen dos listas diferentes, debe tener que crear dos listas en lugar de copiar una referencia:

```
>>> spam = [0, 1, 2, 3, 4, 5]
>>> queso = [0, 1, 2, 3, 4, 5]
```

En el ejemplo anterior, spam y cheese almacenan dos listas diferentes (aunque estas listas son idénticas en contenido).

Ahora, si modifica una de las listas, no afectará a la otra porque spam y cheese tienen referencias a dos listas diferentes:

```
>>> spam = [0, 1, 2, 3, 4, 5]
>>> queso = [0, 1, 2, 3, 4, 5]
>>> queso[1] = '¡Hola!'
>>> spam
[0, 1, 2, 3, 4, 5]
>>> queso
[0, '¡Hola!', 2, 3, 4, 5]
```

La figura 10-8 muestra cómo se configuran las variables y los valores de lista en este ejemplo.

Los diccionarios funcionan de la misma manera. Las variables no almacenan diccionarios; almacenan referencias a diccionarios.

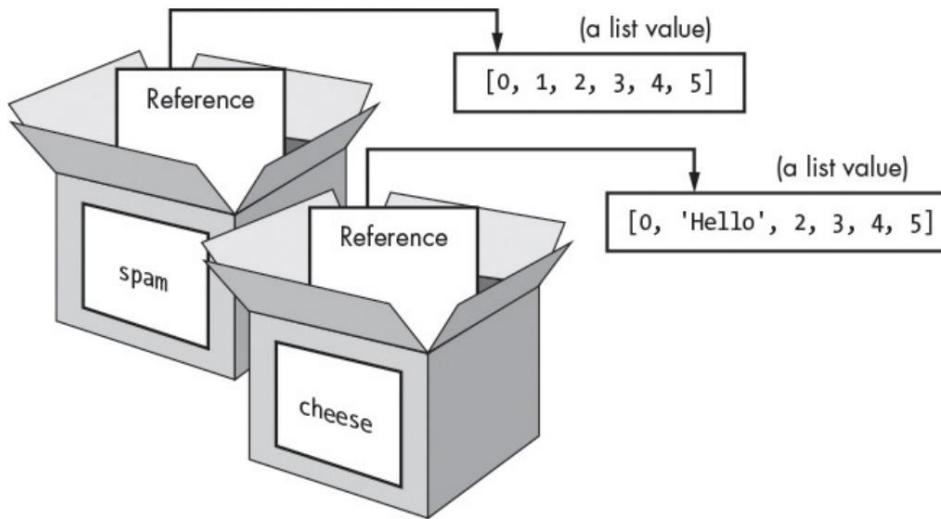


Figura 10-8: Las variables spam y cheese ahora cada tienda hace referencia a dos listas diferentes.

Usando Lista de Referencias en hacerJugada()

Volvamos a la función hacerJugada() :

```

36. def hacerJugada(tablero, letra, jugada):
37.     tablero[jugada] = letra

```

Cuando se pasa un valor de lista para el parámetro del tablero , la variable local de la función es realmente una copia de la referencia a la lista, no una copia de la lista en sí. Por lo tanto, cualquier cambio en el tablero en esta función también se realizará en la lista original. Aunque tablero es una variable local, la función hacerJugada() modifica la lista original.

Los parámetros de letra y movimiento son copias de la cadena y los valores enteros que pasa. Como son copias de valores, si modifica letras o mueve esta función, las variables originales que usó cuando llamó a hacerJugada() no se modifican.

COMPROBAR SI EL JUGADOR GANADO

Las líneas 42 a 49 en la función isWinner() son en realidad una larga

declaración de devolución :

```
39. def isWinner(bo, le): 40.  
    # Dado un tablero y la carta de un jugador, esta función devuelve True si  
    # ese jugador ha ganado.  
41. # Usamos "bo" en lugar de "tablero" y "le" en lugar de "letra" por lo que  
    # no tienes que escribir tanto. 42.  
    return ((bo[7] == le and bo[8] == le and bo[9] == le) or # A través del  
    # cima  
43. (bo[4] == le and bo[5] == le and bo[6] == le) or # En el medio 44. (bo[1] == le and bo[2] == le  
    # y bo[3] == le) or # En la parte inferior 45. (bo[7] == le and bo[4] == le and bo[1] == le) or # Por  
    # el lado izquierdo 46. (bo[8] == le and bo[5] == le and bo[2] == le) or # Por el medio 47. (bo[9]  
    == le and bo[6] == le and bo[3] == le) or # Abajo a la derecha  
  
    # lado  
48. (bo[7] == y and bo[5] == y and bo[3] == y) or # Diagonal 49. (bo[9] == y and bo[5]  
    == y and bo[1] == y) # Diagonal
```

Los nombres bo y le son atajos para el tablero y la letra.

parámetros Estos nombres más cortos significan que tiene menos que escribir en esta función. Recuerda, a Python no le importa el nombre que le des a tus variables.

Hay ocho formas posibles de ganar en Tic-Tac-Toe: puede tener una línea en las filas superior, media o inferior; puede tener una línea en las columnas izquierda, central o derecha; o puede tener una línea a través de cualquiera de las dos diagonales.

Cada línea de la condición verifica si los tres espacios para una línea dada son iguales a la letra provista (combinada con el operador y).

Combina cada línea usando el operador o para verificar las ocho formas diferentes de ganar. Esto significa solo

una de las ocho formas debe ser Verdadera para que podamos decir que el jugador que posee la letra en le es el ganador.

Supongamos que le es 'O' y bo es [' ', 'O', 'O', 'O', ' ', 'X', ' ', 'X', ' ', ' ']. El tablero quedaría así:

X| |
-+-+

|X|
-+-+

O|O|O

Así es como se evaluaría la expresión después de la palabra clave return en la línea 42. Primero Python reemplaza las variables bo y le con los valores en cada variable:

```
volver (('X' == 'O' y ''      == 'O' y ''      == 'O') o
(' ' == 'O' y 'X' == 'O' y ''      == 'O') o
('O' == 'O' y 'O' == 'O' y 'O' == 'O') o
('X' == 'O' y ''      == 'O' y 'O' == 'O') o (' '
== 'O' y 'X' == 'O' y 'O' == 'O') o (' '
== 'O' y ''      == 'O' y 'O' == 'O') o
('X' == 'O' y 'X' == 'O' y 'O' == 'O') o (' '
== 'O' y 'X' == 'O' y 'O' == 'O'))
```

A continuación, Python evalúa todas esas comparaciones == dentro del paréntesis a valores booleanos:

```
volver ((Falso y Falso y Falso) o
(Falso y Falso y Falso) o
(Verdadero y Verdadero y Verdadero) o
(Falso y Falso y Verdadero))
```

Luego, el intérprete de Python evalúa todas las expresiones.

dentro de los paréntesis:

```
volver ((Falso) o  
(Falso) o  
(Verdad o  
(Falso) o  
(Falso) o  
(Falso) o  
(Falso) o  
(Falso))
```

Como ahora solo hay un valor dentro de cada paréntesis interno, puede deshacerse de ellos:

```
volver (Falso o  
Falso o  
Verdad o  
Falso o  
Falso o  
Falso o  
Falso o  
Falso)
```

Ahora Python evalúa la expresión conectada por todos esos operadores:

```
volver (Verdadero)
```

Una vez más, elimine los paréntesis y quedará con un valor:

```
volver verdadero
```

Entonces, dados esos valores para bo y le, la expresión se evaluaría como True. Así es como el programa puede saber si uno de los jugadores ha ganado el juego.

DUPLICACIÓN DE LOS DATOS DEL TABLERO

La función `getBoardCopy()` le permite hacer fácilmente una copia de una lista dada de 10 cadenas que representa un tablero de Tic-Tac-Toe en el juego.

```
51. def getBoardCopy(tablero):
52. # Hacer una copia de la lista del tablero y
53. boardCopy = []
54. for i in board:
55.     boardCopy.append(i)
56. return boardCopy
```

Cuando el algoritmo de IA está planificando sus movimientos, a veces necesitará realizar modificaciones en una copia temporal del tablero sin cambiar el tablero real. En esos casos, llamamos a esta función para hacer una copia de la lista del tablero. La nueva lista se crea en la línea 53.

En este momento, la lista almacenada en `boardCopy` es solo una lista vacía. El ciclo `for` iterará sobre el parámetro del tablero , agregando una copia de los valores de cadena en el tablero real al tablero duplicado. Después de que la función `getBoardCopy()` crea una copia del tablero real, devuelve una referencia a este nuevo tablero en `boardCopy`, no al original en `board`.

COMPROBAR SI UN ESPACIO EN EL TABLERO ES GRATIS

Dado un tablero de Tic-Tac-Toe y un posible movimiento, la función simple `isSpaceFree()` devuelve si ese movimiento está disponible o no.
no:

```
58. def isSpaceFree(board, move):
```

59. # Devuelve True si el movimiento pasado es libre en el tablero pasado.

60. regresar tablero[mover] == "

Recuerde que los espacios libres en las listas del tablero están marcados como una cadena de un solo espacio. Si el elemento en el índice del espacio no es igual a '', entonces se toma el espacio.

PERMITIR AL JUGADOR ENTRAR A MOVERSE

La función getPlayerMove() le pide al jugador que ingrese el número del espacio al que desea avanzar:

```
62. def obtenerJugadaJugador(tablero):
63. # Permitir que el jugador ingrese su jugada.
64. mover = ""
65. while move no está en '1 2 3 4 5 6 7 8 9'.split() or not
       isSpaceFree(board, int(move)): print('¿Cuál es tu
66.     siguiente movimiento? (1-9)') move = entrada() 68.
67.     return int(mover)
```

La condición de la línea 65 es verdadera si cualquiera de las expresiones a la izquierda o a la derecha del operador o es verdadera. El bucle se asegura de que la ejecución no continúe hasta que el jugador haya ingresado un número entero entre 1 y 9. También verifica que el espacio ingresado no esté ocupado, dado que el tablero de Tic-Tac-Toe pasó a la función para el tablero . parámetro. Las dos líneas de código dentro del ciclo while simplemente le piden al jugador que ingrese un número del 1 al 9.

La expresión del lado izquierdo verifica si el movimiento del jugador es igual a '1', '2', '3', y así sucesivamente hasta '9' creando una lista con estas cadenas (con el método split()) y comprobación

si el movimiento está en esta lista. En esta expresión, '1 2 3 4 5 6 7 8 9'.split() se evalúa como ['1', '2', '3', '4', '5', '6', '7', '8', '9'], pero el primero es más fácil de escribir.

La expresión del lado derecho verifica si el movimiento que ingresó el jugador es un espacio libre en el tablero llamando a isSpaceFree(). Recuerda que isSpaceFree() devuelve True si el movimiento que pasas está disponible en el tablero. Tenga en cuenta que isSpaceFree() espera un número entero para mover, por lo que la función int() devuelve una forma de movimiento entera.

Los operadores not se agregan a ambos lados para que la condición sea True cuando cualquiera de estos requisitos no se cumple. Esto hace que el bucle le pida al jugador una y otra vez un número hasta que ingrese un movimiento adecuado.

Finalmente, la línea 68 devuelve la forma entera de cualquier movimiento que haya ingresado el jugador. input() devuelve cadenas, por lo que se llama a la función int() para devolver una forma entera de la cadena.

EVALUACIÓN DE CORTOCIRCUITO

Es posible que haya notado que hay un posible problema en la función getPlayerMove() . ¿Qué pasa si el jugador ingresa 'Z' o alguna otra cadena no entera? La expresión move not in '1 2 3 4 5 6 7 8 9'.split() en el lado izquierdo de or devolvería False como se esperaba, y luego Python evaluaría la expresión en el lado derecho del operador or .

Pero llamar a int('Z') haría que Python diera un error, porque la función int() solo puede tomar cadenas de caracteres numéricos como '9' o '0', no cadenas como 'Z'.

Para ver un ejemplo de este tipo de error, ingrese lo siguiente

en el caparazón interactivo:

```
>>> int('42')
42
>>> int('Z')
Rastreo (llamadas recientes más última):
Archivo "<pyshell#3>", línea 1, en <módulo> int("Z")

ValueError: literal no válido para int() con base 10: 'Z'
```

Pero cuando juegas el juego Tic-Tac-Toe e intentas ingresar 'Z' para tu movimiento, este error no ocurre. Esto se debe a que la condición del ciclo while está siendo cortocircuitada.

El cortocircuito significa que una expresión evalúa solo una parte del camino, ya que el resto de la expresión no cambia lo que evalúa la expresión. Aquí hay un programa corto que da un buen ejemplo de cortocircuito. Ingrese lo siguiente en el shell interactivo:

```
>>> def ReturnsTrue(): print('Se
    llamó a ReturnsTrue().') return True

>>> def DevuelveFalso(): print('Se
    llamó a DevuelveFalso().') devuelve Falso

>>> Devuelve Verdadero()
Se llamó a ReturnsTrue().
Verdadero

>>> Devuelve Falso()
Se llamó a ReturnsFalse().
Falso
```

Cuando se llama a ReturnsTrue() , imprime 'Se llamó a ReturnsTrue()' . y luego también muestra el valor de retorno de ReturnsTrue(). Lo mismo ocurre con ReturnsFalse().

Ahora ingrese lo siguiente en el shell interactivo:

```
>>> Devuelve Falso() o Devuelve Verdadero()
```

Se llamó a ReturnsFalse().

Se llamó a ReturnsTrue().

Verdadero

```
>>> Devuelve Verdadero() o Devuelve Falso()
```

Se llamó a ReturnsTrue().

Verdadero

La primera parte tiene sentido: la expresión ReturnsFalse() o ReturnsTrue() llama a ambas funciones, por lo que verá los dos mensajes impresos.

Pero la segunda expresión solo muestra 'Se llamó a ReturnsTrue()', no 'Se llamó a ReturnsFalse()'. Esto se debe a que Python no llamó a ReturnsFalse() en absoluto. Dado que el lado izquierdo del operador o es True, no importa lo que devuelva ReturnsFalse() , por lo que Python no se molesta en llamarlo. La evaluación fue cortocircuitada.

Lo mismo se aplica para el operador y . Ahora ingrese el siguiente en el shell interactivo:

```
>>> Devuelve Verdadero() y Devuelve Verdadero()
```

Se llamó a ReturnsTrue().

Se llamó a ReturnsTrue().

Verdadero

```
>>> Devuelve Falso() y Devuelve Falso()
```

Se llamó a ReturnsFalse().

Falso

Nuevamente, si el lado izquierdo del operador y es Falso, entonces toda la expresión es Falso. No importa si el lado derecho de y es verdadero o falso, por lo que Python no se molesta en evaluarlo. Tanto Falso como Verdadero y Falso y Falso se evalúan como Falso, por lo que Python cortocircuita la evaluación.

Volvamos a las líneas 65 a 68 del programa Tic-Tac-Toe:

```

65. while move no está en '1 2 3 4 5 6 7 8 9'.split() or not
    isSpaceFree(board, int(move)): print('¿Cuál es tu siguiente
66.     movimiento? (1-9)') move = entrada() 68. return
67.     int(mover)

```

Dado que la parte de la condición en el lado izquierdo del operador o (mover no en '1 2 3 4 5 6 7 8 9'.split()) se evalúa como True, el intérprete de Python sabe que la expresión completa se evaluará como True . No importa si la expresión en el lado derecho de o se evalúa como Verdadero o Falso, porque solo un valor en cada lado del operador o necesita ser Verdadero para que toda la expresión sea Verdadera.

Entonces, Python deja de verificar el resto de la expresión y ni siquiera se molesta en evaluar la parte not isSpaceFree(board, int(move)) . Esto significa que las funciones int() e isSpaceFree() nunca se llaman mientras move not in '1 2 3 4 5 6 7 8 9'.split() sea True.

Esto funciona bien para el programa, porque si el lado derecho de la condición es Verdadero, entonces mover no es una cadena de números de un solo dígito. Eso haría que int() nos diera un error. Pero si move not in '1 2 3 4 5 6 7 8 9'.split() se evalúa como True, los cortocircuitos de Python not isSpaceFree(board, int(move)) e int(move) no se llaman.

ELEGIR UN MOVIMIENTO DE UNA LISTA DE MOVIMIENTOS

Ahora echemos un vistazo a la función chooseRandomMoveFromList() , que es útil para el código AI más adelante en el programa:

```

70. def elegirMovidaAleatoriaDeLista(tablero, listadmovimientos):
71. # Devuelve un movimiento válido de la lista de pasadas en el tablero de pasadas.
72. # Devuelve Ninguno si no hay un movimiento válido.

```

```

73. JugadasPosibles = []
74.
for i en listaMovidas:
    75.     si esEspacioLibre(tablero, i):
        76.         jugadasPosibles.append(i)

```

Recuerde que el parámetro del tablero es una lista de cadenas que representa un tablero de Tic-Tac-Toe. El segundo parámetro, MoveList, es una lista de enteros de posibles espacios entre los que elegir. Por ejemplo, si MovesList es [1, 3, 7, 9], eso significa que chooseRandomMoveFromList() debería devolver el número entero para uno de los espacios de las esquinas.

Sin embargo, chooseRandomMoveFromList() primero verifica que el espacio sea válido para continuar. La lista de posiblesMovidas comienza como una lista en blanco. Luego, el ciclo for itera sobre la lista de movimientos. Los movimientos que hacen que isSpaceFree() devuelva True se agregan a possibleMoves con el método append() .

En este punto, la lista de jugadas posibles tiene todas las jugadas que estaban en la lista de jugadas que también son espacios libres. A continuación, el programa comprueba si la lista está vacía:

```

78. if len(movimientosPosibles) != 0:
    79.     return random.choice(movimientosPosibles)
80.     más:
81.     volver Ninguno

```

Si la lista no está vacía, entonces hay al menos una posible movimiento que se puede hacer en el tablero.

Pero esta lista podría estar vacía. Por ejemplo, si la lista de jugadas era [1, 3, 7, 9] pero el tablero representado por el parámetro tablero ya tenía todos los espacios de las esquinas ocupados, la lista de jugadas posibles sería []. En ese caso, len(movimientosPosibles) se evalúa como 0, y el

La función devuelve el valor Ninguno.

EL NINGUNO VALOR

El valor Ninguno representa la falta de un valor. Ninguno es el único valor del tipo de datos `NoneType`. Puede usar el valor Ninguno cuando necesite un valor que signifique "no existe" o "ninguno de los anteriores".

Por ejemplo, supongamos que tiene una variable llamada `quizAnswer` que contiene la respuesta del usuario a alguna pregunta de prueba emergente de verdadero/falso. La variable podría contener Verdadero o Falso para la respuesta del usuario. Pero si el usuario no respondió la pregunta, no querrá establecer `quizAnswer` en `True` o `False`, porque entonces parecería que el usuario respondió la pregunta. En su lugar, puede establecer `quizAnswer` en Ninguno si el usuario omitió la pregunta.

Como nota al margen, Ninguno no se muestra en el caparazón interactivo como otros valores son:

```
>>> 2 + 2
4
>>> 'Este es un valor de cadena.'
'Este es un valor de cadena.'
>>> Ninguno
>>>
```

Los valores de las dos primeras expresiones se imprimen como salida en la siguiente línea, pero Ninguno no tiene valor, por lo que no se imprime.

Funciones que no parecen devolver nada en realidad devuelven el valor Ninguno . Por ejemplo, `print()` devuelve Ninguno:

```
>>> spam = print('¡Hola mundo!')
¡Hola Mundo!
>>> spam == Ninguno
Verdadero
```

Aquí asignamos print('¡Hola mundo!') al spam. La función print() , como todas las funciones, tiene un valor de retorno. Aunque print() imprime una salida, la llamada a la función devuelve Ninguno. IDLE no muestra Ninguno en el shell interactivo, pero puede decir que el correo no deseado está configurado en Ninguno porque el correo no deseado == Ninguno se evalúa como Verdadero.

CREANDO LA IA DE LA COMPUTADORA

La función getComputerMove() contiene el código de la IA:

```

83. def getComputerMove(board, computerLetter): 84. # Dado un
tablero y la letra de la computadora, determina dónde mover
y devolver ese movimiento.

85. if computerLetter == 'X': 86.
playerLetter = 'O'

87. más:

88.     jugadorLetra = 'X'

```

El primer argumento es un tablero de Tic-Tac-Toe para el parámetro del tablero . El segundo argumento es la letra que usa la computadora , ya sea 'X' o 'O' en el parámetro computerLetter . Las primeras líneas simplemente asignan la otra letra a una variable llamada letraJugador. De esta manera, se puede usar el mismo código ya sea que la computadora sea X u O.

Recuerda cómo funciona el algoritmo de IA Tic-Tac-Toe:

1. Vea si hay un movimiento que la computadora pueda hacer para ganar el juego. Si lo hay, haz ese movimiento. De lo contrario, vaya al paso 2.
2. Vea si hay un movimiento que el jugador pueda hacer que haga que la computadora pierda el juego. Si lo hay, la computadora debe moverse allí para bloquear al jugador. De lo contrario, vaya al paso 3.
3. Compruebe si alguna de las esquinas (espacios 1, 3, 7 o 9) está libre. Si no hay esquina el espacio es libre, vaya al paso 4.
4. Comprobar si el centro está libre. Si es así, muévete allí. Si no es así, vaya al paso 5.

5. Muévete por cualquiera de los lados (espacios 2, 4, 6 u 8). No hay más pasos, porque los espacios laterales son los únicos espacios que quedan si la ejecución ha llegado a este paso.

La función devolverá un número entero del 1 al 9 que representa el movimiento de la computadora. Veamos cómo se implementa cada uno de estos pasos en el código.

Comprobando si la computadora puede ganar en uno Moverse

Antes que nada, si la computadora puede ganar en el próximo movimiento, debe hacer ese movimiento ganador inmediatamente.

```
90. # Aquí está el algoritmo para nuestra IA de Tic-Tac-Toe:  
91. # Primero, verifica si podemos ganar en el siguiente  
movimiento. 92. for i in range(1, 10): 93. boardCopy =  
getBoardCopy(board) if isSpaceFree(boardCopy, i):  
94.  
95.     hacerMovimiento(copiatablero, letraComputadora,  
96.     i) si esGanador(copiatablero, letraComputadora):  
97.         vuelvo yo
```

El ciclo for que comienza en la línea 92 itera sobre cada movimiento posible del 1 al 9. El código dentro del ciclo simula lo que sucedería si la computadora hiciera ese movimiento.

La primera línea del bucle (línea 93) hace una copia de la lista del tablero . Esto es para que el movimiento simulado dentro del bucle no modifique el tablero real de Tic-Tac-Toe almacenado en la variable del tablero . getBoardCopy () devuelve una lista de tableros idéntica pero separada valor.

La línea 94 comprueba si el espacio está libre y, en caso afirmativo, simula hacer el movimiento en la copia del tablero. Si este movimiento da como resultado que la computadora gane, la función devuelve que

entero del movimiento.

Si ninguno de los espacios resulta ganador, el bucle finaliza y la ejecución del programa continúa hasta la línea 100.

Comprobar si el jugador puede ganar en uno
Moverse

A continuación, el código simulará al jugador humano moviéndose en cada uno de los espacios:

```
99. # Comprueba si el jugador podría ganar en su próximo movimiento y bloquéalo.  
100. for i in range(1, 10): copiaTablero = obtenerCopiaTablero(tablero) if  
101.     hayEspacioLibre(copiatablero, i): hacerJugada(copiatablero, letraJugador,  
102.     i) if esGanador(copiaTablero, letraJugador):  
103.  
104.  
105.         vuelvo yo
```

El código es similar al ciclo de la línea 92, excepto que la letra del jugador se coloca en la copia del tablero. Si la función isWinner() muestra que el jugador ganaría con un movimiento, entonces la computadora devolverá ese mismo movimiento para evitar que esto suceda.

Si el jugador humano no puede ganar en un movimiento más, el bucle finaliza y la ejecución continúa hasta la línea 108.

Comprobación de los espacios de las esquinas, el centro y los lados (en ese orden)

Si la computadora no puede hacer un movimiento ganador y no necesita bloquear el movimiento del jugador, se moverá a una esquina, centro o espacio lateral, según los espacios disponibles.

La computadora primero intenta moverse a una de las esquinas espacios:

```
107. # Intenta tomar una de las esquinas, si están libres. 108.  
mover = elegirJugadaAleatoriaDeLista(tablero, [1, 3, 7, 9]) 109. si mover !=  
Ninguno:  
110.     movimiento de regreso
```

La llamada a la función `chooseRandomMoveFromList()` con la lista [1, 3, 7, 9] asegura que la función devuelva el número entero para uno de los espacios de las esquinas: 1, 3, 7 o 9.

Si todas las espacios de las esquinas son tomado, la `elegirMovimientoAleatorioDeLista()` devuelve Ninguno, y el la ejecución pasa a la línea 113:

```
112. # Intenta tomar el centro, si está libre. 113.  
si hay espacio libre (tablero, 5):  
114.     volver 5
```

Si ninguna de las esquinas está disponible, la línea 114 se mueve en el espacio central si está libre. Si el espacio central no está libre, la ejecución pasa a la línea 117:

```
116. # Muévete por uno de los lados.  
117. return elegirMovidaAleatoriaDeLista(tablero, [2, 4, 6, 8])
```

Este código también realiza una llamada a `chooseRandomMoveFromList()`, excepto que le pasa una lista de los espacios laterales: [2, 4, 6, 8]. Esta función no devolverá Ninguno porque los espacios laterales son los únicos espacios que pueden quedar. Esto finaliza la función `getComputerMove()` y el algoritmo de IA.

Comprobando si el tablero está lleno

La última función es `tableroLleno()`:

```
119. def isBoardFull(board): 120.  
# Devuelve True si se han ocupado todos los espacios del tablero. De lo contrario,
```

```
falso retorno.

121. for i in range(1, 10): 122. if
isSpaceFree(board, i):
123.         falso retorno
124. volver Verdadero
```

Esta función devuelve True si la lista de 10 cadenas en el argumento del tablero que se le pasó tiene una 'X' o una 'O' en cada índice (excepto en el índice 0, que se ignora). El bucle for nos permite comprobar los índices del 1 al 9 en la lista de tableros . Tan pronto como encuentre un espacio libre en el tablero (es decir, cuando isSpaceFree(board, i) devuelva True), la función isBoardFull() devolverá False.

Si la ejecución logra pasar por cada iteración del ciclo, entonces ninguno de los espacios está libre. La línea 124 entonces ejecutar devolver True.

EL BUCLE DEL JUEGO

La línea 127 es la primera línea que no está dentro de una función, por lo que es la primera línea de código que se ejecuta cuando ejecuta este programa.

```
127. print('¡Bienvenido a Tic-Tac-Toe!')
```

Esta línea saluda al jugador antes de que comience el juego. El programa entonces entra en un bucle while en la línea 129:

```
129. mientras sea cierto:
130. # Restablecer el tablero.
131. elTablero = [' '] * 10
```

El ciclo while sigue repitiéndose hasta que la ejecución encuentra una declaración de ruptura . La línea 131 configura el tablero principal de Tic-Tac-Toe en una variable llamada theBoard. El tablero comienza vacío, lo cual

representar con una lista de 10 cadenas de un solo espacio. En lugar de escribir esta lista completa, la línea 131 usa la replicación de listas. Es más corto escribir '[' * 10 que [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '].

Elegir la marca del jugador y quién va primero

A continuación, la función `inputPlayerLetter()` le permite al jugador ingresar si quiere ser la X o la O:

```
132. letraJugador, letraEquipo = entradaLetraJugador()
```

La función devuelve una lista de dos cadenas, ya sea ['X', 'O'] o ['O', 'X']. Usamos la asignación múltiple para establecer `playerLetter` en el primer elemento de la lista devuelta y `computerLetter` en el segundo.

A partir de ahí, la función `quién va primero()` decide aleatoriamente quién va primero, devolviendo la cadena 'jugador' o la cadena 'computadora', y luego la línea 134 le dice al jugador quién irá primero:

```
133. turn = quien va primero() 134.  
print('El ' 135. gamelsPlaying = 'Irá primero.')  
True
```

La variable `gamelsPlaying` realiza un seguimiento de si el juego todavía se está jugando o si alguien ha ganado o empelado.

Ejecutar el turno del jugador

El bucle de la línea 137 seguirá yendo y viniendo entre el código para el turno del jugador y el turno de la computadora, siempre que `gamelsPlaying` esté configurado en True:

```
137. while gamelsPlaying: if turno  
138.     == 'jugador': # Turno  
139.         del jugador  
140.         dibujarTablero(elTablero)  
141.         mover = obtenerMovidaJugador(elTablero)
```

142. hacerJugada(elTablero, letraJugador, jugada)

La variable de turno se estableció originalmente en 'jugador' o 'computadora' mediante la llamada whoGoesFirst() en la línea 133. Si turno es igual a 'computadora', entonces la condición de la línea 138 es Falso y la ejecución salta a la línea 156.

Pero si la línea 138 se evalúa como Verdadero, la línea 140 llama a dibujarTablero() y pasa la variable elTablero para imprimir el tablero de Tic-Tac-Toe en la pantalla. Luego , getPlayerMove() le permite al jugador ingresar su movimiento (y también se asegura de que sea un movimiento válido). La función hacerJugada() agrega la X o la O del jugador al Tablero.

Ahora que el jugador ha hecho su jugada, el programa debería comprobar si ha ganado la partida:

144. if esGanador(elTablero, letraJugador):
 145. dibujarTablero(elTablero) print('¡Hurra!
 146. ¡Has ganado el juego!') juegoEstáJugando =
 147. False

Si la función isWinner() devuelve True, el código del bloque if muestra el tablero ganador e imprime un mensaje que le dice al jugador que ha ganado. La variable gameIsPlaying también se establece en Falso para que la ejecución no continúe hasta el turno de la computadora.

Si el jugador no ganó con su último movimiento, tal vez su movimiento llenó todo el tablero y empató el juego. El programa verifica esa condición a continuación con una declaración else :

148. más:
 149. if elTableroLleno(elTablero):
 150. dibujarTablero(elTablero)
 151. print('¡El juego es un empate!')
 152. descanso

En este bloque else , la función isBoardFull() devuelve True si no hay más movimientos que hacer. En ese caso, el bloque if comenzando en la línea 149 muestra el tablero empatado y le dice al jugador que ha ocurrido un empate. La ejecución estalla entonces fuera del tiempo . bucle y salta a la línea 173.

Si el jugador no ha ganado o empatado el juego, el programa ingresa otra sentencia else :

```
153.     más:
154.     turno = 'computadora'
```

La línea 154 establece la variable turno en 'computadora' para que el programa ejecute el código para el turno de la computadora en la siguiente iteración.

Ejecutar el turno de la computadora Si la variable de turno no era 'jugador' para la condición de la línea 138, entonces debe ser el turno de la computadora. El código de este bloque else es similar al código del turno del jugador:

```
156.     más:
157.     # Turno de la
158.     computadora mover = obtenerJugadaComputadora(elTablero,
159.     letraComputadora) hacerJugada(elTablero, letraComputadora, jugada)
160.
161.     if esGanador(elTablero, computadoraLetra):
162.         dibujarTablero(elTablero) print('¡La computadora
163.         te ganó! Tú pierdes.') juegoEstáJugando = False
164.
165.     más:
166.     if elTableroLleno(elTablero):
167.         dibujarTablero(elTablero)
168.         print('¡El juego es un empate!')
169.         descanso
170.     más:
```

171. turno = 'jugador'

Las líneas 157 a 171 son casi idénticas al código del turno del jugador en las líneas 139 a 154. La única diferencia es que este código usa la letra de la computadora y llama a getComputerMove().

Si el juego no se gana o se empata, la línea 171 establece el turno del jugador. No hay más líneas de código dentro del bucle while , por lo que la ejecución vuelve a la instrucción while en línea.

137.

Pedirle al jugador que juegue de nuevo

Finalmente, el programa le pregunta al jugador si quiere jugar otro juego:

173. print('¿Quieres volver a jugar? (sí o no)') 174. if not
input().lower().startswith('y'): 175. break

Las líneas 173 a 175 se ejecutan inmediatamente después del bloque while iniciado por la declaración while en la línea 137. gameIsPlaying se establece en False cuando el juego ha terminado, por lo que en este punto el juego le pregunta al jugador si quiere volver a jugar.

La expresión not input().lower().startswith('y') será verdadera si el jugador ingresa cualquier cosa que no comience con una 'y'. En ese caso, se ejecuta la instrucción break . Eso rompe la ejecución de el ciclo while que se inició en la línea 129. Pero como no hay más líneas de código después de ese bloque while , el programa termina y el juego termina.

RESUMEN

Crear un programa con IA se reduce a cuidadosamente

considerando todas las situaciones posibles que la IA puede encontrar y cómo debe responder en cada una de esas situaciones. La IA de Tic Tac-Toe es simple porque no son posibles tantos movimientos en Tic-Tac-Toe como en un juego como el ajedrez o las damas.

La IA de nuestra computadora verifica cualquier posible movimiento ganador. De lo contrario, comprueba si debe bloquear el movimiento del jugador. Luego, la IA simplemente elige cualquier espacio de esquina disponible, luego el espacio central y luego los espacios laterales. Este es un algoritmo simple para que la computadora lo siga.

La clave para implementar nuestra IA es hacer copias de los datos del tablero y simular movimientos en la copia. De esa manera, el código de IA puede ver si un movimiento resulta en una ganancia o una pérdida. Entonces la AI puede hacer ese movimiento en el tablero real. Este tipo de simulación es eficaz para predecir lo que es o no es un buen moverse.

11

LA DEDUCCIÓN DE LOS BAGELS JUEGO



Bagels es un juego de deducción en el que el jugador trata de adivinar un número aleatorio de tres dígitos (sin dígitos repetidos) generado por la computadora. Después de cada intento, la computadora le da al jugador tres tipos de pistas:

Bagels Ninguno de los tres dígitos adivinados está en el número secreto.

Pico Uno de los dígitos está en el número secreto, pero la conjetura tiene el dígito en el lugar equivocado.

Fermi La conjetura tiene un dígito correcto en el lugar correcto.

La computadora puede dar múltiples pistas, las cuales están ordenadas en orden alfabético. Si el número secreto es 456 y la suposición del jugador es 546, las pistas serían "fermi pico pico". El fermi es del 6 y el pico pico es del 4 y 5.

En este capítulo, aprenderá algunos métodos y funciones nuevos que vienen con Python. También aprenderá sobre

operadores de asignación aumentada e interpolación de cadenas.

Si bien no le permiten hacer nada que no pudiera hacer antes, son buenos atajos para facilitar la codificación.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- La función random.shuffle()
- Operadores de asignación aumentada, `+=`, `-=`, `*=`, `/=`
- El método de lista `sort()`
- El método de cadena `join()`
- Interpolación de cadenas
- El especificador de conversión `%s`
- Bucles anidados

FUNCTION DE MUESTRA DE BAGELS

Esto es lo que ve el usuario cuando ejecuta el programa Bagels.

El texto que ingresa el jugador se muestra en negrita.

Estoy pensando en un número de 3 dígitos. Intenta adivinar qué es.

Las pistas que doy son...

Cuando digo: Eso significa:

Bagels Ninguno de los dígitos es correcto.

Pico Un dígito es correcto pero en la posición incorrecta.

Fermi Un dígito es correcto y está en la posición correcta.

He pensado en un número. Tienes 10 intentos para conseguirlo.

Adivina #1:

123

Fermi

Adivina #2:

453

Pico

Adivina #3:

425

Fermi

Adivina #4:

326

Bagels

Adivina #5:

489

Bagels

Adivina #6:

075

Para para

Adivina #7:

015

Parada Pico

Adivina #8:

175

¡Lo entendiste!

¿Quieres jugar de nuevo? (sí o no)

no

CÓDIGO FUENTE PARA BAGELS

En un archivo nuevo, ingrese el siguiente código fuente y guárdelo como bagels.py. Luego ejecuta el juego presionando F5. Si obtiene errores, compare el código que escribió con el código del libro con la herramienta de comparación en línea en <https://www.nostarch.com/inventwithpython#diff>.



bagels.py

```
1. importar al azar
2.
3. NUM_DIGITOS = 3 4.
MAX_DIGITOS = 10
5.
6. def getSecretNum(): 7.
# Devuelve una cadena de dígitos aleatorios únicos que tiene NUM_DIGITS de longitud.
8. numeros = lista(rango(10)) 9. random.shuffle(numeros) 10. numerosecreto =
    "
11. para i en el rango (NUM_DIGITS):
12     secretNum += str(numeros[i])
13. devolver numerosecreto
14
15. def getClues(guess, secretNum): 16.
# Devuelve una cadena con las pistas de Pico, Fermi y Bagels para el usuario. 17. if
adivinar == secretNum: return '¡Lo tienes!'
18
19
20. pistas = [] 21.
for i in range(len(adivinar)): 22. if
adivinar[i] == numerosecreto[i]: 23.
    pistas.append('Fermi') elif
24.     adivinar[i] in numerosecreto:
25.         pistas.append('Pico')
26. if len(pistas) == 0: devuelve
27.     'Bagels'
28
29. pistas.ordenar()
30. return ''.join(pistas)
31
32. def isOnlyDigits(num): 33.
# Devuelve True si num es una cadena de solo dígitos. De lo contrario, devuelve
    Falso.
34. si num == ":":
35.     falso retorno
36.
37. para i en num:
38.     si no estoy en '0 1 2 3 4 5 6 7 8 9'.split():
39.         devuelve Falso
```

```

40
41. volver Verdadero
42.
43.
44. print('Estoy pensando en un número de %s dígitos. Intenta adivinar cuál es.' %
   (NUM_DIGITS)) 45. print('Las pistas que doy son...') 46. print('Cuando Digo: Eso
significa:') 47. print(' Bagels Ninguno de los dígitos es correcto.') 48. print(' Pico 49. print('
Fermi

      Un dígito es correcto pero está en la posición incorrecta.')
      Un dígito es correcto y está en la posición correcta.')

50
51. mientras sea cierto:
52. secretNum = getSecretNum() 53.
print('He pensado en un número. Tienes %s intentos para obtenerlo' % (MAX_GUESS))

54.
55. conjeturas realizadas = 1
56. while conjeturas realizadas <= MAX_GUESS:
   "
57.     adivinar
58.     = while len(adivina) != NUM_DIGITS o no son solo dígitos(adivina):
59.         print('Adivina #%s: ' % (adivina)) adivinar = entrada()
60
61.
62.         print(obtenerPistas(adivina, númerosecreto))
63.         adivinanzasTomadas += 1
64.

   if adivinar == secretNum:
66.     romper
67.     if conjeturasTomadas >
68.         MAX_CONJETURAS: print('Te quedaste sin conjeturas. La
   respuesta fue %s.' % (secretNum))
69.

70. print('¿Quieres volver a jugar? (sí o no)') 71. if not
input().lower().startswith('y'): 72. break

```

DIAGRAMA DE FLUJO PARA BAGELS

El diagrama de flujo de la figura 11-1 describe lo que sucede en este juego y el orden en que puede suceder cada paso.

El diagrama de flujo para Bagels es bastante simple. La computadora genera un número secreto, el jugador trata de adivinar ese número y la computadora le da pistas al jugador basadas en su suposición. Esto sucede una y otra vez hasta que el jugador gana o pierde. Una vez que finaliza el juego, ya sea que el jugador haya ganado o no, la computadora le pregunta al jugador si quiere volver a jugar.

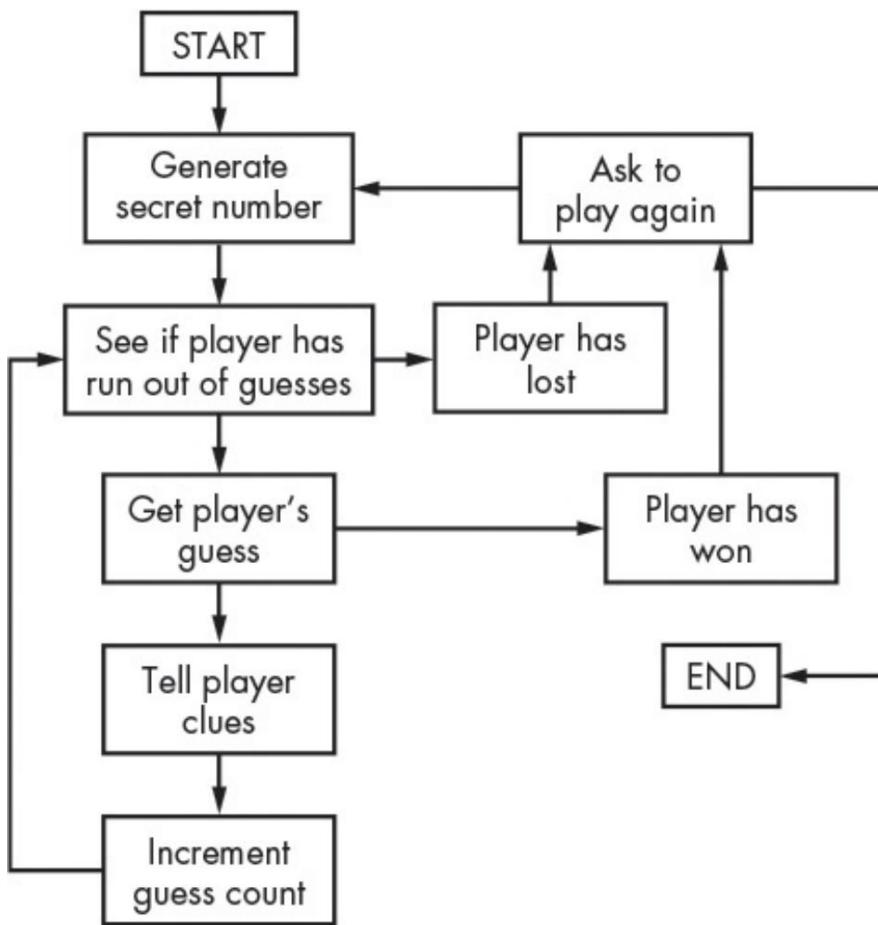


Figura 11-1: Diagrama de flujo para el juego Bagels

IMPORTACIÓN ALEATORIA Y DEFINIENDO GETSECRETNUM()

Al comienzo del programa, importaremos el módulo aleatorio y configuraremos algunas variables globales. Luego definiremos una función llamada `getSecretNum()`.

-
1. importar al azar
 - 2.
 3. NUM_DIGITOS = 3
 4. MÁX._ADIVINIDAD = 10
 - 5.
 6. def getSecretNum(): 7.
- # Devuelve una cadena de dígitos aleatorios únicos que tiene NUM_DIGITS de longitud.
-

En lugar de usar el número entero 3 para el número de dígitos en la respuesta, usamos la variable constante NUM_DIGITS. Lo mismo va por el número de conjeturas que obtiene el jugador; usamos la variable constante MAX_GUESS en lugar del número entero 10. Ahora será fácil cambiar el número de conjeturas o dígitos de números secretos. Simplemente cambie los valores en la línea 3 o 4, y el resto del programa seguirá funcionando sin más cambios.

La función `getSecretNum()` genera un número secreto que contiene solo dígitos únicos. El juego Bagels es mucho más divertido si no tienes dígitos duplicados en el número secreto, como '244' o '333'. Usaremos algunas funciones nuevas de Python para que esto suceda en `getSecretNum()`.

BARAJAR UN CONJUNTO ÚNICO DE DIGITOS

Las dos primeras líneas de `getSecretNum()` mezclan un conjunto de no repetidos números:

-
8. numeros = lista(rango(10)) 9.
 - random.shuffle(numeros)
-

La lista de la línea 8 (rango (10)) se evalúa como [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], por lo que la variable de números contiene una lista de los 10 dígitos.

Cambiar el orden de los elementos de la lista con la función random.shuffle(). La función random.shuffle() cambia aleatoriamente el orden de los elementos de una lista (en este caso, la lista de dígitos). Esta función no devuelve un valor, sino que modifica la lista que le pasa en su lugar. Esto es similar a la forma en que la función makeMove() en el juego Tic-Tac-Toe del Capítulo 10 modificó la lista en la que se pasó, en lugar de devolver una nueva lista con el cambio. Esta es la razón por la que no escribes código como números = random.shuffle(numbers).

Intente experimentar con la función de reproducción aleatoria() ingresando el siguiente código en el shell interactivo:

```
>>> importar aleatorio >>>
spam = lista(rango(10)) >>>
imprimir(spam) [0, 1, 2, 3, 4, 5, 6, 7,
8, 9] >>> aleatorio. barajar(correos
no deseado) >>> imprimir(correos no
deseado) [3, 0, 5, 9, 6, 8, 2, 4, 1, 7]
>>> aleatorio.reproducir(correos no
deseado) >>> imprimir(correos no
deseado) [9, 8, 3, 5, 4, 7, 1, 2, 0, 6]
```

Cada vez que se llama a random.shuffle() en spam, los elementos en el **lista de spam** se barajan. Verás cómo usamos el shuffle() función para hacer un número secreto a continuación.

Obtener el número secreto de Shuffled dígitos

El número secreto será una cadena de los primeros NUM_DIGITS

dígitos de la lista barajada de enteros:

```
10. númerosecreto = ""  
11. for i in range(NUM_DIGITS):  
12     secretNum += str(numbers[i])  
13. return secretNum
```

La variable secretNum comienza como una cadena en blanco. El ciclo for en la línea 11 itera NUM_DIGITS número de veces. En cada iteración a través del bucle, el entero en el índice i se extrae de la lista mezclada, se convierte en una cadena y se concatena al final de secretNum.

Por ejemplo, si los números se refieren a la lista [9, 8, 3, 5, 4, 7, 1, 2, 0, 6], entonces en la primera iteración, se pasarán los números [0] (es decir, 9) a str(); esto devuelve '9', que se concatena al final de secretNum. En la segunda iteración sucede lo mismo con los números[1] (es decir, 8), y en la tercera iteración sucede lo mismo con los números[2] (es decir, 3). El valor final de secretNum que se devuelve es '983'.

Observe que secretNum en esta función contiene una cadena, no un número entero. Esto puede parecer extraño, pero recuerda que no puedes concatenar números enteros. La expresión $9 + 8 + 3$ se evalúa como 20, pero lo que quieres es '9' + '8' + '3', que se evalúa como '983'.

ASIGNACIÓN AUMENTADA OPERADORES

El operador `+=` en la línea 12 es uno de los operadores de asignación aumentada. Normalmente, si desea agregar o concatenar un valor a una variable, usa un código que se ve así:

```
>>> correo no deseado = 42
>>> correo no deseado = correo no deseado +
10 >>> correo no deseado 52
```

```
>>> huevos = 'Hola' >>>
huevos = huevos + '!mundo!'
>>> huevos
'¡Hola Mundo!'
```

Los operadores de asignación aumentada son atajos que lo liberan de tener que volver a escribir el nombre de la variable. El siguiente código hace lo mismo que el código anterior:

```
>>> spam = 42 >>>
```

```
spam += 10 # Lo mismo que spam = spam + 10 >>> spam
```

```
52
```

```
>>> huevos = 'Hola' >>>
huevos += '!mundo!' # Lo mismo que huevos = huevos + '!mundo!'
>>> huevos
'¡Hola mundo!'
```

También hay otros operadores de asignación aumentada.

Ingrese lo siguiente en el shell interactivo:

```
>>> correo basura = 42
>>> correo basura -= 2
>>> correo basura 40
```

La declaración `spam -= 2` es la misma que la declaración `spam = spam - 2`, por lo que la expresión se evalúa como `spam = 42 - 2` y luego como

`correo no deseado = 40.`

Existen operadores de asignación aumentada para multiplicacion y division tambien:

```
>>> spam *= 3
```

```
>>> correo no deseado
120
>>> correo no deseado /= 10
>>> correo no deseado 12.0
```

La declaración `spam *= 3` es lo mismo que `spam = spam * 3`. Entonces, dado que el `spam` se estableció en 40 antes, la expresión completa sería `spam = 40 * 3`, que se evalúa como 120. La expresión `spam /= 10` es lo mismo que `spam = spam / 10`, y `spam = 120 / 10` se evalúa como 12.0 .

Tenga en cuenta que el `spam` se convierte en un número de punto flotante después de que se dividido.

CÁLCULO DE LAS PISTAS PARA DAR

La función `getClues()` devolverá una cadena con pistas de fermi, pico y bagels dependiendo de los parámetros de conjectura y `secretNum` .

```
15. def getClues(guess, secretNum):
16. # Devuelve una cadena con las pistas de Pico, Fermi y Bagels para el
17. if adivinar == secretNum: 18. return '¡Lo tienes!'
```

```
19
20. pistas = []
21. for i in range(len(adivinar)): if
22.     adivinar[i] == númerosecreto[i]:
23.         pistas.append('Fermi') elif
24.         adivinar[i] in númerosecreto:
25.             pistas.agregar ('Pico')
```

El paso más obvio es comprobar si la suposición es la misma que el número secreto, lo que hacemos en la línea 17. En ese caso, la línea 18 devuelve "¡Lo tienes!".

Si la conjectura no es la misma que el número secreto, el

El programa debe averiguar qué pistas darle al jugador. La lista en las pistas comenzará vacía y se agregarán las cadenas 'Fermi' y 'Pico' según sea necesario.

El programa hace esto recorriendo cada posible índice en conjectura y número secreto. Las cadenas en ambas variables tendrán la misma longitud, por lo que la línea 21 podría haber usado len(adivinar) o len(número secreto) y funcionar de la misma manera. A medida que el valor de i cambia de 0 a 1 a 2, y así sucesivamente, la línea 22 verifica si el primer, segundo, tercero, etc. carácter de conjectura es el mismo que el carácter en el índice correspondiente de secretNum. Si es así, la línea 23 agrega la cadena 'Fermi' a las pistas.

De lo contrario, la línea 24 verifica si el número en la i-ésima posición en la conjectura existe en algún lugar de secretNum. Si es así, ya sabes que el número está en algún lugar del número secreto pero no en la misma posición. En ese caso, la línea 25 luego agrega 'Pico' a las pistas.

Si la lista de pistas está vacía después del ciclo, entonces sabrá que no hay dígitos correctos en la conjectura:

```
26. if len(pistas) == 0:  
27     devuelve 'Bagels'
```

En este caso, la línea 27 devuelve la cadena 'Bagels' como el único clave.

EL MÉTODO DE LISTA SORT()

Las listas tienen un método llamado sort() que organiza los elementos de la lista en orden alfabético o numérico. Cuando el método sort() es llamado, no devuelve una lista ordenada sino que ordena la lista en su lugar. Así es como funciona el método shuffle() .

Nunca querría usar `return spam.sort()` porque eso devolvería el valor Ninguno.

En su lugar, desea una línea separada, `spam.sort()`, y luego la línea devuelve `spam`.

Ingrese lo siguiente en el shell interactivo:

```
>>> spam = ['gato', 'perro', 'murciélagos', 'oso  
hormiguero'] >>> spam.sort()  
>>> spam  
['oso hormiguero', 'murciélagos',  
'gato', 'perro'] >>> spam = [9, 8, 3, 5.5, 5, 7, 1, 2.1,  
0, 6] >>> spam.sort() >>> spam [0, 1, 2.1, 3, 5, 5.5,  
6, 7, 8, 9]
```

Cuando ordenamos una lista de cadenas, las cadenas se devuelven en orden alfabético, pero cuando ordenamos una lista de números, los números se devuelven en orden numérico.

En la línea 29, usamos `sort()` en las pistas:

```
29. pistas.ordenar()
```

La razón por la que quieras ordenar alfabéticamente la lista de pistas es para deshacerte de la información adicional que ayudaría al jugador a adivinar el número secreto más fácilmente. Si las pistas fueran ['Pico', 'Fermi', 'Pico'], eso le diría al jugador que el dígito central de la conjectura está en la posición correcta. Dado que las otras dos pistas son Pico, el jugador sabrá que todo lo que tiene que hacer para obtener el número secreto es intercambiar el primer y el tercer dígito.

Si las pistas siempre se ordenan alfabéticamente, el jugador no puede estar seguro de a qué número se refiere la pista de Fermi . Esto hace que el juego sea más difícil y divertido de jugar.

EL MÉTODO DE CADENA JOIN()

El método de cadena `join()` devuelve una lista de cadenas como una sola cadena unida.

```
30. return ''.join(pistas)
```

La cadena a la que se llama el método (en la línea 30, este es un espacio simple, ej: ~~apartir de la siguiente línea en el shell interactivo~~ para tener un

```
>>> ''.join(['Mi', 'nombre', 'es', 'Zophie'])
'Mi nombre es Zophie'
>>> ', '.join(['Life', 'the Universe', 'and Everything'])
'La vida, el universo y todo'
```

Entonces, la cadena que se devuelve en la línea 30 es cada cadena en pista combinada con un solo espacio entre cada cadena. El método de cadena `join()` es algo así como lo opuesto al método de cadena `split()`. Mientras que `split()` devuelve una lista de una cadena dividida, `join()` devuelve una cadena de una lista combinada.

COMPROBAR SI UNA CADENA SOLO TIENE NUMEROS

La función `isOnlyDigits()` ayuda a determinar si el jugador ingresó una suposición válida:

```
32. def isOnlyDigits(num): 33.
# Devuelve True si num es una cadena de solo dígitos. De lo contrario, devuelve
    Falso.
34. si num == "":
35.     falso retorno
```

La línea 34 primero verifica si `num` es la cadena en blanco y, si entonces, devuelve Falso.

Luego, el bucle `for` itera sobre cada carácter de la cadena.

en uno:

-
- 37. para i en num:
 - 38. si no estoy en '0 1 2 3 4 5 6 7 8 9'.split():
 - 39. devuelve Falso
 - 40
 - 41. volver Verdadero
-

El valor de `i` tendrá un solo carácter en cada iteración. Dentro del bloque `for`, el código comprueba si `i` existe en la lista devuelta por `'0 1 2 3 4 5 6 7 8 9'.split()`. (El valor de retorno de `split()` es equivalente a `['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']`.) Si `i` no existe en esa lista, sabe que hay un carácter que no es un dígito en `num`. En ese caso, la línea 39 devuelve Falso.

Pero si la ejecución continúa más allá del ciclo `for`, entonces sabrá que cada carácter en `num` es un dígito. En ese caso, la línea 41 devuelve Verdadero.

COMENZANDO EL JUEGO

Después de todas las definiciones de funciones, la línea 44 es el inicio real del programa:

-
- 44. `print('Estoy pensando en un número de %s dígitos. Intenta adivinar cuál es.' % (NUM_DIGITS))`
 - 45. `print('Las pistas que doy son...')`
 - 46. `print('Cuando Digo: Eso significa:')`
 - 47. `print(' Bagels Ninguno de los dígitos es correcto.')`
 - 48. `print(' Pico Un dígito es correcto pero está en la posición incorrecta.')`
 - 49. `print(' Fermi Un dígito es correcto y en la posición correcta.')`
-

Las llamadas a la función `print()` le dicen al jugador las reglas del juego y lo que significan las pistas pico, fermi y bagels. Línea

La llamada `print()` de 44 tiene `% (NUM_DIGITS)` agregado al final y `%s` dentro de la cuerda. Esta es una técnica conocida como interpolación de cadenas.

INTERPOLACIÓN DE CADENA

La interpolación de cadenas, también conocida como formato de cadenas, es un atajo de codificación. Normalmente, si desea usar los valores de cadena dentro de variables en otra cadena, debe usar el operador de concatenación `+`:

```
>>> nombre = 'Alicia'
>>> evento = 'fiesta'
>>> ubicación = 'la
piscina' >>> día =
'sábado' >>> hora =
'6:00pm' >>> print('Hola' + nombre + '. Will vas a la ' + ubicación + ' para la ' + evento + ' el ' + día + ' a las ' + hora + '?')
```

Como puede ver, puede llevar mucho tiempo escribir una línea que concatene varias cadenas. En su lugar, puede utilizar la interpolación de cadenas, que le permite colocar marcadores de posición como `%s` en la cadena. Estos marcadores de posición se denominan especificadores de conversión. Una vez que haya ingresado los especificadores de conversión, puede colocar todos los nombres de las variables al final de la cadena. Cada `%s` se reemplaza con una variable al final de la línea, en el orden en que ingresó la variable. Por ejemplo, el siguiente código hace lo mismo que el código anterior:

```
>>> nombre = 'Alicia'
>>> evento = 'fiesta'
>>> ubicación = 'la
piscina' >>> día = 'sábado'
```

```
>>> time = '6:00pm' >>>
print('Hola, %s. ¿Irás al %s a las %s este %s?' % (nombre, evento, ubicación, día, hora ))
```

Hola, Alicia. ¿Irás a la fiesta en la piscina este sábado a las 6:00 pm?

Observe que el primer nombre de variable se usa para el primer %s, la segunda variable para el segundo %s, y así sucesivamente. Debe tener la misma cantidad de especificadores de conversión %s que tiene variables

Otro beneficio de usar la interpolación de cadenas en lugar de la concatenación de cadenas es que la interpolación funciona con cualquier tipo de datos, no solo con cadenas. Todos los valores se convierten automáticamente al tipo de datos de cadena. Si concatenaste un número entero a una cadena, obtendrías este error:

```
>>> correo basura =
42 >>> print('Correo basura ==' + correo no deseado)
Rastreo (llamadas recientes más última):
Archivo "<stdin>", línea 1, en <módulo>
TypeError: no se puede convertir el objeto 'int' en str implícitamente
```

La concatenación de cadenas solo puede combinar dos cadenas, pero el spam es un número entero. Debería recordar poner str (spam) en lugar de spam.

Ahora ingrese esto en el shell interactivo:

```
>>> spam = 42 >>>
```

```
print('Spam es %s' % (spam))
```

```
el correo no deseado es 42
```

Con la interpolación de cadenas, esta conversión a cadenas se realiza por usted.

EL BUCLE DEL JUEGO

La línea 51 es un ciclo while infinito que tiene una condición de Verdadero, por lo que se repetirá para siempre hasta que se ejecute una declaración de interrupción :

```
51. mientras sea cierto:  
52. secretNum = getSecretNum() 53. print('He  
pensado en un número. Tienes %s intentos para obtenerlo' % (MAX_GUESS))  
  
54.  
55. conjeturas realizadas = 1  
56. while conjeturas realizadas <= MAX_GUESS:
```

Dentro del ciclo infinito, obtienes un número secreto de la función getSecretNum() . Este número secreto se asigna a secretNum. Recuerde, el valor en secretNum es una cadena, no un entero.

La línea 53 le dice al jugador cuántos dígitos hay en el número secreto usando la interpolación de cadenas en lugar de la concatenación de cadenas. La línea 55 establece la variable conjeturas realizadas en 1 para marcar esto es como la primera conjetura. Luego, la línea 56 tiene un nuevo bucle while que se repite mientras el jugador tenga adivinanzas . En el código, esto es cuando las conjeturas tomadas son menores o iguales a MAX_GUESS.

Observe que el bucle while de la línea 56 está dentro de otro bucle while que empezó en la línea 51. Estos bucles interiores se denominan bucles anidados. Cualquier declaración de interrupción o continuación , como la declaración de interrupción en la línea 66, solo se interrumpirá o continuará fuera del bucle más interno, no de los bucles externos.

Obtener la suposición del jugador La variable de suposición contiene la suposición del jugador devuelta por input(). El código sigue en bucle y le pide al jugador que adivine hasta que ingrese una suposición válida:

```
57.     adivinar = "
```

```

58.     while len(adivinar) != NUM_DIGITS o no son SoloDígitos(adivinar): print('Adivinar
59.         #%%s: ' % (adivinasrealizadas)) adivinar = entrada()
60

```

Una conjetura válida tiene solo dígitos y la misma cantidad de dígitos que el número secreto. El bucle while que comienza en la línea 58 comprueba la validez de la conjetura.

La variable adivinar se establece en la cadena en blanco en la línea 57, por lo que la condición del ciclo while en la línea 58 es Falso la primera vez que se verifica, lo que garantiza que la ejecución ingrese al ciclo que comienza en la línea 59.

Obtener las pistas para la suposición del jugador Después de que la ejecución supera el bucle while que comenzó en la línea 58, la suposición contiene una suposición válida. Ahora el programa pasa adivinar y secretNum a la función getClues() :

```

62.     print(obtenerPistas(adivina, númerosecreto))
63.     adivinanzasTomadas += 1

```

Devuelve una cadena de pistas, que se muestran al jugador en la línea 62. La línea 63 incrementa las conjeturas Tomadas utilizando el operador de asignación aumentada para la suma.

Comprobar si el jugador ganó o perdió

Ahora averiguamos si el jugador ganó o perdió el juego:

```

secret y ciclo.
    if adivinar == secretNum:
66.        romper
67.        if conjeturasTomadas > MAX_CONJETURAS:
68.            print('Te quedaste sin conjeturas. La respuesta fue %s.' % (secretNum))

```

Si adivinar es el mismo valor que númerosecreto, el jugador tiene

adivinó correctamente el número secreto, y la línea 66 sale del bucle while que se inició en la línea 56. Si no, la ejecución continúa hasta la línea 67, donde el programa verifica si el jugador se quedó sin conjeturas.

Si el jugador aún tiene más intentos, la ejecución vuelve al ciclo while en la línea 56, donde le permite al jugador tener otro intento. Si el jugador se queda sin conjeturas (o el programa se sale del ciclo con la instrucción break en la línea 66), la ejecución continúa más allá del ciclo y hasta la línea 70.

Pedirle al jugador que juegue de nuevo

La línea 70 le pregunta al jugador si quiere volver a jugar:

```
70. print('¿Quieres volver a jugar? (sí o no)') 71.  
if not input().lower().startswith('y'): 72. break
```

La respuesta del jugador es devuelta por input(), se llama al método lower() y luego se llama al método beginwith() eso para verificar si la respuesta del jugador comienza con una y. Si no es así, el programa sale del ciclo while que comenzó en la línea 51. Como no hay más código después de este ciclo, el programa finaliza.

Si la respuesta comienza con y, el programa no ejecuta la declaración de interrupción y la ejecución vuelve a la línea 51. Luego, el programa genera un nuevo número secreto para que el jugador pueda jugar un nuevo juego.

RESUMEN

Bagels es un juego simple de programar pero puede ser difícil de ganar.

Pero si sigues jugando, eventualmente descubrirás mejores formas de adivinar usando las pistas que te da el juego. Esto es muy parecido a cómo mejorará en la programación cuanto más lo haga.

Este capítulo presentó algunas funciones y métodos nuevos (`shuffle()`, `sort()` y `join()`) junto con un par de atajos útiles. Los operadores de asignación aumentada requieren menos escritura cuando desea cambiar el valor relativo de una variable; por ejemplo, `spam = spam + 1` se puede acortar a `spam += 1`. Con la interpolación de cadenas, puede hacer que su código sea mucho más legible colocando `%s` (llamado especificador de conversión) dentro de la cadena en lugar de usar muchas operaciones de concatenación de cadenas.

En el [Capítulo 12](#), no haremos ninguna programación, pero los conceptos (coordenadas cartesianas y números negativos) serán necesarios para los juegos en los capítulos posteriores del libro. Estos conceptos matemáticos se usan no solo en los juegos `Sonar Treasure Hunt`, `Reversegam` y `Dodger` que haremos, sino también en muchos otros juegos. Incluso si ya conoce estos conceptos, lea brevemente el [Capítulo 12 para refrescarse](#).

12

LA COORDENADA CARTESIANA SISTEMA



Este capítulo repasa algunos conceptos matemáticos simples que usará en el resto de este libro. En los juegos bidimensionales (2D), los gráficos de la pantalla se pueden mover hacia la izquierda o hacia la derecha y hacia arriba o hacia abajo. Estos juegos necesitan una forma de traducir un lugar en la pantalla en números enteros que el programa pueda manejar.

Aquí es donde entra en juego el sistema de coordenadas cartesianas .

Las coordenadas son números que representan un punto específico en la pantalla. Estos números se pueden almacenar como enteros en las variables de su programa.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Sistemas de coordenadas cartesianas
- El eje x y el eje y
- Números negativos
- Píxeles
- La propiedad conmutativa de la suma
- Valores absolutos y la función abs()

REJILLAS Y CARTESIANOS COORDENADAS

Una forma común de referirse a lugares específicos en un tablero de ajedrez es marcando cada fila y columna con letras y números.

La figura 12-1 muestra un tablero de ajedrez que tiene marcadas cada fila y columna.

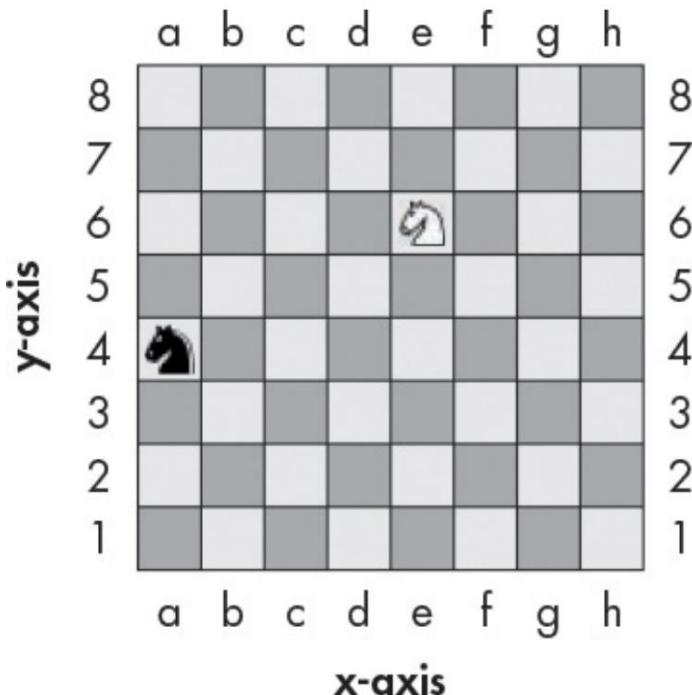


Figura 12-1: Un tablero de ajedrez de muestra con un caballo negro en (a, 4) y un caballo blanco en (e, 6)

Una coordenada para un espacio en el tablero de ajedrez es una combinación de una fila y una columna. En el ajedrez, la pieza del caballo se parece a la cabeza de un caballo. El caballo blanco de la Figura 12-1 está ubicado en el punto (e, 6) ya que está en la columna e y la fila 6, y el caballo negro está ubicado en el punto (a, 4) porque está en la columna a y la fila 4 .

Puedes pensar en un tablero de ajedrez como una coordenada cartesiana sistema. Al usar una etiqueta de fila y una etiqueta de columna, puede dar una

coordenada que es para uno, y solo uno, espacio en el tablero.

Si ha aprendido acerca de los sistemas de coordenadas cartesianas en la clase de matemáticas, puede saber que los números se usan tanto para las filas como para las columnas. Un tablero de ajedrez que usa coordenadas numéricas se parecería a la figura 12-2.

Los números que van de izquierda a derecha a lo largo de las columnas son parte del eje x. Los números que suben y bajan a lo largo de las filas son parte del eje y. Las coordenadas siempre se describen con la coordenada x primero, seguida de la coordenada y. En la Figura 12-2, la coordenada x para el caballo blanco es 5 y la coordenada y es 6, por lo que el caballo blanco está ubicado en las coordenadas (5, 6) y no en (6, 5). De manera similar, el caballo negro está ubicado en la coordenada (1, 4), no en (4, 1), ya que la coordenada x del caballo negro es 1 y su coordenada y es 4.

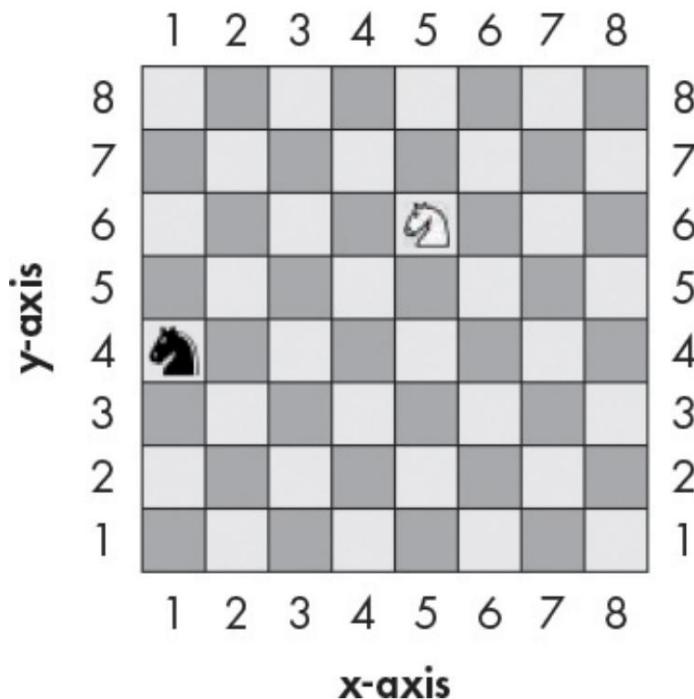


Figura 12-2: El mismo tablero de ajedrez pero con coordenadas numéricas para filas y columnas

Note que para que el caballo negro se mueva hacia el blanco

posición del caballo, el caballo negro debe moverse dos espacios hacia arriba y cuatro espacios hacia la derecha. Pero no es necesario mirar el tablero para darse cuenta de esto. Si sabes que el caballo blanco está ubicado en (5, 6) y el caballo negro está ubicado en (1, 4), puedes usar la resta para calcular esta información.

Resta la coordenada x del caballo negro de la coordenada x del caballo blanco:

$5 - 1 = 4$. El caballo negro tiene que moverse a lo largo del eje x cuatro espacios.

Ahora resta la coordenada y del caballo negro de la coordenada y del caballo blanco:

$6 - 4 = 2$. El caballo negro tiene que moverse a lo largo del eje y por dos

espacios.

Haciendo un poco de matemática con los números de coordenadas, puedes calcular las distancias entre dos coordenadas.

NÚMEROS NEGATIVOS

Las coordenadas cartesianas también usan **números negativos**, es decir, números que son menores que cero. Un signo menos delante de un número indica que es negativo: -1 es menor que 0 . Y -2 es menor que -1 .

Pero el 0 en sí mismo no es positivo ni negativo. En la [Figura 12-3](#), puede ver los números positivos aumentando hacia la derecha y los números negativos disminuyendo hacia la izquierda en una recta numérica.

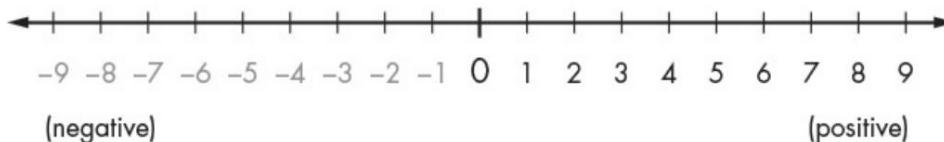


Figura 12-3: Una recta numérica con números positivos y negativos

La recta numérica es útil para ver la resta y la suma.

Puedes pensar en la expresión $5 + 3$ como el caballo blanco que comienza en la posición 5 y se mueve 3 espacios a la derecha. Como puede ver en la [Figura 12-4](#), el caballo blanco termina en la posición 8 .

Esto tiene sentido, porque $5 + 3$ es 8.

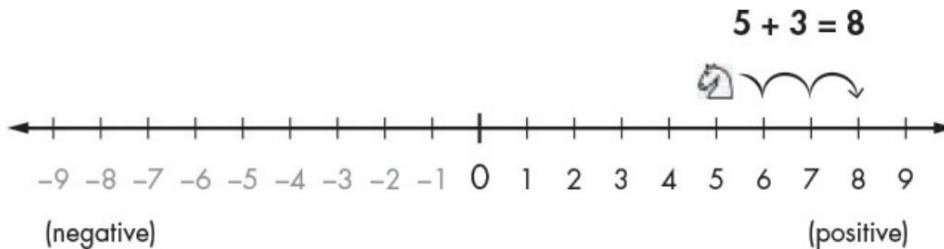


Figura 12-4: Mover el caballo blanco hacia la derecha aumenta la coordenada.

Restas moviendo el caballo blanco hacia la izquierda. Entonces, si la expresión es $5 - 6$, el caballo blanco comienza en la posición 5 y se mueve 6 espacios a la izquierda, como se muestra en la Figura 12-5.

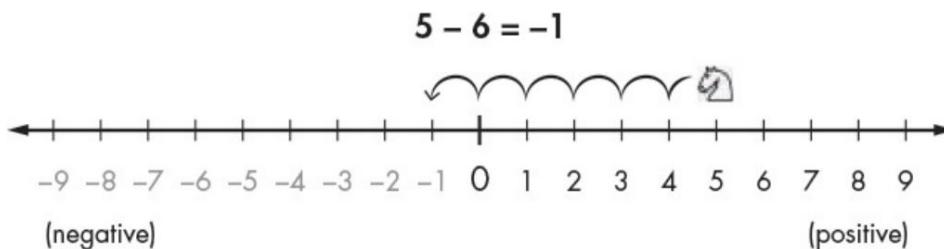


Figura 12-5: Mover el caballo blanco hacia la izquierda resta de la coordenada.

El caballo blanco termina en la posición -1. Eso significa que $5 - 6$ es igual a -1.

Si sumas o restas un número negativo, el caballo blanco se mueve en dirección opuesta a como lo hace con los números positivos. Si sumas un número negativo, el caballo se mueve hacia la izquierda. Si restas un número negativo, el caballo se mueve hacia la derecha. La expresión $-1 - (-4)$ sería igual a 3, como se muestra en la figura 12-6. Note que $-1 - (-4)$ tiene la misma respuesta que $-1 + 4$.

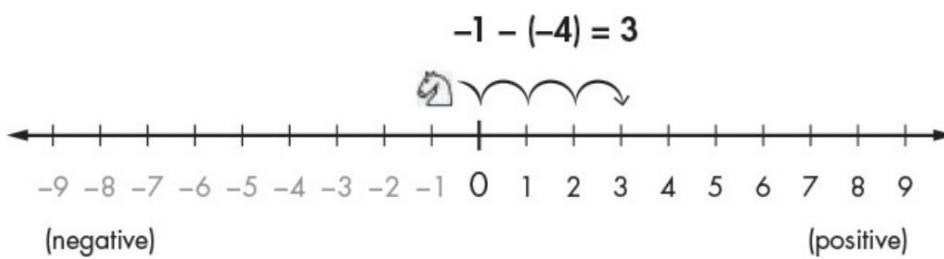


Figura 12-6: El caballo comienza en -6 y se mueve hacia la derecha 4 espacios.

Puedes pensar en el eje x como una recta numérica. Agrega otra recta numérica que sube y baja por el eje y . Si junta estas dos rectas numéricas, tiene un sistema de coordenadas cartesianas como el de la figura 12-7.

Sumar un número positivo (o restar un número negativo) movería al caballo hacia arriba en el eje y o hacia la derecha en el eje x , y restar un número positivo (o sumar un número negativo) movería al caballo hacia abajo en el eje y o a la izquierda en el eje x .

La coordenada $(0, 0)$ en el centro se llama origen. Es posible que hayas usado un sistema de coordenadas como este en tu clase de matemáticas. Como está a punto de ver, los sistemas de coordenadas como estos tienen muchos pequeños trucos que puede usar para facilitar el cálculo de las coordenadas.

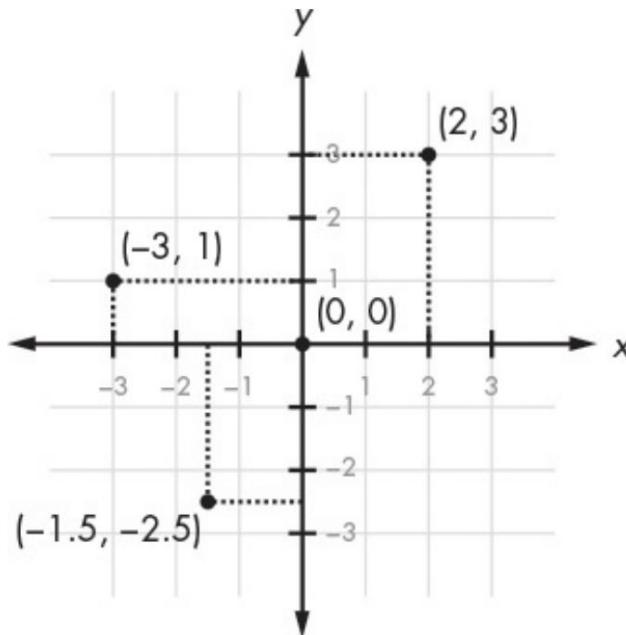


Figura 12-7: Juntar dos rectas numéricas crea un sistema de coordenadas cartesianas.

EL SISTEMA DE COORDENADAS DE UN PANTALLA DE ORDENADOR

La pantalla de su computadora está compuesta de píxeles, el punto de color más pequeño que puede mostrar una pantalla. Es común que las pantallas de las computadoras usen un sistema de coordenadas que tiene el origen $(0, 0)$ en la esquina superior izquierda y que aumenta hacia abajo y hacia la derecha.

Puede ver esto en la Figura 12-8, que muestra una computadora portátil con una resolución de pantalla de 1920 píxeles de ancho y 1080 píxeles de alto.

No hay coordenadas negativas. La mayoría de los gráficos por computadora usan este sistema de coordenadas para los píxeles en la pantalla, y lo usarás en los juegos de este libro. Para la programación, es importante saber cómo funcionan los sistemas de coordenadas, tanto los que se usan para matemáticas como los que se usan para computación.

pantallas

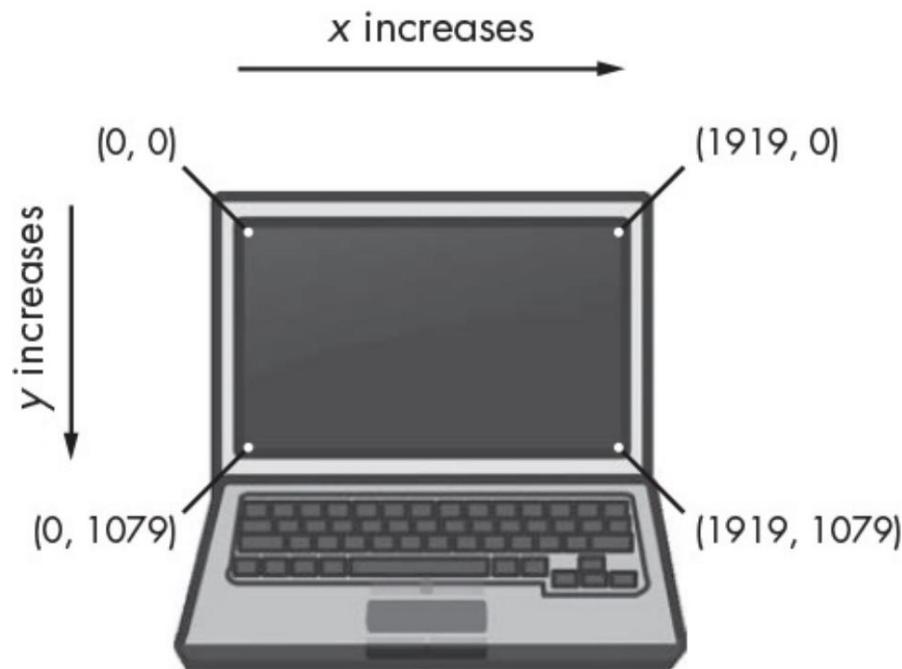


Figura 12-8: El sistema de coordenadas cartesianas en una pantalla de computadora

TRUCOS DE MATEMÁTICAS

Restar y sumar números negativos es fácil cuando tienes una recta numérica frente a ti. También puede ser fácil sin una recta numérica. Aquí hay tres trucos para ayudarte a sumar y restar números negativos por ti mismo.

Truco 1: Un signo menos se come el signo más a su izquierda
 Cuando vea un signo menos con un signo más a la izquierda, puede reemplazar el signo más con un signo menos. Imagina el signo menos “comiéndose” el signo más a su izquierda. La respuesta sigue siendo la misma, porque sumar un valor negativo es lo mismo que restar un valor positivo. Así que $4 + -2$ y $4 - 2$ se evalúan como 2, como puedes ver aquí:

$$\begin{array}{r} 4 + -2 \\ \hline \downarrow \\ 4 - 2 \\ \hline 2 \end{array}$$

Truco 2: dos menos se combinan en un más

Cuando vea los dos signos menos uno al lado del otro sin un número entre ellos, pueden combinarse en un signo más.

La respuesta sigue siendo la misma, porque restar un valor negativo es lo mismo que sumar un valor positivo:

$$\begin{array}{r} 4 - -2 \\ \hline \downarrow \\ 4 + 2 \\ \hline 6 \end{array}$$

Truco 3: dos números que se agregan pueden intercambiarse

Lugares

Siempre puedes intercambiar los números además. Esta es la propiedad commutativa de la suma. Eso significa que hacer un intercambio como $6 + 4$ a $4 + 6$ no cambiará la respuesta, como puede ver cuando cuenta las casillas en la Figura 12-9.

$$6 + 4 = 10$$



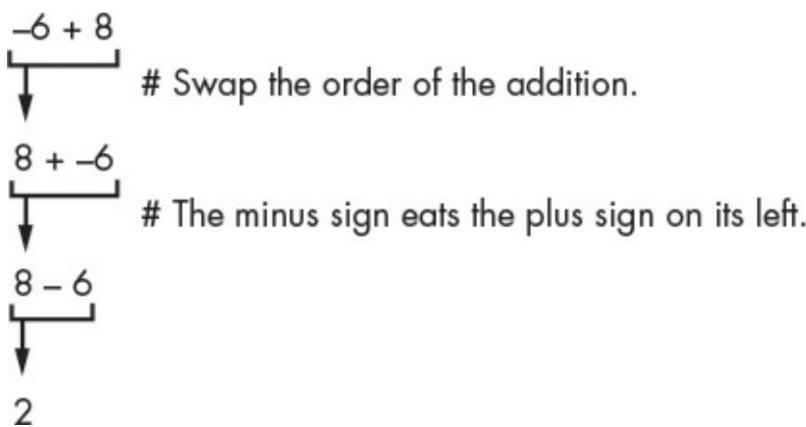
$$4 + 6 = 10$$



Figura 12-9: La propiedad commutativa de la suma te permite intercambiar números.

Digamos que estás sumando un número negativo y un número positivo, como $-6 + 8$. Debido a que estás sumando números, puedes intercambiar el orden de los números sin cambiar la respuesta.

Esto significa que $-6 + 8$ es lo mismo que $8 + -6$. Luego, cuando miras $8 + -6$, ves que el signo menos puede comerse al signo más a su izquierda, y el problema se convierte en $8 - 6 = 2$, como puedes ver aquí:



Has reorganizado el problema para que sea más fácil de resolver sin usar una calculadora o una computadora.

VALORES ABSOLUTOS Y LOS FUNCIÓN ABS()

El valor absoluto de un número es el número sin el menos firmar delante de él. Por lo tanto, los números positivos no cambian, pero los números negativos se vuelven positivos. Por ejemplo, el valor absoluto de -4 es 4. El valor absoluto de -7 es 7. El el valor absoluto de 5 (que ya es positivo) es solo 5.

Puedes calcular la distancia entre dos objetos restando sus posiciones y tomando el valor absoluto de la diferencia. Imagina que el caballo blanco está en la posición 4 y el caballo negro en la posición -2. La distancia sería 6, ya que $4 - -2$ es 6, y el valor absoluto de 6 es 6.

Funciona sin importar el orden de los números. Para ejemplo, $-2 - 4$ (es decir, menos dos menos cuatro) es -6, y el valor absoluto de -6 también es 6.

La función `abs()` de Python devuelve el valor absoluto de un entero. Ingrese lo siguiente en el shell interactivo:

```
>>> abdominales(-5)
5
>>> abs(42)
42
>>> abs(-10.5)
10.5
```

El valor absoluto de -5 es 5. El valor absoluto de un número positivo es solo el número, por lo que el valor absoluto de 42 es 42. Incluso los números con decimales tienen un valor absoluto, por lo que el valor absoluto de -10,5 es 10,5.

RESUMEN

La mayoría de la programación no requiere entender muchas matemáticas. Hasta este capítulo, nos las habíamos arreglado con sumas y multiplicaciones simples.

Se necesitan sistemas de coordenadas cartesianas para describir dónde se encuentra una determinada posición en un área bidimensional. Las coordenadas tienen dos números: la coordenada x y la coordenada y. El eje x corre de izquierda a derecha, y el eje y corre de arriba a abajo. En una pantalla de computadora, el origen está en la esquina superior izquierda y las coordenadas aumentan hacia la derecha y hacia abajo.

Los tres trucos matemáticos que aprendiste en este capítulo facilitan la suma de números enteros positivos y negativos. El primer truco es que un signo menos se comerá el signo más a su izquierda. El segundo truco es que dos menos uno al lado del otro se combinarán en un Signo de más. El tercer truco es que puedes intercambiar la posición de los números que estás sumando.

El resto de los juegos de este libro utilizan estos conceptos porque tienen áreas bidimensionales. Todos los juegos gráficos requieren comprender cómo funcionan las coordenadas cartesianas.

13

BÚSQUEDA DEL TESORO CON SONAR



El juego Sonar Treasure Hunt en este capítulo es el primero en hacer uso del sistema de coordenadas cartesianas que aprendiste en el Capítulo 12. Este juego también usa estructuras de datos, que es solo una forma elegante de decir que tiene valores de lista que contienen otras listas. y variables complejas similares. A medida que los juegos que programa se vuelven más complicados, deberá organizar sus datos en estructuras de datos.

En el juego de este capítulo, el jugador lanza dispositivos de sonar en varios lugares del océano para localizar cofres del tesoro hundidos. El sonar es una tecnología que utilizan los barcos para localizar objetos bajo el mar. Los dispositivos de sonar en este juego le dicen al jugador qué tan lejos está el cofre del tesoro más cercano, pero no en qué dirección. Pero al colocar varios dispositivos de sonda, el jugador puede averiguar la ubicación del cofre del tesoro.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Estructuras de datos
- El teorema de Pitágoras

- El método de lista remove()
- El método de cadena isdigit()
- La función sys.exit()

Hay 3 cofres para recolectar, y el jugador solo tiene 20 dispositivos de sonda para encontrarlos. Imagina que no pudieras ver el cofre del tesoro en la figura 13-1. Debido a que cada dispositivo de sonar puede encontrar solo la distancia desde el cofre, no la dirección del cofre, el tesoro podría estar en cualquier parte del anillo alrededor del dispositivo de sonar.

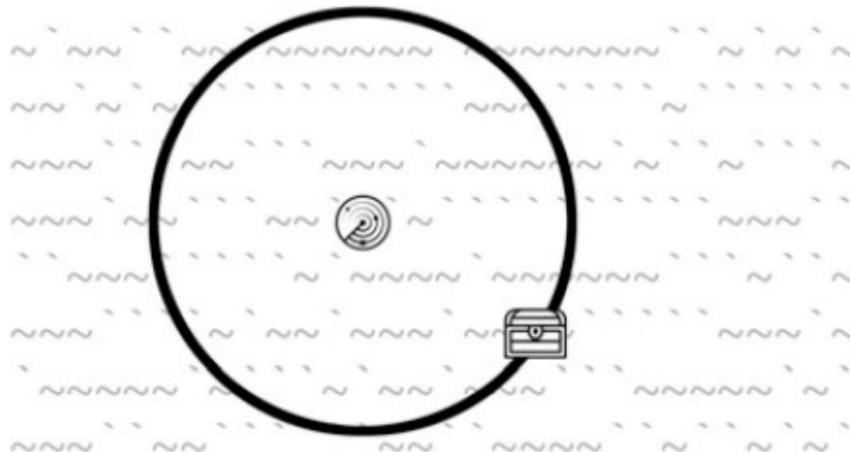


Figura 13-1: El anillo del dispositivo de sonar toca el cofre del tesoro (oculto).

Pero múltiples dispositivos de sonar trabajando juntos pueden reducir la ubicación del cofre a las coordenadas exactas donde el los anillos se cruzan (vea la figura 13-2).

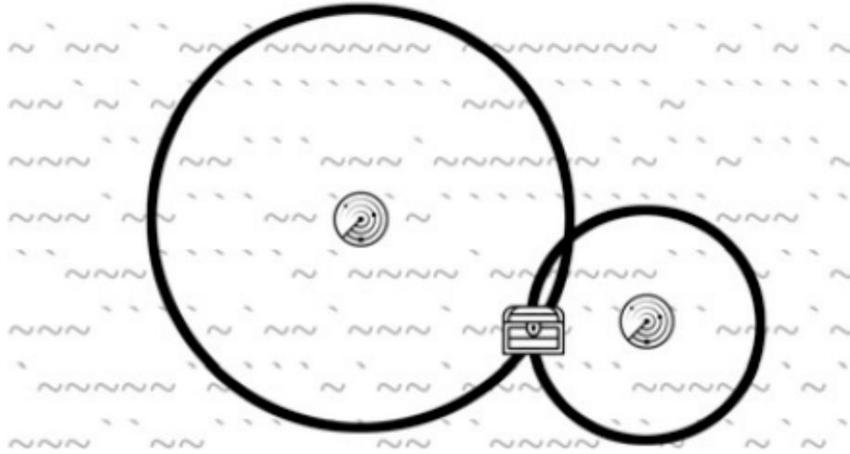


Figura 13-2: La combinación de varios anillos muestra dónde se pueden ocultar los cofres del tesoro.

MUESTRA DE EJECUCIÓN DE SONAR BÚSQUEDA DEL TESORO

Esto es lo que ve el usuario cuando ejecuta el programa Sonar Treasure Hunt. El texto que ingresa el jugador se muestra en negrita.

SONAR!

¿Quieres ver las instrucciones? (sí No)

no

1 2 3 4 5

012345678901234567890123456789012345678901234567890123456789

0 ~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~' 0
1 ~~~~~'~~~~~'~~~~~'~~~~~'~~~~~'~~~~~'~~~~~'~~~~~'~~~~~'~~~~~'~~~~~' 1
2 '~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~' 2
3 '~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~' 3
4 ~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~' 4
5 '~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~' 5
6 ~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~' 6
7 ~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~' 7
8 '~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~' 8
9 ~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~' 9
10 '~~~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~' 10
11 ~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~'~~~' 11

3 `~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 3
 4 ~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 4
 5 `~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 5
 6 `~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 6
 7 `~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 7
 8 `~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 8
 9 `~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 9
 10 `~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 10
 11 `~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 11
 12 ~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 12
 13 `~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 13
 14 ``~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 14

01234567890123456789012345678901234567890123456789

1 2 3 4 5

Tesoro detectado a una distancia de 3 del dispositivo de sonda.

Le quedan 18 dispositivos de sonda. Quedan 3 cofres del tesoro.

¿Dónde quieres colocar el próximo dispositivo de sonda? (0-59 0-14) (o escriba salir)

25 10

1 2 3 4 5

01234567890123456789012345678901234567890123456789

0 ~`~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 0
 1 ~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 1
 2 `~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 2
 3 `~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 3
 4 ~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 4
 5 `~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 5
 6 `~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 6
 7 `~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 7
 8 `~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 8
 9 `~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 9
 10 `~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 10
 11 `~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 11
 12 ~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 12
 13 `~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 13
 14 ``~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~`~~~~~` 14

01234567890123456789012345678901234567890123456789

1 2 3 4 5



sonar.py

```
1. # Búsqueda del tesoro con sonda
2.
3. importar aleatorio 4.
importar sistema 5.
importar matemáticas 6.

7. def obtenerNuevoTablero():
8. # Cree una nueva estructura de datos de placa de 60x15.
9. tablero = []
10. for x in range(60): # La lista principal es una lista de 60 listas. 11
    board.append([]) for
12    y in range(15): # Cada lista en la lista principal tiene 15
        cadenas de un solo carácter.
13    # Usa diferentes personajes para el océano para hacerlo más
        legible.
14    if random.randint(0, 1) == 0:
15        tablero[x].append('~') else:
        diecisésis.
16        tablero[x].append(' ')
17        tabla de retorno
18
19
20. def dibujarTablero(tablero):
21. # Dibujar la estructura de datos del tablero.
22. filaDígitosDecenas =      ' # Espacio inicial para los números abajo a la izquierda
```

```

lado del tablero

23. for i in range(1, 6): 24.
    tensDigitsLine += (' ' * 9) +
    str(i)
    25

26. # Imprime los números en la parte superior de la pizarra.
27. print(tensDigitsLine) 28. print(' ' + ('0123456789' * 6)) 29.
    print() 30.

31. # Imprime cada una de las 15 filas.

32. para fila en rango (15):
33.     # Los números de un solo dígito deben rellenarse con un espacio
34.     adicional. si fila < 10:
35.         extraSpace =
36.         más:
37.             espacio extra =
38.

39.         # Cree la cadena para esta fila en el tablero.
40.         filaDelTablero =
41.             para la columna en el rango (60):
42.                 filaDelTablero += tablero[columna][fila]
43.

44.             print('%s%s %s %s' % (espacioextra, fila, fila del tablero, fila))
45.

46. # Imprime los números en la parte inferior del tablero.

47. print() 48.
print(' ' + ('0123456789' * 6)) 49.
print(tensDigitsLine) 50.

51. def getRandomChests(numChests): 52. #
    Crear una lista de estructuras de datos de cofres (listas de dos elementos de
    coordenadas x, y int). 53. cofres = [] 54. while len(cofres) < numCofres:
55.     nuevoCofre = [random.randint(0, 59), random.randint(0, 14)] si nuevoCofre no
    está en cofres: # Asegúrate de que haya un cofre no es ya

56.

    aquí.

57.         cofres.append(nuevoCofre)
58. devolver cofres
59.

```

```
60. def isOnBoard(x, y): 61. #
Devuelve True si las coordenadas están en el tablero; de lo contrario, regresa
Falso.

62. devuelve x >= 0 y x <= 59 y y >= 0 y y <= 14 63.

64. def hacerMovida(tablero, cofres, x, y): 65. #
Cambia la estructura de datos del tablero con un carácter de dispositivo de sonda.

    Elimina los cofres del tesoro de la lista de cofres a medida que los encuentres.

66. # Devuelve Falso si se trata de un movimiento inválido.

67. # De lo contrario, devuelve la cadena del resultado de este movimiento. 68. la
distancia más pequeña = 100 # Cualquier cofre estará más cerca de 100. 69. para cx,
cy en cofres:
70.     distancia = matemática.sqrt((cx - x) * (cx - x) + (cy - y) * (cy - y))
71.
72.     si la distancia < la distancia más pequeña: # Queremos el tesoro más cercano
tórax.
73.     distancia más pequeña = distancia
74.
75. distancia más pequeña = ronda (distancia más pequeña)
76.
77. si la distancia más pequeña == 0:
78.     ¡# xy está directamente en un cofre del tesoro!
79.     cofres.remove([x, y])
80.     volver '¡Has encontrado un cofre del tesoro hundido!'

81. más:
82.     si la distancia más pequeña < 10:
83.         tablero[x][y] = str(distancia más pequeña)
84.         return 'Tesoro detectado a una distancia de %s del sonar
dispositivo.' % (distancia más pequeña)
85.     más:
86.         tablero[x][y] = 'X' return
87.     'El sonar no detectó nada. Todos los cofres del tesoro están fuera de
alcance.
88.
89. def enterPlayerMove(anteriorMoves): 90. # Permitir
que el jugador ingrese su movimiento. Devuelve una lista de dos elementos de coordenadas
xy int. 91. print('¿Dónde quieras colocar el próximo dispositivo de sonda? (0-59
0-14)
(o escriba quit)')
92. while True:
```


125. 4 ~`~~~~`~~`~~`C`~``~~`~~`~`~`~`~` 4
126.
127. 012345678901234567890123456789012
128. 1 2 3
129. (En el juego real, los cofres no son visibles en el océano).
130.
131. Presiona enter para continuar..."")
132. input() 133.

134. print("Cuando dejas caer un dispositivo de sonar directamente sobre un cofre,
recuperarlo y el otro
135. Los dispositivos de sonda se actualizan para mostrar qué tan lejos está el siguiente cofre más cercano. Él
cofres
136. están fuera del alcance del dispositivo de sonda de la izquierda, por lo que muestra una X.
137.
138. 1 2 3
139. 012345678901234567890123456789012
140.
141. 0 ~~~~`~`~`~~`~~`~~`~~`~~`~~`~~`~~`~~` 0
142. 1 ~`~`~`~`~`~`~`~`~`~`~`~`~`~`~`~`~`~`~` 1
143. 2 `~`X`~`7`~~`C`~~`~~`~~`~~`~~`~~`~~` 2
144. 3 `~~`~~`~~`~~`~~`~~`~~`~~`~~`~~`~~`~~` 3
145. 4 ~`~~~~`~~`~~`C`~``~~`~~`~`~`~`~`~` 4
146.
147. 012345678901234567890123456789012
148. 1 2 3
149.
150. Los cofres del tesoro no se mueven. Los dispositivos de sonda pueden detectar tesoros
cofres
151. hasta una distancia de 9 espacios. Intenta recoger los 3 cofres antes de correr.
fuera de
152. dispositivos de sonar. ¡Buena suerte!
153.
154. Presiona enter para continuar..."")
155. input() 156.

157.
158.
159. imprimir('¡SONAR!') 160.
imprimir()

```
161. print("¿Le gustaría ver las instrucciones? (sí/no)") 162. if
input().lower().startswith('y'): 163. showInstructions() 164.

165. mientras es cierto:
166. # Configuración del juego
167. sonDispositivos = 20
168. elTablero = obtenerNuevoTablero()
169. losCofres = obtenerCofresAl Azar(3) 170.
dibujarTablero(elTablero) 171. JugadasAnteriores =
[]
172.
173. while sonarDevices > 0:
174.     # Mostrar el dispositivo de sonda y los estados del cofre.
175.     print('Te quedan %s dispositivo(s) de sonda. Quedan %s cofre(s) del tesoro.'
% (sonarDevices, len(theCofres)))
176.
177.     x, y = enterPlayerMove(anteriorMoves)
178.     previousMoves.append([x, y]) # Debemos rastrear todos los movimientos para que
los dispositivos de sonda puedan actualizarse.
179.
180.     moverResultado = hacerJugada (elTablero, losCofres, x, y) if
181.     moverResultado == Falso:
182.         Seguir
183.     más:
184.         if moveResult == '|Has encontrado un cofre del tesoro hundido|':
185.             # Actualice todos los dispositivos de sonda actualmente en el
186.             mapa. para x, y en jugadas anteriores:
187.                 hacerMovimiento(elTablero, losCofres, x, y)
188.                 dibujarTablero(elTablero) imprimir(resultadomovimiento)
189.
190.
191.     if len(theCofres) == 0:
192.         print('|Has encontrado todos los cofres del tesoro hundidos!
¡Felicitaciones y buen juego!') romper
193.
194.
195.     sonDispositivos -= 1
196.
197. si sonarDevices == 0:
```

```
198.     print('¡Nos hemos quedado sin dispositivos de sonar! Ahora tenemos  
que dar la vuelta al barco y dirigirnos') print('A casa con los cofres del  
199.     tesoro todavía por ahí! Fin del juego.') print(' Los cofres restantes  
estaban aquí :) para x, y en los Cofres: print(' %s, %s' % (x, y))  
200.  
201.  
202.  
203.  
204. print('¿Quieres volver a jugar? (sí o no)') 205. if not  
input().lower().startswith('y'): 206. sys.exit()
```

DISEÑO DEL PROGRAMA

Antes de tratar de entender el código fuente, juegue el juego varias veces para saber qué está pasando. El juego Sonar Treasure Hunt utiliza listas de listas y otras variables complicadas, llamadas estructuras de datos. Las estructuras de datos almacenan arreglos de valores para representar algo. Por ejemplo, en el Capítulo 10, la estructura de datos de un tablero de Tic-Tac Toe era una lista de cadenas. La cadena representaba una X, una O o un espacio vacío, y el índice de la cadena en la lista representaba el espacio en el tablero. El juego Sonar Treasure Hunt tendrá estructuras de datos similares para las ubicaciones de los cofres del tesoro y los dispositivos de sonda.

IMPORTAR EL ALEATORIO, SYS, Y MÓDULOS DE MATEMÁTICAS

Al comienzo del programa, importamos las funciones random, sys y math . módulos:

-
1. # Búsqueda del tesoro con sonda
 - 2.
 3. importar al azar

4. importar sistema
5. importar matemáticas

El módulo sys contiene la función exit() , que finaliza el programa inmediatamente. Ninguna de las líneas de código después de la llamada sys.exit() se ejecutará; el programa simplemente se detiene como si ha llegado al final. Esta función se utiliza más adelante en el programa.

El módulo matemático contiene la función sqrt() , que se utiliza para encontrar la raíz cuadrada de un número. Las matemáticas detrás de las raíces cuadradas se explican en ["Encontrar el cofre del tesoro más cercano"](#) en la página 186.

CREANDO UN NUEVO TABLERO DE JUEGO

El inicio de cada nuevo juego requiere una nueva estructura de datos del tablero , que es creada por getNewBoard(). El tablero de juego Sonar Treasure Hunt es un océano de arte ASCII con coordenadas x e y a su alrededor.

Cuando usamos la estructura de datos del tablero , queremos poder acceder a su sistema de coordenadas de la misma forma que accedemos a las coordenadas cartesianas. Para hacerlo, usaremos una lista de listas para llamar a cada coordenada en el tablero de esta manera: tablero[x][y]. la x la coordenada viene antes de la coordenada y: para obtener la cadena en la coordenada (26, 12), accede al tablero[26][12], no al tablero[12][26].

```
7. def getNewBoard(): 8. #
Crear una nueva estructura de datos de tablero de 60x15.
9. tablero = [] 10.
for x in range(60): # La lista principal es una lista de 60 listas.
11     board.append([]) for
12         y in range(15): # Cada lista en la lista principal tiene 15
cadenas de un solo carácter.
```

```
13     # Usa diferentes personajes para el océano para hacerlo más
14     legible.
15     if random.randint(0, 1) == 0:
16         tablero[x].append('~') else:
17             decisibis.
18
19     tablero[x].append("")
```

La estructura de datos del tablero es una lista de listas de cadenas. La primera lista representa la coordenada x. Dado que el tablero del juego tiene 60 caracteres, esta primera lista debe contener 60 listas. En la línea 10, creamos un bucle for que le agregará 60 listas en blanco.

Pero el tablero es más que una lista de 60 listas en blanco. Cada una de las 60 listas representa una coordenada x del tablero de juego. Hay 15 filas en el tablero, por lo que cada una de estas 60 listas debe contener 15 cadenas. La línea 12 es otro bucle for que agrega 15 cadenas de un solo carácter que representan el océano.

El océano será un montón de cadenas '~' y '`' elegidas al azar . Los caracteres de tilde (~) y acento grave (`), ubicados junto a la tecla 1 en su teclado, se utilizarán para las olas del océano. Para determinar qué carácter usar, las líneas 14 a 17 aplican esta lógica: si el valor de retorno de random.randint() es 0, agregue la cadena '~' ; de lo contrario, agregue la cadena '`' . Esto le dará al océano un aspecto aleatorio y entrecortado.

Para un ejemplo más pequeño, si el tablero se estableciera en `[['~','~','~'],
[['~','~','~'],[['~','~','~'],[~,~,~],['~','~','~']]]` entonces el tablero que dibujó se vería
Me gusta esto:

~~~~~  
~~~`~  
....

Finalmente, la función devuelve el valor en la variable tablero

en el eje x.

2. Use esa cadena para mostrar las coordenadas del eje x a lo largo de la parte superior de la pantalla.
3. Imprime cada fila del océano junto con las coordenadas del eje y en ambos lados de la pantalla.
4. Imprima el eje x nuevamente en la parte inferior. Tener coordenadas en todos los lados hace es más fácil ver dónde colocar un dispositivo de sonda.

Dibujar las coordenadas X a lo largo de la parte superior de la Junta

La primera parte de drawBoard() imprime el eje x en la parte superior del tablero. Debido a que queremos que cada parte del tablero sea uniforme, cada etiqueta de coordenadas puede ocupar solo un espacio de carácter. Cuando la numeración de coordenadas llega a 10, hay dos dígitos para cada número, así que colocamos los dígitos en el lugar de las decenas en una línea separada, como se muestra en la figura 13-3. El eje x está organizado de manera que la primera línea muestra los dígitos de las decenas y la segunda línea muestra los dígitos de las unidades.

```
++++++1+++++2+++++3      # First line  
++0123456789012345678901234567890123456789 # Second line  
  
+0 ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ ~~~~ 0 # Third line
```

Figura 13-3: El espacio utilizado para imprimir la parte superior del tablero de juego

Las líneas 22 a 24 crean la cadena para la primera línea del tablero, que es la parte de las decenas del eje x:

```
21. # Dibuja la estructura de datos del tablero.  
22. tensDigitsLine = lado      ' # Espacio inicial para los números abajo a la izquierda  
   del tablero  
23. for i in range(1, 6): 24.  
tensDigitsLine += (' ' * 9) + str(i)
```

Los números que marcan la posición de las decenas en la primera línea tienen 9 espacios entre ellos y hay 13 espacios al frente.

copiado al final de boardRow. Para cuando finaliza el ciclo, boardRow tiene las ondas de arte ASCII completas de la fila. La cadena en boardRow luego se imprime junto con los números de fila en la línea 44.

Dibujar las coordenadas X a lo largo de la parte inferior del tablero

Las líneas 46 a 49 son similares a las líneas 26 a 29:

```
46. # Imprime los números en la parte inferior del tablero.  
47. imprimir()  
48. imprimir(' ' + ('0123456789' * 6)) 49.  
imprimir(tensDigitsLine)
```

Estas líneas imprimen las coordenadas x en la parte inferior del tablero.

CREANDO EL ALEATORIO COFRES DEL TESORO

El juego decide aleatoriamente dónde están los cofres del tesoro escondidos. Los cofres del tesoro se representan como una lista de listas de dos números enteros. Estos dos números enteros son las coordenadas x e y de un solo cofre. Por ejemplo, si la estructura de datos del cofre fuera [[2, 2], [2, 4], [10, 0]], esto significaría que había tres tesoros .
cofres, uno en (2, 2), otro en (2, 4) y un tercero en (10, 0).

La función getRandomChests() crea un cierto número de estructuras de datos de cofres en coordenadas asignadas aleatoriamente:

```
51. def getRandomChests(numChests): 52. #  
Crear una lista de estructuras de datos de cofres (listas de dos elementos de x, y int  
coordenadas).  
53. cofres = [] 54. while  
len(cofres) < numCofres:
```

```

55.     newCofre = [random.randint(0, 59), random.randint(0, 14)] if
56.     newCofre no está en cofres: # Asegúrate de que no haya ningún
      cofre aquí.
57.     cofres.append(nuevoCofre)
58. devolver cofres

```

El parámetro numChests le dice a la función cuántos cofres del tesoro generar. El bucle while de la línea 54 iterará hasta que se hayan asignado las coordenadas a todos los cofres. Se seleccionan dos enteros aleatorios para las coordenadas en la línea 55.

La coordenada x puede estar entre 0 y 59, y la coordenada y puede estar entre 0 y 14. La expresión [random.randint(0, 59), random.randint(0, 14)] se evaluará como una lista valor como [2, 2] o [2, 4] o [10, 0]. Si estas coordenadas aún no existen en la lista de cofres , se adjuntan a los cofres en la línea 57.

DETERMINAR SI UN EL MOVIMIENTO ES VÁLIDO

Cuando el jugador ingresa las coordenadas x e y de dónde quiere colocar un dispositivo de sonda, debemos asegurarnos de que los números sean válidos. Como se mencionó anteriormente, hay dos condiciones para que un movimiento sea válido: la coordenada x debe ser entre 0 y 59, y la coordenada y debe estar entre 0 y 14

La función isOnBoard() utiliza una expresión simple con operadores y para combinar estas condiciones en una sola expresión y garantizar que cada parte de la expresión sea verdadera:

```

60. def isOnBoard(x, y):
61. # Devuelve True si las coordenadas están en el tablero; de lo contrario, regresa
      Falso.
62. devuelve x >= 0 y x <= 59 y y >= 0 y y <= 14

```

Debido a que estamos usando el operador booleano y , si incluso una de las coordenadas no es válida, la expresión completa se evalúa como falsa.

COLOCAR UNA MOVIDA EN EL TABLERO

En el juego Sonar Treasure Hunt, el tablero de juego se actualiza para mostrar un número que representa la distancia de cada dispositivo de sonda al cofre del tesoro más cercano. Entonces, cuando el jugador hace un movimiento al darle al programa una coordenada x e y, el tablero cambia según las posiciones de los cofres del tesoro.

```
64. def hacerMovida(tablero, cofres, x, y): 65. #
Cambia la estructura de datos del tablero con un carácter de dispositivo de sonda.
    Elimina los cofres del tesoro de la lista de cofres a medida que los encuentres.
66. # Devuelve Falso si se trata de un movimiento inválido.
67. # De lo contrario, devuelve la cadena del resultado de este movimiento.
```

La función hacerJugada() toma cuatro parámetros: la estructura de datos del tablero de juego, la estructura de datos del cofre del tesoro, la coordenada x y la coordenada y. La función hacerJugada() devolverá un valor de cadena que describe lo que sucedió en respuesta a la jugada:

- Si las coordenadas caen directamente en un cofre del tesoro, hazMovimiento() devuelve '¡Has encontrado un cofre del tesoro hundido!.'
- Si las coordenadas están a una distancia de 9 o menos de un cofre, makeMove() devuelve 'Tesoro detectado a una distancia de %s del sonar dispositivo.' (donde %s se reemplaza con la distancia entera).
- De lo contrario, hacerJugada() devolverá 'Sonar no detectó nada. Todas cofres del tesoro fuera de rango.'

Dadas las coordenadas de dónde el jugador quiere colocar el dispositivo de sonda y una lista de coordenadas x e y para los cofres del tesoro, necesitará un algoritmo para averiguar qué cofre del tesoro está más cerca.

Encontrar el cofre del tesoro más cercano Las líneas

68 a 75 son un algoritmo para determinar qué cofre del tesoro está más cerca del dispositivo de sonda.

68. la distancia más pequeña = 100 # Cualquier cofre estará más cerca de 100. 69.
para cx, cy en cofres: 70.

```
distancia = matemática.sqrt((cx - x) * (cx - x) + (cy - y) * (cy - y))  
71.  
72.     si la distancia < la distancia más pequeña: # Queremos el tesoro más cercano  
tórax.  
73.     distancia más pequeña = distancia
```

Los parámetros x e y son números enteros (digamos, 3 y 5), y juntos representan la ubicación en el tablero de juego donde el jugador adivinó. La variable de cofres tendrá un valor como [[5, 0], [0, 2], [4, 2]], que representa las ubicaciones de tres cofres del tesoro. La Figura 13-4 ilustra este valor.

Para encontrar la distancia entre el dispositivo de sonar y un cofre del tesoro, necesitaremos hacer algunas operaciones matemáticas para encontrar la distancia entre dos coordenadas x e y. Digamos que colocamos un dispositivo de sonar en (3, 5) y queremos encontrar la distancia al cofre del tesoro en (4, 2).

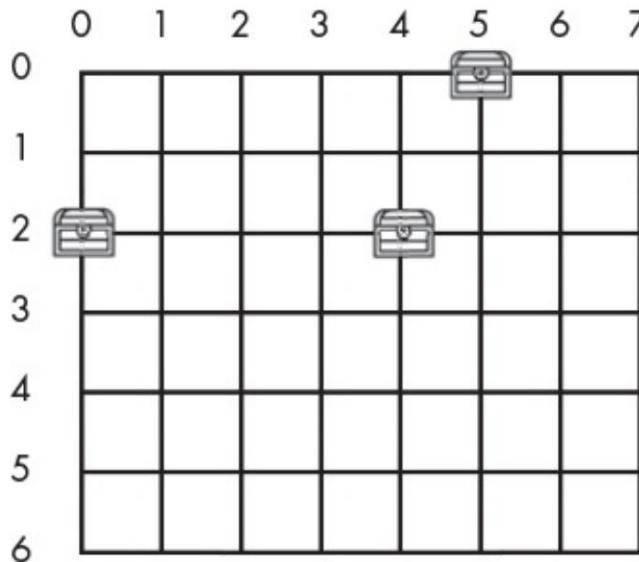


Figura 13-4: Los cofres del tesoro representados por [[5, 0], [0, 2], [4, 2]]

Para encontrar la distancia entre dos conjuntos de coordenadas x e y , usaremos el teorema de Pitágoras. Este teorema se aplica a los triángulos rectángulos : triángulos en los que una esquina mide 90 grados, el mismo tipo de esquina que se encuentra en un rectángulo. El teorema de Pitágoras dice que el lado diagonal del triángulo se puede calcular a partir de las longitudes de los lados horizontal y vertical. La figura 13-5 muestra un triángulo rectángulo dibujado entre el dispositivo de sonar en (3, 5) y el cofre del tesoro en (4, 2).

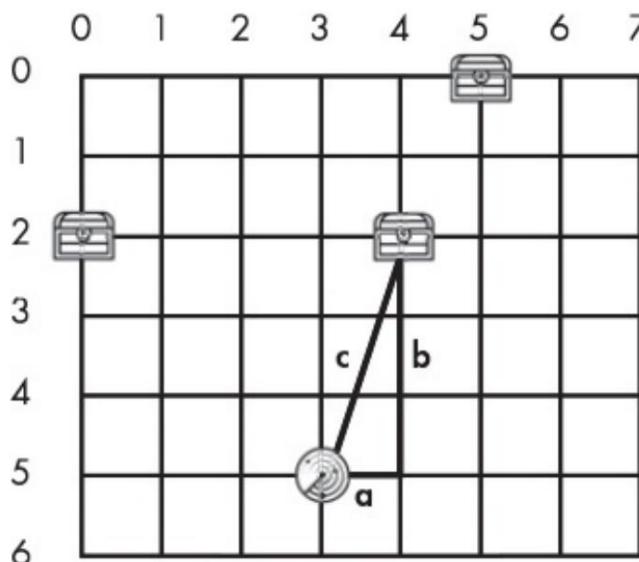


Figura 13-5: El tablero con un triángulo rectángulo dibujado sobre el dispositivo de sonar y un cofre del tesoro

El teorema de Pitágoras es $a + b = c^2$, en el que a es la longitud del lado horizontal, b es la longitud del lado vertical y c es la longitud del lado diagonal o hipotenusa.

Estas longitudes están elevadas al cuadrado, lo que significa que ese número se multiplica por sí mismo. "Descuadrar" un número se llama encontrar la raíz cuadrada del número, como tendremos que hacer para sacar c de c

Usemos el teorema de Pitágoras para encontrar la distancia entre el dispositivo de sonar en (3, 5) y el cofre en (4, 2):

1. Para encontrar a, resta la segunda coordenada x, 4, de la primera coordenada x,
 $3 - 4 = -1$.
2. Para encontrar b^2 , multiplique a por a: $-1 \times -1 = 1$. (Un número negativo multiplicado por un número negativo siempre es un número positivo).
3. Para encontrar b, resta la segunda coordenada y, 2, de la primera coordenada y,
 $5 - 2 = 3$.
4. Para encontrar b^2 , multiplica b por b: $3 \times 3 = 9$.
5. Para encontrar c^2 , a y b: $1 + 9 = 10$.
6. Para sacar c de c^2 , necesitas encontrar la raíz cuadrada de c^2 .

El módulo matemático que importamos en la línea 5 tiene una función de raíz cuadrada llamada `sqrt()`. Ingrese lo siguiente en el concha interactiva:

```
>>> importar matemáticas
>>> matemáticas.sqrt (10)
3.1622776601683795
>>> 3.1622776601683795 * 3.1622776601683795
10.000000000000002
```

Note que multiplicar una raíz cuadrada por sí misma produce el número cuadrado. (El 2 adicional al final del 10 se debe a una ligera imprecisión inevitable en la función `sqrt()`).

Al pasar c a `sqrt()`, podemos decir que el dispositivo de sonda está a 3,16 unidades del cofre del tesoro. El juego redondeará esto a 3.

Veamos de nuevo las líneas 68 a 70:

68. la distancia más pequeña = 100 # Cualquier cofre estará más cerca de 100. 69.

para cx, cy en cofres: 70.

`distancia = matemática.sqrt((cx - x) * (cx - x) + (cy - y) * (cy - y))`

El código dentro del bucle for de la línea 69 calcula la distancia de cada cofre. La línea 68 le da a la distancia más pequeña la distancia imposiblemente larga de 100 al comienzo del bucle, de modo que al menos uno de los cofres del tesoro que encuentre se colocará en la distancia más pequeña en la línea 73. Dado que $cx - x$ representa la distancia horizontal a entre el cofre y el sonar dispositivo, $(cx - x)^2$ es la a de

nuestro cálculo del teorema de Pitágoras. Se suma a $(cy - y)^2$, la b². Esta suma es c y se pasa a `sqrt()` para obtener el

distancia entre el tórax y el dispositivo de sonda.

Queremos encontrar la distancia entre el dispositivo de sonar y el cofre más cercano, por lo que si esta distancia es menor que la distancia más pequeña, se guarda como la nueva distancia más pequeña en la línea 73:

72. si la distancia < la distancia más pequeña: # Queremos el tesoro más cercano
tórax.

73. distancia más pequeña = distancia

Para cuando el ciclo for haya terminado, sabrás que la distancia más pequeña mantiene la distancia más corta entre el sonar dispositivo y todos los cofres del tesoro en el juego.

Eliminar valores con el método de lista remove()

El método de lista `remove()` elimina la primera aparición de un valor

coincidiendo con el argumento pasado. Por ejemplo, ingrese lo siguiente en el shell interactivo:

```
>>> x = [42, 5, 10, 42, 15, 42] >>>
x.quitar(10)
>>> x
[42, 5, 42, 15, 42]
```

El valor 10 ha sido eliminado de la lista x .

Ahora ingrese lo siguiente en el shell interactivo:

```
>>> x = [42, 5, 10, 42, 15, 42] >>>
x.quitar(42)
>>> x
[5, 10, 42, 15, 42]
```

Tenga en cuenta que solo se eliminó el primer valor 42 y el segundo y el tercero todavía están allí. El método remove() elimina la primera, y solo la primera, ocurrencia del valor que le pasa.

Si intenta eliminar un valor que no está en la lista, obtendrá un error:

```
>>> x = [5, 42]
>>> x.quitar(10)
Rastreo (llamadas recientes más última):
  Archivo "<stdin>", línea 1, en <módulo>
ValueError: list.remove(x): x no está en la lista
```

Al igual que el método append() , el método remove() se llama en un list y no devuelve una lista. Quieres usar código como x.remove(42), no x = x.remove(42).

Volvamos a encontrar las distancias entre los dispositivos de sonar y los cofres del tesoro en el juego. La única vez que la distancia más pequeña es igual a 0 es cuando los valores x e y del dispositivo de sonda

Las coordenadas son las mismas que las coordenadas x e y de un cofre del tesoro. Esto significa que el jugador ha adivinado correctamente la ubicación de un cofre del tesoro.

```
77. si la distancia más pequeña == 0:  
78.     ¡# xy está directamente en un cofre del  
79.     tesoro! cofres.remove([x, y])  
80.     volver '¡Has encontrado un cofre del tesoro hundido!'
```

Cuando esto sucede, el programa elimina la lista de dos enteros de este cofre de la estructura de datos de cofres con la lista remove() método. Entonces la función devuelve 'Has encontrado un tesoro hundido ¡tórax!'.

Pero si la distancia más pequeña no es 0, el jugador no adivinó la ubicación exacta de un cofre del tesoro, y el bloque else comienza en la línea 81 ejecuta:

```
81. más:  
82.     si la distancia más pequeña < 10:  
83.         tablero[x][y] = str(distancia más pequeña)  
84.         return 'Tesoro detectado a una distancia de %s del sonar  
             dispositivo.' % (distancia más pequeña)  
85.     más:  
86.         tablero[x][y] = 'X'  
87.     return 'El sonar no detectó nada. Todos los cofres del tesoro están  
             fuera de alcance.'
```

Si la distancia del dispositivo de sonar a un cofre del tesoro es menor que 10, la línea 83 marca el tablero con la versión en cadena de la distancia más pequeña. Si no, el tablero se marca con una 'X'. De esta forma, el jugador sabe qué tan cerca está cada dispositivo de sonda de un cofre del tesoro. Si el jugador ve un 0, sabe que está muy equivocado.

Obtener el movimiento del jugador

La función enterPlayerMove() recopila las coordenadas x e y del próximo movimiento del jugador:

```

89. def enterPlayerMove(anteriorMoves): 90. #
    Permitir que el jugador ingrese su movimiento. Devuelve una lista de dos
    elementos de coordenadas xy int. 91. print('¿Dónde quieras colocar el
    próximo dispositivo de sonda? (0-59 0-14)
        (o escriba salir)')
92. mientras sea cierto:
93.     mover = entrada
94.     () si mover. bajar () == 'salir':
95.         print('¡Gracias por jugar!')
96.         sys.exit()

```

El parámetro PreviousMoves es una lista de dos números enteros de los lugares anteriores en los que el jugador colocó un dispositivo de sonda. Esta información se utilizará para que el jugador no pueda colocar un dispositivo de sonda en un lugar donde ya lo haya colocado.

El ciclo while seguirá preguntando al jugador por su próximo movimiento hasta que ingrese las coordenadas de un lugar que aún no tiene un dispositivo de sonda. El jugador también puede ingresar 'salir' para salir del juego. En ese caso, la línea 96 llama a la función sys.exit() para terminar el programa inmediatamente.

Suponiendo que el jugador no haya ingresado 'salir', el código verifica que la entrada sea dos números enteros separados por un espacio. La línea 98 llama al método split() en movimiento como el nuevo valor de movimiento:

```

98.     move = move.split() if
99.     len(move) == 2 and move[0].isdigit() and move[1].isdigit() and
        isOnBoard(int(move[0]), int(move[1] )): if [int(movimiento[0]), int(movimiento[1])]
100.         en movimientos anteriores:
101.             print('Ya te mudaste allí.')
102.             Seguir
103.             devuelve [int(mover[0]), int(mover[1])]
104.

```

```
105.     print('Ingrese un número del 0 al 59, un espacio, luego un número del 0 al 14.')
```

Si el jugador escribió un valor como '1 2 3', entonces la lista devuelta por `split()` sería ['1', '2', '3']. En ese caso, la expresión `len(movimiento) == 2` sería Falsa (la lista en movimiento debería tener solo dos números porque representa una coordenada), y la expresión completa se evaluaría inmediatamente como Falsa.

Python no comprueba el resto de la expresión debido a un cortocircuito (que se describió en “[Evaluación de cortocircuitos](#)” en la [página 139](#)).

Si la longitud de la lista es 2, entonces los dos valores estarán en índices `move[0]` y `move[1]`. Para comprobar si esos valores son dígitos numéricos (como '2' o '17'), podría usar una función como `isOnlyDigits()` de “[Comprobar si una cadena tiene solo números](#)” en la [página 158](#). Pero Python ya tiene un método que hace esto.

El método de cadena `isdigit()` devuelve `True` si la cadena consta únicamente de números. De lo contrario, devuelve `False`. Ingrese lo siguiente en el shell interactivo:

```
>>> '42'.isdigit()
Verdadero
>>> 'cuarenta'.isdigit()
Falso
>>> ".isdigit()"
Falso
>>> 'hola'.isdigit()
Falso
>>> x = '10'
>>> x.es un dígito()
Verdadero
```

Tanto `move[0].isdigit()` como `move[1].isdigit()` deben ser `True` para el

toda la condición para ser verdadera. La parte final de la condición de la línea 99 llama a la función `isOnBoard()` para verificar si `x` e `y` existen coordenadas en el tablero.

Si toda la condición es verdadera, la línea 100 comprueba si la movimiento existe en la lista de movimientos anteriores . Si es así, entonces la declaración de continuar de la línea 102 hace que la ejecución regrese al inicio del ciclo `while` en la línea 92 y luego solicite el movimiento del jugador nuevamente. Si no es así, la línea 103 devuelve una lista de dos enteros de las coordenadas `x` e `y`.

IMPRESIÓN DEL JUEGO

INSTRUCCIONES PARA EL JUGADOR

La función `showInstructions()` es un par de llamadas `print()` que imprimen cadenas de varias líneas:

```
107. def showInstructions():
108.     print("""Instrucciones: 109.
Eres el capitán del Simon, un barco que busca tesoros. Tu actual
misión
--snip--
154. Presiona enter para
continuar...""") 155. input()
```

La función `input()` le da al jugador la oportunidad de presionar ENTER antes de imprimir la siguiente cadena. Esto se debe a que la ventana IDLE solo puede mostrar una cantidad limitada de texto a la vez, y no queremos que el jugador tenga que desplazarse hacia arriba para leer el comienzo del texto. Después de que el jugador presiona ENTER, la función vuelve a la línea que llamó a la función.

EL BUCLE DEL JUEGO

Ahora que hemos ingresado todas las funciones que llamará nuestro juego, ingresemos a la parte principal del juego. Lo primero que ve el jugador después de ejecutar el programa es el título del juego impreso en la línea 159. Esta es la parte principal del programa, que comienza ofreciendo instrucciones al jugador y luego configurando las variables que usará el juego.

```

159. print('¡SONAR!') 160.
print() 161. print('¿Le
gustaría ver las instrucciones? (sí/no)') 162. if input().lower().startswith('y'):
163. mostrarInstrucciones()

164.

165. mientras es cierto:

166. # Configuración del
juego 167. sonarDevices = 20

168. elTablero = obtenerNuevoTablero()
169. losCofres = obtenerCofresAl Azar(3) 170.
dibujarTablero(elTablero) 171. JugadasAnteriores
= []

```

La expresión `input().lower().startswith('y')` permite que el jugador solicite las instrucciones y se evalúa como True si el jugador ingresa una cadena que comienza con 'y' o 'Y'. Por ejemplo:

```

input().lower().startswith('y')
  ↓
'Y'.lower().startswith('y')
  ↓
'y'.startswith('y')
  ↓
True

```

Si esta condición es verdadera, se llama a `showInstructions()` en la línea 163.

De lo contrario, el juego comienza.

Varias variables se configuran en las líneas 167 a 171; estos son descrito en la Tabla 13-1.

Tabla 13-1: Variables utilizadas en el bucle de juego principal

| Variable | Descripción |
|--------------------|---|
| sonarDevices | El número de dispositivos de sonda (y turnos) que le quedan al jugador. |
| el tablero | La estructura de datos del tablero utilizada para este juego. |
| losCofres | La lista de estructuras de datos de cofres. getRandomChests() devuelve una lista de tres cofres del tesoro en lugares aleatorios del tablero. |
| movidas anteriores | Una lista de todas las movidas x e y que el jugador ha hecho en el juego. |

Vamos a usar estas variables pronto, ¡así que asegúrese de revisar sus descripciones antes de continuar!

Visualización del estado del juego para el jugador

El bucle while de la línea 173 se ejecuta siempre que al jugador le queden dispositivos de sonda e imprime un mensaje que le dice cuántos dispositivos de sonda y cofres del tesoro quedan:

```

173. while sonarDevices > 0:
174.     # Mostrar el dispositivo de sonda y los estados del cofre.
175.     print('Te quedan %s dispositivo(s) de sonda. Quedan %s cofre(s) del
          tesoro.' % (sonarDevices, len(theCofres)))

```

Después de imprimir cuántos dispositivos quedan, el ciclo while sigue ejecutándose.

Manejo del movimiento del jugador

La línea 177 sigue siendo parte del ciclo while y usa múltiples

asignación para asignar las variables x e y a la lista de dos elementos que representan las coordenadas de movimiento del jugador devueltas por enterPlayerMove(). Pasaremos los movimientos anteriores para que el código de enterPlayerMove() pueda garantizar que el jugador no repita un movimiento anterior.

```
177.     x, y = enterPlayerMove(anteriorMoves)
178.     previousMoves.append([x, y]) # Debemos rastrear todos los movimientos para
        que los dispositivos de sonda puedan actualizarse.
179.
180.     moverResultado = hacerJugada (elTablero, losCofres, x, y)
181.     if moverResultado == Falso:
182.         Seguir
```

Las variables x e y se agregan al final de la lista de Jugadas anteriores . La variable anteriorMoves es una lista de coordenadas x e y de cada movimiento que hace el jugador en este juego. Esta lista se usa más adelante en el programa en las líneas 177 y 186.

Las variables x, y, elTablero y losCofres se pasan todas a la función hacerJugada() en la línea 180. Esta función realiza las modificaciones necesarias para colocar un dispositivo de sonda en el tablero.

Si hacerJugada() devuelve Falso, entonces hubo un problema con los valores de x e y que se le pasaron. La instrucción continuar envía la ejecución de regreso al inicio del ciclo while en la línea 173 para pedirle al jugador las coordenadas x e y nuevamente.

Encontrar un cofre del tesoro hundido Si hacerJugada() no devuelve Falso, devuelve una cadena de los resultados de esa jugada. Si esta cadena es '¡Has encontrado un cofre del tesoro hundido!', entonces todos los dispositivos de sonda en el tablero deberían actualizarse para detectar el siguiente cofre del tesoro más cercano en el tablero:

```

183.     más:
184.         if moveResult == '!Has encontrado un cofre del tesoro hundido!':
185.             # Actualice todos los dispositivos de sonda actualmente en el
186.             mapa. para x, y en jugadas anteriores:
187.                 hacerMovimiento(elTablero, losCofres, x, y)
188.                 dibujarTablero(elTablero) imprimir(resultadomovimiento)
189.

```

Las coordenadas x e y de todos los dispositivos de sonda están en movimientos anteriores. Al repetir Jugadas anteriores en la línea 186, puede pasar todas estas coordenadas x e y a la función hacerJugada() nuevamente para volver a dibujar los valores en el tablero. Debido a que el programa no imprime ningún texto nuevo aquí, el jugador no se da cuenta de que el programa está rehaciendo todos los movimientos anteriores. Simplemente parece que la placa se actualiza sola.

Comprobar si el jugador ganó

Recuerda que la función hacerJugada() modifica la lista de losCofres lo enviaste Debido a que theChests es una lista, cualquier cambio realizado dentro de la función persistirá después de que la ejecución regrese de la función. La función makeMove() elimina elementos de los cofres cuando se encuentran cofres del tesoro, por lo que eventualmente (si el jugador sigue adivinando correctamente) todos los cofres del tesoro habrán sido eliminados. (Recuerde, por "cofre del tesoro" nos referimos a las listas de dos elementos de las coordenadas x e y dentro de la lista theChests).

```

191.     if len(theCofres) == 0:
192.         print('!Has encontrado todos los cofres del tesoro hundidos!
193.             ¡Felicitaciones y buen juego!') romper

```

Cuando se hayan encontrado todos los cofres del tesoro en el tablero y eliminado de los Cofres, la lista de los Cofres tendrá una longitud

de 0. Cuando eso sucede, el código muestra un mensaje de felicitación al jugador y luego ejecuta una declaración de interrupción para salir de este bucle while . La ejecución se moverá entonces a la línea 197, la primera línea después del bloque while .

Comprobando si el jugador perdió

La línea 195 es la última línea del ciclo while que comenzó en la línea 173.

```
195.     sonDispositivos == 1
```

El programa reduce la variable sonarDevices porque el jugador ha usado un dispositivo de sonda. Si el jugador sigue perdiendo los cofres del tesoro, eventualmente sonarDevices se reducirá a 0. Después de esta línea, la ejecución vuelve a la línea 173 para que pueda reevaluar la condición de la instrucción while (que es sonarDevices > 0).

Si sonarDevices es 0, entonces la condición será Falsa y la ejecución continuará fuera del bloque while en la línea 197. Pero hasta entonces, la condición seguirá siendo Verdadera y el jugador podrá seguir adivinando:

```
197. si sonarDevices == 0:  
198.     print('¡Nos hemos quedado sin dispositivos de sonar! Ahora tenemos  
         que dar la vuelta al barco y dirigirnos') print('¡A casa con los cofres del  
199.     tesoro todavía por ahí! Fin del juego.') print(' Los cofres restantes  
         estaban aquí :) para x, y en los Cofres: print( %s, %s' % (x, y))  
200.  
201.  
202.
```

La línea 197 es la primera línea fuera del bucle while . Cuando la ejecución llega a este punto, el juego termina. Si sonarDevices es

0, sabes que el jugador se quedó sin dispositivos de sonda antes de encontrar todos los cofres y perdidos.

Las líneas 198 a 200 le dirán al jugador que ha perdido. El bucle for en la línea 201 recorrerá los cofres del tesoro que quedan en el Cofres y mostrará su ubicación para que el jugador pueda ver dónde estaban escondidos los cofres del tesoro.

Terminar el programa con sys.exit() Función

Gane o pierda, el programa le permite al jugador decidir si quiere seguir jugando. Si el jugador no ingresa 'sí' o 'Y' o ingresa alguna otra cadena que no comience con la letra y, entonces not input().lower().startswith('y') se evalúa como True y el sistema Se ejecuta la función .exit() . Esto hace que el programa termine.

```
204. print('¿Quieres volver a jugar? (sí o no)') 205. if  
not input().lower().startswith('y'): sys.exit()  
206.
```

De lo contrario, la ejecución vuelve al comienzo del ciclo while en la línea 165 y comienza un nuevo juego.

RESUMEN

¿Recuerdas cómo nuestro juego de Tic-Tac-Toe numeró los espacios en el tablero de Tic-Tac-Toe del 1 al 9? Este tipo de sistema de coordenadas podría haber estado bien para un tablero con menos de 10 espacios. ¡Pero el tablero Sonar Treasure Hunt tiene 900 espacios!

El sistema de coordenadas cartesianas que aprendimos en el Capítulo 12 realmente hace que todos estos espacios sean manejables, especialmente cuando nuestro juego necesita encontrar la distancia entre dos puntos en el tablero.

Las ubicaciones en los juegos que usan un sistema de coordenadas cartesianas se pueden almacenar en una lista de listas en las que el primer índice es la x coordenada y el segundo índice es la coordenada y. Esto facilita el acceso a una coordenada usando tablero[x][y].

Estas estructuras de datos (como las que se usan para las ubicaciones del océano y el cofre del tesoro) hacen posible representar conceptos complejos como datos, y sus programas de juego se dedican principalmente a modificar estas estructuras de datos.

En el próximo capítulo, representaremos las letras como números. Al representar el texto como números, podemos realizar operaciones matemáticas con ellos para cifrar o descifrar mensajes secretos.

14

CIFRA CÉSAR



El programa de este capítulo no es realmente un juego, pero no obstante es divertido. El programa convertirá el inglés normal en un código secreto. También puede convertir códigos secretos al inglés normal. Solo alguien que conozca la clave de los códigos secretos podrá entender los mensajes.

Debido a que este programa manipula texto para convertirlo en mensajes secretos, aprenderá varias funciones y métodos nuevos para manipular cadenas. También aprenderá cómo los programas pueden hacer operaciones matemáticas con cadenas de texto tal como lo hacen con números.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Criptografía y cifrados
- Texto cifrado, texto sin formato, claves y símbolos
- Cifrado y descifrado
- El cifrado César
- El método de cadena `find()`
- Criptoanálisis
- La técnica de la fuerza bruta

CRIPTOGRAFÍA Y CIFRADO

La ciencia de escribir códigos secretos se llama criptografía. Durante miles de años, la criptografía ha hecho posible enviar mensajes secretos que solo el remitente y el destinatario pueden leer, incluso si alguien capturó al mensajero y leyó el mensaje codificado. Un sistema de código secreto se llama cifrado. El cifrado utilizado por el programa en este capítulo se denomina cifrado César.

En criptografía, llamamos texto plano al mensaje que queremos mantener en secreto. Digamos que tenemos un mensaje de texto sin formato que se ve así:

Hay una pista detrás de la estantería.

Convertir el texto sin formato en el mensaje codificado se denomina cifrar el texto sin formato. El texto plano se cifra en el texto cifrado. El texto cifrado parece letras aleatorias, por lo que no podemos entender cuál era el texto sin formato original con solo mirar el texto cifrado. Aquí está el ejemplo anterior encriptado en texto cifrado:

aolyl pz h jsBl ilopuk AOL ivvrzolsm.

Si conoce el cifrado utilizado para cifrar el mensaje, puede descifrar el texto cifrado y convertirlo en texto sin formato. (El descifrado es lo opuesto al cifrado).

Muchos sistemas de cifrado utilizan claves, que son valores secretos que le permiten descifrar el texto cifrado que se ha cifrado con un cifrado específico. Piense en el cifrado como si fuera una cerradura de puerta. Solo puedes desbloquearlo con una llave en particular.

Si está interesado en escribir programas de criptografía, puede leer mi libro Hacking Secret Ciphers with Python. Es gratis para descargar desde <http://inventwithpython.com/hacking/>.

CÓMO LA CIFRA CÉSAR OBRAS

El cifrado César fue uno de los primeros cifrados jamás inventados. En este cifrado, cifra un mensaje reemplazando cada letra con una letra "cambiada". En criptografía, las letras cifradas se denominan símbolos porque pueden ser letras, números o cualquier otro signo. Si desplaza la letra A un espacio, obtiene la letra B. Si desplaza la letra A dos espacios, obtiene la letra C. La figura 14-1 muestra algunas letras desplazadas tres espacios.

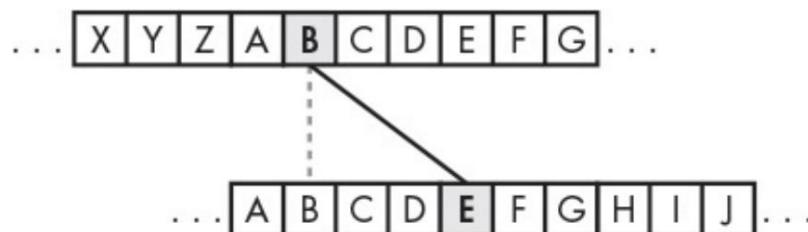


Figura 14-1: Un cifrado César desplazando letras tres espacios. Aquí, B se convierte en E.

Para obtener cada letra desplazada, dibuja una fila de cuadros con cada letra del alfabeto. Luego dibuje una segunda fila de cuadros debajo, pero comience sus letras con un cierto número de espacios por encima. Cuando llegue al final del alfabeto de texto sin formato, vuelva a A. La figura 14-2 muestra un ejemplo con las letras desplazadas tres espacios.

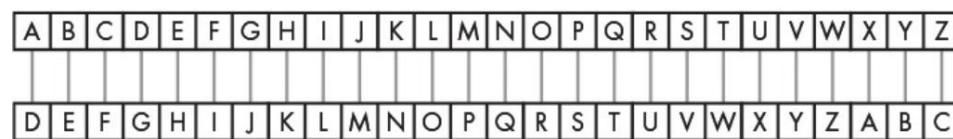


Figura 14-2: Todo el alfabeto desplazado tres espacios

El número de espacios en los que desplazas tus letras (entre 1 y 26) es la clave en el cifrado César. A menos que conozca la clave (el número utilizado para cifrar el mensaje), no podrá descifrar el código secreto. El ejemplo de la figura 14-2 muestra las traducciones de letras para la tecla 3.

NOTA

Si bien hay 26 claves posibles, cifrar su mensaje con 26 dará como resultado un texto cifrado que es exactamente igual que el texto sin formato.

Si cifra la palabra de texto sin formato HOWDY con una clave de 3, después:

- La letra H se convierte en K.
- La letra O se convierte en R.
- La letra W se convierte en Z.
- La letra D se convierte en G.
- La letra Y se convierte en B.

Entonces, el texto cifrado de HOWDY con la clave 3 se convierte en KRZGB. Para descifrar KRZGB con la clave 3, vamos de los cuadros inferiores a los superiores.

Si desea incluir letras minúsculas en lugar de letras mayúsculas, agregue otras 26 casillas a las que ya tiene y rellénelas con las 26 letras minúsculas.

Ahora, con una clave de 3, la letra Y se convierte en b, como se muestra en la figura 14-3.

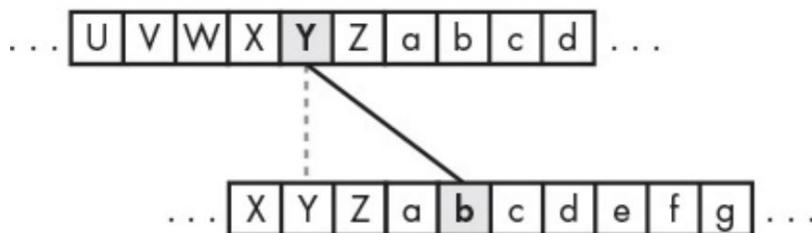


Figura 14-3: Todo el alfabeto, que ahora incluye letras minúsculas, desplazado tres veces
espacios

El cifrado funciona de la misma manera que lo hizo con solo letras mayúsculas. De hecho, si desea utilizar letras del alfabeto de otro idioma, puede escribir cuadros con esas letras para crear su cifrado.

EJECUCIÓN DE MUESTRA DE CIFRO CÉSAR

Aquí hay una ejecución de muestra del cifrado del programa Caesar Cipher un mensaje:

¿Desea cifrar o descifrar un mensaje? cifrar Ingrese su
mensaje: El cielo sobre el puerto era del color de la
televisión, sintonizado en un canal muerto.

Introduzca el número de clave (1-52)

13

Su texto traducido es:

gur FxL noBIR Gur CBEG JnF Gur pByBE Bs GryrlvFvBA, GHArq GB n qrnq punAAry.

Ahora ejecuta el programa y descifra el texto que acabas de cifrar:

¿Desea cifrar o descifrar un mensaje? descifrar Introduzca
su mensaje: gur FxL noBIR Gur CBEG JnF Gur pByBE Bs
GryrlvFvBA, GHArq GB n qrnq punAAry.

Introduzca el número de clave (1-52)

13

Su texto traducido es:

El cielo sobre el puerto era del color de la televisión, sintonizado en un canal muerto.

Si no descifra con la clave correcta, el texto no se descifrará correctamente:

¿Desea cifrar o descifrar un mensaje? descifrar
Introduzca su mensaje: gur FxL noBIr Gur CBEG
JnF Gur pByBE Bs GryrlvFvBA, GHArq GB n
qrnq punAAry.

Introduzca el número de clave (1-52)

15

Su texto traducido es:

Rfc qiw YZmtc rfc nmpr uYq rfc amjmp md rcjctgqqml, rslcb rm Y bcYb afYllcj.

CÓDIGO FUENTE PARA CAESAR CIFRAR

Ingresé este código fuente para el programa Caesar Cipher y luego guarde el archivo como cipher.py.

Si obtiene errores después de ingresar este código, compare el código que escribió con el código del libro con la herramienta de comparación en línea en <https://www.nostarch.com/inventwithpython#diff>.



cifrado.py

```
1. # Cifrado César
2. SÍMBOLOS =
'ABCDEFGHIJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'
3. MAX_KEY_SIZE = len(SÍMBOLOS) 4.

5. def getMode():
6.     mientras sea verdadero:
7.         print('¿Desea encriptar o desencriptar un mensaje?') mode
8.         = input().lower() if mode in ['encrypt', 'e', 'decrypt', 'd']:
9.             return mode
10
11     más:
12     print('Ingrese "cifrar" o "e" o "descifrar" o "d".')
13

14. def getMessage():
15.     print('Ingrese su mensaje:') 16.
return input() 17.

18. def getKey():
19.     tecla = 0 20.
while True:
21     print('Ingrese el número de clave (1-%s)' % (MAX_KEY_SIZE))
22     key = int(input()) if (key >= 1 and key <= MAX_KEY_SIZE):
23
```

```

24      tecla de retorno
25
26. def getTranslatedMessage(modo, mensaje, clave): 27. if
modo[0] == 'd':
28      clave = -clave
29. traducido =
30
31. para el símbolo en el mensaje:
32.     símboloÍndice = SÍMBOLOS.buscar(símbolo) si símboloÍndice ==
33.         -1: # Símbolo no encontrado en SÍMBOLOS.
34.         # Simplemente agregue este símbolo sin ningún
35.         cambio. traducido += símbolo más:
36.
37.         # Cifrar o descifrar.
38.         símboloÍndice += clave
39.
40.     if símboloÍndice >= len(SÍMBOLOS):
41.         símboloÍndice -= len(SÍMBOLOS) elif
42.         símboloÍndice < 0: símboloÍndice +=
43.             len(SÍMBOLOS)
44.
45.     traducido += SÍMBOLOS[índicesímbolo] 46.
volver traducido
47.
48. mode = getMode() 49.
message = getMessage() 50. key
= getKey() 51. print('Tu texto
traducido es:') 52.
print(getTranslatedMessage(mode, message, key))

```

CONFIGURACIÓN DE LA CLAVE MÁXIMA LARGO

Los procesos de cifrado y descifrado son inversos entre sí, pero comparten gran parte del mismo código. Veamos cómo funciona cada línea:

1. #Cifrado César

2. SÍMBOLOS =

'ABCDEFGHIJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'

3. MAX_KEY_SIZE = len(SÍMBOLOS)

MAX_KEY_SIZE es una constante que almacena la longitud de SYMBOLS (52). Esta constante nos recuerda que en este programa, la clave utilizada en el cifrado debe estar siempre entre 1 y 52.

DECIDIR CIFRAR O DESCIFRAR EL MENSAJE

La función getMode() le permite al usuario decidir si quiere usar el modo de cifrado o descifrado del programa:

```
5. def getMode():
6.     while True:
7.         print("¿Desea cifrar o descifrar un mensaje?") mode =
8.         input().lower() if mode in ['encrypt', 'e', 'decrypt', 'd']:
9.
10        modo de retorno
11    más:
12    print("Ingrese \"cifrar\" o \"e\" o \"descifrar\" o \"d\".")
```

La línea 8 llama a input() para permitir que el usuario ingrese el modo que desea. Luego se llama al método lower() en esta cadena para devolver una versión en minúsculas de la cadena . El valor devuelto por input().lower() se almacena en modo. La condición de la declaración if verifica si la cadena almacenada en modo existe en la lista ['encrypt', 'e', 'decrypt', 'd'] .

Esta función devolverá la cadena en modo siempre que el modo sea igual a 'encriptar', 'e', 'desencriptar' o 'd'. Por lo tanto, getMode() devolverá el modo de cadena. Si el usuario escribe algo que no es 'cifrar', 'e', 'descifrar' o 'd', entonces el ciclo while le preguntará nuevamente.

OBTENER EL MENSAJE DE EL JUGADOR

La función getMessage() simplemente obtiene el mensaje para cifrar o descifrar del usuario y lo devuelve:

```
14. def getMessage():
15.     print('Ingrese su mensaje:')
16.
return input()
```

La llamada a input() se combina con return para que usemos sólo una línea en lugar de dos.

OBTENER LA LLAVE DEL JUGADOR

La función getKey() le permite al jugador ingresar la clave que usará para cifrar o descifrar el mensaje:

```
18. def getKey():
19.     tecla = 0
20.
while True:
21     print('Ingrese el número de clave (1-%s)' % (MAX_KEY_SIZE))
22     clave = int(input()) if (clave >= 1 and clave <= MAX_KEY_SIZE):
23     return clave
24
```

El ciclo while asegura que la función siga repitiéndose hasta que el usuario ingrese una clave válida. Una clave válida aquí es una entre los valores enteros 1 y 52 (recuerde que MAX_KEY_SIZE es 52 porque hay 52 caracteres en la variable SYMBOLS). La función getKey() luego devuelve esta clave. La línea 22 establece key en la versión entera de lo que el usuario ingresó, por lo que getKey() devuelve un número entero.

CIFRADO O DESCIFRADO EL MENSAJE

La función getTranslatedMessage() hace el cifrado y descifrado real:

```
26. def getTranslatedMessage(modo, mensaje,
clave): 27. if modo[0] == 'd':
28     clave =
-clave 29. traducida =
```

Tiene tres parámetros:

modo Establece la función en modo de cifrado o modo de descifrado.

mensaje Este es el texto sin formato (o texto cifrado) que se va a cifrar (o descifrar).

clave Esta es la clave que se utiliza en este cifrado.

La línea 27 verifica si la primera letra en la variable de modo es la cadena 'd'. Si es así, entonces el programa está en modo de descifrado. La única diferencia entre el modo de descifrado y cifrado es que en el modo de descifrado, la clave se establece en la versión negativa de sí misma. Por ejemplo, si la clave es el número entero 22, el modo de descifrado lo establece en -22. El motivo se explica en “Cifrado o descifrado de cada letra” en la página 205.

La variable traducida contendrá la cadena del resultado: ya sea el texto cifrado (si está cifrando) o el texto sin formato (si está descifrando). Comienza como una cadena en blanco y tiene caracteres cifrados o descifrados concatenados hasta el final. Sin embargo, antes de que podamos comenzar a concatenar los caracteres para traducir, debemos cifrar o descifrar el texto, lo que haremos

hacer en el resto de `getTranslatedMessage()`.

Encontrar cadenas pasadas con la cadena `find()` Método

Para cambiar las letras para realizar el cifrado o descifrado, primero debemos convertirlas en números. El número de cada letra en la cadena `SYMBOLS` será el índice donde aparece. Como la letra A está en `SÍMBOLOS[0]`, el número 0 representará la A mayúscula. Si quisieramos encriptar esto con la clave 3, simplemente usaríamos $0 + 3$ para obtener el índice de la letra encriptada: `SÍMBOLOS[3]` o 'D'.

Usaremos el método de cadena `find()` , que encuentra la primera aparición de una cadena pasada en la cadena en la que se llama al método. Ingrese lo siguiente en el shell interactivo:

```
>>> '¡Hola mundo!'.find('H')
0
>>> '¡Hola mundo!'.find('o')
4
>>> '¡Hola mundo!'.find('ell')
1
```

'¡Hola mundo!'.`find('H')` devuelve 0 porque la 'H' se encuentra en el primer índice de la cadena '¡Hola mundo!'. Recuerde, los índices comienzan en 0, no en 1. El código '¡Hola mundo!'.`find('o')` devuelve 4 porque la 'o' minúscula se encuentra primero al final de 'Hola'. El método `find()` deja de buscar la primera aparición, por lo que la segunda 'o' en 'mundo' no importa. También puede encontrar cadenas con más de un carácter. La cadena 'ell' se encuentra comenzando en el índice 1.

Si no se puede encontrar la cadena pasada, el método `find()` devuelve -1:

```
>>> '¡Hola mundo!'.find('xyz')
-1
```

Volvamos al programa Caesar Cipher. La línea 31 es una bucle for que itera en cada carácter de la cadena del mensaje :

-
31. para el símbolo en el mensaje:
 32. símboloÍndice = SÍMBOLOS.buscar(símbolo) si
 33. símboloÍndice == -1: # Símbolo no encontrado en SÍMBOLOS.
 34. # Simplemente agregue este símbolo sin ningún cambio. traducido += símbolo
-

El método find() se usa en la línea 32 para obtener el índice de la cadena en el símbolo. Si find() devuelve -1, el carácter en el símbolo simplemente se agregará a la traducción sin ningún cambio. Esto significa que no se cambiarán los caracteres que no formen parte del alfabeto, como las comas y los puntos.

Cifrar o descifrar cada letra Una vez que haya encontrado el número de índice de una letra, agregar la clave al número realizará el cambio y le dará el índice de la letra cifrada.

La línea 38 hace esta adición para obtener la letra cifrada (o descifrada).

-
37. # Cifrar o descifrar.
 38. símboloÍndice += clave
-

Recuerde que en la línea 28, hicimos que el número entero en la clave fuera negativo para el descifrado. El código que suma la clave ahora la restará, ya que sumar un número negativo es lo mismo que restar.

Sin embargo, si esta suma (o resta, si la clave es negativa)

hace que symbolIndex supere el último índice de SYMBOLS, tendremos que volver al principio de la lista en 0. Esto lo maneja la declaración if que comienza en la línea 40:

```

40      if símboloÍndice >= len(SÍMBOLOS):
41          símboloÍndice -= len(SÍMBOLOS)
42.      elif símboloÍndice < 0: símboloÍndice
43.          += len(SÍMBOLOS)
44.
45.      traducido += SÍMBOLOS[índicesímbolo]

```

La línea 40 comprueba si el índice del símbolo ha superado el último índice comparándolo con la longitud de la cadena SÍMBOLOS . Si es así, la línea 41 resta la longitud de SÍMBOLOS de símboloÍndice. Si symbolIndex ahora es negativo, entonces el índice debe ajustarse al otro lado de la cadena SYMBOLS . La línea 42 verifica si el valor de symbolIndex es negativo después de agregar la clave de descifrado. Si es así, la línea 43 suma la longitud de SÍMBOLOS a símboloÍndice.

La variable symbolIndex ahora contiene el índice del símbolo correctamente cifrado o descifrado. SYMBOLS[símboloÍndice] apuntará al carácter de este índice, y este carácter se agrega al final de traducido en la línea 45.

La ejecución regresa a la línea 31 para repetir esto para el siguiente carácter en el mensaje. Una vez que se completa el bucle, la función devuelve la cadena cifrada (o descifrada) traducida en línea

46:

46. volver traducido

La última línea de la función getTranslatedMessage() devuelve el cadena traducida .

INICIO DEL PROGRAMA

El inicio del programa llama a cada una de las tres funciones definidas previamente para obtener el modo, el mensaje y la clave del usuario:

```
48. mode = getMode()
49. message = getMessage()
50. key = getKey() 51.
print('Tu texto traducido es:') 52.
print(getTranslatedMessage(mode, message, key))
```

Estos tres valores se pasan a `getTranslatedMessage()`, cuyo el valor de retorno (la cadena traducida) se imprime al usuario.

EXPANDIENDO LOS SÍMBOLOS

Si desea encriptar números, espacios y signos de puntuación, simplemente agréguelos a la cadena SÍMBOLOS en la línea 2. Por ejemplo, puede hacer que su programa de cifrado encripte números, espacios y signos de puntuación cambiando la línea 2 a lo siguiente:

```
2. SÍMBOLOS = 'ABCDEFGHIJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz 123
```

```
4567890!@#$%^&*'()
```

Tenga en cuenta que la cadena SYMBOLS tiene un carácter de espacio después de la z minúscula .

Si quisiera, podría agregar aún más personajes a esta lista. Y no necesita cambiar el resto de su programa, ya que todas las líneas de código que necesitan la lista de caracteres solo usan la constante SÍMBOLOS .

Solo asegúrese de que cada carácter aparezca solo una vez en la cadena. Además, deberá descifrar su mensaje con la misma cadena de SÍMBOLOS con la que se cifró.

LA TÉCNICA DE LA FUERZA BRUTA

Ese es todo el cifrado César. Sin embargo, si bien este cifrado puede engañar a algunas personas que no entienden la criptografía, no mantendrá un mensaje en secreto para alguien que sepa criptoanálisis. Mientras que la criptografía es la ciencia de hacer

códigos, el criptoanálisis es la ciencia de descifrar códigos.

El objetivo de la criptografía es asegurarse de que si alguien más tiene en sus manos el mensaje cifrado, no pueda descifrar el texto original. Supongamos que somos el descifrador de códigos y todo lo que tenemos es este texto encriptado:

LwCjBA uiG vwB jm xtmiAivB, jCB kmzBiqvBG qA ijACzl.

La fuerza bruta es la técnica de probar todas las claves posibles hasta encontrar la correcta. Debido a que solo hay 52 claves posibles, sería fácil para un criptoanalista escribir un programa de piratería que descifre con todas las claves posibles. Luego, podrían buscar la clave que descifra en inglés simple.

Agreguemos una función de fuerza bruta al programa.

AGREGAR LA FUERZA BRUTA MODO

Primero, cambie las líneas 7, 9 y 12, que se encuentran en la función `getMode()`, para que se vean como las siguientes (los cambios están en negrita):

```
5. def getMode():
6.     mientras sea verdadero:
7.         print('¿Desea cifrar o descifrar o aplicar fuerza bruta a un
8.             mensaje?') mode = input().lower() if mode in ['encrypt', 'e',
9.                 'decrypt', 'd', 'brute ', 'b']: modo de retorno
10
11     más:
12         print('Ingrese "cifrar" o "e" o "descifrar" o "d" o "bruto" o "b".')
```

Este código permitirá al usuario seleccionar la fuerza bruta como modo.

A continuación, realice los siguientes cambios en la parte principal del

programa:

```

48. mode = getMode()
49. message = getMessage()
50. if mode[0] != 'b': 51. key
= getKey() 52. print('Tu texto
traducido es:') 53. if mode [0] !=
'b': 54.
print(getTranslatedMessage(modo, mensaje, clave)) 55. else:

56. for key in range(1, MAX_KEY_SIZE + 1): 57.
print(key, getTranslatedMessage('descifrar', mensaje,
clave))

```

Si el usuario no está en modo de fuerza bruta, se le solicita una clave, se realiza la llamada original a `getTranslatedMessage()` y se imprime la cadena traducida.

Sin embargo, si el usuario está en modo de fuerza bruta, entonces el bucle `getTranslatedMessage()` itera desde 1 hasta `MAX_KEY_SIZE` (que es 52). Recuerde que la función `range()` devuelve una lista de enteros hasta el segundo parámetro, pero sin incluirlo, razón por la cual sumamos + 1. El programa luego imprimirá todas las traducciones posibles del mensaje (incluido el número de clave utilizado en el Traducción). Aquí hay una muestra de ejecución de este programa modificado:

¿Desea cifrar o descifrar o fuerza bruta un mensaje? bruto

Ingrrese su mensaje:

LwCjBA uiG vwB jm xtmiAivB, jCB kmzBiqvBG qA ijACzl.

Su texto traducido es:

- 1 KvBiAz thF uvA il wslhzhuA, iBA jlyAhpuAF pz hizByk.
- 2 JuAhzy sgE tuz hk vrkggygtz, haz ikxzgotzE oy ghyAxj.
- 3 Itzgyx rfD orzuelo gj uqjfxfsy, gzy hjwyfnsyD nx fgxzwi.
- 4 Hsyfxw qeC rsx fi tpiewerx, fyx givxemrxC mw efwyvh.
- 5 Grxewv pdB qrw eh sohdvdqw, exw fhuwdlqwB lv devxug.
- 6 Fqwdvu ocA pqv dg rngcucpv, dwv egtvckpvA ku cduwtf.

7 Epvcut nbz opu cf qmfbtbou, cvu dfsubjouz jt bctvse.

8 Las dudas pueden no ser agradables, pero la certeza es absurda.

9 Cntasr lZx mns ad okdZrZms, ats bdqsZhmsx hr Zartqc.

10 BmsZrq kYw lmr Zc njcYqYlr, Zsr acprYglrw gq YZqsph.

11 AlrYqp jXv klq Yb mibXpXkq, Yrq ZboqXfkqv fp XYproa. 12 zkqXpo iWu jkp

Del WoWjp, Xqp YanpWejpu eo WXoqnZ. --recorte--

Después de revisar cada fila, puede ver que el octavo mensaje no es una tontería sino un lenguaje sencillo. El criptoanalista puede deducir que la clave original para este texto encriptado debe haber sido 8. Este método de fuerza bruta habría sido

difícil de hacer en los días de Julio César y el Imperio Romano, pero hoy en día tenemos computadoras que pueden pasar rápidamente por millones o incluso miles de millones de claves en poco tiempo.

RESUMEN

Las computadoras son buenas para hacer matemáticas. Cuando creamos un sistema para traducir cierta información en números (como hacemos con texto y ordinales o con sistemas espaciales y de coordenadas), los programas de computadora pueden procesar estos números de manera rápida y eficiente. Una gran parte de escribir un programa es descubrir cómo representar la información que desea manipular como valores que Python pueda entender.

Si bien nuestro programa Caesar Cipher puede cifrar mensajes que los mantendrá en secreto para las personas que tienen que descifrarlos con lápiz y papel, el programa no los mantendrá en secreto para las personas que saben cómo hacer que las computadoras procesen la información. (Nuestro modo de fuerza bruta lo demuestra).

En el Capítulo 15, crearemos Reversegam (también conocido como Reversi u Othello). La IA que juega este juego es mucho más

avanzado que la IA que jugó Tic-Tac-Toe en el Capítulo 10.

De hecho, ¡es tan bueno que la mayoría de las veces no podrás superarlo!

15

EL JUEGO REVERSEGAM



En este capítulo, haremos Reversegam, también conocido como Reversi u Othello. Este juego de mesa para dos jugadores se juega en una cuadrícula, por lo que usaremos un sistema de coordenadas cartesianas con coordenadas x e y. Nuestra versión del juego tendrá una IA de computadora que es más avanzada que nuestra IA de Tic-Tac-Toe del Capítulo 10. De hecho, esta IA es tan buena que probablemente te ganará casi cada vez que juegues. (¡Pierdo cada vez que juego contra él!)

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Cómo jugar Reversegam
- La función `bool()`
- Simulación de movimientos en un tablero de Reversegam
- Programación de una IA de Reversegam

CÓMO JUGAR REVERSEGAM

Reversegam tiene un tablero de 8×8 y mosaicos que son negros por un lado y blancos por el otro (nuestro juego usará Os y Xs

en cambio). El tablero inicial se parece a la Figura 15-1.

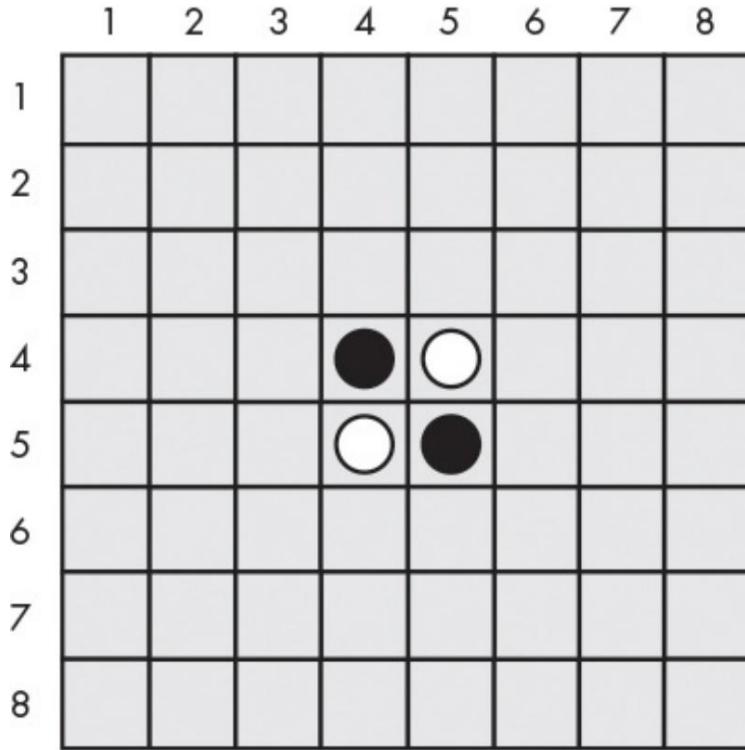


Figura 15-1: El tablero inicial de Reversegam tiene dos mosaicos blancos y dos mosaicos negros.

Dos jugadores se turnan para colocar fichas de su color elegido, negro o blanco, en el tablero. Cuando un jugador coloca una ficha en el tablero, cualquiera de las fichas del oponente que esté entre la nueva ficha y las otras fichas del color del jugador se voltean. Por ejemplo, cuando el jugador blanco coloca una nueva ficha blanca en el espacio (5, 6), como en la Figura 15-2, la ficha negra en (5, 5) está entre dos fichas blancas, por lo que cambiará a blanca, como en la figura 15-3. El objetivo del juego es terminar con más fichas de tu color que del color de tu oponente.

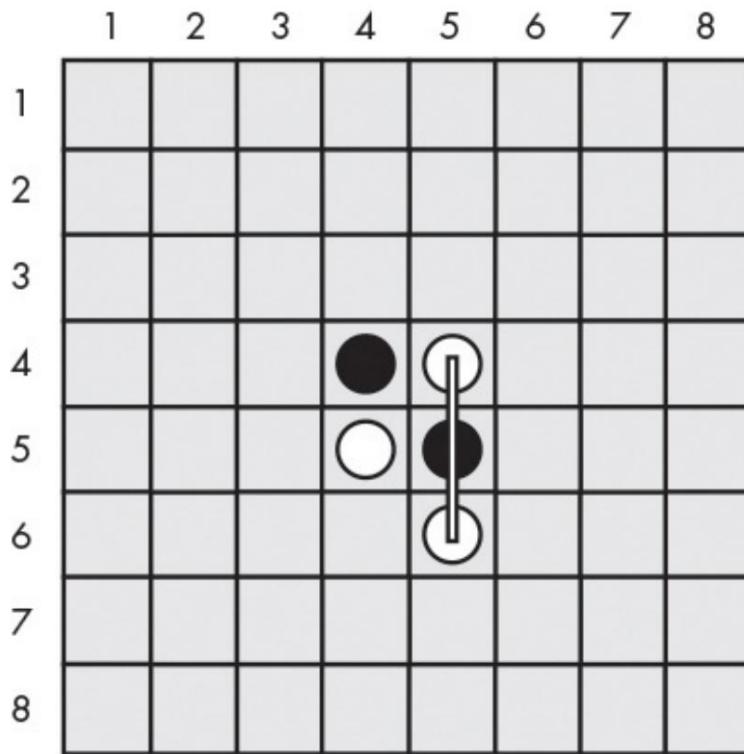


Figura 15-2: Las blancas colocan una nueva ficha.

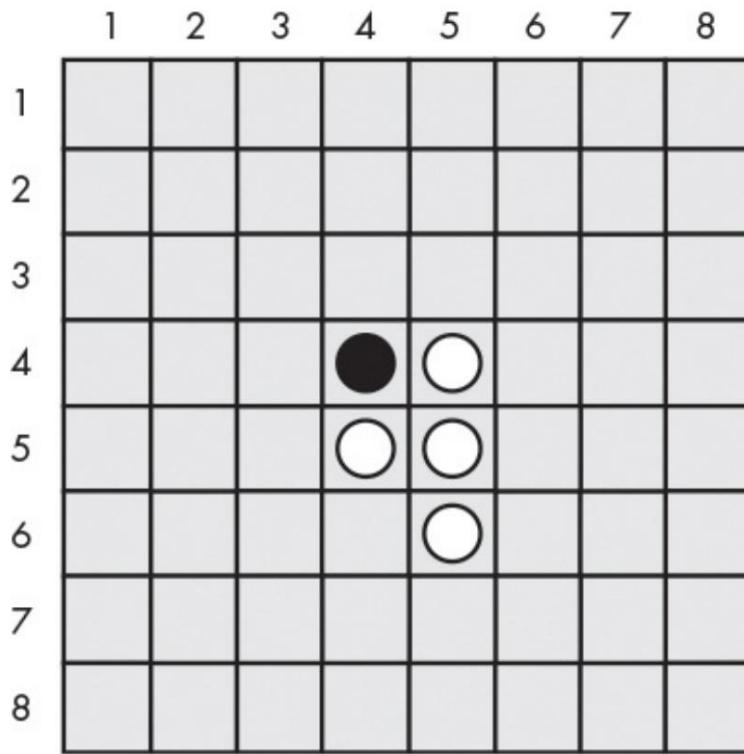


Figura 15-3: El movimiento de las blancas ha provocado que una de las fichas de las negras se voltee.

El negro podría hacer un movimiento similar a continuación, colocando una ficha negra

en (4, 6), lo que daría la vuelta a la ficha blanca en (4, 5). Esto da como resultado un tablero que se parece a la Figura 15-4.

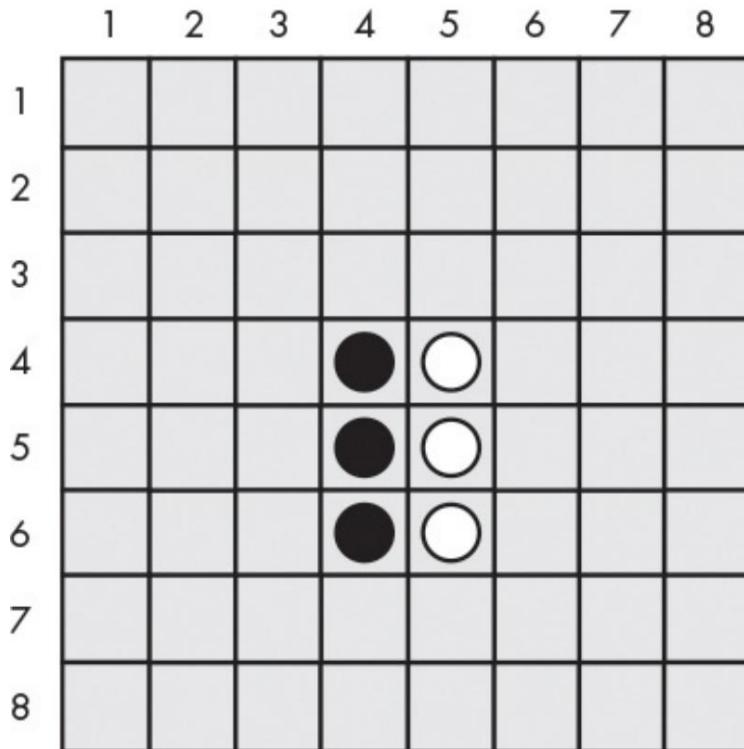


Figura 15-4: Las negras han colocado una nueva ficha, volteando una de las fichas de las blancas.

Las fichas en todas las direcciones se voltean siempre que estén entre la nueva ficha del jugador y una ficha existente de ese color.

En la Figura 15-5, las blancas colocan una ficha en (3, 6) y voltea las fichas negras en dos direcciones (marcadas por las líneas). El resultado se muestra en la figura 15-6.

Cada jugador puede voltear rápidamente muchas fichas en el tablero en uno o dos movimientos. Los jugadores siempre deben hacer un movimiento que voltee al menos una ficha. El juego termina cuando un jugador no puede hacer ningún movimiento o el tablero está completamente lleno. Gana el jugador que tenga más fichas de su color.

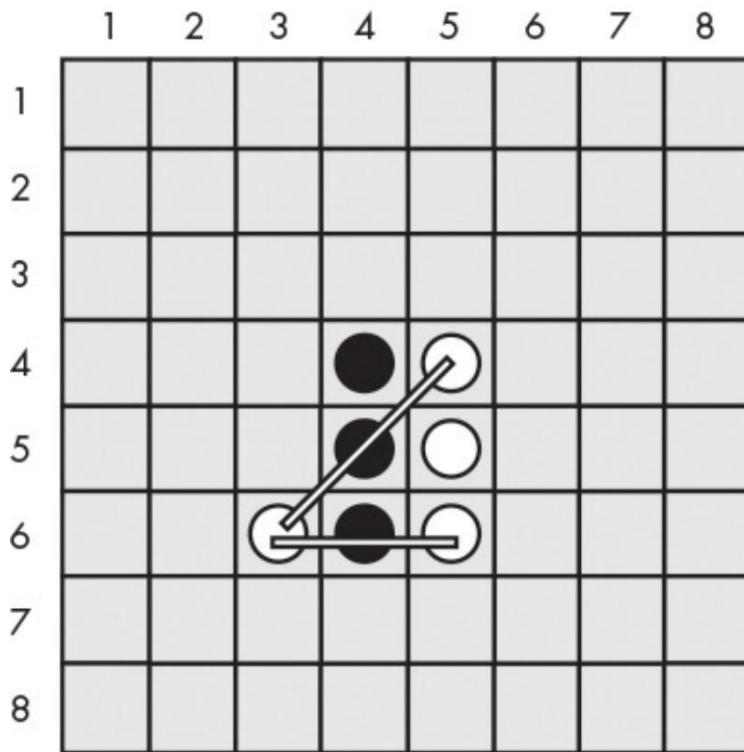


Figura 15-5: El segundo movimiento de las blancas en (3, 6) volteará dos de las fichas de las negras.

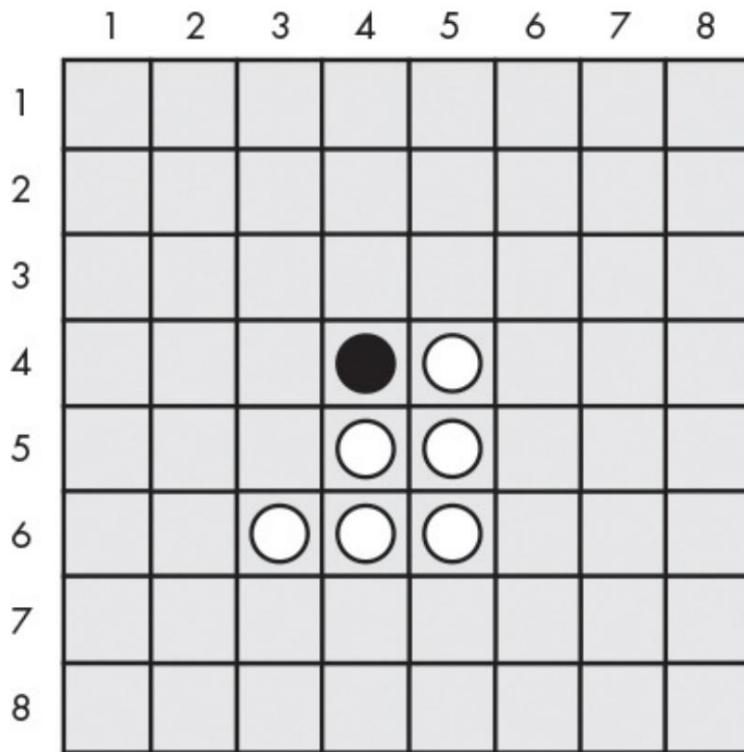


Figura 15-6: El tablero después del segundo movimiento de las blancas.

La IA que fabricamos para este juego buscará cualquier rincón

movimientos en el tablero que puede tomar. Si no hay movimientos de esquina disponible, la computadora seleccionará el movimiento que reclame la mayor cantidad de fichas.

EJECUCIÓN DE MUESTRA DE REVERSEGAM

Esto es lo que ve el usuario cuando ejecuta el programa Reversegam. El texto ingresado por el jugador está en negrita.

¡Bienvenido a Reversegam!

¿Quieres ser X u O?

X

El jugador irá primero.

12345678

+-----+

|1|1 2|2 3|3|

4| XO |4

5| BUEY |5 6|

|6 7| |7 8| |8|

+-----+

12345678

Tú: 2 puntos. Informática: 2 puntos.

Ingrese su movimiento, "salir" para finalizar el juego o "pistas" para cambiar las pistas.

53

12345678

+-----+

|1|1 2|2|

3| x |3

4| XX |4

5| BUEY |5 6|

|6 7| |7 8| |8|

+-----+

12345678

Tú: 4 puntos. Informática: 1 puntos.

Presiona Enter para ver el movimiento de la computadora.

--recorte--

12345678

+-----+

1|00000000|1

2|0XXX0000|2

3|OXOO0000|3

4|OXXOXXOX|4

5|OXOOXOX|5

6|OXXXXOOX|6

7|OOXXOO00|7

8|00000000|8

+-----+

12345678

X anotó 21 puntos. O anotó 43 puntos.

Perdiste. La computadora te ganó por 22 puntos.

¿Quieres jugar de nuevo? (sí o no)

no

Como puede ver, la IA fue bastante buena al vencerme, 43 a 21.

Para ayudar al jugador, programaremos el juego para que proporcione pistas. El jugador puede escribir pistas como su movimiento, lo que activará y desactivará el modo de pistas. Cuando el modo de pistas está activado, todos los movimientos posibles que el jugador puede hacer aparecerán en el tablero como puntos (.), así:

12345678

+-----+

1| |1 2| . |

2 3| XQ|.

3 4| XOX |

4 5| OO |5

6| . . |6 7|

|7 8| 8

+-----+

12345678

Como puede ver, el jugador puede avanzar (4, 2), (5, 3), (4, 6), o (6, 6) según las sugerencias que se muestran en este tablero.

CÓDIGO FUENTE PARA REVERSEGAM

Reversegam es un programa gigantesco en comparación con nuestros juegos anteriores. ¡Tiene casi 300 líneas! Pero no te preocupes: muchos de estos son comentarios o líneas en blanco para espaciar el código y hacerlo más legible.



Al igual que con nuestros otros programas, primero crearemos varias funciones para llevar a cabo tareas relacionadas con Reversegam que llamará la sección principal. Aproximadamente las primeras 250 líneas de código son para estas funciones auxiliares, y las últimas 30 líneas de código implementan el juego Reversegam en sí.

Si obtiene errores después de ingresar este código, compare su

código al código del libro con la herramienta de diferencias en línea en

<https://www.nostarch.com/inventwithpython#diff>.

inversa.py

1. # Reversegam: un clon de Othello/Reversi 2.

importar aleatoriamente 3. importar sys 4. ANCHO =

8 # El tablero tiene 8 espacios de ancho.

5. ALTURA = 8 # El tablero tiene 8 espacios de

altura. 6. def dibujarTablero(tablero): 7. #

Imprime el tablero pasado a esta función. Devolver Ninguno. 8.

imprimir(' 12345678') 9. imprimir(' +-----+') 10. for y en rango(ALTURA):

11. imprimir("%s| % (y+1) , end="") para x en el rango (ANCHO):

12

13 imprimir(tablero[x][y], fin="")

14 imprimir('|%s' % (y+1)) imprimir('

15. +-----+') imprimir(' 12345678')

decidíos.

17

18. def getNewBoard(): 19.

Crear una nueva estructura de datos de tablero en blanco. 20.

tablero = [] 21. for i in range(ANCHO): 22. tablero.append([' ', ' ',

' , ' , ' , ' , ' , ' , '])

23. tabla de retorno

24

25. def esJugadaVálida(tablero, mosaico, xstart, ystart):

26. # Devuelve Falso si la jugada del jugador en el espacio xstart, ystart es
inválido.

27. # Si es un movimiento válido, devuelve una lista de espacios que se convertirían

en el jugador si hiciera un movimiento aquí. 28. si tablero[iniciox][inicioy] != '

' o no está A Bordo(iniciox, inicioy):

29 falso retorno

30

31. si mosaico == 'X':

32. otraBaldosa = 'O'

33. más:

```

34.     otraBaldosa = 'X'
35.
36. tilesToFlip = [] 37. for
    xdirección, ydirección en [[0, 1], [1, 1], [1, 0], [1, -1],
        [0, -1], [-1, -1], [-1, 0], [-1, 1]]:
38.     x, y = xstart, ystart x +==
39.     xdirección # Primer paso en la dirección x y +== ydirección
40.     # Primer paso en la dirección y while isOnBoard(x, y)
41.     and board[x][y] == otherTile: # Keep moviéndose en esta dirección
42.         x e y. x +== dirección x
43.
44.         y +== dirección y si
45.             está en el tablero (x, y) y tablero [x] [y] == mosaico:
46.                 # Hay piezas para voltear. Ve en dirección contraria hasta
                    llegar al espacio original, anotando todas las fichas por el camino.
                    mientras que es cierto:
47.
48.             x -== xdirección
49.             y -== ydirección si
50.             x == xstart y y == ystart: romper
51.
52.             azulejosParaVoltear.append([x, y])
53.

54. if len(tilesToFlip) == 0: # Si no se voltearon fichas, esto no es un
    movimiento válido.
55.     falso retorno
56. volver azulejos a voltear
57.

58. def isOnBoard(x, y): 59. #
    Devuelve True si las coordenadas están ubicadas en el tablero.
59. devuelve x >= 0 y x <= ANCHO - 1 y y >= 0 y y <= ALTO - 1
60.
61.

62. def getBoardWithValidMoves(board, tile): 63. #
    Devuelve un nuevo tablero con puntos que marcan los movimientos válidos del jugador
    poder hacer.
63. CopiaTablero = obtenerCopiaTablero(tablero)

64. para x, y en getValidMoves(boardCopy, mosaico):
65.     boardCopy[x][y] = '.'
66. para x, y en getValidMoves(boardCopy, mosaico):
67.     boardCopy[x][y] = '.'

68. tablero de retornoCopiar

```

69.

70. def getValidMoves(tablero, mosaico):

71. # Devuelve una lista de [x,y] listas de movimientos válidos para el jugador dado
en el tablero dado. 72.

JugadasValidas = [] 73. for x in
range(ANCHO): for y in
74. range(ALTURA): if
75. esJugadaVálida(tablero, mosaico, x, y) != False:
76. Jugadasvalidas.append([x, y])

77. devuelve movimientos válidos

78.

79. def obtenerPuntuaciónDelTablero(tablero):

80. # Determinar la puntuación contando las fichas. Devuelve un diccionario con las
teclas 'X' y 'O'. 81. puntuación x = 0

82. puntuación = 0

83. for x in range(ANCHO): for y in
84. range(ALTO): if tablero[x][y] ==
85. 'X':
86. puntuación x += 1
87. si tablero[x][y] == 'O':
88. puntuación os += 1

89. return {'X':puntuaciónx, 'O':puntuaciónos}

90.

91. def enterPlayerTile(): 92. #
Deja que el jugador ingrese qué mosaico quiere ser.

93. # Devuelve una lista con la ficha del jugador como primer elemento y el
mosaico de la computadora como el segundo.
94. azulejo = "

95. while not (mosaico == 'X' o azulejo == 'O'):
96. print('¿Quieres ser X u O?') mosaico =
97. entrada().superior()
98.

99. # El primer elemento de la lista es la ficha del jugador, y el segundo es la ficha de
la computadora.

100. si mosaico == 'X':
101. devolver ['X', 'O']

102. más:

103. devolver ['O', 'X']

104.

```
105. def quién va primero():
106. # Elige al azar quién va primero. 107. if
random.randint(0, 1) == 0: 108. return 'computadora'

109. más:
110.     devolver 'jugador'
111.

112. def hacerJugada(tablero, loseta, xinicio, yinicio): 113.
# Coloca la loseta en el tablero en xinicio, yinicio y volteá cualquiera de las
piezas del oponente.

114. # Devuelve Falso si este es un movimiento inválido; Verdadero si es válido.

115. tilesToFlip = isValidMove(tablero, mosaico, xinicio, yinicio)
116.

117. if tilesToFlip == Falso: 118.
    falso retorno

119.

120. tablero[xstart][ystart] = mosaico 121.
para x, y en tilesToFlip: tablero[x][y] =
122.     mosaico
123. volver Verdadero
124.

125. def getBoardCopy(board): 126. #
Haz un duplicado de la lista de tableros y devuélvelo. 127. copiatablero
= obtenerNuevoTablero()
128.

129. para x en el rango (ANCHO): para
130.     y en el rango (ALTO):
131.         tableroCopiar[x][y] = tablero[x][y]
132.

133. tablero de retornoCopiar
134.

135. def isOnCorner(x, y): 136. #
Devuelve True si la posición está en una de las cuatro esquinas. 137. return (x ==
0 o x == ANCHO - 1) y (y == 0 o y == ALTO - 1) 138.

139. def getPlayerMove(board, playerTile): 140. #
Permitir que el jugador ingrese su movimiento.

141. # Devuelve el movimiento como [x, y] (o devuelve las cadenas 'sugerencias' o
    'abandonar').

142. DIGITS1TO8 = '1 2 3 4 5 6 7 8'.split()
```

```

143. mientras sea cierto:
144.     print("Ingrese su jugada, \"salir\" para terminar el juego, o \"sugerencias\" para
cambiar las sugerencias.") move = input().lower() if move == 'quit' or move ==
145.     'hints':
146.
147.     movimiento de regreso
148.
149.     if len(mover) == 2 y mover[0] en DIGITS1TO8 y mover[1] en
DIGITOS 1 A 8:
150.         x = int(jugada[0]) - 1 y =
151.         int(jugada[1]) - 1 si
152.             esJugadaVálida(tablero, baldosaJugador, x, y) == Falso:
153.                 Seguir
154.             más:
155.                 descanso
156.             más:
157.                 print('Ese no es un movimiento válido. Ingrese la columna (1-8) y luego la
fila (1-8).') print('Por ejemplo, 81 se moverá en la esquina superior derecha.')
158.
159.
160. devolver [x, y] 161.

162. def getComputerMove(tablero, computerTile): 163. # Dado
un tablero y el mosaico de la computadora, determina dónde 164. # mover y devolver
ese movimiento como una lista [x, y]. 165. jugadasposibles = obtenerJugadasVálidas(tablero,
mosaico de computadora) 166. random.shuffle(movidasposibles) # Aleatoriza el orden
de las jugadas. 167.

168. # Siempre ve a una esquina si está disponible. 169.
for x, y en posiblesMovidas: if estáEnEsquina(x, y): return
170.     [x, y]
171.
172.

173. # Encuentra el movimiento de mayor puntuación posible.
174. mejorpuntuación = -1

175. para x, y en posiblesMovimientos:
176.     copiaTablero = obtenerCopiaTablero(tablero)
177.     realizarMovimiento(CopiaTablero, BaldosaComputadora, x, y)
178.     puntuación = obtenerPuntuaciónTablero(CopiaTablero)[baldosaComputadora]
179.     if puntuación > mejorPuntuación:

```

```
180.        mejorMovida = [x, y]
181.        mejorPuntuación = puntuación
182. volver mejor movimiento
183.
184. def printScore(tablero, playerTile, computerTile): 185.
puntuaciones = getScoreOfBoard(tablero) 186. print('Tú: %s
puntos. Computadora: %s puntos.' % (puntuaciones[playerTile], puntuaciones[computerTile]) )
187.
188. def playGame(playerTile, computerTile): 189.
showHints = False
190. turno = quién va primero() 191.
print('El ' + girar + ' Irá primero.')
192.
193. # Despeja el tablero y coloca las piezas iniciales. 194.
tablero = obtenerNuevoTablero() 195. tablero[3][3] = 'X' 196.
tablero[3][4] = 'O' 197. tablero[4][3] = 'O' 198. tablero[ 4][4] = 'X'
199.
200. mientras es cierto:
201.     jugadorJugadasValidas = obtenerJugadasValidas(tablero, baldosaJugador)
202.     computadoraJugadasValidas = obtenerJugadasValidas(tablero, baldosaComputadora)
203.
204.     if playerValidMoves == [] and computerValidMoves == []: return board #
205.         Nadie puede moverse, así que termina el juego.
206.
207.     elif turn == 'player': # Turno del jugador if
208.         playerValidMoves != []: if showHints:
209.
210.             tableroJugadasVálidas = obtenerTableroConJugadasVálidas(tablero,
BaldosaJugador) dibujarTablero(tableroJugadasVálidas) else:
211.
212.
213.             dibujarTablero(tablero)
214.             imprimirPuntaje(tablero, baldosaJugador, baldosaComputadora)
215.
216.             move = getPlayerMove(board, playerTile) if move
217.             == 'salir':
```

```

218.         print('¡Gracias por jugar!') sys.exit()
219.         # Terminar el programa. elif mover == 'pistas':
220.
221.         mostrar sugerencias = no mostrar sugerencias
222.         Seguir
223.         más:
224.             hacerJugada(tablero, baldosaJugador, jugada[0], jugada[1])
225.             turno = 'computadora'
226.
227.         elif turn == 'computadora': # Turno de la computadora
228.             if computerValidMoves != []: dibujarTablero(tablero)
229.                 imprimirPuntaje(tablero, playerTile, computerTile)
230.
231.
232.             input('Presiona Enter para ver el movimiento de la
233. computadora.') move = getComputerMove(board, computerTile)
234.             makeMove(board, computerTile, move[0], move[1]) turn = 'jugador'
235.
236.
237.
238.

239. print('¡Bienvenido a Reversegam!') 240.

241. BaldosaJugador, BaldosaComputadora = entrarBaldosaJugador()
242.
243. mientras es cierto:
244. tablerofinal = jugarJuego(baldosajugador, baldosaordenador) 245.

246. # Muestra la puntuación final. 247.
dibujarTablero(tablerofinal) 248.
puntuaciones = obtenerPuntuaciónTablero(tablerofinal)
249. print('X anotó %s puntos. O anotó %s puntos.' % (puntuaciones['X'],
puntuaciones['O']))
250. if puntuaciones[baldosaJugador] > puntuaciones[baldosaComputadora]:
251.     print('¡Le ganaste a la computadora por %s puntos! Felicitaciones!' %
(puntuaciones[baldosajugador] - puntuaciones[baldosaComputadora])) 252.
     puntuaciones elif[baldosajugador] < puntuaciones[baldosaordenador]:
```

253. print('Perdiste. La computadora te ganó por %s puntos.' %
(puntajes[computerTile] - puntajes[playerTile]))

254. más:

```
255.     print('El juego fue un empate!')  
256.  
257. print("¿Quieres volver a jugar? (sí o no)") 258. if not  
input().lower().startswith('y'): 259.  
    descanso
```

IMPORTACIÓN DE MÓDULOS Y CONFIGURACIÓN DE CONSTANTES

Al igual que con nuestros otros juegos, comenzamos este programa importando módulos:

```
1. # Reversegam: un clon de Othello/Reversi 2.  
importar aleatoriamente 3. importar sys 4. ANCHO =  
8 # El tablero tiene 8 espacios de ancho.  
  
5. ALTURA = 8 # El tablero tiene 8 espacios de altura.
```

La línea 2 importa el módulo aleatorio para su randint() y choice() funciones La línea 3 importa el módulo sys para su función exit() .

Las líneas 4 y 5 establecen dos constantes, ANCHO y ALTO , que se utilizan para configurar el tablero de juego.

LOS DATOS DEL TABLERO DE JUEGO ESTRUCTURA

Averigüemos la estructura de datos de la placa. Esta estructura de datos es una lista de listas, como la del juego Sonar Treasure Hunt del Capítulo 13. La lista de listas se crea para que tablero[x][y] represente el carácter en el espacio ubicado en la posición x en el eje x (hacia la izquierda/derecha) y la posición y en el eje y (hacia arriba/abajo).).

Este personaje puede ser un ' ' (un espacio que representa un

posición vacía), un '.' (un punto que representa un posible movimiento en el modo de sugerencias), o una 'X' u 'O' (letras que representan fichas).

Cada vez que vea un parámetro llamado tablero, está destinado a ser este tipo de estructura de datos de lista de listas.

Es importante tener en cuenta que mientras que las coordenadas x e y para el tablero de juego variarán de 1 a 8, los índices de la estructura de datos de la lista variarán de 0 a 7. Nuestro código necesitará hacer pequeños ajustes para tener en cuenta esto.

Dibujar la estructura de datos del tablero en la pantalla La estructura de datos del tablero es solo un valor de lista de Python, pero necesitamos una forma más agradable de presentarlo en la pantalla. La función dibujarTablero() toma una estructura de datos del tablero y la muestra en la pantalla para que el jugador sepa dónde se colocan las fichas:

```

6. def dibujarTablero(tablero):
7. # Imprime el tablero pasado a esta función. Devolver Ninguno.
8. imprimir(' 12345678') 9. imprimir(' +-----+') 10. for y en
rango(ALTURA): imprimir("%s| % (y+1), fin =") para x en el rango
(ANCHO):
11
12
13     imprimir(tablero[x][y], fin="")
14     imprimir('|%s' % (y+1)) 15.
imprimir(' +-----+') 16.
imprimir( '12345678')

```

La función dibujarTablero() imprime el tablero de juego actual basado en la estructura de datos a bordo.

La línea 8 es la primera llamada a la función print() ejecutada para cada tablero e imprime las etiquetas para el eje x a lo largo de la parte superior del tablero. La línea 9 imprime la línea horizontal superior del tablero. El ciclo for en la línea 10 se repetirá ocho veces, una para cada fila. Línea

11 imprime la etiqueta para el eje y en el lado izquierdo del tablero, y tiene un argumento de palabra clave end=" para imprimir nada en su lugar de una nueva línea.

Esto es para que otro bucle en la línea 12 (que también se repite ocho veces, una vez por cada columna de la fila) imprima cada posición junto con una X, O, . o un espacio en ~~el bucle interno imprime el tablero[x][y] está llamada a la función print() de la línea 13 dentro de este bucle~~ también tiene un argumento de palabra clave end=" para que el carácter de nueva línea no se imprima. Eso producirá una sola línea en la pantalla que parece '1|XXXXXXX|1' (si cada uno de los valores del tablero[x][y] fuera una 'X').

Una vez que se completa el ciclo interno, la función print() llama a las líneas 15 y 16 para imprimir las etiquetas de la línea horizontal inferior y del eje x.

Cuando el ciclo for en la línea 13 imprime la fila ocho veces, forma todo el tablero:

```
12345678
+-----+
1|XXXXXXXXX|1
2|XXXXXXXXX|2
3|XXXXXXXXX|3
4|XXXXXXXXX|4
5|XXXXXXXXX|5
6|XXXXXXXXX|6
7|XXXXXXXXX|7
8|XXXXXXX|8
+-----+
12345678
```

Por supuesto, en lugar de X, algunos de los espacios en el tablero serán la marca del otro jugador (O), un punto (.) si el modo de pistas está activado, o un espacio para posiciones vacías.

Creación de una nueva estructura de datos del tablero

La función dibujarTablero() mostrará una estructura de datos del tablero en la pantalla, pero también necesitamos una forma de crear estas estructuras de datos del tablero. La función getNewBoard() devuelve una lista de ocho listas, con cada lista que contiene ocho '' cadenas que representarán un tablero en blanco sin movimientos:

```
18. def getNewBoard():
19. # Crear una nueva estructura de datos de tablero en
blanco. 20. tablero = []
21. para i en rango (ANCHO):
22     tablero.append([' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '])
23. regresar tablero
```

La línea 20 crea la lista que contiene las listas internas. el para loop agrega ocho listas internas dentro de esta lista. Estas listas internas tienen ocho cadenas para representar ocho espacios vacíos en el tablero. Juntos, este código crea un tablero con 64 espacios vacíos: un tablero Reversegam en blanco.

COMPROBAR SI HAY UN MOVIMIENTO VÁLIDO

Dada la estructura de datos del tablero, la ficha del jugador y las coordenadas x e y para el movimiento del jugador, la función isValidMove() debería devolver True si las reglas del juego Reversegam permiten un movimiento en esas coordenadas, y False si no lo hacen. Para que un movimiento sea válido, debe estar en el tablero y también voltear al menos una de las fichas del oponente.

Esta función utiliza varias coordenadas x e y en el tablero, por lo que las variables xstart e ystart realizan un seguimiento de las coordenadas x e y del movimiento original.

```

25. def esJugadaVálida(tablero, mosaico, xstart, ystart):
26. # Devuelve Falso si la jugada del jugador en el espacio xstart, ystart es
   inválido.
27. # Si es un movimiento válido, devuelve una lista de espacios que se convertirían
   en el jugador si hiciera un movimiento aquí. 28. si el tablero[iniciox][inicioy] !
= '' o no está A Bordo(iniciox, inicioy): devuelve Falso
29
30
31. si mosaico == 'X':
32.     otraBaldosa = 'O'
33. más:
34.     otraBaldosa = 'X'
35.
36. baldosas para voltear = []

```

La línea 28 verifica si las coordenadas x e y están en el tablero de juego y si el espacio está vacío usando la función `isOnBoard()` (que definiremos más adelante en el programa).

Esta función se asegura de que las coordenadas x e y estén entre 0 y el ANCHO o ALTO del tablero menos 1.

La ficha del jugador (ya sea el jugador humano o el jugador de la computadora) está en ficha, pero esta función necesitará conocer la ficha del oponente. Si la ficha del jugador es X, entonces obviamente la ficha del oponente es O, y viceversa. Usamos la sentencia `if-else` en las líneas 31 a 34 para esto.

Finalmente, si la coordenada x e y dada es un movimiento válido, `isValidMove()` devuelve una lista de todas las fichas del oponente que se voltearían con este movimiento. Creamos una nueva lista vacía, `tilesToFlip`, que usaremos para almacenar todas las coordenadas de mosaico.

Comprobación de cada una de las ocho direcciones

Para que un movimiento sea válido, debe voltear al menos una de las fichas del oponente intercalando la nueva ficha del jugador actual.

ficha con una de las fichas antiguas del jugador. Eso significa que la nueva ficha debe estar al lado de una de las fichas del oponente.

El bucle `for` en la línea 37 itera a través de una lista de listas que representa las direcciones en las que el programa buscará la ficha de un oponente:

37. para direcciónx, direccióny en [[0, 1], [1, 1], [1, 0], [1, -1],
[0, -1], [-1, -1], [-1, 0], [-1, 1]]:

El tablero de juego es un sistema de coordenadas cartesianas con direcciones x e y. Hay ocho direcciones para verificar: arriba, abajo, izquierda, derecha y las cuatro direcciones diagonales. Cada una de las ocho listas de dos elementos en la lista de la línea 37 se usa para verificar una de estas direcciones. El programa comprueba una dirección añadiendo el primer valor de la lista de dos elementos a la coordenada x y el segundo valor a la coordenada y.

Debido a que las coordenadas x aumentan a medida que avanza hacia la derecha, puede verificar la dirección correcta sumando 1 a la x coordinar. Entonces, la lista [1, 0] agrega 1 a la coordenada x y 0 a la coordenada y. Verificar la dirección izquierda es lo contrario: restarías 1 (es decir, sumarías -1) de la coordenada x.

Pero para verificar en diagonal, debe sumar o restar de ambas coordenadas. Por ejemplo, sumar 1 a la coordenada x y sumar -1 a la coordenada y resultaría en verificar la dirección diagonal hacia arriba a la derecha.

La figura 15-7 muestra un diagrama para que sea más fácil recordar qué lista de dos elementos representa qué dirección.

| | | |
|---------------|------------|---------------|
| | | x increases → |
| y increases → | | |
| | [-1, -1] | [0, -1] |
| | [-1, 0] | [0, 0] |
| | [-1, 1] | [0, 1] |
| | [1, -1] | [1, 0] |
| | [1, 1] | |

Figura 15-7: Cada lista de dos elementos representa una de las ocho direcciones.

El ciclo for en la línea 37 itera a través de cada una de las listas de dos elementos para que se verifique cada dirección. Dentro del bucle for , las variables x e y se establecen en los mismos valores que xstart e ystart, respectivamente, utilizando la asignación múltiple en la línea 38. Las variables direcciónx e direccióny se establecen en los valores de una de las dos listas de elementos y cambian las variables x e y de acuerdo con la dirección que se verifica en esa iteración del ciclo for :

-
- ```

37. para direcciónx, direccióny en [[0, 1], [1, 1], [1, 0], [1, -1],
 [0, -1], [-1, -1], [-1, 0], [-1, 1]]:
38. x, y = iniciox, inicioy x
39. += direcciónx # Primer paso en la dirección x y +=
40. direccióny # Primer paso en la dirección y

```
- 

Las variables xstart e ystart permanecerán igual para que el El programa puede recordar desde qué espacio comenzó originalmente.

Recuerda, para que un movimiento sea válido, debe estar tanto en el tablero como al lado de una de las fichas del otro jugador. (De lo contrario, no hay fichas del oponente para voltear, y un movimiento debe voltear al menos una ficha para que sea válido). La línea 41 verifica esta condición y, si no es verdadera, la ejecución vuelve a la por

declaración para comprobar la siguiente dirección.

---

41.     while isOnBoard(x, y) and board[x][y] == otherTile: #
  42.         Sigue moviéndote en esta dirección x e y.
  43.         x += dirección x
  44.         y += dirección y
- 

Pero si el primer espacio marcado tiene la ficha del oponente, entonces el programa debe buscar más fichas del oponente en esa dirección hasta que llegue a una de las fichas del jugador o al final del tablero. Se marca la siguiente ficha en la misma dirección

usando xdirection y ydirection nuevamente para hacer que x e y sean las siguientes coordenadas para verificar. Entonces el programa cambia x e y en las líneas 43 y 44.

Averiguar si hay mosaicos para voltear A continuación, verificamos si hay mosaicos adyacentes que se puedan voltear.

- 
45.     if isOnBoard(x, y) and board[x][y] == mosaico:
  46.         # Hay piezas para voltear. Ve en dirección contraria  
                hasta llegar al espacio original, anotando todas las fichas  
                por el camino. mientras que es cierto:
  - 47.
  48.         x -= xdirección
  49.         y -= ydirección
  50.         si x == xstart y y == ystart:
  51.             romper
  52.         azulejosParaVoltear.append([x, y])
- 

La declaración if en la línea 45 verifica si una coordenada es ocupado por la propia ficha del jugador. Esta ficha marcará el final del sándwich formado por las fichas del jugador que rodean las fichas del oponente. También necesitamos registrar las coordenadas de todas las fichas del oponente que deben voltearse.

El ciclo while mueve x e y en reversa en las líneas 48 y 49.

Hasta que x e y vuelvan a la posición original xstart e ystart , xdirection y ydirection se restan de xey , y cada posición xey se agrega a la lista tilesToFlip . Cuando x e y han llegado a la posición xstart e ystart , la línea 51 interrumpe la ejecución fuera del bucle. Dado que la posición original de xstart e ystart es un espacio vacío (nos aseguramos de que este fuera el caso en las líneas 28 y 29), la condición para el ciclo while de la línea 41 será Falso.

El programa pasa a la línea 37 y el bucle for comprueba

la próxima dirección.

El ciclo for hace esto en las ocho direcciones. Después de que se complete ese bucle, la lista tilesToFlip contendrá las coordenadas x e y de todas las fichas de nuestro oponente que se voltearían si el jugador se moviera en xstart, ystart. Recuerde, la función isValidMove() solo verifica si el movimiento original fue válido; en realidad, no cambia permanentemente la estructura de datos del tablero de juego.

Si ninguna de las ocho direcciones terminó volteando al menos una de las fichas del oponente, entonces tilesToFlip será una lista vacía:

- 
- ```
54. if len(tilesToFlip) == 0: # Si no se voltearon fichas, esto no es un
    movimiento válido.
55.     falso retorno
56. volver azulejos a voltear
```
-

Esta es una señal de que este movimiento no es válido y isValidMove() debe devolver Falso. De lo contrario, isValidMove() devuelve tilesToFlip.

COMPROBACIÓN DE VÁLIDA COORDENADAS

La función `isOnBoard()` se llama desde `isValidMove()`. Realiza una simple verificación para ver si las coordenadas `x` e `y` dadas están en el tablero. Por ejemplo, una coordenada `x` de 4 y una coordenada `y` de 9999 no estarían en el tablero ya que las coordenadas `y` solo llegan hasta 7, que es igual a `ANCHO - 1` o `ALTURA - 1`.

```
58. def isOnBoard(x, y): 59. #
Devuelve True si las coordenadas están ubicadas en el tablero.

60. devuelve x >= 0 y y <= ANCHO - 1 y y >= 0 y y <= ALTO - 1
```

Llamar a esta función es una forma abreviada de la expresión booleana en la línea 72 que verifica si tanto `x` como `y` están entre 0 y el `ANCHO` o `ALTO` restado por 1, que es 7.

Obtener una lista con todos los movimientos válidos

Ahora vamos a crear un modo de sugerencias que muestre un tablero con todos los movimientos posibles marcados en él. La función `getBoardWithValidMoves()` devuelve una estructura de datos de tablero de juego que tiene puntos (.) para todos los espacios que son movimientos válidos:

```
62. def getBoardWithValidMoves(board, tile): 63. #
Devuelve un nuevo tablero con puntos que marcan los movimientos válidos del jugador
poder hacer.

64. copiaTablero = obtenerCopiaTablero(tablero) 65.

66. para x, y en getValidMoves(boardCopy, mosaico):
67.     boardCopy[x][y] = '.'

68. tablero de retornoCopiar
```

Esta función crea una estructura de datos del tablero de juego duplicada llamada `boardCopy` (devuelta por `getBoardCopy()` en la línea 64) en lugar de modificar la que se le pasó en el parámetro del tablero . La línea 66 llama a `getValidMoves()` para obtener una lista de coordenadas `x` e `y` con todas

los movimientos legales que el jugador podría hacer. La copia de la pizarra se marca con puntos en esos espacios y se devuelve.

La función getValidMoves() devuelve una lista de listas de dos elementos.

Las listas de dos elementos contienen las coordenadas x e y de todos los movimientos válidos de la ficha que se le ha dado para la estructura de datos del tablero en el parámetro de la placa :

```
70. def getValidMoves(tablero, mosaico): 71. #
Devuelve una lista de [x,y] listas de movimientos válidos para el jugador dado
en el tablero dado. 72.
jugadasvalidas = [] 73. for x in
range(ANCHO): 74. for y in
range(ALTURA): if esJugadaVálida(tablero,
mosaico, x, y): Falso:
75.
76.         jugadasvalidas.append([x, y])
77. devuelve movimientos válidos
```

Esta función utiliza bucles anidados (en las líneas 73 y 74) para verificar cada coordenada x e y (las 64) llamando a isValidMove() en ese espacio y verificando si devuelve False o una lista de posibles movimientos (en cuyo caso el movimiento es válido).

Cada coordenada x e y válida se agrega a la lista en

movimientos válidos.

Llamar a la función bool() Es posible que haya notado que el programa verifica si isValidMove() en la línea 75 devuelve False aunque esta función devuelve una lista. Para entender cómo funciona esto, necesita aprender un poco más sobre los booleanos y la función bool() .

La función bool() es similar a las funciones int() y str() . Eso devuelve la forma de valor booleano del valor que se le ha pasado.

La mayoría de los tipos de datos tienen un valor que se considera falso .

valor para ese tipo de datos. Cualquier otro valor se considera verdadero.

Por ejemplo, el número entero 0, el número de punto flotante 0.0, una cadena vacía, una lista vacía y un diccionario vacío se consideran falsos cuando se usan como condición para un si o

declaración de bucle. Todos los demás valores son verdaderos. Introduzca la siguiente en el caparazón interactivo:

```
>>> bool(0)
Falso

>>> booleano(0.0)
Falso

>>> bool("")
Falso

>>> booleano([])
Falso

>>> booleano({})
Falso

>>> bool(1)
Verdadero

>>> bool('Hola')
Verdadero

>>> booleano([1, 2, 3, 4, 5])
Verdadero

>>> bool({'correo no deseado':'queso', 'efervescencia':'zumbido'})
Verdadero
```

Las condiciones se interpretan automáticamente como valores booleanos. Por eso la condición de la línea 75 funciona correctamente.

La llamada a la función isValidMove() devuelve el booleano value False o una lista no vacía.

Si imagina que la condición completa se coloca dentro de una llamada a bool(), entonces la condición False de la línea 75 se convierte en bool(False) (que, por supuesto, se evalúa como False). Y una condición de una lista no vacía colocada como parámetro para bool() devolverá True.

OBTENER LA PUNTUACIÓN DE LA JUEGO DE MESA

La función getScoreOfBoard() utiliza bucles for anidados para verificar las 64 posiciones en el tablero y ver qué ficha de jugador (si corresponde) está en ellas:

```
79. def obtenerPuntuaciónDelTablero(tablero):
80. # Determinar la puntuación contando las fichas. Devuelve un diccionario con las
     teclas 'X' y 'O'. 81. puntuación x = 0

82. puntuación = 0
83. para x en el rango (ANCHO):
84.     para y en el rango (ALTO): si
85.         tablero[x][y] == 'X': xscore
86.             += 1
87.         si tablero[x][y] == 'O':
88.             puntuación os += 1
89. return {'X':puntuaciónx, 'O':puntuaciónos}
```

Para cada mosaico X , el código incrementa xscore en la línea 86. Para cada mosaico O , el código incrementa oscore en la línea 88. La función luego devuelve xscore y oscore en un diccionario.

CONSEGUIR LA BALDOSA DEL JUGADOR ELECCIÓN

La función enterPlayerTile() le pregunta al jugador qué mosaico quiere ser, ya sea X u O:

```
91. def enterPlayerTile(): 92. #
Deja que el jugador ingrese qué mosaico quiere ser.
93. # Devuelve una lista con la ficha del jugador como primer elemento y el
     mosaico de la computadora como el segundo.
94. azulejo = ""
95. while not (mosaico == 'X' o azulejo == 'O'):
```

```

96.     print('¿Quieres ser X u O?') mosaico =
97.     entrada().superior()
98.
99. # El primer elemento de la lista es la ficha del jugador, y el segundo es la
    ficha de la computadora.
100. si mosaico == 'X':
101.     devolver ['X', 'O']
102. más:
103.     devolver ['O', 'X']

```

El ciclo for continuará hasta que el jugador ingrese X u O en mayúsculas o minúsculas. La función enterPlayerTile() luego devuelve una lista de dos elementos, donde la elección de ficha del jugador es el primer elemento y la ficha de la computadora es el segundo. Más tarde, la línea 241 llama a enterPlayerTile() y usa asignación múltiple para poner estos dos elementos devueltos en dos variables.

DETERMINAR QUIÉN VA PRIMERO

La función whoGoesFirst() selecciona aleatoriamente quién va primero y devuelve la cadena 'computadora' o la cadena 'jugador':

```

105. def quién va primero():
106. # Elige al azar quién va primero. 107. if
    random.randint(0, 1) == 0: devuelve 'computadora'
108.
109. más:
110.     devolver 'jugador'

```

COLOCANDO UNA BALDOSA EN EL TABLERO

La función makeMove() se llama cuando un jugador quiere colocar una ficha en el tablero y voltear las otras fichas de acuerdo con las reglas de Reversegam:

```

112. def hacerJugada(tablero, loseta, xinicio, yinicio):
113. # Coloca la loseta en el tablero en xinicio, yinicio y volteá cualquiera de las
    piezas del oponente.
114. # Devuelve Falso si este es un movimiento inválido; Verdadero si es
    válido. 115. tilesToFlip = isValidMove(tablero, mosaico, xinicio, yinicio)

```

Esta función modifica en su lugar la estructura de datos del tablero que se pasa. Los cambios realizados en la variable del tablero (porque es una referencia de lista) se realizarán en el ámbito global.

La mayor parte del trabajo lo realiza isValidMove() en la línea 115, que devuelve una lista de coordenadas x e y (en una lista de dos elementos) de mosaicos que deben invertirse. Recuerde, si los argumentos xstart e ystart apuntan a un movimiento no válido, isValidMove() devolverá el valor booleano Falso, que se verifica en la línea 117:

```

117. if tilesToFlip == Falso:
118.     devuelve Falso
119.

120. tablero[xstart][ystart] = mosaico
121. for x, y in tilesToFlip: 122.
    tablero[x][y] = mosaico 123. return True

```

Si el valor de retorno de isValidMove() (ahora almacenado en tilesToFlip) es False, entonces makeMove() también devolverá False en la línea 118.

De lo contrario, esJugadaVálida() devuelve una lista de espacios en el tablero para colocar las fichas (la cadena 'X' u 'O' en la ficha). La línea 120 establece el espacio por el que se ha movido el jugador. La línea 121 for loop establece todos los mosaicos que están en tilesToFlip.

COPIA DE DATOS DE LA TARJETA ESTRUCTURA

La función `getBoardCopy()` es diferente de `getNewBoard()`. La función `getNewBoard()` crea una estructura de datos de tablero de juego en blanco que solo tiene espacios vacíos y las cuatro fichas iniciales. `getBoardCopy()` crea una estructura de datos del tablero de juego en blanco, pero luego copia todas las posiciones en el parámetro del tablero con un bucle anidado. La IA utiliza la función `getBoardCopy()` para que pueda realizar cambios en la copia del tablero de juego sin cambiar el tablero de juego real. Esta técnica también fue utilizada por el programa Tic-Tac-Toe en el Capítulo 10.

```
125. def getBoardCopy(board): 126.  
# Haz un duplicado de la lista de tableros y devuélvelo. 127.  
CopiaTablero = obtenerNuevoTablero() 128.
```

```
129. para x en el rango (ANCHO):  
130.     para y en el rango (ALTO):  
131.         tableroCopiar[x][y] = tablero[x][y]  
132.  
133. tablero de retornoCopiar
```

Una llamada a `getNewBoard()` configura `boardCopy` como una nueva estructura de datos del tablero de juego. Luego, los dos bucles `for` anidados copian cada uno de los 64 mosaicos del tablero a la estructura de datos duplicada del tablero en `tableroCopiar`.

DETERMINAR SI UN EL ESPACIO ESTÁ EN UNA ESQUINA

La función `isOnCorner()` devuelve True si las coordenadas están en un espacio de esquina en las coordenadas (0, 0), (7, 0), (0, 7) o (7, 7):

```
135. def isOnCorner(x, y): 136.  
# Devuelve True si la posición está en una de las cuatro esquinas. 137.  
return (x == 0 o x == ANCHO - 1) y (y == 0 o y == ALTO - 1)
```

De lo contrario, `isOnCorner()` devuelve `False`. Usaremos esta función más tarde para la IA.

OBTENER EL MOVIMIENTO DEL JUGADOR

Se llama a la función `getPlayerMove()` para permitir que el jugador ingrese las coordenadas de su próximo movimiento (y verifique si el movimiento es válido). El jugador también puede ingresar sugerencias para activar el modo de sugerencias (si está desactivado) o desactivarlo (si está activado). Finalmente, el jugador puede ingresar `quit` para salir del juego.

```
139. def getPlayerMove(board, playerTile): 140. #
    Permitir que el jugador ingrese su movimiento.
141. # Devuelve el movimiento como [x, y] (o devuelve las cadenas 'sugerencias' o
    'abandonar').
142. DIGITS1TO8 = '1 2 3 4 5 6 7 8'.split()
```

La variable constante `DIGITS1TO8` es la lista `['1', '2', '3', '4', '5', '6', '7', '8']`. La función `getPlayerMove()` usa `DIGITS1TO8` un par de veces, y esta constante es más legible que el valor de la lista completa. No puede usar el método `isdigit()` porque eso permite ingresar 0 y 9, que no son coordenadas válidas en el tablero de 8×8.

El bucle `while` continúa hasta que el jugador escribe un movimiento válido:

```
143. mientras sea cierto:
144.     print("Ingrese su jugada, \"salir\" para terminar el juego, o \"sugerencias\" para
        cambiar las sugerencias.") move = input().lower() if move == 'quit' or move ==
145.         'hints':
146.
147.         movimiento de regreso
```

La línea 146 comprueba si el jugador quiere salir o cambiar el modo de pistas, y la línea 147 devuelve la cadena 'salir' o 'pistas', respectivamente. Se llama al método lower() en la cadena devuelta por input() para que el jugador pueda escribir SUGERENCIAS o Salir y aún tener la comando entendido.

El código que llamó a getPlayerMove() se encargará de qué hacer si el jugador quiere salir o cambiar el modo de sugerencias. Si el jugador ingresa las coordenadas para continuar, la declaración if en la línea 149 comprueba si el movimiento es válido:

```

149.    if len(mover) == 2 y mover[0] en DIGITS1TO8 y mover[1] en
        DIGITOS 1 A 8:
150.        x = int(jugada[0]) - 1 y
151.        = int(jugada[1]) - 1 si
152.        esJugadaVálida(tablero, baldosaJugador, x, y) == Falso:
153.            Seguir
154.        más:
155.            descanso

```

El juego espera que el jugador haya ingresado las coordenadas x e y de su movimiento como dos números sin nada entre ellos. La línea 149 primero verifica que el tamaño de la cadena que el jugador escribió sea 2. Despues de eso, también verifica que tanto move[0] (el primer carácter de la cadena) como move[1] (el segundo carácter de la cadena) son cadenas que existen en DIGITS1TO8.

Recuerde que las estructuras de datos del tablero de juego tienen índices del 0 al 7, no del 1 al 8. El código imprime del 1 al 8 cuando el tablero se muestra en dibujarTablero() porque los no programadores están acostumbrados a los números que comienzan en 1 en lugar de 0. Entonces, para convertir las cadenas en move[0] y move[1] a números enteros, las líneas 150 y 151 restan 1 de x e y.

Incluso si el jugador ha ingresado un movimiento correcto, el código

necesita comprobar que el movimiento está permitido por las reglas de Reversegam. Esto se hace mediante la función esJugadaVálida() , a la que se le pasa la estructura de datos del tablero de juego, la ficha del jugador y las coordenadas x e y de la jugada.

Si esJugadaVálida() devuelve Falso, se ejecuta la instrucción de continuación de la línea 153 . Luego, la ejecución vuelve al comienzo del ciclo while y le pide al jugador un movimiento válido nuevamente.

De lo contrario, el jugador ingresó un movimiento válido y la ejecución debe salir del bucle while .

Si la condición de la declaración if en la línea 149 era Falsa, entonces la el jugador no ingresó un movimiento válido. Las líneas 157 y 158 les dan instrucciones sobre cómo introducir movimientos correctamente:

-
- ```
156. más:
157. print('Ese no es un movimiento válido. Ingrese la columna (1-8)
y luego la fila (1-8).') print('Por ejemplo, 81 se moverá en la
158. esquina superior derecha.')
```
- 

Luego, la ejecución vuelve a la instrucción while en la línea 143, porque la línea 158 no solo es la última línea del bloque else sino también la última línea del bloque while . El ciclo while continuará hasta que el jugador ingrese un movimiento válido. Si el jugador ingresa las coordenadas x e y, la línea 160 ejecutará:

- 
- ```
160. volver [x, y]
```
-

Finalmente, si se ejecuta la línea 160, obtenerJugadaJugador() devuelve una lista de dos elementos con las coordenadas x e y de la jugada válida del jugador. moverse.

CONSEGUIR LA COMPUTADORA MOVERSE

La función getComputerMove() es donde se implementa el algoritmo de IA:

```
162. def getComputerMove(tablero, computerTile): 163. #
Dado un tablero y el mosaico de la computadora, determina dónde 164. #
mover y devolver ese movimiento como una lista [x, y]. 165. jugadasposibles
= obtenerJugadasVálidas(tablero, mosaico de computadora)
```

Normalmente, usa los resultados de getValidMoves() para el modo de sugerencias, que imprimirá . en tablero para posibles movimientos. Pero si se llama a getValidMoves() con el mosaico de la IA de la computadora (en computerTile), también encontrará todos los movimientos posibles que la computadora puede hacer. La IA seleccionará el mejor movimiento de esta lista.

Primero, la función random.shuffle() aleatorizará el orden de movimientos en la lista de movimientos posibles :

```
166. random.shuffle(possibleMoves) # Aleatoriza el orden de los movimientos.
```

Queremos barajar la lista de movimientos posibles porque hará que la IA sea menos predecible; de lo contrario, el jugador podría simplemente memorizar los movimientos necesarios para ganar porque las respuestas de la computadora siempre serían las mismas. Veamos el algoritmo.

Elaboración de estrategias con movimientos de esquina Los movimientos de esquina son una buena idea en Reversegam porque una vez que se ha colocado una ficha en una esquina, nunca se puede voltear. La línea 169 recorre cada movimiento en posiblesMovidas. Si alguno de ellos está en la esquina, el programa devolverá ese espacio como el movimiento de la computadora:

```
168. # Siempre ve a una esquina si está disponible.
```

```

169. for x, y en posiblesMovidas: if
170.     estáEnEsquina(x, y): return
171.     [x, y]

```

Dado que jugadasPosibles es una lista de listas de dos elementos, usaremos asignación múltiple en el ciclo for para establecer x e y. Si posiblesMoves contiene varios movimientos de esquina, siempre se usa el primero. Pero como jugadasPosibles se barajó en la línea 166, ¿qué esquina mover es el primero en la lista es aleatorio.

Obtener una lista de los movimientos con la puntuación más alta Si no hay movimientos en las esquinas, el programa recorrerá la lista completa de movimientos posibles y descubrirá cuál da como resultado la puntuación más alta. Luego, bestMove se establece en el movimiento con la puntuación más alta que el código haya encontrado hasta el momento, y bestScore se establece en el mejor puntuación de la jugada. Esto se repite hasta que se encuentra el movimiento posible con la puntuación más alta.

```

173. # Encuentra el movimiento de mayor puntuación posible.
174. mejorPuntuación = -1
175. para x, y en movimientos posibles:
176.     copiaTablero = obtenerCopiaTablero(tablero)
177.     realizarMovimiento(CopiaTablero, BaldosaComputadora, x, y)
178.     puntuación = obtenerPuntuaciónTablero(CopiaTablero)[baldosaComputadora]
179.     if puntuación > mejorPuntuación:
180.         mejorMovida = [x, y]
181.         mejorPuntuación = puntuación
182. volver mejor movimiento

```

La línea 174 primero establece bestScore en -1 para que el primero mueva el las comprobaciones de código se establecerán en el primer bestMove. Esto garantiza que bestMove se establece en uno de los movimientos de posiblesMoves cuando vuelve.

En la línea 175, el ciclo for establece x e y para cada movimiento en

posiblesMovimientos. Antes de simular un movimiento, la línea 176 crea una estructura de datos de tablero de juego duplicada llamando a getBoardCopy(). Querrá una copia que pueda modificar sin cambiar la estructura de datos del tablero de juego real almacenada en la variable del tablero .

Luego, la línea 177 llama a realizarJugada(), pasando el tablero duplicado (almacenado en copiaTablero) en lugar del tablero real. Esto simulará lo que sucedería en el tablero real si se hiciera ese movimiento. La función makeMove() se encargará de colocar la ficha de la computadora y voltear las fichas del jugador en el tablero duplicado.

La línea 178 llama a getScoreOfBoard() con el tablero duplicado, que devuelve un diccionario donde las claves son 'X' y 'O', y los valores son los puntajes. Cuando el código en el bucle encuentra un movimiento con una puntuación superior a bestScore, las líneas 179 a 181 almacene ese movimiento y puntúe como los nuevos valores en bestMove y mejor puntuación. Después de que possibleMoves se haya iterado por completo, se devuelve bestMove .

Por ejemplo, digamos que getScoreOfBoard() devuelve el diccionario {'X':22, 'O':8} y computerTile es 'X'. Entonces getScoreOfBoard(boardCopy)[ComputerTile] se evaluaría como {'X':22, 'O':8}['X'], lo que luego evalúa a 22. Si 22 es mayor que bestScore, bestScore se establece en 22 y bestMove se establece en los valores actuales de x e y .

Para cuando finalice este ciclo for , puede estar seguro de que bestScore es el puntaje más alto posible que puede obtener un movimiento, y que el movimiento se almacena en bestMove.

Aunque el código siempre elige el primero en la lista de estos movimientos empatados, la elección parece aleatoria porque el orden de la lista se barajó en la línea 166. Esto asegura que la IA no

ser predecible cuando hay más de un mejor movimiento.

IMPRESIÓN DE LAS PUNTUACIONES A LA PANTALLA

La función `showPoints()` llama a la función `getScoreOfBoard()` y luego imprime las puntuaciones del jugador y de la computadora:

```
184. def printScore(tablero, playerTile, computerTile): 185.  
puntuaciones = getScoreOfBoard(tablero) 186. print('Tú:  
%s puntos. Computadora: %s puntos.' % (puntuaciones[playerTile],  
puntuaciones[computerTile]))
```

Recuerda que `getScoreOfBoard()` devuelve un diccionario con las claves 'X' y 'O' y los valores de las puntuaciones de los jugadores X y O.

Esas son todas las funciones del juego Reversegam. El código en la función `playGame()` implementa el juego real y llama a estas funciones según sea necesario.

COMENZANDO EL JUEGO

La función `playGame()` llama a las funciones anteriores que hemos escrito para jugar un solo juego:

```
188. def playGame(playerTile, computerTile): 189.  
showHints = False  
190. turno = quién va primero()  
191. print('El ' + girar + ' Irá primero.')  
192.  
193. # Despeja el tablero y coloca las piezas iniciales.  
194. tablero = obtenerNuevoTablero() 195. tablero[3][3]  
= 'X' 196. tablero[3][4] = 'O' 197. tablero[4][3] = 'O' 198.  
tablero[ 4][4] = 'X'
```

A la función playGame() se le pasan cadenas 'X' u 'O' para playerTile y computerTile. El jugador que va primero está determinado por la línea 190. La variable de turno contiene la cadena 'computadora' o 'jugador' para realizar un seguimiento de a quién le toca. La línea 194 crea una estructura de datos de tablero en blanco, mientras que las líneas 195 a 198 configuran los cuatro mosaicos iniciales en el tablero. El juego ya está listo para comenzar.

Comprobación de un punto muerto

Antes de obtener el turno del jugador o de la computadora, debemos comprobar si es posible que alguno de ellos se mueva. Si no, entonces el juego está en un punto muerto y debería terminar. (Si solo un lado no tiene movimientos válidos, el turno salta al otro jugador).

200. mientras es cierto:

```

201.     jugadorJugadasValidas = obtenerJugadasValidas(tablero, baldosaJugador)
202.     computadoraJugadasValidas = obtenerJugadasValidas(tablero, baldosaComputadora)
203.
204.     if playerValidMoves == [] and computerValidMoves == []: return board #
205.     Nadie puede moverse, así que termina el juego.

```

La línea 200 es el bucle principal para ejecutar los turnos del jugador y la computadora. Mientras este ciclo continúe, el juego continuará. Pero antes de ejecutar estos turnos, las líneas 201 y 202 verifican si cualquiera de los lados puede hacer un movimiento al obtener una lista de movimientos válidos. Si ambas listas están vacías, ningún jugador puede hacer un movimiento. La línea 205 sale de la función playGame() devolviendo el último tablero, finalizando el juego.

Ejecutar el turno del jugador

Si el juego no está estancado, el programa determina si es el turno del jugador comprobando si el turno está configurado en la cadena 'jugador':

```

207.     elif turn == 'player': # Turno del jugador if
208.         playerValidMoves != []:
209.             si mostrar sugerencias:
210.                 tableroJugadasVálidas = obtenerTableroConJugadasVálidas(tablero,
211.                               BaldosaJugador) dibujarTablero(tableroJugadasVálidas) else:
212.
213.                 dibujarTablero(tablero)
214.                 imprimirPuntaje(tablero, baldosaJugador, baldosaComputadora)

```

La línea 207 comienza un bloque elif que contiene el código que se ejecuta si es el turno del jugador. (El bloque elif que comienza en la línea 227 contiene el código para el turno de la computadora).

Todo este código se ejecutará solo si el jugador tiene un movimiento válido, que la línea 208 determina al verificar que playerValidMoves no esté vacío. Mostramos el tablero en la pantalla llamando a dibujarTablero() en la línea 211 o 213.

Si el modo de sugerencias está activado (es decir, showHints es True), la estructura de datos del tablero debería mostrar . en cada espacio legal que el jugador podría mover, getBoardWithValidMoves() . Se pasa una estructura de datos de tablero de juego y devuelve una copia que también contiene puntos (.). La línea 211 pasa este tablero a la función dibujarTablero() .

Si el modo de sugerencias está desactivado, la línea 213 pasa el tablero a dibujarTablero() en cambio.

Después de imprimir el tablero de juego al jugador, también desea para imprimir la puntuación actual llamando a printScore() en la línea 214.

A continuación, el jugador debe ingresar su movimiento. La función getPlayerMove() maneja esto, y su valor de retorno es una lista de dos elementos de las coordenadas x e y del movimiento del jugador:

```
216.     mover = obtenerJugadaJugador(tablero, baldosaJugador)
```

Cuando definimos getPlayerMove(), ya nos aseguramos de que el movimiento del jugador es válido.

La función getPlayerMove() puede haber devuelto las cadenas 'quit' o 'hints' en lugar de un movimiento en el tablero. Líneas 217 a 222 manejar estos casos:

```

217.     if mover == 'salir':
218.         print('¡Gracias por jugar!')
219.         sys.exit() # Terminar el programa. elif
220.         mover == 'pistas':
221.             mostrar sugerencias = no mostrar sugerencias
222.             Seguir
223.             más:
224.                 hacerJugada(tablero, baldosaJugador, mover[0], mover[1])
225.             turno = 'computadora'

```

Si el jugador ingresó abandono para su movimiento, getPlayerMove() devolvería la cadena 'quit'. En ese caso, la línea 219 llama a sys.exit() para finalizar el programa.

Si el jugador ingresó sugerencias para su movimiento, getPlayerMove() devolvería la cadena 'sugerencias'. En ese caso, desea activar el modo de sugerencias (si estaba desactivado) o desactivarlo (si estaba activado).

La declaración de asignación showHints = not showHints en la línea 221 maneja ambos casos, porque not False se evalúa como True y no True se evalúa como False. Entonces la declaración de continuar se mueve la ejecución hasta el inicio del bucle (el turno no ha cambiado, por lo que seguirá siendo el turno del jugador).

De lo contrario, si el jugador no salió o no cambió el modo de pistas, la línea 224 llama a hacerJugada() para hacer la jugada del jugador en el junta.

Finalmente, la línea 225 cambia a ' computadora'. El flujo de ejecución salta el bloque else y llega al final del bloque while , por lo que

la ejecución vuelve a la instrucción `while` de la línea 200. Esta vez, sin embargo, será el turno de la computadora.

Ejecutar el turno de la computadora Si la variable `turn` contiene la cadena 'computadora', entonces se ejecutará el código para el turno de la computadora. Es similar al código del turno del jugador, con algunos cambios:

```

227.     elif turn == 'computadora': # Turno de la
228.         computadora if computerValidMoves != []:
229.             dibujarTablero(tablero)
230.             imprimirPuntaje(tablero, playerTile, computerTile)
231.
232.             input('Presione Enter para ver el movimiento de la
233.             computadora.') move = getComputerMove(board,
234.             computerTile) makeMove(board, computerTile, move[0], move[1])

```

Después de imprimir el tablero con `drawBoard()`, el programa también imprime la puntuación actual con una llamada a `showPoints()` en la línea 230.

La línea 232 llama a `input()` para pausar el guión para que el jugador pueda mirar el tablero. Esto es muy parecido a cómo se usó `input()` para pausa el programa Jokes en el Capítulo 4. En lugar de usar una llamada `print()` para imprimir una cadena antes de una llamada a `input()`, puedes hacer lo mismo pasando la cadena para imprimir a `input()`.

Después de que el jugador haya mirado el tablero y presionado ENTER, la línea 233 llama a `getComputerMove()` para obtener las coordenadas x e y del próximo movimiento de la computadora. Estas coordenadas se almacenan en las variables x e y mediante asignación múltiple.

Finalmente, x e y, junto con la estructura de datos del tablero de juego y el mosaico de la computadora, se pasan a la función `hacerJugada()`. Esto coloca la ficha de la computadora en el tablero de juego para reflejar el movimiento de la computadora. Llamada de la línea 233 a `getComputerMove()`

obtuvo el movimiento de la computadora (y lo almacenó en las variables x e y). La llamada a hacerJugada() en la línea 234 hace que la jugada se junta.

A continuación, la línea 235 establece la variable turno en 'jugador':

```
235.     turno = 'jugador'
```

No hay más código en el bloque while después de la línea 235, por lo que la ejecución regresa a la instrucción while en la línea 200.

EL BUCLE DEL JUEGO

Esas son todas las funciones que haremos para Reversegam. Comenzando en la línea 239, la parte principal del programa ejecuta un juego llamando a playGame(), pero también muestra el puntaje final y le pregunta al jugador si quiere volver a jugar:

```
239. print('¡Bienvenido a Reversegam!')  
240.  
241. BaldosaJugador, BaldosaComputadora = entrarBaldosaJugador()
```

El programa comienza dando la bienvenida al jugador en la línea 239 y preguntándole si quiere ser X u O. La línea 241 usa el truco de asignación múltiple para establecer baldosaJugador y baldosaComputadora en los dos valores devueltos por EnterPlayerTile().

El ciclo while en la línea 243 ejecuta cada juego:

```
243. mientras es cierto:  
244. tablerofinal = jugarJuego(baldosajugador, baldosaordenador) 245.  
  
246. # Muestra la puntuación final. 247.  
dibujarTablero(tablerofinal) 248. puntuaciones  
= obtenerPuntuaciónTablero(tablerofinal) 249. print('X anotó %s  
puntos. O anotó %s puntos.' % (puntuaciones['X'],
```

```

        puntuaciones['O']))

250. if puntuaciones[baldosaJugador] > puntuaciones[baldosaComputadora]:
251.     print('¡Le ganaste a la computadora por %s puntos! ¡Felicitaciones!')
    % (puntuaciones[baldosaJugador] - puntuaciones[baldosaComputadora]) 252.

        puntuaciones elif[baldosajugador] < puntuaciones[baldosaordenador]:

253.     print('Perdiste. La computadora te ganó por %s puntos.' %
    (puntajes[computerTile] - puntajes[playerTile]))
254. más:
255.     print('¡El juego fue un empate!')

```

Comienza llamando a playGame(). Esta llamada de función no regresa hasta que finaliza el juego. La estructura de datos del tablero devuelta por playGame() se pasará a getScoreOfBoard() para contar las fichas X y O para determinar la puntuación final. Línea 249 muestra esta puntuación final.

Si hay más fichas del jugador que de la computadora, la línea 251 felicita al jugador por ganar. Si la computadora ganó, la línea 253 le dice al jugador que perdió. De lo contrario, la línea 255 le dice al jugador que el juego fue un empate.

PEDIR AL JUGADOR QUE JUGUE OTRA VEZ

Una vez finalizado el juego, se le pregunta al jugador si desea volver a jugar:

```

257. print('¿Quieres volver a jugar? (sí o no)') 258. if not
input().lower().startswith('y'): 259.
descanso

```

Si el jugador no escribe una respuesta que comience con la letra y, como sí o Sí o Y, entonces la condición en la línea 258 se evalúa como Verdadera y la línea 259 sale del ciclo while que comenzó en la línea 243, que termina el juego. De lo contrario, este tiempo

loop se repite de forma natural y se vuelve a llamar a playGame() para comenzar el siguiente juego.

RESUMEN

La IA de Reversegam puede parecer casi imbatible, pero esto no se debe a que la computadora sea más inteligente que nosotros; ¡es mucho más rápido! La estrategia que sigue es simple: muévase en la esquina si puede y, de lo contrario, haga el movimiento que voltee la mayor cantidad de fichas. Un ser humano podría hacer eso, pero llevaría mucho tiempo calcular cuántas fichas se voltearían para cada posible movimiento válido. Para la computadora, calcular este número es simple.

Este juego es similar a Sonar Treasure Hunt porque utiliza una cuadrícula como tablero. También es como el juego Tic-Tac-Toe porque hay una IA que planifica el mejor movimiento que puede hacer la computadora. Este capítulo introdujo solo un nuevo concepto: que las listas vacías, las cadenas en blanco y el número entero 0 se evalúan como Falso en el contexto de una condición. Aparte de eso, este juego usó conceptos de programación que ya conocías!

En el Capítulo 16, aprenderá cómo hacer que las IA jueguen juegos de computadora entre sí.

DEVOLVER SIMULACIÓN DE IA



El algoritmo de IA de Reversegam del Capítulo 15 es simple, pero me supera casi cada vez que lo juego. Debido a que la computadora puede procesar las instrucciones rápidamente, puede verificar fácilmente cada posición posible en el tablero y seleccionar el movimiento con la puntuación más alta. Me llevaría mucho tiempo encontrar el mejor movimiento este manera.

El programa Reversegam tenía dos funciones, getPlayerMove() y getComputerMove(), y ambas devolvían la jugada seleccionada como una lista de dos elementos en el formato [x, y]. Ambas funciones también tenían los mismos parámetros, la estructura de datos del tablero de juego y un tipo de ficha, pero los movimientos devueltos provenían de diferentes fuentes, ya sea el jugador o el algoritmo Reversegam.

¿Qué sucede cuando reemplazamos la llamada a getPlayerMove() con una llamada a getComputerMove()? Entonces el jugador nunca tiene que ingresar un movimiento; se decide por ellos. ¡La computadora está jugando contra sí misma!

En este capítulo, haremos tres nuevos programas en los que

la computadora juega contra sí misma, cada uno basado en el Programa Reversegam en el Capítulo 15:

- Simulación 1: AISim1.py realizará cambios en reversegam.py.
- Simulación 2: AISim2.py realizará cambios en AISim1.py.
- Simulación 3: AISim3.py realizará cambios en AISim2.py.

Los pequeños cambios de un programa a otro le mostrarán cómo convertir un juego de "jugador contra computadora" en una simulación de "computadora contra computadora". El programa final, AISim3.py, comparte la mayor parte de su código con reversegam.py pero tiene un propósito bastante diferente. La simulación no nos permite jugar a Reversegam pero nos enseña más sobre el juego en sí.

Puede escribir estos cambios usted mismo o descargarlos del sitio web ~~del libro~~ la [a](https://www.nostarch.com/inventwithpython/)

<https://www.nostarch.com/inventwithpython/>.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Simulaciones
- Porcentajes
- División entera
- La función ronda()
- Juegos de computadora contra computadora

HACIENDO JUGAR A LA COMPUTADORA CONTRA SI MISMO

Nuestro programa AISim1.py tendrá unos sencillos cambios para que el ordenador juegue contra sí mismo. Tanto las funciones getPlayerMove() como getComputerMove() toman una estructura de datos de tablero y el

ficha del jugador, y luego devolver el movimiento a realizar. Esta es la razón por la que `getComputerMove()` puede reemplazar a `getPlayerMove()` y el programa aún funciona. En el programa `AlSim1.py`, se llama a la función `getComputerMove()` para los jugadores X y O.

También hacemos que el programa deje de imprimir el tablero de juego para los movimientos que se realizan. Dado que un ser humano no puede leer los tableros de juego tan rápido como una computadora hace los movimientos, no es útil imprimir cada movimiento, por lo que solo imprimimos el tablero final al final del juego.

Estos son solo cambios mínimos en el programa, por lo que aún dirá cosas como El jugador irá primero. aunque la computadora esté jugando como computadora y como jugador.

Ejemplo de ejecución de la simulación 1

Esto es lo que ve el usuario cuando ejecuta el programa `AlSim1.py`. El texto ingresado por el jugador está en negrita.

```
¡Bienvenido a Reversegam!
La computadora irá primero.
12345678
+-----+
1|XXXXXXXXX|1
2|OXXXXXXX|2
3|XOXOOXXX|3
4|XXOOXOOX|4
5|XXOOXXXX|5
6|XXOXOXXX|6
7|XXXOXOXX|7
8|XXXXXXX|8
+-----+
12345678
X anotó 51 puntos. O anotó 13 puntos.
¡Le ganaste a la computadora por 38 puntos! ¡Felicidades!
¿Quieres jugar de nuevo? (sí o no)
no
```

Código fuente para la simulación 1

Guarde el antiguo archivo reversegam.py como AISim1.py de la siguiente manera:

1. Seleccione Archivo ➔ Guardar como.
2. Guarde este archivo como AISim1.py para que pueda realizar cambios sin afectar a reversegam.py. (En este punto, reversegam.py y AISim1.py todavía tienen el mismo código).
3. Realice cambios en AISim1.py y guarde ese archivo para conservar los cambios. (AISim1.py tendrá los nuevos cambios y reversegam.py tendrá el código original sin cambios).

Este proceso creará una copia de nuestro código fuente de Reversegam como un nuevo archivo en el que puede realizar cambios, mientras deja el juego original de Reversegam igual (es posible que desee jugarlo nuevamente para probarlo). Por ejemplo, cambie la línea 216 en AISim1.py a lo siguiente (el cambio está en negrita):

216. move = getComputerMove(tablero, playerTile)

Ahora ejecuta el programa. Tenga en cuenta que el juego aún le pregunta si desea ser X u O, pero no le pedirá que ingrese ningún movimiento. Cuando reemplaza la función getPlayerMove() con la función getComputerMove() , ya no llama a ningún código que tome esta entrada del jugador. El jugador aún presiona ENTER después de los movimientos de la computadora original (debido a la entrada ("Presione Enter para ver el movimiento de la computadora") en la línea 232), pero el juego continúa .

¡sí mismo!



Hagamos otros cambios en `AISim1.py`. Cambie las siguientes líneas en negrita. Los cambios comienzan en la línea 209. La mayoría de estos cambios simplemente comentan el código, lo que significa convertir el código en un comentario para que no se ejecute.

Si obtiene errores después de escribir este código, compare el código que escribió con el código del libro con la herramienta de diferencias en línea en <https://www.nostarch.com/inventwithpython#diff>.

`AISim1.py`

```

207.     elif turn == 'player': # Turno del jugador
208.         if playerValidMoves != []: #if showHints:
209.
210.             # tableroJugadasVálidas = obtenerTableroConJugadasVálidas(tablero,
211.                                         FichaJugador) # dibujarTablero(tableroJugadasVálidas) #else:
212.
213.             # dibujarTablero(tablero)
214.             #imprimirPuntuación(tablero, baldosaJugador, baldosaComputadora)
215.
216.             move = getComputerMove(board, playerTile) #if move
217.             == 'quit': # print('¡Gracias por jugar!') # sys.exit() #
218.             Finaliza el programa. #elif mover == 'pistas':
219.
220.

```

```

221.      # mostrarSugerencias = no mostrarSugerencias
222.      # Seguir
223.      #más:
224.      hacerJugada(tablero, baldosaJugador, jugada[0], jugada[1])
225.      turno = 'computadora'
226.
227.      elif turno == 'computadora': # Turno de la computadora
228.          si la computadora realiza movimientos válidos != None:
229.              []: #drawBoard(board) #printScore(board,
230.                  playerTile, computerTile)
231.
232.          #input('Presiona Enter para ver el movimiento de la computadora.')
233.          move = getComputerMove(board, computerTile) makeMove(board,
234.              computerTile, move[0], move[1]) turn = 'player'
235.
236.
237.
238.
239. print('¡Bienvenido a Reversegam!') 240.

```

241. BaldosaJugador, BaldosaComputadora = ['X', 'O'] #enterBaldosaJugador()

Eliminación de las indicaciones del reproductor y adición de un reproductor de computadora Como puede ver, el programa AISim1.py es prácticamente igual que el programa Reversegam original, excepto que reemplazamos la llamada a getPlayerMove() con una llamada a getComputerMove(). también hemos realizado algunos cambios en el texto que se imprime en la pantalla para que el juego sea más fácil de seguir. ¡Cuando ejecutas el programa, todo el juego se juega en menos de un segundo!

Una vez más, la mayoría de los cambios son simplemente comentar el código. Dado que la computadora juega contra sí misma, el programa ya no necesita ejecutar código para obtener movimientos del jugador o mostrar el estado del tablero. Todo esto se omite para que el tablero se muestre solo al final del juego. Nosotros

comente el código en lugar de eliminarlo porque es más fácil restaurar el código descomentándolo si necesitamos reutilizarlo más adelante.

Comentamos las líneas 209 a 214 porque no necesitamos para dibujar un tablero de juego para el jugador, ya que no jugará. También comentamos las líneas 217 a 223 porque no necesitamos verificar si el jugador ingresa a salir o cambia el modo de sugerencias. Pero necesitamos eliminar la sangría de la línea 224 por cuatro espacios ya que estaba en el bloque else que acabamos de comentar. Las líneas 229 a 232 también dibujan el tablero de juego para el jugador, así que también comentamos esas líneas.

El único código nuevo está en las líneas 216 y 241. En la línea 216, simplemente sustituimos la llamada a obtenerJugadaJugador() con obtenerJugadaComputadora(), como se discutió anteriormente. En la línea 241, en lugar de preguntarle al jugador si quiere ser X u O, simplemente siempre asignamos 'X' a playerTile y 'O' a computerTile. (Sin embargo, ambos jugadores serán jugados por la computadora, por lo que puede cambiar el nombre de playerTile a computerTile2 o secondComputerTile si lo desea). Ahora que tenemos la computadora jugando contra sí misma, podemos seguir modificando nuestro programa para que sea más interesante. cosas.

HACIENDO JUGAR A LA COMPUTADORA SÍ MISMO VARIAS VECES

Si creamos un nuevo algoritmo, podríamos compararlo con la IA implementada en getComputerMove() y ver cuál es mejor. Sin embargo, antes de hacerlo, necesitamos una forma de evaluar a los jugadores. No podemos evaluar qué IA es mejor basándonos en un solo juego, por lo que deberíamos hacer que las IA jueguen entre sí más de una vez. Para hacer eso, haremos algunos cambios en el

código fuente. Siga estos pasos para hacer AISim2.py:

1. Seleccione Archivo ➔ Guardar como.
2. Guarde este archivo como AISim2.py para que pueda realizar cambios sin afectar a AISim1.py. (En este punto, AISim1.py y AISim2.py todavía tienen el mismo código).

Ejemplo de ejecución de la simulación

2 Esto es lo que ve el usuario cuando ejecuta el AISim2.py programa.

```
¡Bienvenido a Reversegam!
#1: X anotó 45 puntos. O anotó 19 puntos.
#2: X anotó 38 puntos. O anotó 26 puntos.
#3: X anotó 20 puntos. O anotó 44 puntos.
#4: X anotó 24 puntos. O anotó 40 puntos.
#5: X anotó 8 puntos. O anotó 56 puntos. --
recorte-- #249: X anotó 24 puntos. O anotó 40
puntos.
#250: X anotó 43 puntos. O anotó 21 puntos.
X victorias: 119 (47,6%)
O gana: 127 (50,8%)
Empates: 4 (1,6%)
```

Debido a que los algoritmos incluyen aleatoriedad, su ejecución no tendrá exactamente los mismos números.

Código fuente para la simulación 2

Cambie el código en AISim2.py para que coincida con lo siguiente. Asegúrese de cambiar el código línea por línea en orden numérico. Si obtiene errores después de escribir este código, compare el código que escribió con el código del libro con la herramienta de diferencias en línea en <https://www.nostarch.com/inventwithpython#diff>.

AISim2.py

```
235.         turno = 'jugador'
236.
237. NUM_JUEGOS = 250
238. xGanancias = oGanancias = empates = 0
239. print('¡Bienvenido a Reversegam!')
240.
241. BaldosaJugador, BaldosaComputadora = ['X', 'O'] #enterBaldosaJugador()
242.
243. for i in range(NUM_GAMES): #while True: 244.
finalBoard = playGame(playerTile, computerTile) 245.

246. # Muestra la puntuación final.
247. #dibujarTablero(tablerofinal) 248.
puntuaciones = obtenerPuntajeDelTablero(tablerofinal)
249. print('#%s: X obtuvo %s puntos. O obtuvo %s puntos.' % (i + 1,
    puntuaciones['X'],
    puntuaciones['O'])) 250. if puntuaciones[baldosaJugador] >
251.     puntuaciones[baldosaComputadora]: xGanancias += 1 #print('¡Le ganaste a la computadora por %s puntos!
    ¡Felicidades!' % (puntuaciones[playerTile] -
    puntuaciones[computerTile])) 252. elif
puntuaciones[playerTile] < puntuaciones[computerTile]: 253.
oWins += 1 #print('Perdiste. La computadora te ganó por %s
    puntos.' % (puntuaciones[computerTile] - puntuaciones[playerTile]))

254. más:
255.     empates += 1 #print('El juego fue un empate!')
256.
257. #print('¿Quieres volver a jugar? (sí o no)') 258. #if not
input().lower().startswith('y'): 259. # break

260.
261. print('X gana: %s (%s%%)' % (xGana, ronda(xGana / NUM_JUEGOS * 100, 1))) 262.
print('O gana: %s (%s%%)' % (oWins, round(oWins / NUM_GAMES * 100, 1))) 263.
print('Empates: %s (%s%%)' % (empates, round(empates / NUM_GAMES * 100, 1)))
```

Si esto es confuso, siempre puede descargar el código fuente de AISim2.py del sitio web del libro en

[https://www.nostarch.com/inventwithpython/.](https://www.nostarch.com/inventwithpython/)

Seguimiento de varios juegos

La información principal que queremos de la simulación es cómo muchas victorias para X, victorias para O y empates en un cierto número de juegos. Estos se pueden rastrear en cuatro variables, que se crean en las líneas 237 y 238.

237. NUM_JUEGOS = 250

238. xGanancias = oGanancias = empates = 0

La constante NUM_GAMES determina cuántos juegos jugará la computadora. Ha agregado las variables xWins, oWins y ties para realizar un seguimiento de cuándo gana X , cuándo gana O y cuándo empatan. Puede encadenar la instrucción de asignación para establecer empates iguales a 0 y oWins iguales a empates, luego xWins iguales a oGana. Esto establece las tres variables en 0.

NUM_GAMES se usa en un ciclo for que reemplaza el ciclo del juego en la línea 243:

243. for i in range(NUM_GAMES): #while True:

El ciclo for ejecuta el juego la cantidad de veces en NUM_GAMES. Esto reemplaza el ciclo while que solía repetirse hasta que el jugador decía que no quería jugar otro juego.

En la línea 250, una declaración if compara el puntaje de los dos jugadores, y las líneas 251 a 255 en los bloques if-elif-else incrementan las variables xWins , oWins y ties al final de cada juego antes de regresar para comenzar uno nuevo. juego:

250. if puntuaciones[playerTile] > puntuaciones[computerTile]:

251. xWins += 1 #print('¡Le ganaste a la computadora por %s puntos!

 ¡Felicitaciones!' % (puntuaciones[playerTile] -

```

        puntuaciones[computerTile]))  

252. elif puntuaciones[playerTile] < puntuaciones[computerTile]:  

253. oWins += 1 #print('Perdiste la computadora te ganó por %s puntos.' % (Scores[computerTile] - puntuaciones[playerTile]))  

254. más:  

255.     empates += 1 #print('¡El juego fue un empate!')

```

Comentamos los mensajes impresos originalmente en el bloque, por lo que ahora solo se imprime un resumen de una línea de los puntajes para cada juego. Usaremos las variables xWins , oWins y ties más adelante en el código para analizar cómo se desempeñó la computadora contra sí misma.

Comentando las llamadas a la función print() También comentó las líneas 247 y 257 a la 259. Al hacerlo, eliminó la mayoría de las llamadas a la función print() del programa, así como las llamadas a dibujarTablero(). No necesitamos ver cada uno de los juegos ya que hay muchos en juego.

El programa aún ejecuta todos los juegos en su totalidad usando la IA que codificamos, pero solo se muestran los puntajes resultantes.

Después de ejecutar todos los juegos, el programa muestra cuántos juegos ganó cada lado y las líneas 251 a 253 imprimen información sobre las ejecuciones del juego.

Imprimir cosas en la pantalla ralentiza la computadora, pero ahora que eliminó ese código, la computadora puede ejecutar un juego completo de Reversegam en aproximadamente uno o dos segundos. Cada vez que el programa imprimía una de esas líneas con el puntaje final, ejecutaba un juego completo (verificando alrededor de 50 o 60 movimientos individualmente para elegir el que obtiene la mayor cantidad de puntos). Ahora que la computadora no tiene que hacer tanto trabajo, puede funcionar mucho más rápido.

Los números que el programa imprime al final son estadísticas, números que se utilizan para resumir cómo funcionan los juegos .

fueron jugados. En este caso, mostramos los puntajes resultantes de cada juego jugado y los porcentajes de victorias y empates para las fichas.

Uso de porcentajes para calificar los Al Los
Los porcentajes son una parte de una cantidad total. Los porcentajes de un todo pueden variar de 0 por ciento a 100 por ciento. Si tuviera el 100 por ciento de un pastel, tendría todo el pastel; si tuviera el 0 por ciento de un pastel, no tendría ningún pastel; y si tuvieras el 50 por ciento del pastel, tendrías la mitad.

Podemos calcular porcentajes con división. Para obtener un porcentaje, divide la parte que tienes por el total y luego multiplícalo por 100. Por ejemplo, si X ganó 50 de 100 juegos, calcularías la expresión $50/100$, que se evalúa como 0,5. Multiplique esto por 100 para obtener el porcentaje (en este caso, 50 por ciento).

Si X ganó 100 de 200 juegos, calcularía el porcentaje con $100/200$, que también se evalúa como 0,5. Cuando multiplicas 0,5 por 100 para obtener el porcentaje, obtienes el 50 por ciento.

Ganar 100 de 200 juegos es el mismo porcentaje (es decir, la misma porción) que ganar 50 de 100 juegos.

En las líneas 261 a 263, usamos porcentajes para imprimir información sobre los resultados de los juegos:

```
261. print('X gana: %s (%s%%)' % (xGana, round(xGana / NUM_JUEGOS * 100, 1)))  
262. print('O gana: %s (%s%%)' % (oWins, round(oWins / NUM_GAMES * 100, 1)))  
263. print('Empates: %s (%s%%)' % (empates, round(empates / NUM_GAMES * 100,  
1)))
```

Cada instrucción `print()` tiene una etiqueta que le dice al usuario si los datos que se imprimen corresponden a victorias X, victorias O o empates. Usamos

interpolación de cadenas para insertar el número de juegos ganados o empatados y luego insertar el porcentaje calculado que las victorias o los empates componen del total de juegos, pero puede ver que no estamos simplemente dividiendo xWins, oWins o empates por el total de juegos y multiplicar por 100. Esto se debe a que queremos imprimir solo un lugar decimal para cada porcentaje, lo que no podemos hacer con la división normal.

La división se evalúa como un número de coma flotante

Cuando usa el operador de división (/), la expresión siempre se evaluará como un número de coma flotante. Por ejemplo, la expresión 10/2 se evalúa como el valor de punto flotante 5.0, no como el valor entero 5.

Es importante recordar esto, porque agregar un número entero a un valor de punto flotante con el operador de suma + también siempre se evaluará como un valor de punto flotante. Por ejemplo, 3 + 4,0 se evalúa como el valor de punto flotante 7,0, no como el número entero 7.

Ingrese el siguiente código en el shell interactivo:

```
>>> correo basura = 100 / 4
>>> correo basura
25,0

>>> correo no deseado = correo no deseado + 20
>>> correo no deseado
45,0
```

En el ejemplo, el tipo de datos del valor almacenado en spam siempre es un valor de punto flotante. Puede pasar el valor de coma flotante a la función int() , que devuelve una forma entera del valor de coma flotante. Pero esto siempre redondeará el valor del punto flotante hacia abajo. Por ejemplo, las expresiones int(4.0), int(4.2),

e int(4.9) todos dan como resultado 4, no 5. Pero en AlSim2.py, necesitamos redondear cada porcentaje al lugar de las décimas. Como no podemos simplemente dividir para hacer esto, necesitamos usar la función round() .

La función redonda ()

La función round() redondea un número de punto flotante al número entero más cercano. Ingrese lo siguiente en el shell interactivo:

```
>>> ronda(10.0)
10
>>> ronda(10.2)
10
>>> ronda(8.7)
9
>>> ronda(3.4999)
3
>>> ronda(2.5422, 2)
2.54
```

La función round() también tiene un segundo parámetro opcional, donde puede especificar a qué lugar desea redondear el número. Esto hará que el número redondeado sea un número de coma flotante en lugar de un número entero. Por ejemplo, la expresión round(2.5422, 2) da como resultado 2.54 y round(2.5422, 3) da como resultado 2.542. En las líneas 261 a 263 de AlSim2.py, usamos este round() con un parámetro de 1 para encontrar la fracción de juegos ganados o empatados por X y O hasta un decimal, lo que nos da porcentajes precisos.

COMPARACIÓN DE DIFERENTES IA ALGORITMOS

Con solo unos pocos cambios, podemos hacer que la computadora juegue cientos de juegos contra sí misma. En este momento, cada jugador gana aproximadamente la mitad de los juegos, ya que ambos ejecutan exactamente el mismo algoritmo para los movimientos. Pero si agregamos diferentes algoritmos, podemos ver si una IA diferente ganará más juegos.

Agreguemos algunas funciones nuevas con nuevos algoritmos. Pero primero, en AISim2.py seleccione Archivo Guardar como para guardar este nuevo archivo como AISim3.py.

Cambiaremos el nombre de la función getComputerMove() a getCornerBestMove(), ya que este algoritmo primero intenta moverse en las esquinas y luego elige el movimiento que volteá la mayor cantidad de fichas. Llamaremos a esta estrategia el mejor algoritmo de la esquina. También agregaremos varias otras funciones que implementan diferentes estrategias, incluido un algoritmo de peor movimiento que obtiene el movimiento con peor puntaje; un algoritmo de movimiento aleatorio que obtiene cualquier movimiento válido; y un algoritmo de esquina-lado-mejor, que funciona igual que la IA de esquina-mejor excepto que busca un movimiento lateral después de un movimiento de esquina y antes de realizar el movimiento de mayor puntuación.

En AISim3.py, la llamada a getComputerMove() en la línea 257 se cambiará a getCornerBestMove(), y getComputerMove () de la línea 274 se convertirá en getWorstMove(), que es la función que escribiremos para el algoritmo de peor movimiento. De esta manera, tendremos el mejor algoritmo regular de la esquina en contra de un algoritmo que elige a propósito el movimiento que volteará la menor cantidad de fichas.

Código fuente para la simulación 3

A medida que ingresa el código fuente de AISim3.py en su copia renombrada de AISim2.py, asegúrese de escribir su código línea por línea en orden numérico para que los números de línea coincidan. Si obtiene errores después de escribir este código, compare el código que escribió con

el código del libro con la herramienta de diferencias en línea en

<https://www.nostarch.com/inventwithpython#diff>.

AlSim3.py

```
162. def obtenerMejorMovimientoEsquina(tablero, mosaico de
computadora): --snip-- 184. def obtenerPeorJugada(tablero,
mosaico): 185. # Devuelve el movimiento que volteo la menor
cantidad de mosaicos. 186. jugadasposibles = obtenerJugadasVálidas(tablero,
mosaico) 187. random.shuffle(jugadasposibles) # Aleatoriza el orden de las
jugadas.
188.
189. # Encuentra el movimiento con la puntuación más baja
 posible. 190. peor puntuación = 64
191. para x, y en movimientos posibles: 192.
    copiaTablero = obtenerCopiaTablero(tablero)
193.     hacerMovimiento(CopiaTablero, mosaico, x, y)
194.     puntuación = obtenerPuntuaciónTablero(copiaTablero)
195.     [mosaico] si puntuación < peorPuntuación:
196.         peorMovimiento = [x, y]
197.         peorPuntuación = puntuación
198.
199. volver PeorMovimiento
200.
201. def obtenerJugadaAleatoria(tablero, loseta):
202.     jugadasposibles = obtenerJugadasValidas(tablero, loseta) 203.
return elección.aleatoria(jugadasposibles) 204.
205. def isOnSide(x, y): 206.
return x == 0 o x == ANCHO - 1 o y == 0 o y == ALTO - 1
207.
208. def obtenerMejorJugadaLadoEsquina(tablero, loseta):
209. # Devuelve una jugada de esquina, una jugada lateral o la mejor jugada. 210.
jugadasposibles = obtenerJugadasVálidas(tablero, mosaico) 211.
random.shuffle(jugadasposibles) # Aleatoriza el orden de las jugadas.
212.
213. # Siempre ve por una esquina si esta disponible. 214.
for x, y en posiblesMovidas: 215. if isOnCorner(x, y): return
[x, y]
216.
```

```

217.

218. # Si no hay movimiento de esquina para hacer, devuelve un movimiento
lateral. 219. para x, y en movimientos posibles: 220.

    si está del lado (x, y):
221.         devuelve [x, y]
222.

223. return obtenerMejorMovimientoEsquina(tablero, mosaico) # Hacer lo que la IA normal
    haría.

224.

225. def printScore(tablero, playerTile, computerTile): --snip--

257.         mover = obtenerEsquinaMejorJugada(tablero, baldosaJugador)
--recorte--
274.         mover = obtenerPeorJugada(tablero, mosaico de computadora)

```

Ejecutar AISim3.py da como resultado el mismo tipo de salida que AISim2.py, excepto que diferentes algoritmos reproducirán el juegos.

Cómo funcionan las IA en Simulation 3

Las funciones obtenerMejorJugadaEsquina(), obtenerPeorJugada(), obtenerJugadaAleatoria(), y getCornerSideBestMove() son similares entre sí pero usan estrategias ligeramente diferentes para jugar. Uno de ellos usa la nueva función isOnSide() . Esto es similar a nuestra función isOnCorner() , pero verifica los espacios a lo largo del costado del tablero antes de seleccionar el movimiento de mayor puntuación.

La esquina-mejor IA

Ya tenemos el código para una IA que elige moverse en una esquina y luego elige el mejor movimiento posible, ya que eso es lo que hace getComputerMove() . Simplemente podemos cambiar el nombre de getComputerMove() a algo más descriptivo, así que cambie la línea 162 para cambiar el nombre de nuestra función a getCornerBestMove():

```
162. def obtenerMejorMovimientoEsquina(tablero, mosaicoDeComputadora):
```

Dado que getComputerMove() ya no existe, necesitamos actualizar el código en la línea 257 para getCornerBestMove():

```
257.     mover = obtenerEsquinaMejorJugada(tablero, baldosaJugador)
```

Ese es todo el trabajo que tenemos que hacer para esta IA, así que avancemos en.

La peor jugada de IA

La IA del peor movimiento simplemente encuentra el movimiento con la menor cantidad de puntos y lo devuelve. Su código es muy parecido al código que usamos para encontrar el movimiento de mayor puntuación en nuestro algoritmo getComputerMove() original :

```
184. def obtenerPeorJugada(tablero, ficha):
185. # Devuelve la jugada que voltea la menor cantidad de fichas. 186.
    jugadasPosibles = obtenerJugadasVálidas(tablero, mosaico) 187.
    random.shuffle(jugadasPosibles) # Aleatoriza el orden de las jugadas.
188.
189. # Encuentra el movimiento con la puntuación más baja
    posible. 190. peor puntuación = 64
191. para x, y en posiblesMovimientos:
192.     copiaTablero = obtenerCopiaTablero(tablero)
193.     hacerMovimiento(CopiaTablero, mosaico, x, y)
194.     puntuación = obtenerPuntuaciónTablero(copiaTablero)
195.     [mosaico] si puntuación < peorPuntuación:
196.         peorMovimiento = [x, y]
197.         peorPuntuación = puntuación
198.
199. volver PeorMovimiento
```

El algoritmo para obtenerPeorJugada() comienza de la misma manera para las líneas 186 y 187, pero luego se desvía en la línea 190. Configuramos una variable para contener la peorPuntuación en lugar de la mejorPuntuación y la configuramos

a 64, porque ese es el número total de posiciones en el tablero y la mayor cantidad de puntos que se podrían anotar si se llenara todo el tablero. Las líneas 191 a 194 son iguales al algoritmo original, pero luego la línea 195 comprueba si la puntuación es inferior a la peor puntuación en lugar de si la puntuación es mayor. Si la puntuación es menor, la peor jugada se reemplaza con la jugada en el tablero que el algoritmo está probando actualmente, y la peor puntuación también se actualiza.

Entonces la función devuelve peor Jugada.

Finalmente, getComputerMove() de la línea 274 debe cambiarse a

obtenerPeorJugada():

274. mover = obtenerPeorJugada(tablero, mosaico de computadora)

Cuando haya terminado y ejecute el programa, getCornerBestMove() y getWorstMove() se enfrentarán entre sí.

La IA de movimiento aleatorio

La IA de movimientos aleatorios simplemente encuentra todos los movimientos posibles válidos y luego elige uno al azar.

201. def obtenerJugadaAleatoria(tablero, loseta): 202.
jugadasposibles = obtenerJugadasVálidas(tablero, loseta) 203. return
elección.aleatoria(jugadasposibles)

Utiliza getValidMoves(), tal como lo hacen todas las demás IA, y luego usa choice() para devolver uno de los posibles movimientos en el devuelto . lista.

Comprobación de movimientos laterales

Antes de entrar en los algoritmos, veamos una nueva función auxiliar que hemos agregado. La función auxiliar isOnSide() es como la

isOnCorner() , excepto que verifica si un movimiento está en marcha los lados de una tabla:

```
205. def isOnSide(x, y):
206.     return x == 0 o x == ANCHO - 1 o y == 0 o y == ALTO - 1
```

Tiene una expresión booleana que verifica si el valor x o el valor y de los argumentos de coordenadas que se le pasan es igual a 0 o 7. Cualquier coordenada con un 0 o un 7 está en el borde del tablero.

Usaremos esta función a continuación en la mejor IA del lado de la esquina.

La mejor IA del lado de la esquina

La mejor IA del lado de la esquina funciona de manera muy similar a la mejor IA de la esquina, por lo que podemos reutilizar parte del código que ya hemos ingresado. Definimos esta IA en la función getCornerSideBestMove():

```
208. def obtenerMejorJugadaLadoEsquina(tablero,
loseta): 209. # Devuelve una jugada de esquina, una jugada lateral o la
mejor jugada. 210. jugadasposibles = obtenerJugadasVálidas(tablero,
mosaico) 211. random.shuffle(jugadasposibles) # Aleatoriza el orden de las jugadas. 212.

213. # Siempre ve por una esquina si esta disponible.
214. for x, y en posiblesMovidas: if estáEnEsquina(x,
215.         y): return [x, y]
216.
217.

218. # Si no hay movimiento de esquina para hacer, devuelve un movimiento
lateral. 219. para x, y en movimientos posibles: 220.
220.         si está del lado (x, y):
221.             devuelve [x, y]
222.

223. return obtenerMejorMovimientoEsquina(tablero, mosaico) # Hacer lo que la IA normal
haría.
```

Las líneas 210 y 211 son las mismas que en nuestra esquina, la mejor IA,

y las líneas 214 a 216 son idénticas a nuestro algoritmo para verificar un movimiento de esquina en nuestra IA `getComputerMove()` original . Si no hay un movimiento de esquina, entonces las líneas 219 a 221 buscan un movimiento lateral usando la función auxiliar `isOnSide()` . Una vez que se ha verificado la disponibilidad de todos los movimientos de esquina y laterales, si todavía no hay movimiento, entonces reutilizamos nuestra función `getCornerBestMove()` . Ya que hay no hubo movimientos de esquina antes y todavía no habrá ninguno cuando el código llegue a la función `getCornerBestMove()` , esta función simplemente buscará el movimiento con la puntuación más alta para hacer y devolverlo.

La tabla 16-1 repasa los nuevos algoritmos que hemos creado.

Tabla 16-1: Funciones utilizadas para las IA de Reversegam

| Función | Descripción |
|--|--|
| <code>obtenerEsquinaMejorMovimiento()</code> | Tome un movimiento de esquina si está disponible. Si no hay esquina, encuentra el movimiento de mayor puntuación. |
| <code>getCornerSideBestMove()</code> | Realiza un movimiento de esquina si está disponible. Si no hay esquina, toma un espacio al costado. Si no hay lados disponibles, use el algoritmo regular <code>getCornerBestMove()</code> . |
| <code>obtenerMovimientoAleatorio()</code> | Elija al azar un movimiento válido para hacer. |
| <code>obtenerPeorJugada()</code> | Tome la posición que resulte en la menor cantidad de fichas volteadas. |

Ahora que tenemos nuestros algoritmos, podemos enfrentarlos contra El uno al otro.

Comparación de las IA

Hemos escrito nuestro programa para que la IA de la esquina mejor juegue contra la IA del peor movimiento. Podemos ejecutar el programa para simular qué tan bien funcionan las IA entre sí y analizar

los resultados con las estadísticas impresas.

Además de estas dos IA, hemos creado otras que no utilizamos. Estas IA existen en el código, pero no se utilizan, por lo que si queremos ver cómo les va en una partida, tendremos que editar el código para llamarlas. Como ya tenemos una comparación configurada, veamos cómo funciona la IA de peor movimiento contra la IA de esquina mejor.

IA de peor movimiento vs. IA de esquina mejor

Ejecute el programa para comparar la función obtenerEsquinaMejorJugada() con la función obtenerPeorJugada(). Como era de esperar, la estrategia de voltear la menor cantidad de fichas en cada turno perderá la mayoría de los juegos:

X victorias: 206 (82,4%)

O gana: 41 (16,4%)

Empates: 3 (1,2%)

¡Lo sorprendente es que a veces la estrategia del peor movimiento funciona! En lugar de ganar el 100 por ciento de las veces, el algoritmo de la función obtenerEsquinaMejorJugada() gana solo alrededor del 80 por ciento de las veces. Aproximadamente 1 de cada 5 veces, pierde!

Este es el poder de ejecutar programas de simulación: puede encontrar ideas novedosas que le llevarían mucho más tiempo darse cuenta si solo estuviera jugando juegos por su cuenta. ¡La computadora es mucho más rápida!

IA de movimiento aleatorio frente a la mejor IA de esquina

Probemos una estrategia diferente. Cambie la llamada a obtenerPeorJugada() en la línea 274 a obtenerJugadaAleatoria():

274. move = getRandomMove(tablero, computerTile)

Cuando ejecute el programa ahora, se verá algo como esto:

```
¡Bienvenido a Reversegam!
#1: X anotó 32 puntos. O anotó 32 puntos.
#2: X anotó 44 puntos. O anotó 20 puntos.
#3: X anotó 31 puntos. O anotó 33 puntos.
#4: X anotó 45 puntos. O anotó 19 puntos.
#5: X anotó 49 puntos. O anotó 15 puntos. --recorte--
#249: X anotó 20 puntos. O anotó 44 puntos.

#250: X anotó 38 puntos. O anotó 26 puntos.
X victorias: 195 (78,0%)
O gana: 48 (19,2%)
Empates: 7 (2,8%)
```

El algoritmo de movimiento aleatorio getRandomMove() se desempeñó ligeramente mejor contra el algoritmo de mejor esquina que el peor algoritmo de movimiento. Esto tiene sentido porque hacer elecciones inteligentes suele ser mejor que simplemente elegir movimientos al azar, pero hacer elecciones al azar es ligeramente mejor que elegir el peor movimiento a propósito.

La mejor IA de esquina lateral frente a la mejor IA de esquina

Elegir un espacio de esquina si está disponible es una buena idea, porque una ficha en la esquina nunca se puede voltear. Poner un mosaico en los espacios laterales también parece ser una buena idea, ya que hay menos formas de rodearlo y voltearlo. Pero, ¿este beneficio supera el dejar pasar movimientos que voltearían más fichas?

Averigüémoslo comparando el mejor algoritmo de la esquina con el mejor algoritmo del lado de la esquina.

Cambie el algoritmo en la línea 274 para usar
getEsquinaLadoMejorMovimiento():

274. mover = obtenerLadoEsquinaMejorMovimiento(tablero, mosaicoDeComputadora)

A continuación, ejecute el programa de nuevo:

¡Bienvenido a Reversegam!
#1: X anotó 27 puntos. O anotó 37 puntos.
#2: X anotó 39 puntos. O anotó 25 puntos.
#3: X anotó 41 puntos. O anotó 23 puntos. --
recorte-- #249: X anotó 48 puntos. O anotó 16
puntos.
#250: X anotó 38 puntos. O anotó 26 puntos.
X victorias: 152 (60,8%)
O gana: 89 (35,6%)
Empates: 9 (3,6%)

¡Guau! Eso es inesperado. Parece que elegir los espacios laterales sobre un espacio que voltear más fichas es una mala estrategia. El beneficio del espacio lateral no supera el costo de voltear menos fichas del oponente. ¿Podemos estar seguros de estos resultados?

Ejecutemos el programa nuevamente, pero esta vez juegue 1,000 juegos cambiando la línea 278 a NUM_GAMES = 1000 en AISim3.py. El programa ahora puede tardar unos minutos en ejecutarse en su computadora, ¡pero le tomaría semanas hacerlo a mano!

Verá que las estadísticas más precisas de la serie de 1000 juegos son casi las mismas que las estadísticas de la serie de 250 juegos. Parece que elegir el movimiento que voltear la mayor cantidad de fichas es una mejor idea que elegir un lado para seguir adelante.

Acabamos de usar la programación para averiguar qué estrategia de juego funciona mejor. Cuando escuchas que los científicos usan modelos de computadora, esto es lo que están haciendo. Usan una simulación para recrear algún proceso del mundo real y luego hacen pruebas en la simulación para obtener más información sobre el mundo real.

RESUMEN

Este capítulo no cubrió un juego nuevo, pero modeló varias estrategias para Reversegam. Si pensáramos que hacer movimientos laterales en Reversegam era una buena idea, tendríamos que pasar semanas, incluso meses, jugando cuidadosamente juegos de Reversegam a mano y anotando los resultados para probar esta idea. Pero si sabemos cómo programar una computadora para jugar Reversegam, entonces podemos hacer que pruebe diferentes estrategias para nosotros.

Si lo piensa, ¡la computadora está ejecutando millones de líneas de nuestro programa Python en segundos! Tus experimentos con las simulaciones de Reversegam pueden ayudarte a aprender más sobre cómo jugarlo en la vida real.

De hecho, este capítulo sería un buen proyecto para una feria de ciencias. Podría investigar qué conjunto de movimientos conduce a la mayor cantidad de victorias frente a otros conjuntos de movimientos, y podría hacer una hipótesis sobre cuál es la mejor estrategia. Después de ejecutar varias simulaciones, podría determinar qué estrategia funciona mejor. ¡Con la programación, puedes hacer un proyecto de feria de ciencias a partir de una simulación de cualquier juego de mesa! Y todo es porque sabes cómo instruir a la computadora para que lo haga, paso a paso, línea a línea. Puede hablar el idioma de la computadora y hacer que haga grandes cantidades de procesamiento de datos y procesamiento de números por usted.

Eso es todo sobre los juegos basados en texto de este libro. Los juegos que solo usan texto pueden ser divertidos, aunque sean simples. Pero la mayoría de los juegos modernos usan gráficos, sonido y animación para hacerlos más emocionantes. En el resto de los capítulos de este libro, aprenderá a crear juegos con gráficos usando un módulo de Python llamado pygame.

17

CREAR GRÁFICOS



Hasta ahora, todos nuestros juegos han usado solo texto. El texto se muestra en la pantalla como salida y el jugador ingresa el texto como entrada. El solo uso de texto hace que la programación sea fácil de aprender. Pero en este capítulo, crearemos algunos programas más emocionantes con gráficos avanzados usando el módulo pygame .

Los capítulos 17, 18, 19 y 20 le enseñarán cómo usar pygame para crear juegos con gráficos, animaciones, entrada de mouse y sonido. En estos capítulos, escribiremos código fuente para programas simples que demuestran los conceptos de pygame . Luego, en el Capítulo 21, reuniremos todos los conceptos que aprendimos para crear un juego

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Instalación de pygame
- Colores y fuentes en Pygame
- Gráficos con alias y suavizados
- Atributos
- Los tipos de datos pygame.font.Font, pygame.Surface, pygame.Rect y pygame.PixelArray
- Funciones constructoras

- funciones de dibujo de pygame
- El método blit() para objetos de superficie
- Eventos

INSTALANDO PYGAME

El módulo pygame ayuda a los desarrolladores a crear juegos al facilitar el dibujo de gráficos en la pantalla de su computadora o agregar música a los programas. El módulo no viene con Python, pero al igual que Python, se puede descargar gratis. Descarga pygame en <https://www.nostarch.com/inventwithpython/>, y siga las instrucciones para su sistema operativo.

Una vez que el archivo de instalación termine de descargarse, ábralo y siga las instrucciones hasta que pygame haya terminado de instalarse.

Para verificar que se instaló correctamente, ingrese lo siguiente en el shell interactivo:

```
>>> importar pygame
```

Si no aparece nada después de presionar ENTER, entonces sabrá que pygame se instaló correctamente. Si aparece el error ImportError: No module named pygame , intente instalar pygame nuevamente (y asegúrese de haber escrito import pygame correctamente).

NOTA

Al escribir sus programas de Python, no guarde su archivo como pygame.py. Si lo hace, la línea import pygame importará ~~pygame~~ pygame en lugar del módulo real, y ninguno de sus códigos

HOLA MUNDO EN PYGAME

Primero, haremos un nuevo programa Hello World de pygame como el

uno que creaste al principio del libro. Esta vez, usarás pygame para hacer "¡Hola mundo!" aparecer en una ventana gráfica en lugar de como texto. Solo usaremos pygame para dibujar algunas formas y líneas en la ventana de este capítulo, pero pronto usará estas habilidades para crear su primer juego animado.

El módulo pygame no funciona bien con el interactivivo shell, por lo que solo puede escribir programas usando pygame en un editor de archivos; no puede enviar instrucciones de una en una a través del shell interactivo.

Además, los programas Pygame no usan las funciones print() o input(). No hay entrada o salida de texto. En cambio, pygame muestra la salida dibujando gráficos y texto en una ventana separada. La entrada a pygame proviene del teclado y el mouse a través de eventos, que se describen en "Eventos y el bucle de juego" en la página 270.

MUESTRA DE EJECUCIÓN DE PYGAME HELLO MUNDO

Cuando ejecute el programa gráfico Hello World, debería ver una nueva ventana que se parece a la Figura 17-1.

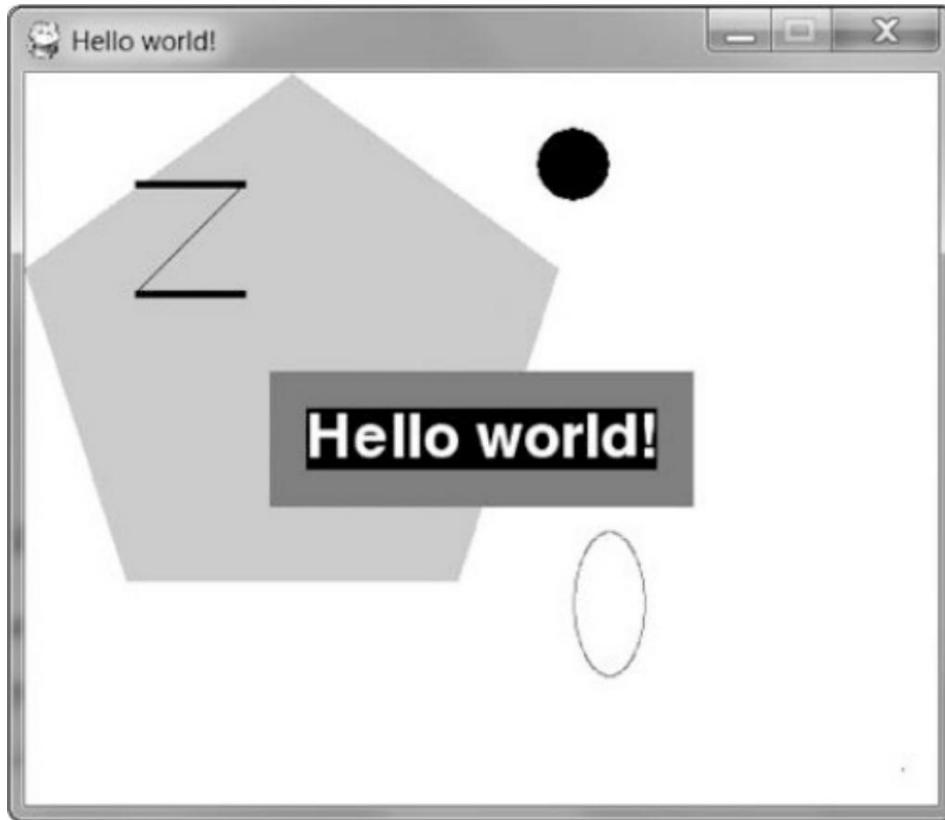


Figura 17-1: El programa pygame Hello World

Lo bueno de usar una ventana en lugar de una consola es que el texto puede aparecer en cualquier lugar de la ventana, no solo después del texto anterior que imprimió. El texto también puede ser de cualquier color y tamaño. La ventana es como un lienzo, y puedes dibujar lo que quieras en él.

CÓDIGO FUENTE PARA PYGAME HOLA MUNDO

Ingrrese el siguiente código en el editor de archivos y guárdelo como `pygameHelloWorld.py`. Si obtiene errores después de escribir este código, compare el código que escribió con el código del libro con el código en línea.

diferencia

herramienta

a

<https://www.nostarch.com/inventwithpython#diff>.



pygame HolaMundo.py

```
1. importar pygame, sys
2. desde pygame.locals importar *
3.
4. # Configurar pygame.
5. pygame.init() 6.

7. # Configure la ventana.
8. superficieVentana = pygame.display.set_mode((500, 400), 0, 32) 9.
pygame.display.set_caption('¡Hola mundo!')
10

11. # Configure los colores.
12. NEGRO = (0, 0, 0)
13. BLANCO = (255, 255, 255)
14. ROJO = (255, 0, 0)
15. VERDE = (0, 255, 0)
16. AZUL = (0, 0, 255) 17.

18. # Configure las
fuentes. 19. basicFont = pygame.font.SysFont(Ninguno, 48)
20

21. # Configure el texto.
22. text = basicFont.render('¡Hola mundo!', Verdadero, BLANCO, AZUL) 23.
textRect = text.get_rect() 24. textRect.centerx = windowSurface.get_rect().centerx
```

```
25. textRect.centery = superficieVentana.get_rect().centery
26
27. # Dibuja el fondo blanco sobre la superficie. 28.
superficieventana.relleno(BLANCO) 29.

30. # Dibuja un polígono verde en la superficie. 31.
pygame.draw.polygon(superficieventana, VERDE, ((146, 0), (291, 106), (236, 277), (56, 277),
(0, 106)))
32.
33. # Dibuja algunas líneas azules en la superficie.
34. pygame.draw.line(superficieventana, AZUL, (60, 60), (120, 60), 4) 35.
pygame.draw.line(superficieventana, AZUL, (120, 60), (60, 120)) 36.
pygame.draw.line(superficieventana, AZUL, (60, 120), (120, 120), 4)
37.
38. # Dibuja un círculo azul en la superficie.
39. pygame.draw.circle(ventanaSuperficie, AZUL, (300, 50), 20, 0) 40.

41. # Dibuja una elipse roja sobre la superficie. 42.
pygame.draw.ellipse(superficieventana, ROJO, (300, 250, 40, 80), 1)
43.
44. # Dibuja el rectángulo de fondo del texto en la superficie. 45.
pygame.draw.rect(windowSurface, RED, (textRect.left - 20, textRect.top - 20,
textRect.width + 40, textRect.height + 40))
46.
47. # Obtenga una matriz de píxeles de la
superficie. 48. pixArray = pygame.PixelArray(ventanaSuperficie)
49. pixArray[480][380] = NEGRO 50. del pixArray 51.

52. # Dibuja el texto en la superficie.
53. SuperficieVentana.blit(texto, rectangulotexto)
54.
55. # Dibuja la ventana en la pantalla.
56. Pygame.display.update() 57.

58. # Ejecuta el ciclo del juego.
59. mientras sea cierto:
60. for event in pygame.event.get(): if event.type
61.     == QUIT: pygame.quit()
62.
```

63. sys.exit()

IMPORTAR EL PYGAME MÓDULO

Repasemos cada una de estas líneas de código y descubramos qué hacen.

Primero, necesita importar el módulo pygame para poder llamar a sus funciones.

Puede importar varios módulos en la misma línea separando los nombres de los módulos con comas. La línea 1 importa los módulos pygame y sys :

-
1. importar pygame, sys
 2. desde pygame.locals importar *
-

La segunda línea importa el módulo pygame.locals . Este módulo contiene muchas variables constantes que usará con pygame, como QUIT, que lo ayuda a salir del programa, y K_ESCAPE, que representa la tecla ESC .

La línea 2 también le permite usar el módulo pygame.locals sin tener que escribir pygame.locals. delante de cada método, constante o cualquier otra cosa que llame desde el módulo.

Si tiene from sys import * en lugar de import sys en su programa, podría llamar a exit() en lugar de sys.exit() en su código.

Pero la mayoría de las veces es mejor usar el nombre completo de la función para que sabes en qué módulo está la función.

INICIALIZANDO PYGAME

Todos los programas de pygame deben llamar a pygame.init() después de importar el módulo de pygame pero antes de llamar a cualquier otra función de pygame :

```
4. # Configurar  
pygame. 5. pygame.init()
```

Esto inicializa pygame para que esté listo para usar. No necesita saber lo que hace init() ; solo necesita recordar llamarlo antes de usar cualquier otra función de pygame .

CONFIGURAR EL PYGAME VENTANA

La línea 8 crea una ventana de interfaz gráfica de usuario (GUI) llamando al método set_mode() en el módulo pygame.display . (Él display module es un módulo dentro del módulo pygame . Incluso el ¡El módulo pygame tiene sus propios módulos!)

```
7. # Configure la ventana.  
8. superficieVentana = pygame.display.set_mode((500, 400), 0, 32) 9.  
pygame.display.set_caption('¡Hola mundo!')
```

Estos métodos ayudan a configurar una ventana para que pygame se ejecute. Al igual que en el juego Sonar Treasure Hunt, las ventanas usan un sistema de coordenadas, pero el sistema de coordenadas de la ventana está organizado en píxeles.

Un píxel es el punto más pequeño en la pantalla de su computadora. Un solo píxel en su pantalla puede iluminarse en cualquier color. Todos los píxeles de su pantalla trabajan juntos para mostrar las imágenes que ve. Crearemos una ventana de 500 píxeles de ancho y 400 píxeles de alto usando una tupla.

Tuplas

Los valores de tupla son similares a las listas, excepto que usan paréntesis en lugar de corchetes. Además, al igual que las cadenas, las tuplas no pueden ser

modificado. Por ejemplo, ingrese lo siguiente en el shell interactivo:

```
>>> spam = ('Vida', 'Universo', 'Todo', 42)
>>> spam[0]
'La vida'
>>> correo_no_deseado[3]
42
>>> spam[1:3]
('Universo', 'Todo')    >>>
spam[3] = 'Hola'      Rastreo
(última llamada más reciente):
Archivo "<stdin>", línea 1, en <módulo>
TypeError: el objeto 'tuple' no admite la asignación de elementos
```

Como puede ver en el ejemplo, si desea obtener solo un elemento de una tupla o un rango de elementos , aún debe usar corchetes como lo haría con una lista. Sin embargo, si intenta cambiar el elemento en el índice 3 a la cadena 'Hola' , Python generar un error .

Usaremos tuplas para configurar las ventanas de pygame . Hay tres parámetros para el método pygame.display.set_mode() . El primero es una tupla de dos enteros para el ancho y el alto de la ventana, en píxeles. Para configurar una ventana de 500 x 400 píxeles, utiliza la tupla (500, 400) como primer argumento para set_mode(). Los parámetros segundo y tercero son opciones avanzadas que están más allá del alcance de este libro. Simplemente pase 0 y 32 para ellos, respectivamente.

Objetos de superficie

La función set_mode() devuelve un objeto pygame.Surface (que llamaremos objetos de superficie para abreviar). Objeto es simplemente otro nombre para un valor de un tipo de datos que tiene métodos. Por ejemplo, las cadenas son objetos en Python porque tienen datos (la cadena

mismo) y métodos (como `lower()` y `split()`). El objeto `Surface` representa la ventana.

Las variables almacenan referencias a objetos del mismo modo que almacenan referencias para listas y diccionarios (consulte “[Lista de referencias](#)” en la [página 132](#)).

El método `set_caption()` en la línea 9 simplemente establece el título de la ventana para que diga ‘¡Hola mundo!’. El título está en la parte superior izquierda de la ventana.

CONFIGURAR VARIABLES DE COLOR

Hay tres colores primarios de luz para los píxeles: rojo, verde y azul. Al combinar diferentes cantidades de estos tres colores (que es lo que hace la pantalla de tu computadora), puedes formar cualquier otro color. En pygame, los colores están representados por tuplas de tres enteros. Estos se denominan **valores de color** RGB y los usaremos en nuestro programa para asignar colores a los píxeles. Como no queremos reescribir una tupla de tres números cada vez que queramos usar un color específico en nuestro programa, crearemos constantes para contener tuplas que llevan el nombre del color que representa la tupla:

-
- 11. # Configure los colores.
 - 12. NEGRO = (0, 0, 0)
 - 13. BLANCO = (255, 255, 255)
 - 14. ROJO = (255, 0, 0)
 - 15. VERDE = (0, 255, 0)
 - 16. AZUL = (0, 0, 255)
-

El primer valor de la tupla determina cuánto rojo hay en el color. Un valor de 0 significa que no hay rojo en el color y un valor de 255 significa que hay una cantidad máxima de rojo en el color. El segundo valor es para verde, y el tercer valor es para

azul. Estos tres enteros forman una tupla RGB.

Por ejemplo, la tupla (0, 0, 0) no tiene rojo, verde ni azul.

El color resultante es completamente negro, como en la línea 12. La tupla (255, 255, 255) tiene una cantidad máxima de rojo, verde y azul, resultando en blanco, como en la línea 13.

También usaremos rojo, verde y azul, que se asignan en las líneas 14 a 16. La tupla (255, 0, 0) representa la cantidad máxima de rojo pero no verde ni azul, por lo que el color resultante es rojo. De manera similar, (0, 255, 0) es verde y (0, 0, 255) es azul.

Puede mezclar la cantidad de rojo, verde y azul para obtener cualquier tono de cualquier color. La tabla 17-1 tiene algunos colores comunes y sus ~~RGB~~ valores. Él web página <https://www.nostarch.com/inventwithpython/> enumera varios valores de tupla más para diferentes colores.

Tabla 17-1: Colores y sus valores RGB

| Color | valor RGB |
|---------------|-----------------|
| Negro | (0, 0, 0) |
| Azul | (0, 0, 255) |
| Gris | (128, 128, 128) |
| Verde | (0, 128, 0) |
| Lima | (0, 255, 0) |
| Púrpura | (128, 0, 128) |
| Rojo | (255, 0, 0) |
| verde azulado | (0, 128, 128) |

| | |
|----------|-----------------|
| Blanco | (255, 255, 255) |
| Amarillo | (255, 255, 0) |

Solo usaremos los cinco colores que ya hemos definido, pero en sus programas, puede usar cualquiera de estos colores o incluso inventar colores diferentes.

ESCRIBIENDO TEXTO EN EL PYGAME VENTANA

Escribir texto en una ventana es un poco diferente a simplemente usar `print()`, como lo hemos hecho en nuestros juegos basados en texto. Para escribir texto en una ventana, primero debemos hacer alguna configuración.

Uso de fuentes para aplicar estilo al texto Una fuente es un conjunto completo de letras, números, símbolos y caracteres dibujados en un solo estilo. Usaremos fuentes cada vez que necesitemos imprimir texto en una ventana de pygame . La figura 17-2 muestra la misma oración impresa en diferentes fuentes.

Programming is fun!
Programming is fun!

Figura 17-2: Ejemplos de diferentes fuentes

En nuestros juegos anteriores, solo le dijimos a Python que imprimiera texto. El color, el tamaño y la fuente que se usaron para mostrar este texto fueron completamente determinados por su sistema operativo. El programa Python no pudo cambiar la fuente. Sin embargo, pygame puede dibujar texto en cualquier fuente en su computadora.

La línea 19 crea un objeto `pygame.font.Font` (llamado objeto `Font` para abreviar) llamando a la función `pygame.font.SysFont()` con dos parámetros:

```
18. # Configure las
fuentes. 19. basicFont = pygame.font.SysFont(Ninguno, 48)
```

El primer parámetro es el nombre de la fuente, pero pasaremos el valor Ninguno para usar la fuente predeterminada del sistema. El segundo parámetro es el tamaño de la fuente (que se mide en unidades llamadas puntos). Dibujaremos '¡Hola mundo!' en la ventana en la fuente predeterminada en 48 puntos. Generar una imagen de letras para texto como "¡Hola mundo!" se llama representación.

Renderizar un objeto `Font`

El objeto `Font` que ha almacenado en la variable `basicFont` tiene un método llamado `render()`. Este método devolverá un objeto de superficie con el texto dibujado en él. El primer parámetro de `render()` es la cadena del texto a dibujar. El segundo parámetro es un valor booleano que indica si se suaviza o no la fuente. El suavizado desenfoca ligeramente el texto para que se vea más suave. La figura 17-3 muestra cómo se ve una línea con y sin suavizado.

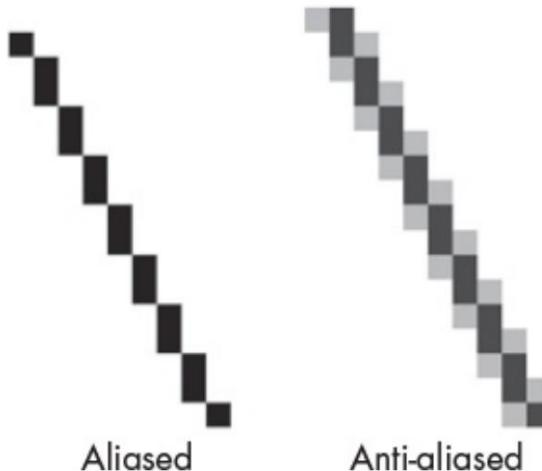


Figura 17-3: Vista ampliada de una línea con alias y una línea con suavizado

En la línea 22, pasamos True para usar suavizado:

```
21. # Configure el
texto. 22. text = basicFont.render('¡Hola mundo!', Verdadero, BLANCO, AZUL)
```

Los parámetros tercero y cuarto en la línea 22 son tuplas RGB. El tercer parámetro es el color en el que se representará el texto (blanco, en este caso), y el cuarto es el color de fondo detrás del texto (azul). Asignamos el objeto Font al texto variable .

Una vez que hayamos configurado el objeto Font , debemos colocarlo en un ubicación en la ventana.

Configuración de la ubicación del texto con atributos Rect El tipo de datos pygame.Rect (llamado Rect para abbreviar) representa áreas rectangulares de cierto tamaño y ubicación. Esto es lo que usamos para establecer la ubicación de los objetos en una ventana.

Para crear un nuevo objeto Rect , llame a la función pygame.Rect(). Observe que la función pygame.Rect() tiene el mismo nombre que el tipo de datos pygame.Rect . Funciones que tienen el mismo nombre que su tipo de dato y crean objetos o valores de su

tipo de datos se denominan funciones constructoras. Los parámetros para la función pygame.Rect() son números enteros para las coordenadas x e y de la esquina superior izquierda, seguidos por el ancho y el alto, todo en píxeles. El nombre de la función con los parámetros se ve así: pygame.Rect(left, top, width, height).

Cuando creamos el objeto Font , ya se creó un objeto Rect para él, por lo que todo lo que tenemos que hacer ahora es recuperarlo. Para hacer eso, usamos el método get_rect() en el texto y asignamos Rect a textRect:

```
23. textRect = text.get_rect()  
24. textRect.centerx = windowSurface.get_rect().centerx  
25. textRect.centery = windowSurface.get_rect().centery
```

Así como los métodos son funciones que están asociadas con un objeto, los atributos son variables que están asociadas con un objeto. El tipo de datos Rect tiene muchos atributos que describen el rectángulo que representan. Para establecer la ubicación de textRect en la ventana, debemos asignar sus valores centrales x e y a las coordenadas de píxeles en la ventana. Dado que cada objeto Rect ya tiene atributos que almacenan las coordenadas x e y del centro de Rect , llamados centerx y centery, respectivamente, todo lo que tenemos que hacer es asignar esos valores de coordenadas.

Queremos poner textRect en el centro de la ventana, por lo que necesitamos obtener windowSurface Rect, obtener sus atributos centerx y centery , y luego asignarlos a centerx y centery atributos de textRect. Hacemos eso en las líneas 24 y 25.

Hay muchos otros atributos Rect que podemos usar. Mesa 17-2 es una lista de atributos de un objeto Rect llamado myRect.

Tabla 17-2: Atributos Rect

| pygame.Rect | Descripción |
|-------------|-------------|
|-------------|-------------|

| atributo | |
|--------------------|---|
| miRect.izquierda | Valor entero de la coordenada x del lado izquierdo del rectángulo |
| miRect.derecho | Valor entero de la coordenada x del lado derecho del rectángulo |
| myRect.superior | Valor entero de la coordenada y del lado superior del rectángulo |
| myRect.bottom | Valor entero de la coordenada y del lado inferior del rectángulo |
| miRect.centerx | Valor entero de la coordenada x del centro del rectángulo |
| myRect.centery | Valor entero de la coordenada y del centro del rectángulo |
| miRect.ancho | Valor entero del ancho del rectángulo |
| myRect.height | Valor entero de la altura del rectángulo. |
| myRect.size | Una tupla de dos enteros: (ancho, alto) |
| myRect.topleft | Una tupla de dos enteros: (izquierda, arriba) |
| myRect.topright | Una tupla de dos enteros: (derecha, arriba) |
| myRect.bottomleft | Una tupla de dos enteros: (izquierda, abajo) |
| myRect.bottomright | Una tupla de dos enteros: (derecha, abajo) |
| myRect.midleft | Una tupla de dos enteros: (izquierda, centrada) |
| myRect.midright | Una tupla de dos enteros: (right, centery) |

myRect.midtop Una tupla de dos enteros: (centerx, top)

myRect.midbottom Una tupla de dos enteros: (centerx, bottom)

Lo mejor de los objetos Rect es que si modifica cualquiera de estos atributos, todos los demás atributos también se modificarán automáticamente. Por ejemplo, si crea un objeto Rect que tiene 20 píxeles de ancho y 20 píxeles de alto y tiene la esquina superior izquierda en las coordenadas (30, 40), entonces la coordenada x de el lado derecho se establecerá automáticamente en 50 (porque $20 + 30 = 50$).

O si, en cambio, cambia el atributo de la izquierda con la línea myRect.left = 100, Pygame cambiará automáticamente el atributo de la derecha a 120 (porque $20 + 100 = 120$). Todos los demás atributos para ese objeto Rect también se actualizan.

MÁS SOBRE MÉTODOS, MÓDULOS Y TIPOS DE DATOS

Dentro del pygame módulo son los módulos de fuente y superficie , y dentro de esos módulos están los tipos de datos Fuente y Superficie . Los programadores de pygame comenzaron los módulos con una letra minúscula y los tipos de datos con una letra mayúscula para que sea más fácil distinguir los tipos de datos y los módulos.

Observe que tanto el objeto Fuente (almacenado en la variable de texto en la línea 23) como el objeto Superficie (almacenado en la variable superficieventana en la línea 24) tienen un método llamado get_rect (). Técnicamente, estos son dos métodos diferentes, pero los programadores de pygame les dieron el mismo nombre porque ambos hacen lo mismo: devolver objetos Rect que representan el tamaño y la posición del objeto Font o Surface .

LLENAR UN OBJETO DE SUPERFICIE CON UN COLOR

Para nuestro programa, queremos llenar toda la superficie almacenada en windowSurface con el color blanco. La función de relleno ()

cubre completamente la superficie con el color que pasas como parámetro.

(Recuerde, la variable WHITE se estableció en el valor (255, 255, 255) en la línea 13).

27. # Dibuja el fondo blanco sobre la superficie. 28.

superficieventana.rellenar(BLANCO)

Tenga en cuenta que en pygame, la ventana en la pantalla no cambiará cuando llame al método fill() o cualquiera de las otras funciones de dibujo. Más bien, estos cambiarán el objeto de superficie , y debe mostrar el nuevo objeto de superficie en la pantalla con la función pygame.display.update() para ver los cambios.

Esto se debe a que modificar el objeto Surface en la memoria de la computadora es mucho más rápido que modificar la imagen en la pantalla. Es mucho más eficiente dibujar en la pantalla.

una vez después de que todas las funciones de dibujo se hayan dibujado en el objeto de superficie .

FUNCIONES DE DIBUJO DE PYGAME

Hasta ahora, hemos aprendido cómo llenar una ventana de pygame con un color y agregar texto, pero pygame también tiene funciones que le permiten dibujar formas y líneas. Cada forma tiene su propia función y puede combinar estas formas en diferentes imágenes para su diseño gráfico. juego.

Dibujar un polígono La función

pygame.draw.polygon() puede dibujar cualquier forma de polígono que le des. Un polígono es una figura de varios lados cuyos lados son líneas rectas. Los círculos y las elipses no son polígonos, por lo que necesitamos usar diferentes funciones para esas formas.

Los argumentos de la función, en orden, son los siguientes:

1. El objeto de superficie sobre el que dibujar el polígono.
2. El color del polígono.
3. Una tupla de tuplas que representa las coordenadas x e y de los puntos a dibujar en orden.
La última tupla se conectará automáticamente a la primera tupla para completar la forma.
4. Opcionalmente, un número entero para el ancho de las líneas del polígono. Sin esto, el se rellenará el polígono.

En la línea 31, dibujamos un polígono verde sobre nuestro objeto Surface blanco .

```
30. # Dibuja un polígono verde en la superficie. 31.  
pygame.draw.polygon(superficieventana, VERDE, ((146, 0), (291, 106), (236, 277), (56,  
277), (0, 106)))
```

Queremos que se llene nuestro polígono, por lo que no damos el último entero opcional para los anchos de línea. La figura 17-4 muestra algunos ejemplos de polígonos.

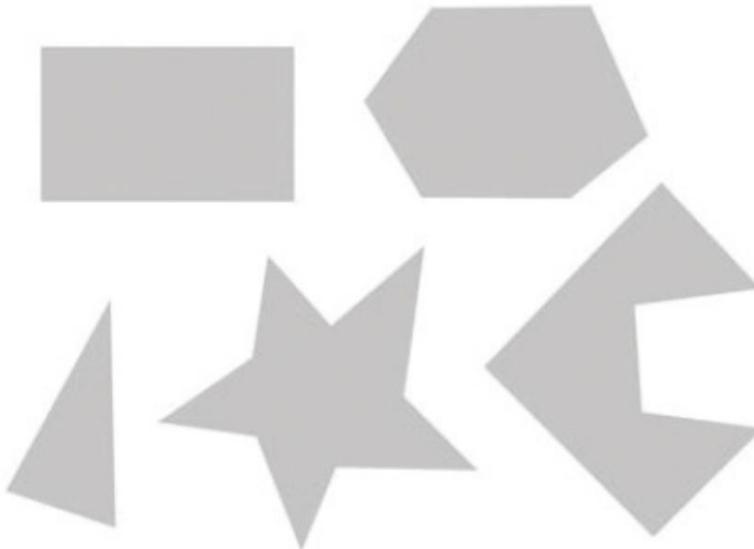


Figura 17-4: Ejemplos de polígonos

Dibujar una línea

La función `pygame.draw.line()` simplemente dibuja una línea desde un punto de la pantalla hasta otro punto. Los parámetros para `pygame.draw.line()`, en orden, son los siguientes:

1. El objeto de superficie sobre el que dibujar la línea.
2. El color de la línea.
3. Una tupla de dos enteros para las coordenadas x e y de un extremo de la línea.
4. Una tupla de dos enteros para las coordenadas x e y del otro extremo del línea.
5. Opcionalmente, un número entero para el ancho de la línea en píxeles.

En las líneas 34 a 36, llamamos a `pygame.draw.line()` tres veces:

```
33. # Dibuja algunas líneas azules en la superficie.  
34. pygame.draw.line(superficieventana, AZUL, (60, 60), (120, 60), 4) 35.  
     pygame.draw.line(superficieventana, AZUL, (120, 60), (60, 120)) 36.  
     pygame.draw.line(superficieventana, AZUL, (60, 120), (120, 120), 4)
```

Si no especifica el parámetro de ancho , tomará el valor predeterminado de 1. En las líneas 34 y 36, pasamos 4 para el ancho, por lo que las líneas tendrán un grosor de 4 píxeles. Las tres llamadas `pygame.draw.line()` en las líneas 34, 35 y 36 dibujan la Z azul en la superficie objeto.

Dibujar un círculo La función `pygame.draw.circle()` dibuja círculos en objetos de superficie . Sus parámetros, en orden, son los siguientes:

1. El objeto de superficie sobre el que dibujar el círculo.
2. El color del círculo.
3. Una tupla de dos enteros para las coordenadas x e y del centro de la círculo.
4. Un número entero para el radio (es decir, el tamaño) del círculo.
5. Opcionalmente, un número entero para el ancho de la línea. Un ancho de 0 significa que el

se rellenará el círculo.

La línea 39 dibuja un círculo azul en el objeto Superficie :

```
38. # Dibuja un círculo azul en la superficie.  
39. pygame.draw.circle(superficieventana, AZUL, (300, 50), 20, 0)
```

El centro de este círculo está en una coordenada x de 300 y una coordenada y de 50. El radio del círculo es de 20 píxeles y está lleno de azul.

Dibujar una elipse La función

pygame.draw.ellipse() es similar a pygame.draw.circle() función, pero en su lugar dibuja una elipse, que es como un círculo aplastado. Los parámetros de la función pygame.draw.ellipse() , en orden, son los siguientes:

1. El objeto de superficie sobre el que dibujar la elipse.
2. El color de la elipse.
3. Una tupla de cuatro enteros para la esquina izquierda y superior del Rect de la elipse objeto y el ancho y alto de la elipse.
4. Opcionalmente, un número entero para el ancho de la línea. Un ancho de 0 significa que la elipse se rellenará.

La línea 42 dibuja una elipse roja en el objeto Superficie :

```
41. # Dibuja una elipse roja sobre la superficie.  
42. pygame.draw.ellipse(superficieventana, ROJO, (300, 250, 40, 80), 1)
```

La esquina superior izquierda de la elipse está en una coordenada x de 300 y una coordenada y de 250. La forma tiene 40 píxeles de ancho y 80 píxeles de alto. El contorno de la elipse tiene 1 píxel de ancho.

dibujar un rectángulo

La función `pygame.draw.rect()` dibujará un rectángulo. Los parámetros de la función `pygame.draw.rect()`, en orden, son los siguientes:

1. El objeto Surface sobre el que dibujar el rectángulo.
2. El color del rectángulo.
3. Una tupla de cuatro enteros para las coordenadas x e y de la esquina superior izquierda y el ancho y alto del rectángulo. En lugar de una tupla de cuatro enteros para el tercer parámetro, también puede pasar un objeto Rect.

En el programa Hello World, queremos que el rectángulo que dibujamos sea visible 20 píxeles alrededor de todos los lados del texto.

Recuerde, en la línea 23 creamos un `textRect` para contener nuestro texto.

En la línea 45, establecemos los puntos izquierdo y superior del rectángulo como la izquierda y la parte superior de `textRect` menos 20 (restamos porque las coordenadas disminuyen a medida que avanza hacia la izquierda y hacia arriba):

```
44. # Dibuja el rectángulo de fondo del texto en la superficie. 45.  
pygame.draw.rect(windowSurface, RED, (textRect.left - 20, textRect.top  
- 20, textRect.width + 40, textRect.height + 40))
```

El ancho y la altura del rectángulo son iguales al ancho y la altura de `textRect` más 40. Usamos 40 y no 20 porque la izquierda y la parte superior se movieron hacia atrás 20 píxeles, por lo que debe compensar ese espacio.

Coloring Pixels Line

48 crea un objeto `pygame.PixelArray` (llamado objeto `PixelArray` para abreviar). El objeto `PixelArray` es una lista de listas de tuplas de color que representa el objeto `Surface` que le pasaste.

El objeto `PixelArray` le brinda un alto nivel de control por píxel, por lo que es una buena opción si necesita dibujar imágenes muy detalladas o personalizadas en la pantalla en lugar de imágenes grandes.

formas

Usaremos un PixelArray para colorear un píxel en windowSurface black.

Puede ver este píxel en la parte inferior derecha de la ventana cuando ejecuta pygame Hello World.

La línea 48 pasa windowSurface a la llamada pygame.PixelArray() , por lo que asignar NEGRO a pixArray[480][380] en la línea 49 hará que el píxel en las coordenadas (480, 380) sea negro:

```
47. # Obtenga una matriz de píxeles de la  
superficie. 48. pixArray = pygame.PixelArray(ventanaSuperficie)  
49. pixArray[480][380] = NEGRO
```

El módulo pygame modificará automáticamente el objeto windowSurface con este cambio.

El primer índice en el objeto PixelArray es para la coordenada x. El segundo índice es para la coordenada y. Los objetos PixelArray facilitan la configuración de píxeles individuales en un objeto Surface en un color específico.

Cada vez que crea un objeto PixelArray a partir de un objeto de superficie , ese objeto de superficie se bloquea. Eso significa que no se pueden realizar llamadas al método blit() (que se describen a continuación) en ese objeto de superficie . Para desbloquear el objeto Surface , debe eliminar el objeto PixelArray con el operador del :

```
50. del pixArray
```

Si olvida hacerlo, obtendrá un mensaje de error que dice
pygame.error: las superficies no deben bloquearse durante el blit.

EL MÉTODO BLIT() PARA OBJETOS DE SUPERFICIE

El método blit() dibujará el contenido de un objeto de superficie en otro objeto de superficie . Todos los objetos de texto creados por el método render() existen en su propio objeto de superficie . Todos los métodos de dibujo de pygame pueden especificar el objeto de superficie para dibujar una forma o una línea, pero nuestro texto se almacenó en la variable de texto en lugar de dibujarse en windowSurface. Para dibujar texto en el Superficie en la que queremos que aparezca, debemos usar el método blit() :

52. # Dibuja el texto en la superficie.
53. SuperficieVentana.blit(texto, rectangulotexto)

La línea 53 dibuja el '¡Hola mundo!' Objeto de superficie en la variable de texto (definida en la línea 22) en el objeto de superficie almacenado en la variable superficieventana .

El segundo parámetro de blit() especifica en qué parte de windowSurface se debe dibujar la superficie del texto . El objeto Rect que obtuvo al llamar a text.get_rect() en la línea 23 se pasa para esto parámetro.

DIBUJANDO EL OBJETO DE SUPERFICIE A LA PANTALLA

Dado que en pygame no se dibuja nada en la pantalla hasta que se llama a la función pygame.display.update() , la llamamos en la línea 56 para mostrar nuestro objeto Surface actualizado :

55. # Dibuja la ventana en la pantalla.
56. Pygame.display.update()

Para ahorrar memoria, no desea actualizar la pantalla después de cada función de dibujo; en cambio, quieres

actualice la pantalla solo una vez, después de que se hayan llamado todas las funciones de dibujo.

EVENTOS Y EL BUCLE DEL JUEGO

En nuestros juegos anteriores, todos los programas imprimían todo inmediatamente hasta que llegaban a una llamada a la función `input()`. En ese momento, el programa se detendría y esperaría a que el usuario escriba algo y presione ENTER. Pero los programas pygame se ejecutan constantemente a través de un bucle de juego, que ejecuta cada línea de código en el bucle unas 100 veces por segundo.

El bucle del juego busca constantemente nuevos eventos, actualiza el estado de la ventana y dibuja la ventana en la pantalla.

Pygame genera eventos cada vez que el usuario presiona una tecla, hace clic o mueve el mouse, o realiza alguna otra acción reconocida por el programa que debería hacer que suceda algo en el juego. Un evento es un objeto del tipo de datos `pygame.event.Event`.

La línea 59 es el comienzo del ciclo del juego:

58. # Ejecuta el ciclo del juego.

59. mientras sea cierto:

La condición para la declaración `while` se establece en `True` para que bucles para siempre. La única vez que el bucle saldrá es si un evento hace que el programa termine.

Obtención de objetos de evento

La función `pygame.event.get()` comprueba si hay nuevos objetos `pygame.event.Event` (llamados objetos de evento para abreviar) que tienen

generado desde la última llamada a `pygame.event.get()`. Estos eventos se devuelven como una lista de objetos `Event`, que luego el programa ejecutará para realizar alguna acción en respuesta al evento. Todos los objetos de evento tienen un atributo llamado `type`, que nos dice el tipo de evento. En este capítulo, solo necesitamos usar el tipo de evento `QUIT`, que nos dice cuándo el usuario sale del programa:

```
60. for evento en pygame.event.get():
61.     if event.type == SALIR:
```

En la línea 60, usamos un ciclo `for` para iterar sobre cada objeto de evento en la lista devuelta por `pygame.event.get()`. Si el atributo de tipo del evento es igual a la variable constante `SALIR`, que estaba en el módulo `pygame.locals` que importamos al comienzo del programa, entonces sabrá que se ha generado el evento `SALIR`.

El módulo `pygame` genera el evento `QUIT` cuando el usuario cierra la ventana del programa o cuando la computadora se apaga e intenta finalizar todos los programas en ejecución. A continuación, le diremos al programa qué hacer cuando detecte el mensaje `QUIT` evento.

Salir del programa

Si se ha generado el evento `QUIT`, el programa llamará tanto `pygame.quit()` como `sys.exit()`:

```
62.     pygame.quit()
63.     sys.exit()
```

La función `pygame.quit()` es algo opuesta a `init()`. Debe llamarlo antes de salir de su programa. Si lo olvida, puede hacer que IDLE se cuelgue después de que su programa haya finalizado. Líneas

62 y 63 salga de pygame y finalice el programa.

RESUMEN

En este capítulo, hemos cubierto muchos temas nuevos que nos permitirán hacer mucho más de lo que podíamos con nuestros juegos anteriores. En lugar de simplemente trabajar con texto llamando a `print()` e `input()`, un programa pygame tiene una ventana en blanco, creada por `pygame.display.set_mode()`, en la que podemos dibujar. Las funciones de dibujo de pygame le permiten dibujar formas en muchos colores en esta ventana. También puede crear texto de varios tamaños. Estos dibujos pueden estar en cualquier coordenada x e y dentro de la ventana, a diferencia del texto creado por `imprimir()`.

Aunque el código es más complicado, los programas Pygame pueden ser mucho más divertidos que los juegos de texto. A continuación, aprendamos a crear juegos con gráficos animados.

18

GRÁFICOS ANIMADOS



Ahora que ha aprendido algunas habilidades de pygame , escribiremos un programa para animar cuadros que rebotan alrededor de una ventana. Las cajas son de diferentes colores y tamaños y se mueven solo en direcciones diagonales. Para animar los cuadros, los moveremos unos pocos píxeles en cada iteración a través del ciclo del juego. Esto hará que parezca que las cajas se mueven por la pantalla.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Animación de objetos con el bucle del juego
- Cambiar la dirección de un objeto

MUESTRA DE EJECUCIÓN DE LA ANIMACIÓN PROGRAMA

Cuando ejecute el programa Animación, se parecerá a la Figura 18-1. Los bloques rebasarán en los bordes de la ventana.

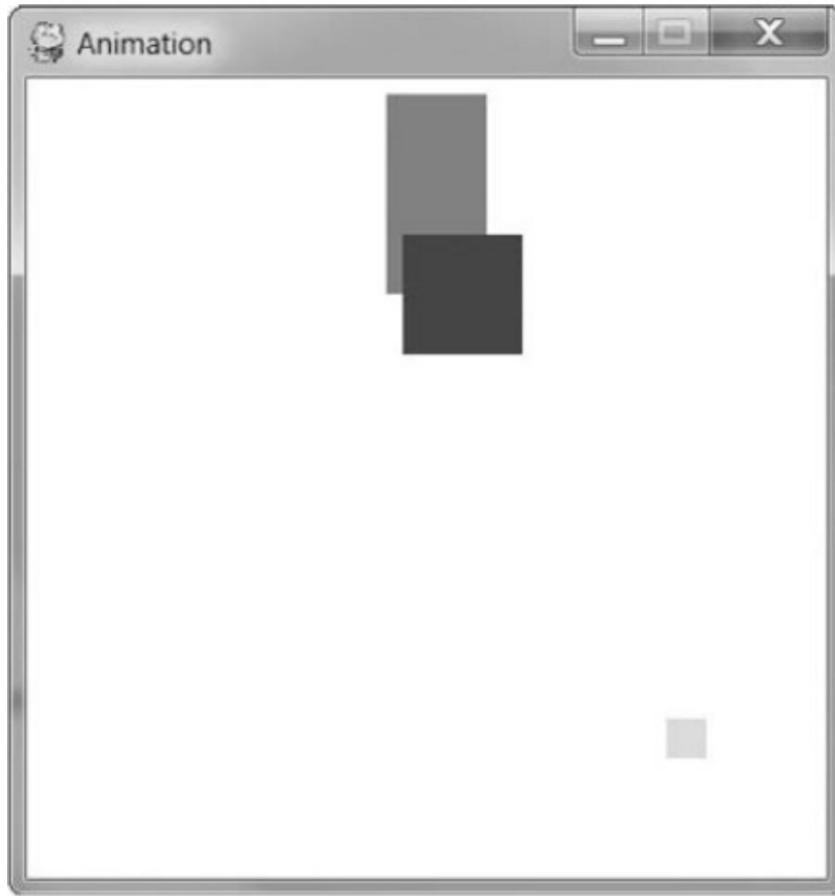


Figura 18-1: Una captura de pantalla del programa Animación

CÓDIGO FUENTE DEL PROGRAMA DE ANIMACIÓN

Ingrese el siguiente programa en el editor de archivos y guárdelo como `animation.py`. Si obtiene errores después de escribir este código, compare el código que escribió con el código del libro con la herramienta de diferencias en línea en <https://www.nostarch.com/inventwithpython#diff>.



animacion.py

```
1. importar pygame, sys, time 2.  
desde pygame.locals importar *  
3.  
4. # Configurar pygame.  
5. pygame.init() 6.  
  
7. # Configure la ventana.  
8. ANCHO DE VENTANA = 400  
9. ALTURA DE LA VENTANA = 400  
10. superficieVentana = pygame.display.set_mode((ANCHOVENTANA,  
ALTOVENTANA), 0, 32)  
  
11. pygame.display.set_caption('Animación')  
12  
13. # Configurar variables de dirección.  
14. IZQUIERDA ABAJO = 'izquierda abajo'  
15. DOWNRIGHT = 'directamente'  
16. ARRIBA = 'arriba a la izquierda'  
17. VERTICAL = 'vertical'  
18  
19. VELOCIDAD DE MOVIMIENTO = 4  
20  
21. # Configure los colores.  
22. BLANCO = (255, 255, 255)
```

```
23. ROJO = (255, 0, 0)
24. VERDE = (0, 255, 0)
25. AZUL = (0, 0, 255) 26.

27. # Configure la estructura de datos del
cuadro. 28. b1 = {'rect':pygame.Rect(300, 80, 50, 100), 'color':RED, 'dir':UPRIGHT} 29. b2 =
{'rect':pygame.Rect(200, 200 , 20, 20), 'color':VERDE, 'dir':ARRIBA} 30. b3 = {'rect':pygame.Rect(100,
150, 60, 60), 'color':AZUL, 'dir': ABAJO IZQUIERDA} 31. cajas = [b1, b2, b3]

32.

33. # Ejecuta el ciclo del juego.
34. mientras sea cierto:

35. # Comprobar el evento QUIT. 36. para
evento en pygame.event.get(): if event.type ==
37.     QUIT: pygame.quit() sys.exit()
38.
39.
40

41. # Dibuja el fondo blanco sobre la superficie. 42.
superficieventana.relleno(BLANCO)
43.

44. para b en casillas:
45.     # Mueve la estructura de datos del cuadro.
46.     si b['dir'] == ABAJO IZQUIERDA:
47.         b['rect'].left -= MOVESPEED
48.         b['rect'].top += MOVESPEED
49.     si b['dir'] == HACIA LA DERECHA:
50.         b['rect'].left += MOVESPEED
51.         b['rect'].top += MOVESPEED
52.     if b['dir'] == ARRIBA:
53.         b['rect'].left -= MOVESPEED
54.         b['rect'].top -= MOVESPEED if b['dir'] ==
== UPRIGHT: b['rect'].izquierda ++
55.         VELOCIDAD DE MOVIMIENTO
56.         b['rect'].arriba -= VELOCIDAD DE MOVIMIENTO
57.
58.

59.     # Compruebe si la caja se ha movido fuera de la ventana.
60.     si b['rect'].superior < 0:
61.         # La caja se ha movido más allá de la parte
superior. si b['dir'] == HACIA ARRIBA:
```

```

63.           b['dir'] = ABAJO IZQUIERDA
64.       si b['dir'] == VERTICAL:
65.           b['dir'] = HACIA ABAJO
66.       if b['rect'].bottom > WINDOWHEIGHT: # El cuadro
67.           se ha movido más allá de la parte inferior. if b['dir']
68.           == ABAJO IZQUIERDA: b['dir'] = ARRIBA if b['dir']
69.           == ABAJO DERECHA: b['dir'] = ARRIBA
70.
71.
72.       si b['rect'].izquierda < 0:
73.           # La caja se ha movido más allá del lado izquierdo.
74.           if b['dir'] == ABAJO IZQUIERDA: b['dir'] = ABAJO
75.           DERECHA
76.           if b['dir'] == HACIA ARRIBA:
77.               b['dir'] = HACIA ARRIBA
78.       if b['rect'].right > ANCHO DE VENTANA:
79.           # La caja se ha movido más allá del lado derecho. if
80.           b['dir'] == ABAJO DERECHA: b['dir'] = ABAJO
81.           IZQUIERDA
82.           if b['dir'] == ARRIBA: b['dir'] =
83.           ARRIBA
84.
85.       # Dibuja la caja en la superficie.
86.       pygame.draw.rect(ventanaSuperficie, b['color'], b['rect'])
87.
88.   # Dibuja la ventana en la pantalla.
89.   pygame.display.update() 90.
time.sleep(0.02)

```

MOVER Y REBOTAR EL CAJAS

En este programa tendremos tres cajas de diferentes colores moviéndose y rebotando en las paredes de una ventana. En los próximos capítulos, usaremos este programa como base para hacer un juego en el que controlemos una de las cajas. Para hacer esto, primero necesitamos

para considerar cómo queremos que se muevan las cajas.

Cada cuadro se moverá en una de las cuatro direcciones diagonales.

Cuando una caja golpea el costado de la ventana, debe rebotar y moverse en una nueva dirección diagonal. Las cajas rebotarán como se muestra en la Figura 18-2.

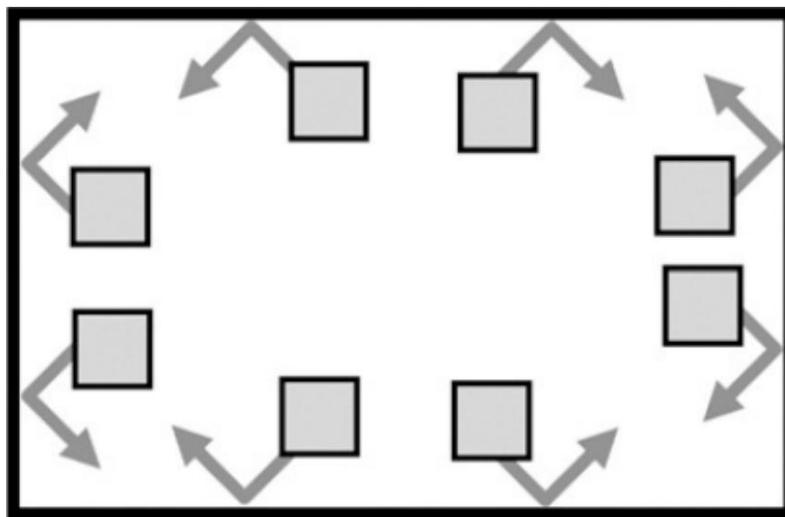


Figura 18-2: Cómo rebotan las cajas

La nueva dirección en la que se mueve una caja después de rebotar depende de dos cosas: en qué dirección se movía antes del rebote y en qué pared rebotó. Hay ocho formas posibles en que una caja puede rebotar: dos formas diferentes para cada una de las cuatro paredes. Por ejemplo, si un cuadro se mueve hacia abajo y hacia la derecha y luego rebota en el borde inferior de la ventana, queremos que la nueva dirección del cuadro sea hacia arriba y hacia la derecha.

Podemos usar un objeto Rect para representar la posición y el tamaño del cuadro, una tupla de tres enteros para representar el color del cuadro y un número entero para representar en cuál de las cuatro direcciones diagonales se mueve el cuadro actualmente.

El bucle del juego ajustará la posición x e y del cuadro en el objeto Rect y dibujará todos los cuadros en la pantalla en su posición actual en cada iteración.

Como la ejecución del programa

itera sobre el bucle, las cajas se moverán gradualmente por la pantalla para que parezca que se mueven y rebotan suavemente.

CONFIGURACIÓN DE LA CONSTANTE VARIABLES

Las líneas 1 a 5 solo configuran nuestros módulos e inicializan pygame como hicimos en el [Capítulo 17](#):

```
1. importar pygame, sys, time 2.  
desde pygame.locals importar *  
3.  
4. # Configurar pygame.  
5. pygame.init()  
6.  
7. # Configure la ventana.  
8. ANCHO DE VENTANA = 400  
9. ALTURA DE LA VENTANA = 400  
10. superficieVentana = pygame.display.set_mode((ANCHOVENTANA,  
ALTOVENTANA), 0, 32)  
11. pygame.display.set_caption('Animación')
```

En las líneas 8 y 9, definimos las dos constantes para el ancho y el alto de la ventana, y luego, en la línea 10, usamos esas constantes para configurar windowSurface, que representará nuestra ventana de pygame . La línea 11 usa set_caption() para establecer el título de la ventana en 'Animación'.

En este programa, verá que el tamaño del ancho y el alto de la ventana se usa para algo más que la llamada a set_mode(). Usaremos variables constantes para que, si alguna vez desea cambiar el tamaño de la ventana, solo tenga que cambiar las líneas 8 y 9. Dado que el ancho y la altura de la ventana nunca cambian durante el

ejecución del programa, las variables constantes son una buena idea.

Variables constantes para la dirección

Usaremos variables constantes para cada una de las cuatro direcciones que

Las cajas pueden moverse en:

-
- 13. # Configurar variables de dirección.
 - 14. IZQUIERDA ABAJO = 'izquierda abajo'
 - 15. DOWNRIGHT = 'directamente'
 - 16. ARRIBA = 'arriba a la izquierda'
 - 17. VERTICAL = 'vertical'
-

Podrías haber usado cualquier valor que quisieras para estas direcciones en lugar de usar una variable constante. Por ejemplo, podría usar la cadena 'abajo a la izquierda' directamente para representar la dirección diagonal hacia abajo y hacia la izquierda y volver a escribir la cadena cada vez que necesite especificar esa dirección. Sin embargo, si alguna vez escribió mal la cadena 'abajo a la izquierda' , terminaría con un error que haría que su programa se comportara de manera extraña, aunque el programa no fallara.

Si usa variables constantes y accidentalmente escribe mal el nombre de la variable, Python notará que no hay ninguna variable con ese nombre y bloqueará el programa con un error.

Esto seguiría siendo un error bastante malo, pero al menos lo sabría de inmediato y podría solucionarlo.

También creamos una variable constante para determinar qué tan rápido las cajas deben moverse:

-
- 19. VELOCIDAD DE MOVIMIENTO = 4
-

El valor 4 en la variable constante MOVESPEED le dice al programador cuántos píxeles debe moverse cada cuadro en cada iteración a través del ciclo del juego.

Variables constantes para el color

Las líneas 22 a 25 configuran variables constantes para los colores.

Recuerde, pygame usa una tupla de tres valores enteros para las cantidades de rojo, verde y azul, llamado valor RGB. Los números enteros van de 0 a 255.

- 21. # Configure los colores.
- 22. BLANCO = (255, 255, 255)
- 23. ROJO = (255, 0, 0)
- 24. VERDE = (0, 255, 0)
- 25. AZUL = (0, 0, 255)

Las variables constantes se utilizan para la legibilidad, al igual que en el Programa pygame Hola Mundo.

CONFIGURACIÓN DE LOS DATOS DE LA CAJA ESTRUCTURAS

A continuación definiremos las cajas. Para simplificar las cosas, configuraremos un diccionario como estructura de datos (consulte “[El tipo de datos del diccionario](#)” en la página 112) para representar cada cuadro en movimiento. El diccionario tendrá las claves 'rect' (con un objeto Rect para un valor), 'color' (con una tupla de tres enteros para un valor) y 'dir' (con una de las variables constantes de dirección para un valor). Configuraremos solo tres cuadros por ahora, pero puede configurar más cuadros definiendo más estructuras de datos. El código de animación que usaremos más adelante se puede usar para animar tantos cuadros como defina cuando configure sus estructuras de datos.

La variable b1 almacenará una de estas estructuras de datos de cuadro:

- 27. # Configure la estructura de datos del cuadro.
- 28. b1 = {'rect':pygame.Rect(300, 80, 50, 100), 'color':RED, 'dir':UPRIGHT}

La esquina superior izquierda de este cuadro se encuentra en una coordenada x de 300 y una coordenada y de 80. Tiene un ancho de 50 píxeles y una altura de 100 píxeles. Su color es ROJO, y su dirección inicial es VERTICAL.

Las líneas 29 y 30 crean dos estructuras de datos más similares para cajas que son de diferentes tamaños, posiciones, colores y direcciones:

```
29. b2 = {'rect':pygame.Rect(200, 200, 20, 20), 'color':GREEN, 'dir':UPLEFT}
30. b3 = {'rect':pygame.Rect(100, 150 , 60, 60), 'color':AZUL, 'dir':ABAJO
IZQUIERDA} 31. cajas = [b1, b2, b3]
```

Si necesita recuperar un cuadro o un valor de la lista, puede hacerlo mediante índices y claves. Ingresar casillas [0] accedería a la estructura de datos del diccionario en b1. Si ingresamos boxes[0] ['color'], eso accedería a la tecla 'color' en b1, por lo que la expresión boxes[0]['color'] se evaluaría como (255, 0, 0). Puede hacer referencia a cualquiera de los valores en cualquiera de las estructuras de datos de cuadro comenzando con cuadros. Los tres diccionarios, b1, b2 y b3, se almacenan en un lista en la variable boxes .

EL BUCLE DEL JUEGO

El bucle del juego maneja la animación de las cajas en movimiento. Las animaciones funcionan dibujando una serie de imágenes con ligeras diferencias que se muestran una tras otra. En nuestra animación, las imágenes serán de los cuadros en movimiento y las ligeras diferencias estarán en la posición de cada cuadro. Cada cuadro se moverá 4 píxeles en cada imagen. Las imágenes se muestran tan rápido que parecerá que los cuadros se mueven suavemente por la pantalla. Si una caja golpea el costado de la ventana, el bucle del juego hará que la caja rebote cambiando su dirección.

Ahora que sabemos un poco sobre cómo funciona el ciclo del juego
funcionará, ¡vamos a codificarlo!

Manejo cuando el jugador sale Cuando el jugador sale
cerrando la ventana, necesitamos detener el programa de la misma manera que hicimos
con el programa pygame Hello World. Necesitamos hacer esto en el bucle del juego
para que nuestro programa esté constantemente comprobando si ha habido un evento
QUIT . La línea 34 inicia el bucle y las líneas 36 a 39 manejan la salida:

```
33. # Ejecuta el ciclo del juego.  
34. mientras sea cierto:  
35. # Comprobar el evento QUIT.  
36. para evento en pygame.event.get():  
37.     if event.type == QUIT:  
38.         pygame.quit() sys.exit()  
39.
```

Después de eso, queremos asegurarnos de que windowSurface esté listo para
dibujar. Más tarde, dibujaremos cada cuadro en windowSurface
con el método rect() . En cada iteración a través del ciclo del juego, el código vuelve a
dibujar toda la ventana con nuevos cuadros que se ubican unos pocos píxeles cada vez.
Cuando hacemos eso, no estamos redibujando todo el objeto Surface ; en cambio, solo
estamos agregando un dibujo del objeto Rect a windowSurface . Pero cuando el bucle
del juego itera para dibujar todos los objetos Rect nuevamente, vuelve a dibujar cada
Rect y no borra el antiguo dibujo Rect . Si dejamos que el bucle del juego siga dibujando
objetos Rect en la pantalla, terminaremos con un rastro de objetos Rect en lugar de una
animación fluida. Para evitar eso, debemos borrar la ventana para cada iteración del
ciclo del juego.

Para hacer eso, la línea 42 llena toda la superficie con blanco para que se borre todo lo dibujado anteriormente:

41. # Dibuja el fondo blanco sobre la superficie. 42.
superficieventana.relleno(BLANCO)

Sin llamar a windowSurface.fill(WHITE) para blanquear toda la ventana antes de dibujar los rectángulos en su nueva posición, tendría un rastro de objetos Rect . Si quiere probarlo y ver qué sucede, puede comentar la línea 42 poniendo un # al comienzo de la línea.

Una vez que se llena la superficie de la ventana, podemos comenzar a dibujar todos nuestros Rectificar objetos.

Mover cada caja

Para actualizar la posición de cada cuadro, debemos iterar sobre la lista de cuadros dentro del ciclo del juego:

44. para b en casillas:

Dentro del bucle for , se referirá al cuadro actual como b para que el código sea más fácil de escribir. Necesitamos cambiar cada cuadro dependiendo de la dirección en la que ya se está moviendo, por lo que usaremos declaraciones if para averiguar la dirección del cuadro al verificar la clave dir dentro de la estructura de datos del cuadro. Luego cambiaremos la posición de la caja dependiendo de la dirección en la que se mueva la caja.

45. # Mueve la estructura de datos del cuadro.
46. si b['dir'] == ABAJO IZQUIERDA:
47. b['rect'].left -= MOVESPEED
48. b['rect'].top += MOVESPEED
49. si b['dir'] == HACIA LA DERECHA:
50. b['rect'].left += MOVESPEED
51. b['rect'].top += MOVESPEED

```

52.     si b['dir'] == HACIA ARRIBA:
53.         b['rect'].left -= MOVESPEED
54.         b['rect'].top -= MOVESPEED if b['dir']
55.             == UPRIGHT: b['rect'].left +=
56.                 MOVESPEED b['rect'].arriba -=
57. VELOCIDAD DE MOVIMIENTO

```

El nuevo valor para establecer los atributos izquierdo y superior de cada cuadro depende de la dirección del cuadro. Si la dirección es ABAJO IZQUIERDA o ABAJO DERECHA, desea aumentar el atributo superior . Si la dirección es ARRIBA o VERTICAL, desea disminuir el atributo superior .

Si la dirección de la caja es HACIA ABAJO o HACIA ARRIBA , desea para aumentar el atributo izquierdo . Si la dirección es ABAJO IZQUIERDA o ARRIBA, desea disminuir el atributo izquierdo .

El valor de estos atributos aumentará o disminuirá según la cantidad del entero almacenado en MOVESPEED, que almacena cuántos píxeles se mueven las cajas en cada iteración a través del bucle del juego.

Configuramos MOVESPEED en la línea 19.

Por ejemplo, si b['dir'] se establece en 'downleft', b['rect'].left en 40 y b['rect'].top en 100, entonces la condición en la línea 46 será True. Si MOVESPEED se establece en 4, las líneas 47 y 48 cambiarán el objeto Rect de modo que b['rect'].left sea 36 y b['rect'].top sea 104. Cambiar el valor de Rect hace que el código de dibujo en la línea 86 para dibujar el rectángulo ligeramente hacia abajo ya la izquierda de su posición anterior.

Rebotar una caja

Después de que las líneas 44 a 57 hayan movido la caja, debemos verificar si la caja ha pasado el borde de la ventana. Si es así, querrás hacer rebotar la caja. En el código, esto significa que el ciclo for establecerá un nuevo valor para la tecla 'dir' del cuadro . la caja

moverse en la nueva dirección en la siguiente iteración del ciclo del juego.
 Esto hace que parezca que la caja ha rebotado en el costado de la ventana.

En la sentencia if de la línea 60, determinamos que la casilla se ha movido más allá del borde superior de la ventana si el atributo superior del objeto Rect del cuadro es menor que 0:

```

59.      # Compruebe si la caja se ha movido fuera de la ventana.
60.      si b['rect'].superior < 0:
61.          # La caja se ha movido más allá de la parte
62.          superior. si b['dir'] == HACIA ARRIBA:
63.              b['dir'] = ABAJO IZQUIERDA
64.          si b['dir'] == VERTICAL:
65.              b['dir'] = HACIA ABAJO

```

En ese caso, la dirección cambiará según la dirección en la que se movía la caja. Si la caja se movía hacia ARRIBA, ahora se moverá hacia ABAJO A LA IZQUIERDA; si se movía en posición VERTICAL , ahora se moverá HACIA ABAJO.

Las líneas 66 a 71 tratan la situación en la que la caja tiene movido más allá del borde inferior de la ventana:

```

66.      if b['rect'].bottom > WINDOWHEIGHT: # El cuadro
67.          se ha movido más allá de la parte inferior. if
68.          b['dir'] == ABAJO IZQUIERDA: b['dir'] = ARRIBA
69.          if b['dir'] == ABAJO DERECHA: b['dir'] =
70.              ARRIBA
71.

```

Estas líneas verifican si el atributo inferior (no el atributo superior) es mayor que el valor en WINDOWHEIGHT.

Recuerde que las coordenadas y comienzan en 0 en la parte superior de la ventana y aumente a WINDOWHEIGHT en la parte inferior.

Las líneas 72 a 83 manejan el comportamiento de las cajas cuando

rebota en los lados:

```

72.     si b['rect'].izquierda < 0:
73.         # La caja se ha movido más allá del lado izquierdo.
74.         if b['dir'] == ABAJO IZQUIERDA: b['dir'] = ABAJO
75.             DERECHA
76.             if b['dir'] == HACIA ARRIBA:
77.                 b['dir'] = HACIA ARRIBA
78.             if b['rect'].right > ANCHO DE VENTANA:
79.                 # La caja se ha movido más allá del lado derecho.
80.                 si b['dir'] == HACIA LA DERECHA:
81.                     b['dir'] = ABAJO IZQUIERDA
82.                     si b['dir'] == VERTICAL:
83.                         b['dir'] = ARRIBA

```

Las líneas 78 a 83 son similares a las líneas 72 a 77 pero verifique si el lado derecho del cuadro se ha movido más allá del borde derecho de la ventana. Recuerde, las coordenadas x comienzan en 0 en el borde izquierdo de la ventana y aumentan hasta ANCHO DE VENTANA en el borde derecho de la ventana.

Dibujar las cajas en la ventana en su nuevo Posiciones

Cada vez que los cuadros se mueven, debemos dibujarlos en sus nuevas posiciones en windowSurface llamando a pygame.draw.rect() función:

```

85.     # Dibuja la caja en la superficie.
86.     pygame.draw.rect(windowSurface, b['color'], b['rect'])

```

Debe pasar windowSurface a la función porque es el objeto Surface para dibujar el rectángulo. Pasa b['color'] a la función porque es el color del rectángulo. Finalmente, pase b['rect'] porque es el objeto Rect con la posición y el tamaño del rectángulo a dibujar.

La línea 86 es la última línea del bucle for .

Dibujar la ventana en la pantalla Después del ciclo
for , se dibujará cada cuadro en la lista de cuadros , por lo que debe llamar
a pygame.display.update() para dibujar windowSurface en el
pantalla:

```
88. # Dibuja la ventana en la pantalla.  
89. pygame.display.update() 90.  
time.sleep(0.02)
```

La computadora puede mover, rebotar y dibujar los cuadros tan rápido que si el programa se ejecutara a toda velocidad, todos los cuadros se verían borrosos. Para hacer que el programa se ejecute lo suficientemente lento como para que podamos ver las casillas, necesitamos agregar time.sleep(0.02). Puede intentar comentar la línea time.sleep(0.02) y ejecutar el programa para ver cómo se ve. La llamada a time.sleep() pausará el programa durante 0,02 segundos, o 20 milisegundos, entre cada movimiento de las cajas.

Después de esta línea, la ejecución vuelve al inicio del ciclo del juego y comienza el proceso nuevamente. De esta manera, las cajas se mueven un poco constantemente, rebotan en las paredes y se dibujan en la pantalla en sus nuevas posiciones.

RESUMEN

Este capítulo ha presentado una forma completamente nueva de crear programas de computadora. Los programas de los capítulos anteriores se detendrían y esperarían a que el jugador ingresara texto. Sin embargo, en nuestro programa de animación, el programa actualiza constantemente las estructuras de datos sin esperar la entrada del reproductor.

Recuerde que teníamos estructuras de datos que representarían el estado del tablero en nuestros juegos Hangman y Tic-Tac-Toe. Estas estructuras de datos se pasaron a una función drawBoard() para que se mostraran en la pantalla. Nuestro programa de Animación es similar. La variable de cuadros contiene una lista de estructuras de datos que representan cuadros que se dibujarán en la pantalla, y estos se dibujan dentro del bucle del juego.

Pero sin llamadas a input(), ¿cómo obtenemos información del reproductor? En el Capítulo 19, cubriremos cómo los programas saben cuándo el jugador presiona las teclas del teclado. También aprenderemos sobre un nuevo concepto llamado detección de colisiones.

19

DETECCIÓN DE COLISIONES



La detección de colisiones implica averiguar cuándo dos cosas en la pantalla se han tocado (es decir, chocado) entre sí.

La detección de colisiones es realmente útil para los juegos. Por ejemplo, si el jugador toca a un enemigo, puede perder salud. O si el jugador toca una moneda, debe recogerla automáticamente.

La detección de colisiones puede ayudar a determinar si el personaje del juego está parado sobre suelo firme o si no hay nada más que aire vacío debajo de él.

En nuestros juegos, la detección de colisiones determinará si dos rectángulos se superponen entre sí. El programa de ejemplo de este capítulo cubrirá esta técnica básica. También veremos cómo nuestros programas pygame pueden aceptar entradas del jugador a través del teclado y el mouse. Es un poco más complicado que llamar a la función `input()`, como hicimos con nuestros programas de texto. Pero usar el teclado es mucho más interactivo en los programas GUI, y usar el mouse ni siquiera es posible en nuestros juegos de texto. Estos dos conceptos harán que tu

juegos más emocionantes!

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Objetos de reloj
- Entrada de teclado en [Pygame](#)
- Entrada de ratón en [Pygame](#)
- Detección de colisiones
- No modificar una lista mientras se itera sobre ella

MUESTRA DE EJECUCIÓN DE LA COLISIÓN PROGRAMA DE DETECCIÓN

En este programa, el jugador usa las teclas de flecha del teclado para mover un cuadro negro alrededor de la pantalla. Cuadrados verdes más pequeños, que representan comida, aparecen en la pantalla y la caja los “come” cuando los toca. El jugador puede hacer clic en cualquier parte de la ventana para crear nuevos cuadrados de comida. Además, ESC sale del programa y la tecla X teletransporta al jugador a un lugar aleatorio en la pantalla.

La figura 19-1 muestra el aspecto que tendrá el programa una vez finalizado.

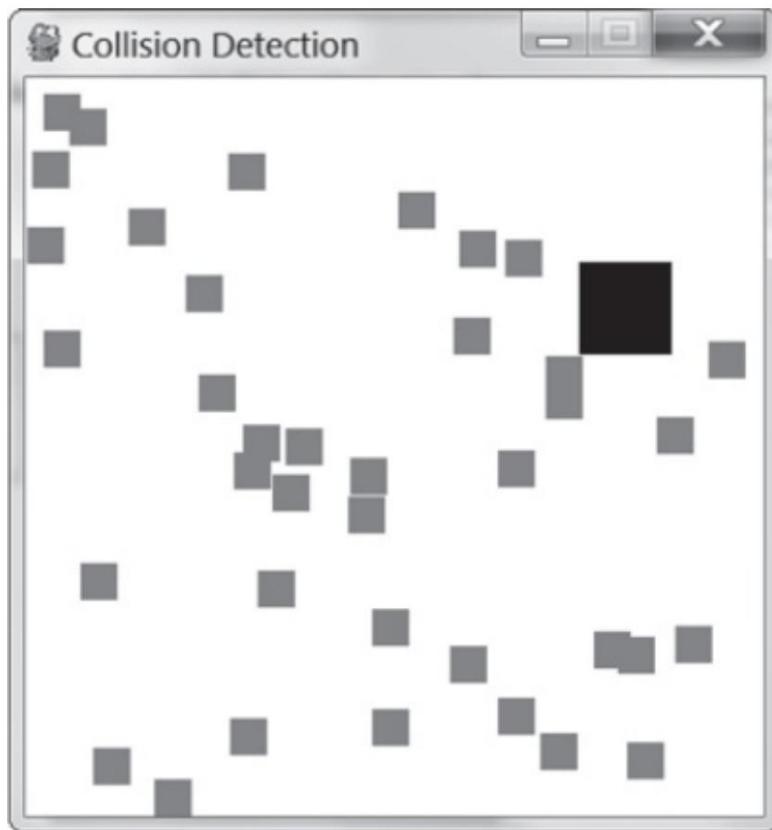


Figura 19-1: Una captura de pantalla del programa pygame Collision Detection

CÓDIGO FUENTE DEL DETECCIÓN DE COLISIONES PROGRAMA

Inicie un nuevo archivo, ingrese el siguiente código y luego guárdelo como `colisionDetection.py`. Si obtiene errores después de escribir este código, compare el código que escribió con el código del libro con el código en línea.

diferencia

herramienta

a

<https://www.nostarch.com/inventwithpython#diff>.



Detección de colisión.py

```
1. importar pygame, sys, aleatorio 2.  
desde pygame.locals importar *  
3.  
4. # Configurar pygame.  
5. pygame.init() 6.  
mainClock = pygame.time.Clock()  
7.  
8. # Configure la ventana.  
9. ANCHO DE VENTANA = 400  
10. ALTURA DE LA VENTANA = 400  
11. superficieVentana = pygame.display.set_mode((ANCHOVENTANA,  
ALTOVENTANA), 0, 32)  
  
12. pygame.display.set_caption('Detección de colisiones')  
13  
14. # Configura los colores.  
15. NEGRO = (0, 0, 0)  
16. VERDE = (0, 255, 0)  
17. BLANCO = (255, 255, 255)  
18  
19. # Configure las estructuras de datos del jugador y de la comida.  
20. contador de comida = 0  
21. COMIDA NUEVA = 40  
22. TAMAÑO DE COMIDA = 20
```

```
23. jugador = pygame.Rect(300, 100, 50, 50) 24.  
comidas = [] 25. for i in range(20): 26.  
comidas.append(pygame.Rect(random.randint(0,  
ANCHO DE VENTANA - TAMAÑO DEL COMIDA), random.randint(0, ALTURA DE LA  
VENTANA - TAMAÑO DEL COMIDA), TAMAÑO DEL COMIDA, TAMAÑO DE LA COMIDA))
```

27

```
28. # Configurar variables de movimiento.  
29. moveLeft = Falso  
30. moverDerecha = Falso  
31. moverArriba = Falso 32.  
moverAbajo = Falso  
33.  
34. VELOCIDAD DE MOVIMIENTO = 6  
35.  
36.  
37. # Ejecuta el ciclo del juego.  
38. mientras sea cierto:  
39. # Comprobar eventos.  
40. for event in pygame.event.get(): 41. if  
event.type == QUIT: pygame.quit() sys.exit() if  
event.type == KEYDOWN:  
42.  
43.  
44.  
45.     # Cambiar las variables del teclado. if  
46.     event.key == K_LEFT or event.key == K_a: moveRight =  
47.         False moveLeft = True  
48.  
49.     if event.key == K_RIGHT o event.key == K_d: moveLeft =  
50.         False  
51.     moverDerecha =  
52.     Verdadero si evento.clave == K_ARRIBA o evento.clave  
53.     == K_w: moverAbajo = Falso  
54.     mover hacia arriba = verdadero  
55.     if evento.clave == K_DOWN o evento.clave == K_s:  
56.         mover hacia arriba = Falso  
57.         mover hacia abajo = verdadero  
58.     si event.type == KEYUP:  
59.         si evento.key == K_ESCAPE:
```

```
60         pygame.quit()
61.     sys.exit() si
62.     event.key == K_LEFT o event.key == K_a: moveLeft =
63.         False
64.     if event.key == K_RIGHT o event.key == K_d: moveRight
65.         = False if event.key == K_UP o event.key == K_w:
66.         moveUp = False
67.
68.     if event.key == K_DOWN or event.key == K_s: moveDown
69.         = False
70.     if event.key == K_x:
71.         player.top = random.randint(0, WINDOWHEIGHT - player.height)
72.         player.left = random.randint(0, WINDOWWIDTH - player.width)
73.
74.     if event.type == MOUSEBUTTONUP:
75.         comidas.append(pygame.Rect(event.pos[0], event.pos[1],
76.                                     FOODSIZE, FOODSIZE))
77.     contador de comida += 1
78. si contadordecomida >= COMIDA NUEVA:
79.     # Agregar comida nueva.
80.     contador de comida = 0
81.     alimentos.append(pygame.Rect(random.randint(0, ANCHO DE VENTANA - TAMAÑO DE
82.                                     COMIDA), random.randint(0, ALTO DE VENTANA - TAMAÑO DE COMIDA), TAMAÑO DE
83.                                     COMIDA, TAMAÑO DE COMIDA))
84.
85. # Dibuja el fondo blanco sobre la superficie. 84.
superficieventana.relleno(BLANCO)
86. # Mueve al jugador. 87. if
moveDown and player.bottom < WINDOWHEIGHT: 88. player.top +=
MOVESPEED 89. if moveUp and player.top > 0: 90. player.top -=
MOVESPEED 91. if moveLeft and player.left > 0: 92. jugador.left -=
MOVESPEED 93. if moveRight y player.right < ANCHO DE VENTANA:
94.         jugador.derecha += VELOCIDAD DE MOVIMIENTO
```

```
95.  
96. # Dibuja al jugador sobre la superficie. 97.  
pygame.draw.rect(ventanaSuperficie, NEGRO, jugador) 98.  
  
99. # Comprueba si el jugador se ha cruzado con algún cuadrado de comida. 100.  
para alimentos en alimentos[:]:  
101.     if player.colliderect(comida):  
102.         comidas.remove(comida)  
103.  
104. # Dibuja la comida.  
105. for i in range(len(alimentos)):  
106.     pygame.draw.rect(ventanaSuperficie, VERDE, alimentos[i])  
107.  
108. # Dibuja la ventana en la pantalla.  
109. pygame.display.update() 110.  
mainClock.tick(40)
```

IMPORTACIÓN DE LOS MÓDULOS

El programa de detección de colisiones de pygame importa los mismos módulos que el programa de animación del capítulo 18, más el módulo aleatorio :

```
1. importar pygame, sys, aleatorio 2.  
desde pygame.locals importar *
```

UTILIZAR UN RELOJ PARA MARCAR EL RITMO PROGRAMA

Las líneas 5 a 17 en su mayoría hacen las mismas cosas que hizo el programa Animation: inicializan pygame, establecen WINDOWHEIGHT y WINDOWWIDTH, y asignan las constantes de color y dirección.

Sin embargo, la línea 6 es nueva:

```
6. relojprincipal = pygame.time.Clock()
```

En el programa de animación, una llamada a `time.sleep(0.02)` ralentizaba el programa para que no se ejecutara demasiado rápido. Si bien esta llamada siempre se detendrá durante 0,02 segundos en todas las computadoras, la velocidad del resto del programa depende de qué tan rápida sea la computadora. Si queremos que este programa se ejecute a la misma velocidad en cualquier computadora, necesitamos una función que haga pausas más largas en las computadoras rápidas y más cortas en las lentas.

Un objeto `pygame.time.Clock` puede pausar una cantidad de tiempo apropiada en cualquier computadora. La línea 110 llama a `mainClock.tick(40)` dentro del ciclo del juego. Esta llamada al método `tick()` del objeto `Clock` espera el tiempo suficiente para que se ejecute a unas 40 iteraciones por segundo, sin importar la velocidad de la computadora. Esto asegura que el juego nunca se ejecute más rápido de lo esperado. Una llamada a `tick()` debería aparecer solo una vez en el ciclo del juego.

CONFIGURAR LA VENTANA Y ESTRUCTURAS DE DATOS

Las líneas 19 a 22 configuran algunas variables para los cuadrados de alimentos que aparecen en la pantalla:

```
19. # Configure las estructuras de datos del jugador y de la comida.  
20. contador de comida = 0  
21. COMIDA NUEVA = 40  
22. TAMAÑO DE COMIDA = 20
```

La variable `foodCounter` comenzará en el valor 0, `NEWFOOD` en 40 y `FOODSIZE` en 20. Veremos cómo se usan más adelante cuando nosotros creamos la comida.

La línea 23 configura un objeto `pygame.Rect` para la ubicación del jugador:

```
23. jugador = pygame.Rect(300, 100, 50, 50)
```

La variable `jugador` tiene un objeto `pygame.Rect` que representa el tamaño y la posición del cuadro. El cuadro del jugador se moverá como lo hicieron los cuadros en el programa Animación (consulte “Mover cada cuadro” en la página 280), pero en este programa, el jugador puede controlar dónde se mueve el cuadro.

A continuación, configuramos un código para realizar un seguimiento de la comida. cuadrícula:

```
24. comidas = []
25. for i in range(20): 26.
    comidas.append(pygame.Rect(random.randint(0, ANCHO DE VENTANA - TAMAÑO DE
    COMIDA), random.randint(0, ALTURA DE VENTANA - TAMAÑO DE COMIDA), TAMAÑO DE
    COMIDA, TAMAÑO DE COMIDA))
```

El programa hará un seguimiento de cada cuadro de alimentos con una lista de objetos `Rect` en los alimentos. Las líneas 25 y 26 crean 20 cuadrados de alimentos colocados al azar alrededor de la pantalla. Puede usar la función `random.randint()` para generar x e y aleatorios coordenadas

En la línea 26, el programa llama a la función constructora `pygame.Rect()` para devolver un nuevo objeto `pygame.Rect`. Representará la posición y el tamaño de un nuevo cuadrado de comida. Los primeros dos parámetros para `pygame.Rect()` son las coordenadas x e y de la esquina superior izquierda. Desea que la coordenada aleatoria esté entre 0 y el tamaño de la ventana menos el tamaño del cuadrado de comida.

Si establece la coordenada aleatoria entre 0 y el tamaño de la ventana, entonces el cuadrado de comida podría salir de la ventana por completo, como en la figura 19-2.

Los parámetros tercero y cuarto para `pygame.Rect()` son el ancho y el alto del cuadrado de comida. Tanto el ancho como la altura

son los valores en la constante FOODSIZE .

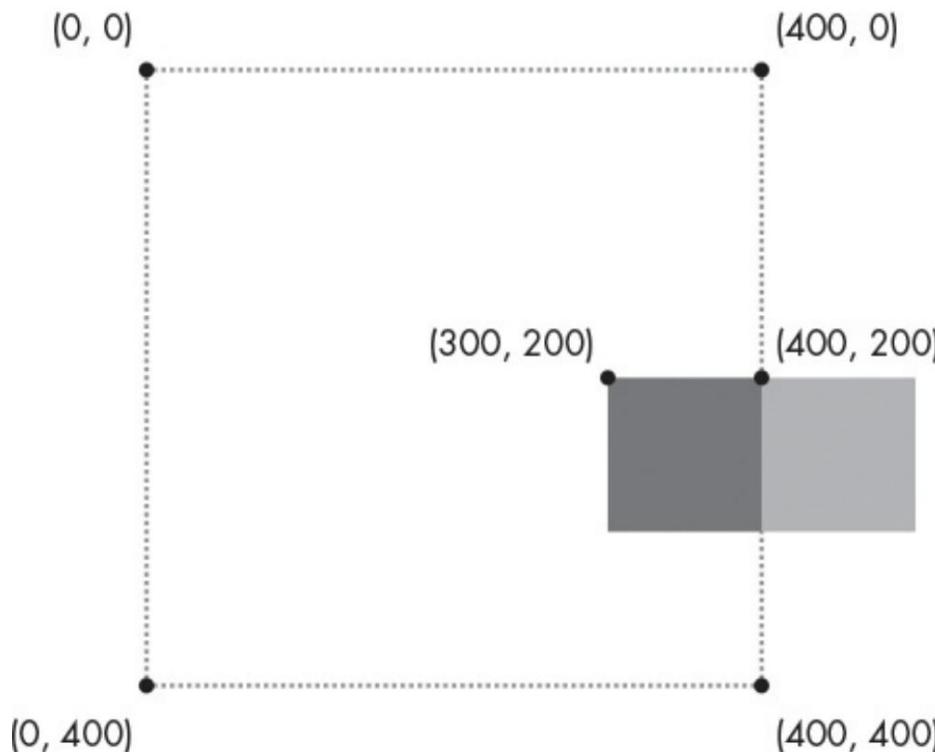


Figura 19-2: Para un cuadrado de 100×100 en una ventana de 400×400 , establecer el borde superior izquierdo en 400 colocaría el rectángulo fuera de la ventana. Para estar dentro, el borde izquierdo debe establecerse en 300 en su lugar.

Los parámetros tercero y cuarto para `pygame.Rect()` son el ancho y el alto del cuadrado de comida. Tanto el ancho como la altura son los valores de la constante FOODSIZE .

CONFIGURAR VARIABLES PARA MOVIMIENTO DE LA VÍA

A partir de la línea 29, el código configura algunas variables que rastrean el movimiento de la caja del jugador para cada dirección en la que la caja puede moverse:

28. # Configurar variables de movimiento.

29. moveLeft = Falso

```
30. moverDerecha = Falso  
31. moverArriba = Falso  
32. moverAbajo = Falso
```

Las cuatro variables tienen valores booleanos para realizar un seguimiento de qué tecla de flecha se presiona y se establecen inicialmente en False. Por ejemplo, cuando el jugador presiona la tecla de flecha izquierda en su teclado para mover el cuadro, moveLeft se establece en True. Cuando sueltan la tecla, moveLeft vuelve a establecerse en False.

Las líneas 34 a 43 son casi idénticas al código de los programas pygame anteriores . Estas líneas manejan el inicio del ciclo del juego y qué hacer cuando el jugador sale del programa. Omitiremos la explicación de este código ya que lo cubrimos en el capítulo anterior.

MANEJO DE EVENTOS

El módulo pygame puede generar eventos en respuesta a la entrada del usuario desde el mouse o el teclado. Los siguientes son los eventos que pygame.event.get() puede devolver :

SALIR Generado cuando el jugador cierra la ventana.

KEYDOWN Se genera cuando el jugador presiona una tecla. Tiene un atributo clave que indica qué tecla se presionó. También tiene un atributo mod que indica si SHIFT, CTRL, ALT o otras teclas se mantuvieron presionadas cuando se presionó esta tecla.

KEYUP Generado cuando el jugador suelta una tecla. tiene llave y atributos mod que son similares a los de KEYDOWN.

MOUSEMOTION Generado cada vez que el mouse se mueve sobre la ventana. Tiene un atributo pos (abreviatura de posición) que

devuelve una tupla (x, y) para las coordenadas de dónde se encuentra el mouse en la ventana. El atributo rel también devuelve una tupla (x, y) , pero proporciona coordenadas relativas desde el último evento MOUSEMOTION . Por ejemplo, si el mouse se mueve 4 píxeles hacia la izquierda desde (200, 200) hasta (196, 200), entonces rel será el valor de tupla (-4, 0). El atributo de botón devuelve una tupla de tres enteros. El primer entero en la tupla es para el botón izquierdo del mouse, el segundo entero es para el botón central del mouse (si existe) y el tercer entero es para el botón derecho del mouse. Estos enteros serán 0 si no se presionan cuando se mueve el mouse y 1 si se presionan.

MOUSEBUTTONDOWN Se genera cuando se pulsa un botón del ratón . presionado en la ventana. Este evento tiene un atributo pos , que es una tupla (x, y) para las coordenadas de dónde se colocó el mouse cuando se presionó el botón. También hay un atributo de botón , que es un número entero del 1 al 5 que indica qué botón del mouse se presionó, como se explica en la Tabla 19-

1.

MOUSEBUTTONUP Se genera cuando se pulsa el botón del ratón . publicado. Esto tiene los mismos atributos.

como

BOTÓN DEL RATÓN.

Cuando se genera el evento MOUSEBUTTONDOWN , tiene un atributo de botón . El atributo de botón es un valor que está asociado con los diferentes tipos de botones que puede tener un ratón. Por ejemplo, el botón izquierdo tiene el valor 1 y el botón derecho tiene el valor 3. La Tabla 19-1 enumera todos los atributos de botón para eventos del mouse, pero tenga en cuenta que es posible que un mouse no tenga todos los

valores de los botones enumerados aquí.

Tabla 19-1: Los valores de los atributos del botón

| valor del botón | Botón del ratón |
|-----------------|---|
| 1 | Botón izquierdo |
| 2 | Botón central |
| 3 | Botón derecho |
| 4 | Rueda de desplazamiento movida hacia arriba |
| 5 | Rueda de desplazamiento movida hacia abajo |

Usaremos estos eventos para permitir que el jugador controle la caja con eventos KEYDOWN y con clics del botón del mouse.

Manejo del evento KEYDOWN

El código para manejar los eventos de pulsación y liberación de tecla comienza en la línea 44; incluye el tipo de evento KEYDOWN :

44. si evento.tipo == TECLADO:

Si el tipo de evento es KEYDOWN, entonces el objeto Event tiene un atributo clave que indica qué tecla se presionó. Cuando el jugador presiona una tecla de flecha o una tecla WASD (pronunciado wazz-dee, estas teclas están en el mismo diseño que las teclas de flecha pero en el lado izquierdo del teclado), entonces queremos que la caja se mueva. Usaremos declaraciones if para verificar la tecla presionada en orden para decir en qué dirección debe moverse la caja.

La línea 46 compara este atributo clave con K_LEFT y K_a, que son las constantes de pygame.locals que representan la flecha izquierda

en el teclado y la A en WASD, respectivamente. Las líneas 46 a 57 verifican cada una de las teclas de flecha y WASD:

```
45.     # Cambiar las variables del teclado. if
46.     event.key == K_LEFT or event.key == K_a: moveRight = False
47.         moveLeft = True
48.
49.     if evento.clave == K_DERECHO o evento.clave == K_d:
50.         moverIzquierda = Falso
51.             moveRight = True si
52.             event.key == K_UP o event.key == K_w:
53.                 mover hacia abajo = falso
54.                     mover hacia arriba = verdadero
55.                 if evento.clave == K_DOWN o evento.clave == K_s:
56.                     mover hacia arriba = Falso
57.                         mover hacia abajo = verdadero
```

Cuando se presiona una de estas teclas, el código le dice a Python que establezca la variable de movimiento correspondiente en True.

Python también establecerá la variable de movimiento de la dirección opuesta a Falso.

Por ejemplo, el programa ejecuta las líneas 47 y 48 cuando se presiona la tecla de flecha izquierda. En este caso, Python establecerá moveLeft en True y moveRight en False (aunque moveRight ya sea False, Python lo establecerá en False nuevamente solo para estar seguro).

En la línea 46, event.key puede ser igual a K_LEFT o K_a. El valor en event.key se establece en el mismo valor que K_LEFT si se presiona la tecla de flecha izquierda o el mismo valor que K_a si se presiona la tecla A.

Al ejecutar el código en las líneas 47 y 48 si la pulsación de tecla es K_LEFT o K_a, hace que la tecla de flecha izquierda y la tecla A hagan lo mismo. Las teclas W, A, S y D se utilizan como

se alterna para cambiar las variables de movimiento, permitiendo que el jugador use su mano izquierda en lugar de la derecha si lo prefiere.

Puede ver una ilustración de ambos juegos de llaves en la [Figura 19-3](#).

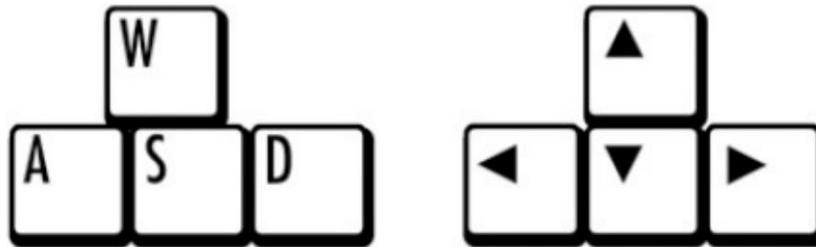


Figura 19-3: Las teclas WASD se pueden programar para hacer lo mismo que las teclas de flecha.

Las constantes para las teclas de letras y números son fáciles de calcular: la constante de la tecla A es K_a, la constante de la tecla B es K_b, y así sucesivamente. La constante de la tecla 3 es K_3. La [Tabla 19-2](#) enumera las variables constantes comúnmente utilizadas para las otras teclas del teclado.

Tabla 19-2: Variables constantes para teclas de teclado

| variable constante pygame | tecla del teclado |
|---------------------------|--------------------|
| K_IZQUIERDA | Flecha izquierda |
| K_DERECHO | Flecha correcta |
| K_UP | flecha arriba |
| K_ABAJO | Flecha hacia abajo |
| K_ESCAPE | ESC |
| K_RETROCESO | Retroceso |
| K_TAB | PESTAÑA |
| K_RETORNO | VOLVER o ENTRAR |

| | |
|-------------------|-------------------|
| ESPACIO_K | barra espaciadora |
| K_DELETE | DEL |
| K_MAYÚS | Shift izquierdo |
| K_RSHIFT | Giro a la derecha |
| K_LCTRL | CTRL izquierdo |
| K_RCTRL | CTRL derecho |
| K_LALT | ALT izquierdo |
| K_RALT | ALT derecho |
| K_HOME | CASA |
| K_END | FIN |
| K_PÁGINA ARRIBA | RE PÁG |
| K_ABAJO DE PÁGINA | PGDN |
| K_F1 | F1 |
| K_F2 | F2 |
| K_F3 | F3 |
| K_F4 | F4 |
| K_F5 | F5 |
| K_F6 | F6 |

| | |
|-------|-----|
| K_F7 | F7 |
| K_F8 | F8 |
| K_F9 | F9 |
| K_F10 | F10 |
| K_F11 | F11 |
| K_F12 | F12 |

Manejo del evento KEYUP

Cuando el jugador suelta la tecla que estaba presionando, un

Se genera el evento KEYUP :

58. si event.type == KEYUP:

Si la tecla que soltó el jugador fue ESC, entonces Python debería terminar el programa. Recuerde, en pygame debe llamar a la función pygame.quit() antes de llamar a la función sys.exit() , lo que hacemos en las líneas 59 a 61:

59. si evento.key == K_ESCAPE:
60 pygame.quit() sys.exit()
61.

Las líneas 62 a 69 establecen una variable de movimiento en False si eso Se soltó la tecla de dirección:

62. if evento.clave == K_IZQUIERDA o evento.clave == K_a:
63. moverIzquierda = Falso
64. if event.key == K_RIGHT o event.key == K_d: moveRight
 = False if event.key == K_UP or event.key == K_w:
66.

```
67.     moveUp = False
68.     si event.key == K_DOWN o event.key == K_s:
69.         moveDown = False
```

Establecer la variable de movimiento en False a través de un evento KEYUP hace que la caja deje de moverse.

TELEPORTAR AL JUGADOR

También puedes agregar teletransportación al juego. Si el jugador presiona la tecla X, las líneas 71 y 72 establecen la posición de la casilla del jugador en un lugar aleatorio en la ventana:

```
70.     if event.key == K_x:
71.         player.top = random.randint(0, WINDOWHEIGHT -
72.                                         player.height) player.left = random.randint(0,
72.                                         WINDOWWITH - player.width)
```

La línea 70 verifica si el jugador presionó la tecla X. Luego, la línea 71 establece una coordenada x aleatoria para teletransportar al jugador entre 0 y la altura de la ventana menos la altura del rectángulo del jugador. La línea 72 ejecuta un código similar, pero para la coordenada y. Esto le permite al jugador teletransportarse alrededor de la ventana presionando la tecla X, pero no puede controlar a dónde se teletransportará, es completamente aleatorio.

AGREGAR NUEVOS CUADRADOS DE COMIDA

Hay dos formas en que el jugador puede agregar nuevos cuadrados de comida a la pantalla. Pueden hacer clic en un punto de la ventana donde quieren que aparezca el nuevo cuadro de comida, o pueden esperar hasta que el ciclo del juego haya iterado NEWFOOD varias veces, en cuyo caso un

Se generará aleatoriamente un nuevo cuadro de comida en la ventana.

Primero veremos cómo se agrega la comida a través de la entrada del mouse del jugador:

```
74.     if event.type == MOUSEBUTTONUP:
75.         comidas.append(pygame.Rect(event.pos[0], event.pos[1], FOODSIZE,
FOODSIZE))
```

La entrada del mouse es manejada por eventos al igual que la entrada del teclado. El evento MOUSEBUTTONUP ocurre cuando el jugador suelta el botón del mouse después de hacer clic en él.

En la línea 75, la coordenada x se almacena en event.pos[0], y la coordenada y se almacena en event.pos[1]. La línea 75 crea un nuevo objeto Rect para representar un nuevo cuadrado de comida y lo coloca donde ocurrió el evento MOUSEBUTTONUP . Al agregar un nuevo objeto Rect a la lista de alimentos , el código muestra un nuevo cuadrado de alimentos en el pantalla.

Además de agregarse manualmente a discreción del jugador, los cuadrados de comida se generan automáticamente a través del código en las líneas 77 a 81:

```
77. contador de comida += 1
78. si contadordecomida >= COMIDA NUEVA:
79.     # Agregar comida nueva.
80.     contador de comida = 0
81.     alimentos.append(pygame.Rect(random.randint(0, ANCHO DE VENTANA - TAMAÑO DE
COMIDA), random.randint(0, ALTO DE VENTANA - TAMAÑO DE COMIDA), TAMAÑO DE
COMIDA, TAMAÑO DE COMIDA))
```

La variable foodCounter realiza un seguimiento de la frecuencia con la que se deben agregar los alimentos. Cada vez que el bucle del juego itera, foodCounter se incrementa en 1 en la línea 77.

Una vez que foodCounter es mayor o igual a la constante

NEWFOOD, foodCounter se restablece y se genera un nuevo cuadrado de comida en la línea 81. Puede cambiar la velocidad a la que se agregan nuevos cuadrados de comida ajustando NEWFOOD nuevamente en la línea 21.

La línea 84 simplemente llena la superficie de la ventana con blanco, que cubrimos en "[Manejo cuando el jugador sale](#)" en la página 279, por lo que pasaremos a analizar cómo se mueve el jugador por la pantalla.

MOVER AL JUGADOR ALREDEDOR LA VENTANA

Hemos establecido las variables de movimiento (moveDown, moveUp, moveLeft y moveRight) en True o False según las teclas que haya presionado el jugador. Ahora necesitamos mover el cuadro del jugador, que está representado por el objeto pygame.Rect almacenado en el jugador. Haremos esto ajustando las coordenadas x e y del jugador.

```

86. # Mueve al jugador.
87. if moveDown and player.bottom < WINDOWHEIGHT:
88.     player.top += MOVESPEED 89. if moveUp and
player.top > 0: 90. player.top -= MOVESPEED 91. if moveLeft
y player.left > 0: 92. player.left -= MOVESPEED 93. if
player.right < VENTANA: 94.

```

jugador.derecha += VELOCIDAD DE MOVIMIENTO

Si moveDown se establece en True (y la parte inferior del cuadro del jugador no está debajo del borde inferior de la ventana), entonces la línea 88 mueve el cuadro del jugador hacia abajo agregando MOVESPEED al atributo superior actual del jugador . Las líneas 89 a 94 hacen lo mismo para las otras tres direcciones.

Dibujar el jugador en la ventana

La línea 97 dibuja la casilla del jugador en la ventana:

```
96. # Dibuja al jugador sobre la superficie. 97.
pygame.draw.rect(ventanaSuperficie, NEGRO, jugador)
```

Después de mover el cuadro, la línea 97 lo dibuja en su nueva posición. La superficie de ventana pasada como primer parámetro le dice a Python en qué objeto de superficie dibujar el rectángulo. La variable BLACK , que tiene (0, 0, 0) almacenado, le dice a Python que dibuje un rectángulo negro. El objeto Rect almacenado en la variable del jugador le dice a Python la posición y el tamaño del rectángulo a dibujar.

Comprobación de colisiones Antes de dibujar los cuadrados de alimentos, el programa debe comprobar si el cuadro del jugador se ha superpuesto con alguno de los cuadrados. Si es así, entonces ese cuadrado debe eliminarse de la lista de alimentos . De esta manera, Python no dibujará ningún cuadrado de comida que la caja ya haya comido.

Usaremos el método de detección de colisiones que todos Rect los objetos tienen, collidrect(), en la línea 101:

```
99. # Comprueba si el jugador se ha cruzado con algún cuadrado de comida. 100.
para alimentos en alimentos[:]:
101.     if player.collidrect(comida):
102.         comidas.remove(comida)
```

En cada iteración a través del bucle for , el cuadro de comida actual de la lista de comidas (plural) se coloca en la variable comida (singular). El método collidrect () para los objetos pygame.Rect se pasa al objeto pygame.Rect del rectángulo del jugador como argumento

y devuelve True si los dos rectángulos chocan y False si no lo hacen. Si es Verdadero, la línea 102 elimina el cuadrado de comida superpuesto de la lista de alimentos .

NO CAMBIAS UNA LISTA MIENTRAS LA ITARAS

Tenga en cuenta que este bucle for es ligeramente diferente de cualquier otro bucle for que hayamos visto. Si observa detenidamente la línea 100, no itera sobre alimentos, sino sobre alimentos[:].

Recuerda cómo funcionan las rebanadas. foods[:2] se evalúa como una copia de la lista con los elementos desde el principio y hasta (pero sin incluir) el elemento en el índice 2.foods [:] le dará una copia de la lista con los elementos desde el principio hasta el final. Básicamente, alimentos[:] crea una nueva lista con una copia de todos los elementos en alimentos. Esta es una forma más corta de copiar una lista que, digamos, lo que hizo la función getBoardCopy() en el juego Tic-Tac-Toe del Capítulo 10.

No puede agregar o eliminar elementos de una lista mientras la itera. Python puede perder la noción de cuál debería ser el próximo valor de la variable de alimentos si el tamaño de la lista de alimentos siempre está cambiando. Piense en lo difícil que sería contar la cantidad de gominolas en un frasco mientras alguien agrega o quita gominolas.

Pero si itera sobre una copia de la lista (y la copia nunca cambia), agregar o eliminar elementos de la lista original no será un problema.

DIBUJANDO LOS CUADRADOS DE COMIDA EN LA VENTANA

El código de las líneas 105 y 106 es similar al código que usamos para dibujar la caja negra para el jugador:

```
104. # Dibuja la comida.
105. for i in range(len(alimentos)): 106.
    pygame.draw.rect(VERDE, ventanasuperficie,
```

La línea 105 recorre cada cuadrado de alimentos en la lista de alimentos , y la línea 106 dibuja el cuadrado de comida en superficieventana.

Ahora que el jugador y los cuadrados de comida están en la pantalla, la ventana está lista para actualizarse, así que llamamos al método update() en la línea 109 y finalizamos el programa llamando al método tick() en el objeto Clock que creamos anteriormente:

```
108. # Dibuja la ventana en la pantalla.  
109. pygame.display.update() 110.  
mainClock.tick(40)
```

El programa continuará a través del ciclo del juego y seguirá actualizándose hasta que el jugador se cierre.

RESUMEN

Este capítulo introdujo el concepto de detección de colisiones.

La detección de colisiones entre dos rectángulos es tan común en los juegos gráficos que pygame proporciona su propio método de detección de colisiones llamado `colliderect()` para objetos `pygame.Rect`.

Los primeros juegos de este libro estaban basados en texto. La salida del programa era texto impreso en la pantalla y la entrada era texto escrito por el jugador en el teclado. Pero los programas gráficos también pueden aceptar entradas de teclado y mouse.

Además, estos programas pueden responder a pulsaciones de teclas individuales cuando el jugador presiona o suelta una sola tecla. El jugador no tiene que escribir una respuesta completa y presionar ENTER. Esto permite una retroalimentación inmediata y mucho más.

juegos interactivos

Este programa interactivo es divertido, pero vayamos más allá de dibujar rectángulos. En el [Capítulo 20](#), aprenderá cómo cargar imágenes y reproducir efectos de sonido con pygame.

20

USO DE SONIDOS E IMÁGENES



En los capítulos 18 y 19, aprendió cómo crear programas GUI que tengan gráficos y puedan aceptar entradas desde el teclado y el mouse. También aprendiste a dibujar diferentes formas. En este capítulo, aprenderá a agregar sonidos, música e imágenes a sus juegos.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- Archivos de imagen y sonido
- Dibujar y cambiar la escala de sprites
- Agregar música y sonidos
- Activar y desactivar el sonido

AGREGAR IMÁGENES CON SPRITES

Un sprite es una sola imagen bidimensional que se utiliza como parte de los gráficos en una pantalla. La Figura 20-1 muestra algunos sprites de ejemplo.



Figura 20-1: Algunos ejemplos de sprites

Las imágenes de sprites se dibujan sobre un fondo. Puede voltear la imagen del sprite horizontalmente para que quede mirando hacia el otro lado. También puede dibujar la misma imagen de sprite varias veces en la misma ventana y puede cambiar el tamaño de los sprites para que sean más grandes o más pequeños que la imagen de sprite original. La imagen de fondo también puede considerarse un sprite grande.

La figura 20-2 muestra los sprites que se usan juntos.

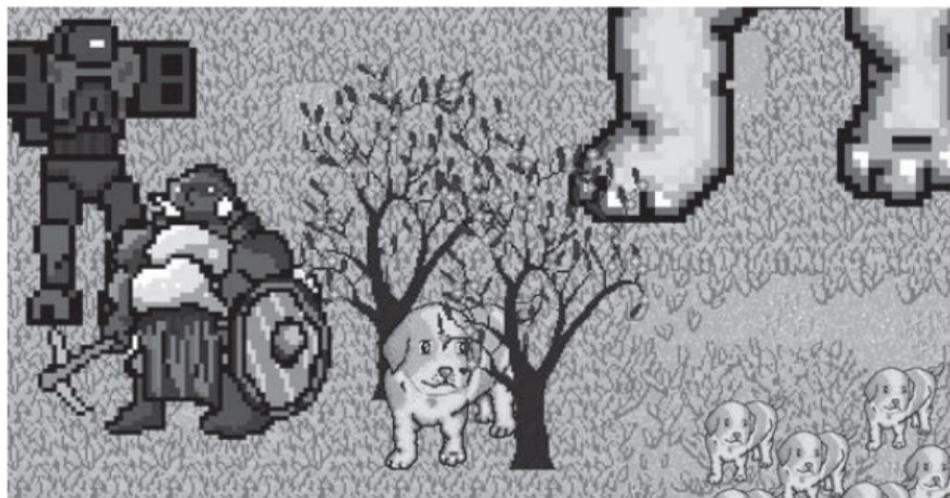


Figura 20-2: Una escena completa, con sprites dibujados sobre un fondo

El siguiente programa demostrará cómo reproducir sonidos.
y dibujar sprites usando pygame.

ARCHIVOS DE IMAGEN Y SONIDO

Los sprites se almacenan en archivos de imagen en su computadora. Hay varios formatos de imagen que pygame puede usar. Para saber qué formato usa un archivo de imagen, mire al final del nombre del archivo (después del último punto). Esto se llama la extensión del archivo. Por ejemplo, el archivo player.png está en formato PNG. Los formatos de imagen compatibles con pygame incluyen BMP, PNG, JPG y GIF.

Puede descargar imágenes desde su navegador web. En la mayoría de los navegadores web, lo hace haciendo clic con el botón derecho en la imagen de la página web y seleccionando Guardar en el menú que aparece. Recuerde en qué parte del disco duro guardó el archivo de imagen, ya que deberá copiar el archivo de imagen descargado en la misma carpeta que el archivo .py de su programa Python . También puedes crear tus propias imágenes con un programa de dibujo como Microsoft Paint o Tux Paint.

Los formatos de archivo de sonido que admite pygame son MIDI, WAV y MP3. Puede descargar efectos de sonido de Internet al igual que puede descargar archivos de imagen, pero los archivos de sonido deben estar en uno de estos tres formatos. Si su computadora tiene un micrófono, también puede grabar sonidos y crear sus propios archivos WAV para usar en sus juegos.

MUESTRA DE EJECUCIÓN DE LOS SPRITES PROGRAMA DE SONIDOS

El programa de este capítulo es el mismo que el programa de Detección de colisiones del [Capítulo 19](#). Sin embargo, en este programa usaremos sprites en lugar de simples cuadrados. Usaremos un sprite de una persona para representar al jugador en lugar de la caja negra y un sprite de cerezas en lugar de los cuadrados de comida verde. También reproduciremos música de fondo y agregaremos un efecto de sonido cuando el

el sprite del jugador come una de las cerezas.

En este juego, el sprite del jugador comerá sprites de cereza y, a medida que coma las cerezas, crecerá. Cuando ejecute el programa, el juego se parecerá a la Figura 20-3.

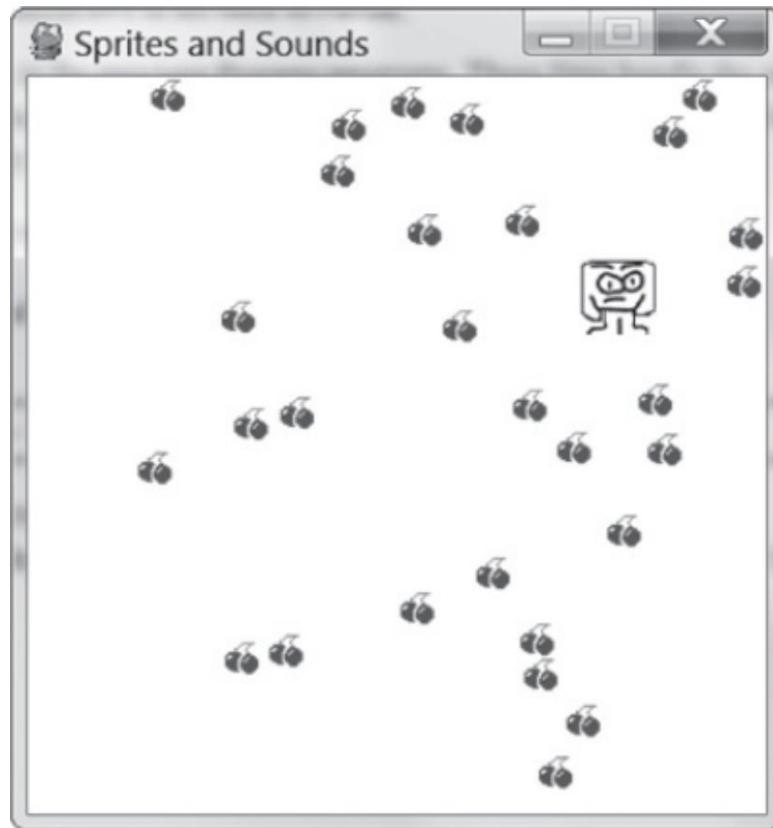


Figura 20-3: Una captura de pantalla del juego Sprites and Sounds

CÓDIGO FUENTE DE LOS SPRITES PROGRAMA DE SONIDOS

Inicie un nuevo archivo, ingrese el siguiente código y luego guárdelo como `spritesAndSounds.py`. Puede descargar los archivos de imagen y sonido que usaremos en este programa desde el sitio web de este libro en <https://www.nostarch.com/inventwithpython/>. Coloque estos archivos en la misma carpeta que el programa `spritesAndSounds.py`.



Si obtiene errores después de ingresar este código, compare el código que escribió con el código del libro con la herramienta de comparación en línea en <https://www.nostarch.com/inventwithpython#diff>.

spritesY Sonidos.py

```
1. import pygame, sys, time, random 2. from
pygame.locals import *
3.
4. # Configurar pygame.
5. pygame.init() 6.
mainClock = pygame.time.Clock()
7.
8. # Configure la ventana.
9. ANCHO DE VENTANA = 400
10. ALTURA DE LA VENTANA = 400
11. superficieVentana = pygame.display.set_mode((ANCHOVENTANA,
ALTOVENTANA), 0, 32)

12. pygame.display.set_caption('Sprites y Sonidos') 13.

14. # Configura los colores.
15. BLANCO = (255, 255, 255)
16.
17. # Configure la estructura de datos del
bloque. 18. jugador = pygame.Rect(300, 100, 40, 40)
```

```
19. playerImage = pygame.image.load('player.png') 20.  
playerStretchedImage = pygame.transform.scale(playerImage, (40, 40)) 21. foodImage =  
pygame.image.load('cherry.png') 22. alimentos = [] 23. para i en rango (20): 24. alimentos.  
20))  
  
25  
26. contador de comida = 0  
27. COMIDA NUEVA = 40  
28  
29. # Configurar variables de teclado.  
30. moverIzquierda = Falso  
31. moverDerecha = Falso  
32. moverArriba = Falso 33.  
moverAbajo = Falso  
34.  
35. VELOCIDAD DE MOVIMIENTO = 6  
36.  
37. # Configura la música.  
38. pickUpSound = pygame.mixer.Sound('pickup.wav') 39.  
pygame.mixer.music.load('background.mid') 40.  
pygame.mixer.music.play(-1, 0.0) 41. musicPlaying = Verdadero 42.  
  
43. # Ejecuta el ciclo del juego.  
44. mientras sea cierto:  
45. # Comprobar el evento QUIT. 46. for  
event in pygame.event.get(): if event.type ==  
47.     QUIT: pygame.quit() sys.exit() if  
48.         event.type == KEYDOWN:  
49.  
50  
51.         # Cambiar las variables del teclado. if  
52.         event.key == K_LEFT or event.key == K_a: moveRight =  
53.             False moveLeft = True  
54.  
55.         if evento.clave == K_DERECHO o evento.clave == K_d:  
56.             moverIzquierda = Falso  
57.             moverDerecha = Verdadero
```

```
58.         if evento.clave == K_UP o evento.clave == K_w:
59.             mover hacia abajo = falso
60.
61.             mover hacia arriba = verdadero
62.
63.             if evento.clave == K_DOWN o evento.clave == K_s:
64.                 moveUp = Falso
65.                 moveDown = Verdadero
66.
67.                 si event.type == KEYUP:
68.
69.                     if event.key == K_ESCAPE:
70.
71.                         pygame.quit() sys.exit() if event.key
72.                         == K_LEFT or event.key == K_a:
73.                         moveLeft = False
74.
75.                         if event.key == K_RIGHT o event.key == K_d: moveRight =
76.                             False if event.key == K_UP o event.key == K_w: moveUp =
77.                                 False
78.
79.                         if event.key == K_DOWN or event.key == K_s: moveDown =
80.                             False
81.
82.                         if event.key == K_x:
83.
84.                             player.top = random.randint(0, WINDOWHEIGHT - player.height)
85.                             player.left = random.randint(0, WINDOWWITH - player.width) if
86.                             event.key == K_m: if musicPlaying : pygame.mixer.music.stop()
87.
88.
89.         contador de comida += 1
90.         si contadordecomida >= COMIDA NUEVA:
91.             # Agregar comida nueva.
92.             contador de comida = 0
93.             alimentos.append(pygame.Rect(random.randint(0, ANCHO DE VENTANA - 20),
94.                                         random.randint(0, ALTO DE VENTANA - 20), 20, 20))
```

```
94.  
95. # Dibuja el fondo blanco sobre la superficie. 96.  
superficieventana.rellenar(BLANCO)  
97.  
98. # Mueve al jugador. 99. if  
moveDown and player.bottom < WINDOWHEIGHT: 100. player.top +=  
MOVESPEED 103. 101. if moveUp and player.top > 10: player.top -= MOVESPEED  
104. 102. VELOCIDAD DE MOVIMIENTO  
102.  
  
105. si moverDerecha y jugador.derecha < ANCHO DE VENTANA: 106.  
jugador.derecha += VELOCIDAD DE MOVIMIENTO  
107.  
108.  
109. # Dibuja el bloque sobre la superficie.  
110. superficieVentana.blit(imagenestiradadeljugador, jugador) 111.  
  
112. # Comprueba si el bloque se ha cruzado con algún cuadrado de comida. 113. para  
alimento en alimentos[:]:  
114.     if  
115.  
116.         player.colliderect(food):foods.remove(food) player =  
pygame.Rect(player.left, player.top, player.width + 2,  
117.         player.height + 2) playerStretchedImage = pygame.transform.scale(playerImage,  
(jugador.ancho, jugador.altura)) if musicPlaying: pickUpSound.play()  
118.  
119.  
120.  
121. # Dibuja la comida.  
122. para alimentos en alimentos:  
123.     superficieVentana.blit(comidalmagen, comida)  
124.  
125. # Dibuja la ventana en la pantalla.  
126. pygame.display.update() 127.  
mainClock.tick(40)
```

CONFIGURAR LA VENTANA Y LA ESTRUCTURA DE DATOS

La mayor parte del código de este programa es el mismo que el programa de detección de colisiones del [capítulo 19](#). Nos centraremos solo en las partes que agregan sprites y sonidos. Primero, en la línea 12 establezcamos el título de la barra de título en una cadena que describa este programa:

```
12. pygame.display.set_caption('Sprites y Sonidos')
```

Para establecer el título, debe pasar la cadena 'Sprites and Sounds' a la función `pygame.display.set_caption()`.

Agregando un Sprite

Ahora que tenemos la leyenda configurada, necesitamos los sprites reales. Usaremos tres variables para representar al jugador, a diferencia de los programas anteriores que usaban solo una.

```
17. # Configure la estructura de datos  
del bloque. 18. jugador = pygame.Rect(300,  
100, 40, 40) 19. playerImage =  
pygame.image.load('player.png') 20. playerStretchedImage =  
pygame.transform.scale(playerImage, (40, 40)) 21. comidalmagen = pygame.image.load('cereza.png')
```

La variable del jugador en la línea 18 almacenará un objeto Rect que realiza un seguimiento de la ubicación y el tamaño del jugador. La variable del jugador no contiene la imagen del jugador. Al comienzo del programa, la esquina superior izquierda del reproductor se encuentra en (300, 100) y el reproductor tiene una altura y un ancho iniciales de 40 píxeles.

La segunda variable que representa al jugador es `playerImage` en la línea 19. A la función `pygame.image.load()` se le pasa una cadena del nombre de archivo de la imagen a cargar. El valor devuelto es un objeto Surface que tiene los gráficos del archivo de imagen dibujados en su superficie. Almacenamos este objeto Surface dentro de `playerImage`.

Cambiando el Tamaño de un Sprite En la línea 20, usaremos una nueva función en el módulo pygame.transform . La función pygame.transform.scale() puede reducir o ampliar un sprite. El primer argumento es un objeto Surface con la imagen dibujada en él. El segundo argumento es una tupla para el nuevo ancho y alto de la imagen en el primer argumento. La función scale() devuelve un objeto Surface con la imagen dibujada en un nuevo tamaño. En el programa de este capítulo, haremos que el sprite del jugador se estire más a medida que come más cerezas. Guardaremos la imagen original en la variable playerImage pero la imagen estirada en la variable playerStretchedImage .

En la línea 21, volvemos a llamar a load() para crear un objeto Surface con la imagen de la cereza dibujada en él. Asegúrese de tener los archivos player.png y cherry.png en la misma carpeta que el archivo spritesAndSounds.py ; de lo contrario, pygame no podrá encontrarlos y dará un error.

CONFIGURACIÓN DE LA MÚSICA Y SONIDOS

A continuación, debe cargar los archivos de sonido. Hay dos módulos para sonido en pygame. El módulo pygame.mixer puede reproducir efectos de sonido cortos durante el juego. El módulo pygame.mixer.music puede reproducir música de fondo.

Adición de archivos de sonido Llame a la función constructora pygame.mixer.Sound() para crear un objeto pygame.mixer.Sound (denominado objeto de sonido para abreviar). Este objeto tiene un método play() que reproducirá el efecto de sonido cuando

llamó.

```
37. # Configura la
música. 38. pickUpSound = pygame.mixer.Sound('pickup.wav')
39. pygame.mixer.music.load('background.mid') 40.
pygame.mixer.music.play(-1, 0.0) 41. musicPlaying =
Verdadero
```

La línea 39 llama a `pygame.mixer.music.load()` para cargar la música de fondo y la línea 40 llama a `pygame.mixer.music.play()` para comenzar a reproducirla. El primer parámetro le dice a pygame cuántas veces reproducir la música de fondo después de la primera vez que la tocamos. Entonces, pasar 5 haría que pygame reprodujera la música de fondo seis veces.

Aquí le pasamos el parámetro -1, que es un valor especial que hace que la música de fondo se repita eternamente.

El segundo parámetro para reproducir () es el punto en el archivo de sonido para comenzar a reproducir. Pasar 0.0 reproducirá la música de fondo desde el principio. Pasar 2.5 iniciaría la música de fondo 2.5 segundos desde el principio.

Finalmente, la variable `musicPlaying` tiene un valor booleano que le dice al programa si debe reproducir la música de fondo y los efectos de sonido o no. Es bueno darle al reproductor la opción de ejecutar el programa sin que se reproduzca el sonido.

Activar y desactivar el sonido La tecla M activará o desactivará la música de fondo. Si `musicPlaying` está establecido en True, entonces la música de fondo se está reproduciendo actualmente y deberíamos detenerla llamando a `pygame.mixer.music.stop()`.

Si `musicPlaying` está establecido en False, entonces la música de fondo no se está reproduciendo actualmente y deberíamos iniciarla llamando a `play()`. Líneas 79 a 84 use sentencias if para hacer esto:

```

79.     if event.key == K_m: if
80.         musicPlaying:
81.             pygame.mixer.music.stop() else:
82.
83.                 pygame.mixer.music.play(-1, 0.0)
84.             música en reproducción = no música en reproducción

```

Ya sea que la música se esté reproduciendo o no, queremos alternar el valor en musicPlaying. Alternar un valor booleano significa establecer un valor en el opuesto de su valor actual. La línea musicPlaying = not musicPlaying establece la variable en False si actualmente es True o la establece en True si actualmente es False. Piense en alternar como lo que sucede cuando enciende o apaga un interruptor de luz: alternar el interruptor de luz lo establece en la configuración opuesta.

DIBUJANDO AL JUGADOR EN EL VENTANA

Recuerde que el valor almacenado en playerStretchedImage es una superficie objeto. La línea 110 dibuja el sprite del jugador en el objeto Surface de la ventana (que se almacena en windowSurface) usando

blit():

109. # Dibuja el bloque sobre la superficie.

110. superficieVentana.blit(jugadorImagenEstirada, jugador)

El segundo parámetro del método blit() es un objeto Rect que especifica en qué parte del objeto Surface debe dibujarse el sprite. El programa utiliza el objeto Rect almacenado en el reproductor, que realiza un seguimiento de la posición del reproductor en la ventana.

COMPROBACIÓN DE COLISIONES

Este código es similar al código de los programas anteriores, pero hay un par de líneas nuevas:

```
114.     if
115.
116.         player.colliderect(food):foods.remove(food)
117.             player = pygame.Rect(player.left, player.top,
118.                 player.width + 2, player.height + 2) playerStretchedImage =
119.                     pygame.transform.scale(playerImage, (jugador.ancho,
118.                         jugador.altura)) if musicPlaying: pickUpSound.play()
```

Cuando el sprite del jugador come una de las cerezas, su tamaño aumenta en dos píxeles de alto y ancho. En la línea 116, se asignará un nuevo objeto Rect que es dos píxeles más grande que el antiguo objeto Rect como el nuevo valor de player.

Mientras que el objeto Rect representa la posición y el tamaño del reproductor, la imagen del reproductor se almacena en un playerStretchedImage como un objeto Surface . En la línea 117, el programa crea una nueva imagen estirada llamando a scale().

Estirar una imagen a menudo la distorsiona un poco. Si sigue estirando una imagen ya estirada, las distorsiones se suman rápidamente. Pero al estirar la imagen original a un nuevo tamaño cada vez, al pasar playerImage, no playerStretchedImage, como primer argumento para scale(), distorsiona la imagen solo una vez.

Finalmente, la línea 119 llama al método play() en el objeto Sound almacenado en la variable pickUpSound . Pero hace esto solo si musicPlaying está configurado en True (lo que significa que el sonido está activado).

DIBUJANDO LAS CEREZAS EN EL

VENTANA

En los programas anteriores, llamó a la función `pygame.draw.rect()` para dibujar un cuadrado verde para cada objeto `Rect` almacenado en la lista de alimentos . En este programa, sin embargo, desea dibujar los sprites de cereza en su lugar. Llame al método `blit()` y pase el objeto `Surface` almacenado en `foodImage`, que tiene dibujada la imagen de las cerezas en eso:

```
121. # Dibuja la comida.  
122. para alimentos en alimentos:  
123.     superficieVentana.blit(comidalmagen, comida)
```

La variable `comida` , que contiene cada uno de los objetos `Rect` en `comidas` en cada iteración a través del ciclo `for` , le dice al método `blit()` dónde dibujar la imagen `comida`.

RESUMEN

Ha agregado imágenes y sonido a su juego. Las imágenes, llamadas sprites, se ven mucho mejor que las simples formas dibujadas que se usaban en los programas anteriores. Los sprites se pueden escalar (es decir, estirar) a un tamaño más grande o más pequeño, por lo que podemos mostrar los sprites en cualquier tamaño que queramos. El juego presentado en este capítulo también tiene un fondo y reproduce efectos de sonido.

Ahora que sabemos cómo crear una ventana, mostrar sprites, dibujar primitivos, recopilar entradas de teclado y mouse, reproducir sonidos e implementar la detección de colisiones, estamos listos para crear un juego gráfico en pygame. El Capítulo 21 reúne todos estos elementos para nuestro juego más avanzado hasta el momento.

21

UN JUEGO DE DODGER CON SONIDOS E IMÁGENES



Los cuatro capítulos anteriores repasaron el módulo pygame y demostraron cómo usar sus muchas funciones. En este capítulo, usaremos ese conocimiento para crear un juego gráfico llamado Dodger.

TEMAS CUBIERTOS EN ESTE CAPÍTULO

- La bandera pygame.PANTALLA COMPLETA
- El método Move_ip() Rect
- Implementación de códigos de trucos
- Modificación del juego de los Dodgers

En el juego Dodger, el jugador controla un sprite (el personaje del jugador) que debe esquivar un montón de malos que caen desde la parte superior de la pantalla. Cuanto más tiempo el jugador pueda seguir esquivando a los malos, mayor será su puntuación.

Solo por diversión, también agregaremos algunos modos de trucos a este juego. Si el jugador mantiene presionada la tecla X, la velocidad de cada malo es

reducido a un ritmo súper lento. Si el jugador mantiene presionada la tecla Z, los malos invertirán su dirección y viajarán hacia arriba en la pantalla en lugar de hacia abajo.

REVISIÓN DEL PYGAME BÁSICO

TIPOS DE DATOS

Antes de comenzar a crear Dodger, repasemos algunos de los tipos de datos básicos que se usan en pygame:

pygame.Rect

Los objetos Rect representan la ubicación y el tamaño de un espacio rectangular.

La ubicación está determinada por el atributo topleft del objeto Rect (o los atributos topright, bottomleft y bottomright) .

Estos atributos de esquina son una tupla de enteros para las coordenadas x e y. El tamaño está determinado por los atributos de ancho y alto , que son números enteros que indican cuántos píxeles de largo o alto tiene el rectángulo. Los objetos Rect tienen un método colliderect () que verifica si están colisionando con otro objeto Rect .

pygame.Surface

Los objetos de superficie son áreas de píxeles de colores. Un objeto Surface representa una imagen rectangular, mientras que un objeto Rect representa solo un espacio y una ubicación rectangulares. Los objetos de superficie tienen un método blit() que se utiliza para dibujar la imagen de un objeto de superficie en otro objeto de superficie . El objeto de superficie devuelto por la función pygame.display.set_mode() es especial porque cualquier cosa dibujada en ese objeto de superficie se muestra en la pantalla del usuario cuando pygame.display.update() está

llamó.

pygame.event.Evento

El módulo pygame.event genera objetos de evento cada vez que el usuario proporciona el teclado, el mouse u otra entrada. La función pygame.event.get() devuelve una lista de estos objetos Event .

Puede determinar el tipo del objeto Evento comprobando su atributo de tipo .

SALIR, TECLADO ABAJO y MOUSEBUTTONDOWN son

ejemplos de algunos tipos de eventos. (Consulte “[Manejo de eventos](#)” en la página 292 para obtener una lista completa de todos los tipos de eventos).

pygame.font.Fuente

El módulo pygame.font utiliza el tipo de datos Font , que representa el tipo de letra utilizado para el texto en pygame. Los argumentos para pasar a pygame.font.SysFont() son una cadena del nombre de la fuente (es común pasar None para que el nombre de la fuente obtenga la fuente predeterminada del sistema) y un número entero del tamaño de la fuente.

pygame.time.Reloj

El objeto Reloj en el módulo pygame.time es útil para evitar que nuestros juegos se ejecuten más rápido de lo que el jugador puede ver. El objeto Clock tiene un método tick() , al que se le puede pasar el número de fotogramas por segundo (FPS) que queremos que se ejecute el juego. Cuanto mayor sea el FPS, más rápido será el juego carreras.

MUESTRA DE EJECUCIÓN DE DODGER

Cuando ejecute este programa, el juego se verá como la Figura 21-

1.



Figura 21-1: Una captura de pantalla del juego de los Dodgers

CÓDIGO FUENTE PARA DODGER

Ingrrese el siguiente código en un archivo nuevo y guárdelo como `dodger.py`.

Puede descargar los archivos de código, imagen y sonido desde <https://www.nostarch.com/inventwithpython/>. Coloque los archivos de imagen y sonido en la misma carpeta que `dodger.py`.



Si obtiene errores después de ingresar este código, compare el código que escribió con el código del libro con la herramienta de comparación en línea en <https://www.nostarch.com/inventwithpython#diff>.

dodger.py

```
1. importar pygame, random, sys 2.  
desde pygame.locals importar *  
3.  
4. ANCHO DE VENTANA = 600  
5. ALTURA DE LA VENTANA = 600  
6. COLOR DEL TEXTO = (0, 0, 0)  
7. COLOR DE FONDO = (255, 255, 255)  
8FPS = 60  
9. TAMAÑO DE MALO = 10  
10. TAMAÑO MÁXIMO DEL MALO = 40  
11. MALO VELOCIDAD = 1  
12. VELOCIDAD MÁXIMA DEL MALO = 8  
13. AGREGARNUEVOBADDIERATE = 6  
14. TASA DE MOVIMIENTO DEL JUGADOR = 5  
15.  
16. def terminar(): 17.  
pygame.quit() 18.  
sys.exit()  
19  
20. def esperaParaPlayerToPressKey():
```

```
21. mientras sea cierto:  
22     for event in pygame.event.get(): if  
23         event.type == QUIT: terminar() if  
24             event.type == KEYDOWN: if  
25                 event.key == K_ESCAPE: #  
26                     Presionar ESC para salir. Terminar()  
27  
28     retorno  
29  
30. def playerHasHitBaddie(playerRect, baddies): 31. for b  
in baddies:  
32.     if playerRect.colliderect(b['rect']):  
33.         devuelve True  
34. volver Falso  
35.  
36. def dibujarTexto(texto, fuente, superficie, x,  
y): 37. textobj = fuente.render(texto, 1, TEXTCOLOR) 38.  
textrect = textobj.get_rect() 39. textrect.topleft = (x, y) 40.  
superficie.blit(textobj, textrect)  
  
41.  
42. # Configure pygame, la ventana y el cursor del mouse. 43.  
pygame.init() 44. mainClock = pygame.time.Clock() 45.  
windowSurface = pygame.display.set_mode((ANCHOVENTANA,  
ALTOVENTANA)) 46. pygame.display.set_caption('Dodger') 47. pygame.  
ratón.set_visible(Falso)  
  
48.  
49. # Configure las  
fuentes. 50. fuente = pygame.font.SysFont(Ninguno,  
48) 51.  
52. # Configurar  
sonidos. 53. gameOverSound = pygame.mixer.Sound('gameover.wav')  
54. pygame.mixer.music.load('background.mid')  
55.  
56. # Configurar  
imágenes. 57. playerImage = pygame.image.load('player.png')  
58. playerRect = playerImage.get_rect() 59. baddieImage =  
pygame.image.load("baddie.png")
```

60
61. # Muestra la pantalla "Inicio".
62. superficieventana.fill(COLOR DE FONDO) 63.
dibujarTexto('Dodger', fuente, superficieventana, (ANCHOVENTANA / 3), (ALTOVENTANA /
3)) 64. dibujarTexto('Presione una tecla para comenzar.', fuente, superficieventana,
(ANCHO DE VENTANA / 3) - 30, (ALTO DE VENTANA / 3) + 50)
65. pygame.display.update() 66.
waitForPlayerToPressKey()
67.
68. topScore = 0 69.
while True:
70. # Configura el inicio del juego. 71. malos
= []
72. puntuación = 0
73. playerRect.topleft = (WINDOWWIDTH / 2, WINDOWHEIGHT - 50) 74. moveLeft = moveRight
= moveUp = moveDown = False 75. reverseCheat = slowCheat = False
76. baddieAddCounter = 0
77. pygame.mixer.music.play(-1, 0.0)
78.
79. while True: # El bucle del juego se ejecuta mientras se reproduce la parte del juego.
80. puntuación += 1 # Incrementar puntuación.
81.
82. para evento en pygame.event.get(): if
83. event.type == SALIR: terminar()
84.
85.
86. if event.type == KEYDOWN: if
87. event.key == K_z:
88. truco inverso = Verdadero
89. if evento.clave == K_x:
90. trampaleta = Verdadero
91. if event.key == K_LEFT or event.key == K_a: moveRight =
92. False moveLeft = True
93.
94. if evento.clave == K_DERECHO o evento.clave == K_d:
95. moverIzquierda = Falso
96. moveRight = True si
97. event.key == K_UP o event.key == K_w:

```

98.         mover hacia abajo = falso
99.         mover hacia arriba = verdadero
100.        if evento.clave == K_DOWN o evento.clave == K_s:
101.            mover hacia arriba = Falso
102.            mover hacia abajo = verdadero
103.
104.        if evento.tipo == KEYUP: if
105.            evento.clave == K_z:
106.                truco inverso = Falso
107.                puntuación = 0
108.            if event.key == K_x:
109.                slowCheat = False
110.                puntuación = 0
111.            si evento.clave == K_ESCAPE:
112.                terminar()
113.
114.        if event.key == K_LEFT or event.key == K_a: moveLeft =
115.            False
116.        if event.key == K_RIGHT o event.key == K_d: moveRight =
117.            False if event.key == K_UP o event.key == K_w: moveUp =
118.            False
119.
120.        if event.key == K_DOWN or event.key == K_s: moveDown =
121.            False
122.
123.        if evento.tipo == MOVIMIENTO DEL RATÓN:
124.            # Si el mouse se mueve, mueve al jugador al cursor.
125.            rect.jugador.centerx = evento.pos[0] rect.jugador.centery =
126.            evento.pos[1]
127.            # Agregue nuevos malos en la parte superior de la pantalla, si es
128.            necesario. si no reverseCheat y no slowCheat:
129.                baddieAddCounter += 1
130.            if baddieAddCounter == AGREGARNUEVOBADDIERATE:
131.                baddieAddCounter = 0
132.                baddieSize = random.randint(BADDIEMINSIZE,
BADDIEMAXSIZE) 133.
nuevoMalo = {'rect': pygame.Rect(random.randint(0,
ANCHO DE VENTANA - tamaño del malo), 0 - tamaño del malo,
tamaño del malo, tamaño del malo), 'velocidad': aleatorio.randint
134. (VELOCIDAD DEL MALO,
```

```
    BADDIEMAXSPEED),
135.        'superficie':pygame.transform.scale(imagenMalo,
136.            (TamañoMalo, TamañoMalo)), }

137.

138.        malos.append(nuevoMalo)
139.

140.    # Mueve al jugador. si moveLeft
141.    y playerRect.left > 0:
142.        playerRect.move_ip(-1 * PLAYEROVERRATE, 0) if
143.        moveRight y playerRect.right < WINDOWWITH:
144.            playerRect.move_ip(PLAYEROVERRATE, 0) if moveUp y
145.        playerRect.top > 0:
146.            playerRect.move_ip(0, -1 * PLAYEROVERRATE) if
147.        moveDown and playerRect.bottom < WINDOWHEIGHT:
148.            playerRect.move_ip(0, PLAYEROVERRATE)

149.

150.    # Mueve a los malos hacia abajo.
151.    para b en malos:
152.        si no reverseCheat y no slowCheat:
153.            b['rect'].move_ip(0, b['speed']) elif
154.        reverseCheat:
155.            b['rect'].move_ip(0, -5)
156.        elif truco lento:
157.            b['rect'].move_ip(0, 1)
158.

159.    # Eliminar los malos que han caído más allá del fondo.
160.    para b en malos[:]:
161.        if b['rect'].top > WINDOWHEIGHT:
162.            malos.remove(b)
163.

164.    # Dibuja el mundo del juego en la ventana.
165.    superficieventana.fill(COLOR DE FONDO)
166.

167.    # Dibuja la puntuación y la puntuación
168.    máxima. dibujarTexto('Puntuación: %s' % (puntuación), fuente, superficieventana,
169.    10, 0) dibujarTexto('Puntuación máxima: %s' % (puntuación), fuente,
170.    superficieventana, 10, 40)
171.    # Dibuja el rectángulo del jugador.
```

```

172.     superficieVentana.blit(ImagenJugador, RectoJugador)
173.
174.     # Dibuja a cada malo.
175.     para b en malos:
176.         superficieVentana.blit(b['superficie'], b['rect'])
177.
178.     pygame.display.update()
179.
180.     # Comprueba si alguno de los malos ha golpeado al
181.     jugador. if playerHasHitBaddie(playerRect, baddies): if
182.         score > topScore:
183.             topScore = puntuación # Establecer nueva puntuación máxima.
184.             descanso
185.
186.             relojprincipal.tick(FPS)
187.

188. # Detener el juego y mostrar la pantalla "Game Over". 189.
    pygame.mixer.music.stop() 190. gameOverSound.play()

191.
192. drawText('GAME OVER', fuente, superficieVentana, (ANCHOVENTANA / 3),
    (ALTOVENTANA / 3)) 193. dibujarText('Presiona una tecla para volver a jugar.', fuente,
    superficieVentana, (ANCHOVENTANA / 3) - 80 , (ALTURA DE LA VENTANA / 3) + 50) 194.
    pygame.display.update() 195. waitForPlayerToPressKey()

196.
197. juegoOverSound.stop()

```

IMPORTACIÓN DE LOS MÓDULOS

El juego Dodger importa los mismos módulos que los programas pygame anteriores : pygame, random, sys y pygame.locals.

```

1. importar pygame, random, sys 2.
desde pygame.locals importar *

```

El módulo pygame.locals contiene varias variables constantes

que usa pygame , como los tipos de eventos (QUIT, KEYDOWN, etc.) y las teclas del teclado (K_ESCAPE, K_LEFT, etc.). Al usar la sintaxis from pygame.locals import * , puede usar QUIT en el código fuente en lugar de pygame.locals.QUIT.

CONFIGURACIÓN DE LA CONSTANTE VARIABLES

Las líneas 4 a 7 configuran constantes para las dimensiones de la ventana, el color del texto y el color de fondo:

-
- 4. ANCHO DE VENTANA = 600
 - 5. ALTURA DE LA VENTANA = 600
 - 6. COLOR DEL TEXTO = (0, 0, 0)
 - 7. COLOR DE FONDO = (255, 255, 255)
-

Usamos variables constantes porque son mucho más descriptivas que si hubiéramos escrito los valores. Por ejemplo, la línea windowSurface.fill(BACKGROUNDCOLOR) es más comprensible que windowSurface.fill((255, 255, 255)).

Puedes cambiar fácilmente el juego cambiando las variables constantes. Al cambiar ANCHO DE VENTANA en la línea 4, cambia automáticamente el código en todos los lugares donde se usa ANCHO DE VENTANA . Si hubiera utilizado el valor 600 en su lugar, tendría que cambiar cada aparición de 600 en el código. es mas facil cambiar el valor en la constante una vez.

En la línea 8, configura la constante para el FPS, la cantidad de cuadros por segundo que desea que se ejecute el juego:

8FPS = 60

Un marco es una pantalla que se dibuja para una sola iteración

a través del bucle del juego. Pasas FPS al método mainClock.tick () en la línea 186 para que la función sepa cuánto tiempo pausar el programa. Aquí el FPS está configurado en 60, pero puede cambiar el FPS a un valor más alto para que el juego se ejecute más rápido o a un valor más bajo. valor para ralentizarlo.

Las líneas 9 a 13 establecen algunas variables más constantes para el Cayendo malos:

-
- 9. TAMAÑO DE MALO = 10
 - 10. TAMAÑO MÁXIMO DEL MALO = 40
 - 11. MALO VELOCIDAD = 1
 - 12. VELOCIDAD MÁXIMA DEL MALO = 8
 - 13. AGREGARNUEVOBADDIERATE = 6
-

El ancho y la altura de los malos estarán entre BADDIEMINSIZE y BADDIEMINSIZE y BADDIEMAXSIZE. La tasa a la que el los malos caen la pantalla estará entre BADDIEMINSPEED y BADDIEMAXSPEED píxeles por iteración a través del bucle del juego. Y se agregará un nuevo villano en la parte superior de la ventana cada iteración ADDNEWBADDIERATE a través del ciclo del juego.

Finalmente, PLAYEROVERRATE almacena la cantidad de píxeles que el personaje del jugador se mueve en la ventana en cada iteración a través del ciclo del juego (si el personaje se está moviendo):

-
- 14. TASA DE MOVIMIENTO DEL JUGADOR = 5
-

Al aumentar este número, puede aumentar la velocidad en que se mueve el personaje.

DEFINICIÓN DE FUNCIONES

Hay varias funciones que crearás para este juego. Él

Las funciones `finish()` y `waitForPlayerToPressKey()` finalizarán y pausarán el juego, respectivamente, la función `playerHasHitBaddie()` rastreará las colisiones del jugador con los malos, y `drawText()`. La función dibujará la partitura y otro texto en la pantalla.

Finalización y pausa del juego El módulo pygame requiere que llame tanto a `pygame.quit()` como a `sys.exit()` para finalizar el juego. Las líneas 16 a 18 los colocan a ambos en una función llamada `terminar()`.

```
16. def terminar():
17.     pygame.quit() 18.
18.     sys.exit()
```

Ahora solo necesita llamar a `terminar()` en lugar de ambos `pygame.quit()` y `sys.exit()`.

A veces querrás pausar el programa hasta que el jugador presione una tecla, como al comienzo del juego cuando aparece el texto del título de los Dodgers o al final cuando se muestra Game Over . Las líneas 20 a 24 crean una nueva función llamada

`esperarParaJugadorToPressKey()`:

```
20. def waitForPlayerToPressKey():
21.     while True:
22.         for event in pygame.event.get():
23.             if event.type == SALIR:
24.                 terminar()
```

Dentro de esta función, hay un ciclo infinito que se interrumpe solo cuando se recibe un evento KEYDOWN o QUIT . Al comienzo del ciclo, `pygame.event.get()` devuelve una lista de objetos de evento para verificar afuera.

Si el jugador ha cerrado la ventana mientras el programa está

esperando que el jugador presione una tecla, pygame generará un evento QUIT , que verifica en la línea 23 con event.type. Si el jugador ha salido, Python llama a la función terminar() en la línea 24.

Si el juego recibe un evento KEYDOWN , primero debe verificar si se presionó ESC :

```

25     if event.type == KEYDOWN: if
26         event.key == K_ESCAPE: # Presionar ESC para salir.
27             Terminar()
28         retorno

```

Si el jugador presionó ESC, el programa debería terminar. Si ese no fuera el caso, entonces la ejecución omitirá el bloque if en la línea 27 e irá directamente a la declaración de retorno , que sale de la función waitForPlayerToPressKey() .

Si no se genera un evento QUIT o KEYDOWN , el código continúa en bucle. Dado que el bucle no hace nada, parecerá que el juego se ha congelado hasta que el jugador presione una tecla.

Hacer un seguimiento de las colisiones de Baddie

La función playerHasHitBaddie() devolverá True si el jugador el personaje ha chocado con uno de los malos:

```

30. def playerHasHitBaddie(playerRect, malos):
31. para b en malos:
32.     if jugadorRect.colliderect(b['rect']):
33.         volver verdadero
34. volver Falso

```

El parámetro baddies es una lista de estructuras de datos del diccionario baddie. Cada uno de estos diccionarios tiene una clave 'rect' , y el valor de esa clave es un objeto Rect que representa la clave del malo. tamaño y ubicación.

playerRect también es un objeto Rect . Los objetos Rect tienen un método llamado collidrect() que devuelve True si el objeto Rect ha colisionado con el objeto Rect que se le pasa. De lo contrario, colisionar () devuelve Falso.

El bucle for en la línea 31 itera a través de cada diccionario de malos en la lista de malos. Si alguno de estos malos choca con el personaje del jugador, playerHasHitBaddie() devuelve True.

Si el código logra iterar a través de todos los malos en la lista de malos sin detectar una colisión, playerHasHitBaddie() devuelve Falso.

Dibujar texto en la ventana Dibujar texto en la ventana implica algunos pasos, que realizamos con drawText(). De esta manera, solo hay una función para llamar cuando queremos mostrar la puntuación del jugador o el texto de Game Over en la pantalla.

```
36. def dibujarTexto(texto, fuente,  
superficie, x, y): 37. textobj = fuente.render(texto, 1,  
TEXTCOLOR) 38. textrect = textobj.get_rect() 39.  
textrect.topleft = (x, y) 40. superficie.blit(textobj,  
textrect)
```

Primero, la llamada al método render() en la línea 37 crea una superficie objeto que representa el texto en una fuente específica.

A continuación, debe conocer el tamaño y la ubicación del objeto de superficie . Puede obtener un objeto Rect con esta información utilizando el método de superficie get_rect() .

El objeto Rect devuelto por get_rect() en la línea 38 tiene una copia de la información de ancho y alto del objeto Surface .

La línea 39 cambia la ubicación del objeto Rect estableciendo un nuevo

valor de tupla para su atributo superior izquierdo .

Finalmente, la línea 40 dibuja el objeto Surface del texto representado en el objeto Surface que se pasó a la función dibujarTexto() .

Mostrar texto en pygame requiere algunos pasos más que simplemente llamar a la función print() . Pero si coloca este código en una sola función llamada dibujarTexto(), entonces solo necesita llamar a esta función para mostrar el texto en la pantalla.

INICIALIZANDO PYGAME Y CONFIGURACIÓN DE LA VENTANA

Ahora que las variables y funciones constantes están terminadas, comenzaremos a llamar a las funciones de pygame que configuran la ventana. y reloj:

```
42. # Configure pygame, la ventana y el cursor del  
mouse. 43. pygame.init() 44. mainClock =  
pygame.time.Clock()
```

La línea 43 configura pygame llamando a la función pygame.init() . La línea 44 crea un objeto pygame.time.Clock() y lo almacena en la variable mainClock . Este objeto nos ayudará a evitar que el programa se ejecute demasiado rápido.

La línea 45 crea un nuevo objeto de superficie que se usa para la visualización de la ventana:

```
45. superficieVentana = pygame.display.set_mode((ANCHOVENTANA,  
ALTOVENTANA))
```

Tenga en cuenta que solo se pasa un argumento a pygame.display.set_mode(): una tupla. Los argumentos para pygame.display.set_mode() no son dos enteros sino una tupla de dos

números enteros Puede especificar el ancho y el alto de este objeto de superficie (y la ventana) pasando una tupla con las variables constantes ANCHO DE VENTANA y ALTO DE VENTANA.

La función `pygame.display.set_mode()` tiene un segundo parámetro opcional. Puede pasar la constante `pygame.FULLSCREEN` a hacer que la ventana ocupe toda la pantalla. Mira este modificación a la línea 45:

```
45. superficieVentana = pygame.display.set_mode((ANCHO DE LA VENTANA,  
ALTO DE LA VENTANA), pygame.PANTALLA COMPLETA)
```

Los parámetros ANCHO DE VENTANA y ALTO DE VENTANA todavía se pasan para el ancho y el alto de la ventana, pero la imagen se estirará más para ajustarse a la pantalla. Intente ejecutar el programa con y sin modo de pantalla completa.

La línea 46 establece el título de la ventana en la cadena 'Dodger':

```
46. pygame.display.set_caption('Dodger')
```

Este título aparecerá en la barra de título en la parte superior de la ventana.

En Dodger, el cursor del mouse no debería estar visible. Desea que el mouse pueda mover el personaje del jugador por la pantalla, pero el cursor del mouse se interpondría en el camino de la imagen del personaje. Podemos hacer que el mouse sea invisible con solo una línea de código:

```
47. pygame.mouse.set_visible(False)
```

Llamar a `pygame.mouse.set_visible(False)` le dice a pygame que haga el cursor invisible.

CONFIGURACIÓN DE FUENTE, SONIDO Y OBJETOS DE IMAGEN

Dado que estamos mostrando texto en la pantalla en este programa, debemos darle al módulo pygame un objeto Font para usar con el texto.

La línea 50 crea un objeto Font llamando a `pygame.font.SysFont()`:

```
49. # Configure las
fuentes. 50. fuente = pygame.font.SysFont(Ninguno, 48)
```

Pasar Ninguno usa la fuente predeterminada. Pasar 48 le da a la fuente un tamaño de 48 puntos.

A continuación, crearemos los objetos de sonido y configuraremos la música de fondo:

```
52. # Configurar
sonidos. 53. gameOverSound = pygame.mixer.Sound('gameover.wav')
54. pygame.mixer.music.load('background.mid')
```

La función constructora `pygame.mixer.Sound()` crea un nuevo objeto Sound y almacena una referencia a este objeto en la variable `gameOverSound`. En tus propios juegos, puedes crear tantos objetos de sonido como quieras, cada uno con un archivo de sonido diferente.

La función `pygame.mixer.music.load()` carga un archivo de sonido para jugar para la música de fondo. Esta función no devuelve ningún objeto y solo se puede cargar un archivo de sonido de fondo a la vez. La música de fondo se reproducirá constantemente durante el juego, pero los objetos de sonido se reproducirán solo cuando el jugador pierda el juego al encontrarse con un malo.

Puedes usar cualquier archivo WAV o MIDI para este juego. Algunos archivos de sonido están disponibles en el sitio web de este libro en <https://www.nostarch.com/inventwithpython/>. También puedes usar

sus propios archivos de sonido para este juego, siempre y cuando nombre los archivos gameover.wav y background.mid o cambie las cadenas utilizadas en las líneas 53 y 54 para que coincidan con el nombre de archivo que desea.

A continuación, cargará los archivos de imagen que se utilizarán para el reproductor. personaje y los malos:

```
56. # Configurar
imágenes. 57. playerImage = pygame.image.load('player.png')
58. playerRect = playerImage.get_rect() 59. baddieImage =
pygame.image.load('baddie.png')
```

La imagen del personaje se almacena en player.png, y la imagen de los malos se almacena en baddie.png. Todos los malos se ven iguales, por lo que solo necesita un archivo de imagen para ellos. Puede descargar estas imágenes del sitio web de este libro en <https://www.nostarch.com/inventwithpython/>.

MOSTRAR LA PANTALLA DE INICIO

Cuando el juego comienza por primera vez, Python debería mostrar el título de Dodger en la pantalla. También desea decirle al jugador que puede comenzar el juego presionando cualquier tecla. Esta pantalla aparece para que el jugador tenga tiempo de prepararse para empezar a jugar después de ejecutar el programa.

En las líneas 63 y 64, escribimos código para llamar a drawText() función:

```
61. # Muestra la pantalla "Inicio".
62. superficieventana.fill(COLOR DE FONDO) 63.
dibujarTexto('Dodger', fuente, superficieventana, (ANCHOVENTANA / 3),
(ALTOVENTANA / 3)) 64. dibujarTexto('Presione una tecla para comenzar.',
fuente, superficieventana,
(ANCHO DE VENTANA / 3) - 30, (ALTO DE VENTANA / 3) + 50)
65. Pygame.display.update()
```

66. esperar al jugador para presionar la tecla ()

Pasamos a esta función cinco argumentos:

1. La cadena del texto que desea que aparezca
2. La fuente en la que desea que aparezca la cadena
3. El objeto de superficie en el que se representará el texto
4. La coordenada x en el objeto de superficie en la que dibujar el text
5. La coordenada y en el objeto Surface en el que dibujar el texto

Esto puede parecer muchos argumentos para pasar por una llamada de función, pero tenga en cuenta que esta llamada de función reemplaza cinco líneas de código cada vez que la llama. Esto acorta el programa y facilita la búsqueda de errores, ya que hay menos código para verificar.

La función `waitForPlayerToPressKey()` detiene el juego haciendo un bucle hasta que se genera un evento `KEYDOWN`. Entonces la ejecución sale del bucle y el programa continúa correr.

COMENZANDO EL JUEGO

Con todas las funciones ahora definidas, podemos comenzar a escribir el código principal del juego. Las líneas 68 y siguientes llamarán a las funciones que definimos anteriormente. El valor en la variable `topScore` comienza en 0

cuando el programa se ejecuta por primera vez. Cada vez que el jugador pierde y tiene un puntaje mayor que el puntaje máximo actual, el puntaje máximo se reemplaza con este puntaje mayor.

68. puntuación máxima = 0

69. mientras sea cierto:

El ciclo infinito iniciado en la línea 69 técnicamente no es el

bucle de juego El bucle del juego maneja eventos y dibuja la ventana mientras se ejecuta el juego. En cambio, este ciclo while se repite cada vez que el jugador comienza un nuevo juego. Cuando el jugador pierde y el juego se reinicia, la ejecución del programa regresa a la línea 69.

Al principio, también querrás configurar a los malos para que estén vacíos . lista:

```
70. # Configura el inicio del juego. 71. malos  
= []  
72. puntuación = 0
```

La variable baddies es una lista de objetos de diccionario con las siguientes claves:

'rect' El objeto Rect que describe dónde y qué tamaño malo es.

'velocidad' Qué tan rápido cae el malo por la pantalla. Este número entero representa píxeles por iteración a través del bucle del juego.

'superficie' El objeto de superficie que tiene la imagen del malo escalada dibujado en él. Esta es la superficie que se dibuja en la superficie. objeto devuelto por pygame.display.set_mode().

La línea 72 restablece la puntuación del jugador a 0.

La ubicación inicial del reproductor está en el centro de la pantalla y 50 píxeles por encima de la parte inferior, que se establece en la línea 73:

```
73. playerRect.topleft = (ANCHO DE VENTANA / 2, ALTO DE VENTANA - 50)
```

El primer elemento en la tupla de la línea 73 es la coordenada x de la

borde izquierdo, y el segundo elemento es la coordenada y del borde superior.

A continuación, configuraremos variables para los movimientos del jugador y el trucos:

-
- 74. moveLeft = moveRight = moveUp = moveDown = False
 - 75. reverseCheat = slowCheat = False
 - 76. baddieAddCounter = 0
-

Las variables de movimiento moveLeft, moveRight, moveUp y moveDown se establece en False. Las variables reverseCheat y slowCheat también se establecen en False. Se establecerán en Verdadero solo cuando el jugador habilite estos trucos manteniendo presionadas las teclas Z y X, respectivamente.

La variable baddieAddCounter es un contador que le dice al programa cuándo agregar un nuevo villano en la parte superior de la pantalla. El valor de baddieAddCounter se incrementa en 1 cada vez que se itera el bucle del juego. (Esto es similar al código en “Aregar nuevos cuadrados de alimentos” en la página 295).

Cuando baddieAddCounter es igual a ADDNEWBADDIERATE, baddieAddCounter se restablece a 0 y se agrega un nuevo villano en la parte superior de la pantalla. (Esta verificación se realiza más adelante en la línea 130).

La música de fondo comienza a reproducirse en la línea 77 con una llamada a la función pygame.mixer.music.play() :

-
- 77. pygame.mixer.music.play(-1, 0.0)
-

Debido a que el primer argumento es -1, pygame repite la música sin cesar. El segundo argumento es un flotador que dice cuántos segundos de música quieras que empiece a sonar. Pasar 0.0 significa que la música comienza a reproducirse desde el principio.

EL BUCLE DEL JUEGO

El código del bucle del juego actualiza constantemente el estado del mundo del juego cambiando la posición del jugador y los malos, manejando eventos generados por pygame y dibujando el mundo del juego en la pantalla. Todo esto sucede varias docenas de veces por segundo, lo que hace que el juego se ejecute en tiempo real.

La línea 79 es el comienzo del bucle principal del juego:

```
79. while True: # El bucle del juego se ejecuta mientras se reproduce la parte del juego.  
80.     puntuación += 1 # Incrementar puntuación.
```

La línea 80 aumenta la puntuación del jugador en cada iteración del ciclo del juego. Cuanto más tiempo pueda pasar el jugador sin perder, mayor será su puntuación. El bucle saldrá solo cuando el jugador pierda el juego o salga del programa.

Manejo de eventos de teclado Hay cuatro tipos de eventos que manejará el programa: SALIR, TECLADO ABAJO , TECLADO ARRIBA y MOVIMIENTO DEL MOUSE.

La línea 82 es el comienzo del código de manejo de eventos:

```
82.     para evento en pygame.event.get(): if  
83.         event.type == SALIR: terminar()  
84.
```

Llama a pygame.event.get(), que devuelve una lista de objetos Event . Cada objeto Event representa un evento que ha ocurrido desde la última llamada a pygame.event.get(). El código verifica el atributo de tipo del objeto Evento para ver qué tipo de evento es y luego lo maneja en consecuencia.

Si el atributo de tipo del objeto Evento es igual a SALIR, entonces

el usuario ha cerrado el programa. La variable constante QUIT se importó del módulo pygame.locals .

Si el tipo de evento es KEYDOWN, el jugador ha presionado una tecla:

```

86.     if event.type == KEYDOWN: if
87.         event.key == K_z:
88.             truco inverso = Verdadero
89.         if evento.clave == K_x:
90.             trampalenta = Verdadero

```

La línea 87 verifica si el evento describe la tecla Z que se presiona con event.key == K_z. Si esta condición es True, Python establece la variable reverseCheat en True para activar el truco inverso. De manera similar, la línea 89 verifica si se presionó la tecla X para activar el truco lento.

Las líneas 91 a 102 verifican si el jugador generó el evento presionando una de las teclas de flecha o WASD. Este código es similar al código relacionado con el teclado de los capítulos anteriores.

Si el tipo de evento es KEYUP, el jugador ha soltado una tecla:

```

104.    if evento.tipo == KEYUP: if
105.        evento.clave == K_z:
106.            truco inverso = Falso
107.            puntuación = 0
108.        if evento.clave == K_x:
109.            trampalenta = Falso
110.            puntuación = 0

```

La línea 105 comprueba si el jugador ha soltado la tecla Z, lo que desactivará el truco inverso. En ese caso, la línea 106 establece reverseCheat en False y la línea 107 restablece el puntaje a 0. El reinicio de puntaje es para disuadir al jugador de usar los trucos.

Las líneas 108 a 110 hacen lo mismo para la tecla X y el truco lento. Cuando se suelta la tecla X, slowCheat se establece en False y la puntuación del jugador se restablece a 0.

En cualquier momento durante el juego, el jugador puede presionar ESC para abandonar.

```
111.     si evento.clave == K_ESCAPE:
112.         terminar()
```

La línea 111 determina si la tecla que se soltó fue ESC comprobando event.key == K_ESCAPE. Si es así, la línea 112 llama a la función terminar() para salir del programa.

Las líneas 114 a 121 verifican si el jugador ha dejado de presionar una de las teclas de flecha o WASD. En ese caso, el código establece la variable de movimiento correspondiente en False.

Esto es similar al código de movimiento en los [programas del Capítulo 19](#) y del [Capítulo 20](#).

Manejo del movimiento del mouse

Ahora que ha manejado los eventos del teclado, gestionemos cualquier evento del mouse que pueda haberse generado. El juego Dodger no hace nada si el jugador ha hecho clic en un botón del mouse, pero responde cuando el jugador mueve el mouse.

Esto le da al jugador dos formas de controlar al personaje en el juego: el teclado o el mouse.

El evento MOUSEMOTION se genera cada vez que el mouse se mueve:

```
123.     if evento.tipo == MOVIMIENTO DEL RATÓN:
124.         # Si el mouse se mueve, mueve al jugador al cursor.
125.         rect.jugador.centerx = evento.pos[0] rect.jugador.centery =
126.             evento.pos[1]
```

Los objetos de evento con un tipo establecido en MOUSEMOTION también tienen un atributo denominado pos para la posición del evento del mouse. El atributo pos almacena una tupla de las coordenadas x e y de dónde se movió el cursor del mouse en la ventana. Si el tipo de evento es MOUSEMOTION, el personaje del jugador se mueve a la posición de el cursor del ratón.

Las líneas 125 y 126 establecen las coordenadas x e y del centro del personaje del jugador en las coordenadas x e y del ratón. cursor.

AÑADIR NUEVOS MALOS

En cada iteración del ciclo del juego, el código incrementa la variable baddieAddCounter en uno:

```

127.     # Agregue nuevos malos en la parte superior de la pantalla, si es
128.     necesario. si no reverseCheat y no slowCheat:
129.         baddieAddCounter += 1

```

Esto sucede solo si los trucos no están habilitados. Recuerde que reverseCheat y slowCheat se establecen en True siempre que se mantengan presionadas las teclas Z y X, respectivamente. Mientras se mantienen presionadas las teclas Z y X, baddieAddCounter no se incrementa.

Por lo tanto, no aparecerán nuevos malos en la parte superior de la pantalla.

Cuando baddieAddCounter alcanza el valor en ADDNEWBADDIERATE, es hora de agregar un nuevo villano en la parte superior de la pantalla. Primero, baddieAddCounter se restablece a 0:

```

130.     if baddieAddCounter == AGREGARNUEVOBADDIERATE:
131.         baddieAddCounter = 0
132.         baddieSize = random.randint(BADDIEMINSIZE,
BADDIEMAXSIZE) 133.
nuevoMalo = {'rect': pygame.Rect(random.randint(0,

```

```
ANCHO DE VENTANA - tamaño del villano), 0 - tamaño del  
villano, tamaño del villano, tamaño del villano), 'velocidad':  
134.     random.randint(VELOCIDAD DEL MALO, VELOCIDAD MÁXIMA  
DEL MALO), 'superficie':pygame.transform.scale(ImagenDelMalo,  
135.     (TamañoDelMalo, TamañoDelMalo)), }  
  
136.
```

La línea 132 genera un tamaño para el malo en píxeles. El tamaño será un número entero aleatorio entre BADDIEMINSIZE y BADDIEMAXSIZE, que son constantes establecidas en 10 y 40 en las líneas 9 y 10, respectivamente.

La línea 133 es donde se crea una nueva estructura de datos de malos. Recuerde, la estructura de datos para los malos es simplemente un diccionario con las claves 'rect', 'speed' y 'surface'. La tecla 'rect' contiene una referencia a un objeto Rect que almacena la ubicación y el tamaño del malo. La llamada a la función constructora pygame.Rect() tiene cuatro parámetros: la coordenada x del borde superior del área, la coordenada y del borde izquierdo del área, el ancho en píxeles y la altura en píxeles.

El malo debe aparecer en un punto aleatorio en la parte superior de la ventana, así que pasa random.randint(0, WINDOWWIDTH - baddieSize) para la coordenada x del borde izquierdo del malo. La razón por la que pasa ANCHO DE VENTANA - Tamaño del malo en lugar de ANCHO DE VENTANA es que si el borde izquierdo del malo está demasiado hacia la derecha, parte del malo estará fuera del borde de la ventana y no será visible en la pantalla.

El borde inferior del malo debe quedar justo encima del borde superior de la ventana. La coordenada y del borde superior de la ventana es 0. Para colocar el borde inferior del malo allí, establezca el borde superior en 0 - baddieSize.

El ancho y la altura del malo deben ser iguales (la imagen es un cuadrado), así que pasa baddieSize para el tercero y el cuarto argumentos

La velocidad a la que el malo se mueve por la pantalla se establece en la tecla 'velocidad' . Establézcalo en un número entero aleatorio entre BADDIEMINSPEED y BADDIEMAXSPEED.

La línea 138 luego agregará los datos de villano recién creados estructura a la lista de estructuras de datos malos:

```
138.     malos.append(nuevoMalo)
```

El programa usa esta lista para verificar si el jugador ha chocado con alguno de los malos y para determinar dónde dibujar a los malos en la ventana.

MOVER EL JUGADOR PERSONAJE Y LOS MALOS

Las cuatro variables de movimiento moveLeft, moveRight, moveUp y moveDown se establecen en True y False cuando pygame genera los eventos KEYDOWN y KEYUP , respectivamente.

Si el personaje del jugador se mueve hacia la izquierda y el borde izquierdo del personaje del jugador es mayor que 0 (que es el borde izquierdo de la ventana), entonces playerRect debería moverse hacia la izquierda:

```
140. # Mueve al jugador. si
141. moveLeft y playerRect.left > 0:
142.     RectJugador.move_ip(-1 * TASA DE JUGADOR, 0)
```

El método move_ip() moverá la ubicación del objeto Rect horizontal o verticalmente una cantidad de píxeles. El primer argumento de move_ip() es cuántos píxeles mover el Rect

objeto a la derecha (para moverlo a la izquierda, pase un entero negativo). El segundo argumento es cuántos píxeles mover el objeto Rect hacia abajo (para moverlo hacia arriba, pase un número entero negativo). Por ejemplo, playerRect.move_ip(10, 20) movería el objeto Rect 10 píxeles a la derecha y 20 píxeles hacia abajo y playerRect.move_ip(-5, -15) movería el objeto Rect 5 píxeles a la izquierda y 15 píxeles hacia arriba. .

La ip al final de move_ip() significa "en su lugar". Esto se debe a que el método cambia el propio objeto Rect , en lugar de devolver un nuevo objeto Rect con los cambios. También hay un método move() , que no cambia el objeto Rect sino que crea y devuelve un nuevo objeto Rect en la nueva ubicación.

Siempre moverá el objeto playerRect por la cantidad de píxeles en PLAYEROVERRATE. Para obtener la forma negativa de un entero, multiplícalo por -1. En la línea 142, dado que 5 está almacenado en PLAYEROVERRATE, la expresión -1 * PLAYEROVERRATE se evalúa como -5. Por lo tanto, llamar a playerRect.move_ip(-1 * PLAYEROVERRATE, 0) cambiará la ubicación de playerRect 5 píxeles a la izquierda de su ubicación actual.

Las líneas 143 a 148 hacen lo mismo con las otras tres direcciones: derecha, arriba y abajo.

-
- ```

143. if moveRight y playerRect.right < ANCHO DE VENTANA:
144. playerRect.move_ip(PLAYEROVERRATE, 0) if
145. moveUp y playerRect.top > 0:
146. playerRect.move_ip(0, -1 * PLAYEROVERRATE)
147. if moveDown and playerRect.bottom < WINDOWHEIGHT:
148. playerRect.move_ip(0, PLAYEROVERRATE)

```
- 

Cada una de las tres sentencias if en las líneas 143 a 148 verifica que su variable de movimiento se establece en True y que el borde de la

El objeto recto del jugador está dentro de la ventana. Luego llama a move\_ip() para mover el objeto Rect .

Ahora el código recorre cada estructura de datos del malo en la lista de malos para bajarlos un poco:

---

```

150. # Mueve a los malos hacia abajo.
151. para b en malos:
152. si no reverseCheat y no slowCheat:
153. b['rect'].move_ip(0, b['speed'])

```

---

Si ninguno de los trucos se ha activado, la ubicación del malo se mueve hacia abajo un número de píxeles igual a su velocidad (almacenada en la tecla 'velocidad' ).

## IMPLEMENTANDO EL TRUCO CÓDIGOS

Si se activa el truco inverso, entonces el malo debería moverse 5 píxeles hacia arriba:

---

```

154. elif reverseCheat:
155. b['rect'].move_ip(0, -5)

```

---

Pasar -5 como segundo argumento a move\_ip() moverá el objeto Rect hacia arriba 5 píxeles.

Si se ha activado el truco lento, entonces el malo aún debe moverse hacia abajo, pero a la velocidad lenta de 1 píxel por iteración a través del bucle del juego:

---

```

156. elif truco lento:
157. b['rect'].move_ip(0, 1)

```

---

La velocidad normal del malo (nuevamente, esto se almacena en el

clave de 'velocidad' de la estructura de datos del malo) se ignora cuando el truco lento está activado.

## ELIMINAR LOS MALOS

Los malos que se encuentren por debajo del borde inferior de la ventana deben eliminarse de la lista de malos . Recuerde que no debe agregar o eliminar elementos de la lista mientras recorre la lista. En lugar de iterar a través de la lista de malos con el bucle for , iterar a través de una copia de la lista de malos . Para hacer esta copia, use el operador de corte en blanco [:]:

---

```
159. # Eliminar los malos que han caído más allá del
160. fondo. para b en malos[:]:
```

---

El ciclo for en la línea 160 usa la variable b para el elemento actual en la iteración a través de baddies[:]. Si el malo está debajo del borde inferior de la ventana, debemos eliminarlo, lo que hacemos en la línea 162:

---

```
161. if b['rect'].top > WINDOWHEIGHT:
162. malos.remove(b)
```

---

El diccionario b es la estructura de datos baddie actual de la lista de los malos[:]. Cada estructura de datos de villanos en la lista es un diccionario con una tecla 'rect' , que almacena un objeto Rect . Entonces b['rect'] es el objeto Rect para el malo. Finalmente, el atributo superior es la coordenada y del borde superior del área rectangular. Recuerda que las coordenadas y aumentan al disminuir. Entonces b['rect'].top > WINDOWHEIGHT verificará si el borde superior del malo está debajo de la parte inferior de la ventana. Si esta condición es Verdadero, entonces la línea 162 elimina la estructura de datos del malo de

la lista de los malos .

## DIBUJANDO LA VENTANA

Después de que se hayan actualizado todas las estructuras de datos, el mundo del juego debe dibujarse utilizando las funciones de imagen de pygame .

Debido a que el ciclo del juego se ejecuta varias veces por segundo, cuando los malos y el jugador se colocan en nuevas posiciones, parece que se mueven sin problemas.

Antes de que se dibuje cualquier otra cosa, la línea 165 llena toda la pantalla para borrar todo lo que se haya dibujado anteriormente:

---

```
164. # Dibuja el mundo del juego en la
165. ventana. superficieventana.fill(COLOR DE FONDO)
```

---

Recuerde que el objeto Surface en windowSurface es especial porque es el que devuelve pygame.display.set\_mode().

Por lo tanto, todo lo que se dibuje en ese objeto Surface aparecerá en la pantalla después de llamar a pygame.display.update() .

Dibujar las líneas de puntuación del jugador 168 y 169 muestra el texto de la puntuación actual y la puntuación máxima en la esquina superior izquierda de la ventana.

---

```
167. # Dibuja la puntuación y la puntuación
168. máxima. dibujarTexto('Puntuación: %s' % (puntuación), fuente, superficieventana,
169. 10, 0) dibujarTexto('Puntuación máxima: %s' % (puntuación), fuente,
 superficieventana, 10, 40)
```

---

La expresión 'Puntuación: %s' % (puntuación) utiliza la interpolación de cadenas para insertar el valor de la variable de puntuación en la cadena. Esta cadena, el objeto Font almacenado en la variable font , el objeto Surface sobre el que dibujar el texto y las coordenadas x e y de donde

el texto que debe colocarse se pasa al método `drawText()` , que manejará la llamada a los métodos `render()` y `blit()` .

Para obtener la puntuación más alta, haz lo mismo. Pase 40 para la coordenada y en lugar de 0 para que aparezca el texto de la puntuación más alta debajo del texto de la partitura actual.

Dibujar el personaje del jugador y los malos La información sobre el jugador se mantiene en dos variables diferentes. `playerImage` es un objeto `Surface` que contiene todos los píxeles de colores que forman la imagen del personaje del jugador. `playerRect` es un objeto `Rect` que almacena el tamaño y la ubicación del personaje del jugador.

El método `blit()` dibuja la imagen del personaje del jugador (en `playerImage`) en `windowSurface` en la ubicación en `playerRect`:

---

```
171. # Dibuja el rectángulo del jugador.
172. superficieVentana.blit(ImagenJugador, RectoJugador)
```

---

El bucle `for` de la línea 175 atrae a todos los malos en el objeto `superficieventana`:

---

```
174. # Dibuja a cada malo.
175. para b en malos:
176. superficieVentana.blit(b['superficie'], b['rect'])
```

---

Cada elemento de la lista de malos es un diccionario. Las teclas '`superficie`' y '`rect`' de los diccionarios contienen el objeto `Surface` con la imagen del malo y el objeto `Rect` con la información de posición y tamaño, respectivamente.

Ahora que todo ha sido dibujado a `windowSurface` , necesita actualizar la pantalla para que el jugador pueda ver lo que hay allí:

---

```
178. pygame.display.update()
```

---

Dibuja este objeto Surface en la pantalla llamando a update().

## COMPROBACIÓN DE COLISIONES

La línea 181 comprueba si el jugador ha chocado con algún malo llamando a playerHasHitBaddie(). Esta función devolverá True si el personaje del jugador ha chocado con cualquiera de los malos en la lista de malos . De lo contrario, la función devuelve False.

---

```
180. # Comprueba si alguno de los malos ha golpeado al
181. jugador. if playerHasHitBaddie(playerRect, baddies):
182. if score > topScore:
183. topScore = puntuación # Establecer nueva puntuación máxima.
184. descanso
```

---

Si el personaje del jugador ha golpeado a un malo y si el puntaje actual es más alto que el puntaje máximo, las líneas 182 y 183 actualizan el puntaje máximo. La ejecución del programa sale del bucle del juego en la línea 184 y pasa a la línea 189, finalizando el juego.

Para evitar que la computadora ejecute el bucle del juego lo más rápido posible (lo que sería demasiado rápido para que el jugador se mantuviera al día), llama a mainClock.tick() para pausar el juego muy brevemente:

---

```
186. relojprincipal.tick(FPS)
```

---

Esta pausa será lo suficientemente larga para garantizar que se produzcan unas 40 iteraciones (el valor almacenado dentro de la variable FPS ) a través del bucle del juego cada segundo.

## LA PANTALLA DE JUEGO TERMINADO

Cuando el jugador pierde, el juego deja de reproducir la música de fondo y reproduce el efecto de sonido "game over":

---

```
188. # Detener el juego y mostrar la pantalla "Game Over".
189. pygame.mixer.music.stop() 190. gameOverSound.play()
```

---

La línea 189 llama a la función stop() en el módulo pygame.mixer.music para detener la música de fondo. La línea 190 llama al método play() en el objeto Sound almacenado en gameOverSound.

Luego, las líneas 192 y 193 llaman a la función dibujarTexto() para dibujar el texto de "juego terminado" al objeto windowSurface :

---

```
192. drawText('GAME OVER', fuente, superficieVentana, (ANCHOVENTANA / 3),
 (ALTOVENTANA / 3)) 193. dibujarTexto('Presiona una tecla para volver a jugar.',
fuente, superficieVentana, (ANCHOVENTANA / 3) - 80 , (ALTURA DE LA VENTANA /
3) + 50) 194. pygame.display.update() 195. waitForPlayerToPressKey()
```

---

La línea 194 llama a update() para dibujar este objeto Surface en la pantalla. Después de mostrar este texto, el juego se detiene hasta que el jugador presiona una tecla llamando a waitForPlayerToPressKey() función.

Después de que el jugador presiona una tecla, la ejecución del programa regresa de la llamada waitForPlayerToPressKey() en la línea 195.

Dependiendo de cuánto tiempo le tome al jugador presionar una tecla, el efecto de sonido de "juego terminado" puede o no seguir sonando. Para detener este efecto de sonido antes de que comience un nuevo juego, la línea 197 llama juegoOverSound.stop():

---

```
197. juegoOverSound.stop()
```

---

¡Eso es todo para nuestro juego gráfico!

## MODIFICANDO EL JUEGO DE LOS DODGER

Puede encontrar que el juego es demasiado fácil o demasiado difícil.

Afortunadamente, el juego es fácil de modificar porque nos tomamos el tiempo de usar variables constantes en lugar de ingresar los valores directamente. Ahora todo lo que tenemos que hacer para cambiar el juego es modificar los valores establecidos en las variables constantes.

Por ejemplo, si quieras que el juego funcione más lento en general, cambia la variable FPS en la línea 8 a un valor más pequeño, como 20. Esto hará que tanto los malos como el personaje del jugador se muevan más lento, ya que se ejecutará el bucle del juego. sólo 20 veces por segundo en lugar de 40.

Si solo desea ralentizar a los malos y no al jugador, cambie BADDIEMAXSPEED a un valor más pequeño, como 4. Esto hará que todos los malos se muevan entre 1 (el valor en BADDIEMINSPEED) y 4 píxeles por iteración a través del bucle del juego. , en lugar de entre 1 y 8.

Si desea que el juego tenga menos malos pero más grandes en lugar de muchos malos más pequeños, aumente ADDNEWBADDIERATE a 12, BADDIEMINSIZE a 40 y BADDIEMAXSIZE a 80. Ahora los malos se agregan cada 12 iteraciones a través del ciclo del juego en lugar de cada 6 iteraciones, entonces habrá la mitad de malos que antes. Pero para mantener el juego interesante, los malos son mucho más grandes.

Manteniendo el juego básico igual, puedes modificar cualquiera de las variables constantes para afectar dramáticamente cómo se juega el juego. Siga probando nuevos valores para las variables constantes hasta que encuentre el conjunto de valores que más le guste.

## RESUMEN

A diferencia de nuestros juegos basados en texto, Dodger realmente parece un juego de computadora moderno. Tiene gráficos y música y utiliza el ratón. Si bien pygame proporciona funciones y tipos de datos como bloques de construcción, es usted el programador quien los une para crear juegos divertidos e interactivos.

Y puedes hacer todo esto porque sabes cómo instruir a la computadora para que lo haga, paso a paso, línea a línea. Al hablar el idioma de la computadora, puede hacer que haga el cálculo y dibujo de números por usted. Esta es una habilidad útil y espero que continúes aprendiendo más sobre la programación de Python. (¡Y todavía hay mucho más por aprender!)

Ahora ponte en marcha e inventa tus propios juegos. ¡Buena suerte!

# ÍNDICE

## SÍMBOLOS

- + (suma), 2, 13
  - asignación aumentada, 155
  - propiedad conmutativa de, 169
- \ (barra invertida), para caracteres de escape, 41–42
- : (dos puntos), en declaraciones for , 28–29
- { (corchetes), para diccionarios, 113
- / (división), 2, 4, 246–247
  - asignación aumentada, 156
- " (comillas dobles), 42–43
- = (signo igual), como operador de asignación, 5, 34
- == (igual a) operador, 32, 34
- > (mayor que) operador, 30, 32
  - operador >= (mayor o igual que), 32
- # (marca de almohadilla), para comentarios, 18, 23
- < (menor que) operador, 30, 32
- <= (menor o igual que) operador, 32
- \* (multiplicación), 2
  - asignación aumentada, 156
- != (no igual a) operador, 32, 36 ()
- (paréntesis), y orden de operación, 4
- >>> solicitud, 5, 14
- ' (comillas simples), 42–43

[] (corchetes), para índices, 92  
- (resta), 2  
asignación aumentada, 155

## A

función abs() , 170  
valor absoluto del número, 170  
suma (+), 2, 13  
asignación aumentada, 155  
propiedad conmutativa de, 169

AI. Ver inteligencia artificial (IA)

programa AISim1.py  
agregando un reproductor de computadora,  
242–243 comparando algoritmos, 247–254  
computadora jugando contra sí misma, 240–247  
corrida de muestra, 240–241  
código fuente, 241–242

programa AISim2.py  
seguimiento de varios juegos, 245  
ejecución de muestra, 243–244  
código fuente, 244

programa AISim3.py  
cómo funciona la IA, 249–252  
código fuente, 248–249  
algoritmos, 128  
orden alfabético, método sort() para, 157–158  
y operador, 52–53

evaluación de cortocircuito, 140

## Programa de animación

configuración de la estructura de datos del cuadro, 278–279

variables constantes, 277–278

mover y rebotar cajas, 276–277, 280–282 bucle de juego

en ejecución, 279–283 ejecución de muestra, 274

código fuente, 274–276

suavizado, 263

método append(), 95

argumentos, 18

teclas de flecha en el teclado, 293, 294

inteligencia artificial (IA), 121, 209

para Reversegam, 232–233

comparando algoritmos, 247–254

computadora jugando contra sí misma, 240–247

para tres en raya

crear, 142–145

elaboración de estrategias, 128–129

Arte ASCII, 79, 108

operador de asignación (=), 5, 34

atributos, 264

operadores de asignación aumentada, 155–156

## B

color de fondo, para texto, 263

música de fondo, 322, 325

repitiendo para siempre, 308

barra invertida (\), para caracteres de escape, 41–42

Comprobación del juego de

- deducción de bagels para ganar o perder, 161–162
- cadena de verificación solo para números, 158–159
- pistas, 149
- calcular, 156–157
- conseguir, 161
- diagrama de flujo para, 152–153
- conseguir la conjetura del jugador, 161
- método join(), 158
- jugando de nuevo, 162
- ejecución de muestra, 150–
- 151 número secreto, creación, 160–161
- mezclar un conjunto único de dígitos, 154–155
- código fuente, 151–152
- partida inicial, 159

línea en blanco, impresión, 41

método blit(), 269–270, 331

bloques de código, agrupando con, 27

formato de archivo BMP, 302

función bool(), 227

tipo de datos booleano, 31

- operadores de comparación y, 33–34 condiciones
- para verificar, 32 en evaluación Tic-Tac-Toe, 136–137

Operadores booleanos, 52–55

romper declaración, 35, 37, 109, 161  
puntos de interrupción en el depurador, 73–  
75 técnica de fuerza bruta, para cifrados, 206–208 errores, 63  
hallazgo, 70–72  
tipos, 64–65  
atributo de botón , para evento MOUSEBUTTONDOWN , 292

## C

Técnica de fuerza bruta del  
programa Caesar Cipher, 206–208 cifrado  
o descifrado, 202–205  
obtener la clave del jugador, 203  
obtener un mensaje del jugador, 203 cómo  
funciona, 199–200  
ejecución de muestra, 200–201  
establecer la longitud máxima de la clave, 202  
código fuente, 201–202  
programa de inicio, 206  
vocación  
funciones, 49–50, 60–61  
métodos, 94–96  
caso camello, 20  
subtítulos, para windows, 261  
Sistema de coordenadas cartesianas, 163, 209  
cuadrículas y, 164–165  
trucos matemáticos, 168–169

números negativos en, 166–167

distinción entre mayúsculas y minúsculas, 61

de nombres de variables, 20

atributo centerx , del objeto Rect , 264

atributo centrado, de objeto Rect , 264

modos de trucos, 312

tablero de ajedrez, coordenadas, 164–165

función choice() , módulo aleatorio , 115–117

cifrado, 198

función circle() , módulo pygame.draw , 268

ventana de limpieza, para iteración de animación, 280

Reloj() función, 289–290

Objeto reloj , 313

código. Ver código fuente

lanzamientos de moneda, simulación de programa,

73–75 función collidrect() , 297, 320

Programa de detección de colisiones

agregando cuadrados de alimentos,

295–296 verificando colisiones, 297

programa reloj a ritmo, 289–290

dibujar cuadros de comida, 298

dibujar jugador en pantalla, 297 manejo

de eventos, 292–295 mover jugador

alrededor de pantalla, 296–297

ejecución de muestra, 286

código fuente, 287–289

jugador teletransportador, 295  
variables para seguimiento de movimiento, 291  
configuración de ventanas y estructuras de datos, 290–291  
dos puntos (:), en declaraciones for , 28–29  
color  
    objeto de superficie de relleno con, 266 de  
    píxeles, 269–270  
    Valores RGB, en pygame, 261–262  
    de texto, 263  
comentando código, 242  
comentarios, 17–18  
propiedad conmutativa de la suma, 169  
operadores de comparación, 32, 33–34  
computadora  
    vs. simulación de IA por computadora, 240  
    sistema de coordenadas de pantalla, 167–168  
concatenación  
    de listas, 94  
    de cuerdas, 13, 26, 159  
condiciones, 33–34  
variables constantes, 91  
    para teclas de teclado, 294  
funciones constructoras, 264  
continuar declaración, 161  
especificadores de conversión, 159–160  
sistema coordinado  
    Cartesiano. Ver sistema de coordenadas cartesianas

de pantalla de computadora, 167–168  
coordenadas, 163  
algoritmo de esquina mejor, 248, 249  
contra el mejor algoritmo del lado de la esquina, 253–254  
contra algoritmo de movimiento aleatorio, 252–253  
contra algoritmo de peor movimiento, 252  
algoritmo de esquina-lado-mejor, 248, 251–252  
contra el mejor algoritmo de la esquina, 253–254  
bloqueo de programas, 64  
criptoanálisis, 206 criptografía,  
198  
cursor, en el editor de archivos, 14

## D

estructuras de datos, 180  
para cajas en el programa Animación, 278–279  
para detección de colisiones, 290–291  
copiando en Reversegam, 229–230 para las olas  
del océano en Sonar Treasure Hunt, 184  
para el programa Sprites and Sounds, 306–307  
para tablero de tres en raya, 127–128  
tipos de datos, 13  
Booleano, 31–34  
diccionario, 112–115 enteros,  
2–3, 30–31  
listas, 92–94  
pygame.Rect, 264–265

instrumentos de cuerda. Ver cadenas

depurador

puntos de corte, 73–75

encontrar errores, 70–72

juego de carrera bajo, 66

comenzando,

65 recorriendo el programa con, 67–70

descifrado, 198

en Caesar Cipher, 202–205 bloque

def , 49, 50

declaración de devolución dentro, 55

declaración de definición , 49, 50

eliminación de elementos en la lista, 117–118

del comunicado, 117

tipo de datos de diccionario, 112–113

evaluando con la función choice() , 115–117

métodos keys() y valores() , 114–115

contra listas, 113–114

variables que almacenan referencias a, 134

herramienta de diferencias, 16–17

división (/), 2, 4, 246–247

asignación aumentada, 156

juego de los esquivadores

malos

adición, 327–328

dibujo, 331–332

en movimiento, 328–329

eliminación, 330  
seguimiento de colisiones, 320  
modos de trucos, implementación, 329–330  
detección de colisiones, 332  
variables constantes, 318–319  
dibujar carácter, 331–332 dibujar texto  
en ventana, 320–321  
ventana de dibujo, 330–332  
finalización y pausa, 319–320 funciones,  
319–321  
bucle de juego, 324, 325–327  
juego terminado pantalla, 332–333  
importar módulos, 317–318 modificar,  
333 mover caracteres, 328–329  
  
representación de texto para partitura, 331  
ejecución de muestra,  
313 código fuente, 313–317  
juego inicial, 324–325  
comillas dobles ("'), 42–43  
  
Reino del Dragón  
pidiendo volver a jugar, 61  
Evaluación de operadores booleanos, 54  
comprobación de cuevas, 59 visualización  
de resultados, 58–59  
diagrama de flujo para, 46–47  
funciones, 49–50

obtener la entrada del jugador, [54–55](#)  
cómo jugar, [45](#)  
importar módulos aleatorios y de tiempo , [48](#)  
declaración de devolución , [55–56](#)  
ejecutando bajo depurador, [66](#)  
ejecución de muestra, [46](#)  
código fuente, [47–48](#)  
inicio de programa, [60](#)

## Y

declaraciones elif , [103](#)  
función ellipse() , módulo pygame.draw , [268](#)  
sentencia else , [59](#), [103](#)  
cifrado, [198](#)  
en Cifrado César, [202–205](#)  
parámetro de palabra clave end , para la función print() , [43–44](#)  
fin de programas, [19](#)  
sentencias end=" , [97](#)  
método termina con() , [105](#)  
signo igual (=), como operador de asignación, [5](#), [34](#)  
operador igual a (==) , [32](#), [34](#)  
mensajes de error, [64](#)  
    Error de importación, [256](#)  
    ÍndiceError, [93](#)  
    Error de nombre, [6](#), [16](#)  
errores de sintaxis, [4–5](#)

Error de valor, 139

errores Ver errores

Tecla ESC , para terminar el programa, 295, 326

caracteres de escape, 41–42

evaluación, cortocircuito, 139–141

manejo de eventos, 292–295, 325–326

Objeto de evento , 271, 312

eventos, 270–271

ejecución de programa, 17 función

exit() , módulo sys , 180

salir del programa, 19, 271

expresiones, 3, 36

evaluar, 3–4

en llamadas de función, 19

llamada de función, 18

## F

Valor booleano falso , 31

para tipos de datos, 227

while palabra clave y, 51

editor de archivos, 13–16

extensiones de archivo, para imágenes,

302 función fill() , 266

método find() , 204–205

función flotante() , 29–31

números de punto flotante, 2–3, 4

de la división (/), 246–247

redondeo, 247  
sentencias de control de flujo, 26  
declaraciones elif , 103  
si declaración, 34, 37  
romper declaración y, 35  
declaración else después, 59  
para declaración, 26, 28–29, 37  
declaración while , 51–52, 60  
diagramas de flujo  
para el juego de deducción de bagels, 152–153  
beneficios, 86  
para diseño Hangman, 80–85 para  
diseño Tic-Tac-Toe, 127  
Objeto de fuente , 312, 322  
representación, 263  
fuentes, 262–263  
para declaración, 26, 28–29, 37  
cuadro, 318  
funciones, 18–19, 24. Véanse también nombres de funciones individuales  
llamando, 18, 49–50  
declaraciones def , 49, 50  
parámetros, 57–58  
  
GRAMO  
bucle de juego, 270–271  
para el juego Dodger, 325–327 para  
el juego Reversegam, 237–238

para el programa de animación, 279–283

juegos. Ver nombres de juegos individuales

método `get_rect()`, 264, 321

formato de archivo GIF, 302

alcance mundial, 56–57

variables globales, 67

ventana de interfaz gráfica de usuario (GUI), 260 gráficos

Véase también módulo pygame

Arte ASCII para Hangman, 79

descarga desde navegadores web, 303

sprites para agregar, 302

Juego Adivina el número, 21

comprobación de pérdidas, 35–36

comprobación de ganancias, 35

conversión de valores, 29–31

declaraciones de control de flujo, 26–29

generar números aleatorios, 24–26 obtener la

suposición del jugador, 29 importar módulo

aleatorio , 23–24

ejecución de muestra, 22

código fuente, 22–23

para declaración, 28–29

jugador de bienvenida, 26

## H

Hackear cifrados secretos con Python, 198

juego del ahorcado

arte ASCII, 79  
pidiendo volver a jugar, 104–105  
comprobación de pérdidas, 108  
comprobar si gana, 107  
confirmar la entrada de una conjetura válida, 104  
diccionarios de palabras en, 115  
tablero de exhibición al jugador, 97–101, 106  
mostrando palabra secreta con espacios en blanco, 99–101  
declaraciones elif , 103  
finalizar o reiniciar, 83, 108–109  
extender  
agregar más conjeturas, 112 tipo de  
datos de diccionario, 112–115  
diccionario de evaluación de listas, 115–117  
categoría de palabras impresas, 119  
comentarios al jugador, 85  
diagramas de flujo para el diseño, 80–85  
obtener la suposición del jugador, 101–103  
obtener la palabra secreta de la lista, 96 cómo  
jugar, 78  
conjeturas incorrectas, 107–108  
programa principal, 105–109  
categoría de palabra de impresión para reproductor,  
119 revisión de funciones, 105  
corrida de muestra, 78–79  
código fuente, 88–91  
almohadilla (#), para comentarios, 18, 23

programa hola mundo

crear, 14–15

cómo funciona, 17–19

en pygame, 256–259. Véase también módulo pygame

hipotenusa, 187

yo

INACTIVO

editor de archivos, 13–14

concha interactiva, 1–9, 13

a partir, xxvi

si declaración, 34, 37

romper declaración y, 35

declaración else después, 59

archivos de imagen, 302–303

imágenes Ver gráficos

declaración de importación , 24

Error de importación, 256

módulos de importación, 24

en operador, 94

sangría de código, 27

ÍndiceError, 93

índices

para acceder a los elementos de la lista,

92–93 valor después de eliminar un elemento de

la lista, 117 bucle infinito, 64, 160–161

entrada, 37, 39

validación, 52

función de entrada(), 18–19

instalando

Pygame, 256

Python, función int() xxv–

xxvi , 29–31, 139

enteros, 2–3

convertir cadenas a, 30–31

concha interactiva, 1–9, 13

elementos en listas

acceder con índices, 92–93 cambiar con

asignación de índice, 93

supresión, 117–118

iteración, 29

## j

método join(), 158

Programa de chistes

palabra clave final , 43–44

caracteres de escape, 41–42 cómo

funciona, 41

ejecución de muestra, 40

código fuente, 40

formato de archivo JPG, 302

## k

pares clave-valor, en diccionarios, 113

teclado

manejo de eventos, 325–326  
entrada del usuario con, 292–295  
Evento KEYDOWN , 292, 293–294

llaves

para cífrados, 198  
para diccionarios, 113  
método keys() , 114–115  
Evento KEYUP , 292, 294–295

## L

función len() , 57, 113  
saltos de línea, en cadena impresa, 51  
función line() , módulo pygame.draw , 267  
números de línea, xxiv  
función lista() , 98  
liza  
acceder a elementos con índices, 92–93 cambiar  
el orden de los elementos, 154  
concatenación, 94  
versus diccionarios, 113–114  
y en operador, 94  
iteración y cambios, 298 métodos,  
95  
referencias, 132–135  
rebanar, 98–99  
método sort() para, 157–158

función de carga() , 307  
cargar programa previamente guardado, 15  
ámbito local, 56–57 bucles, 26

romper declaración para salir, 35  
anidado, 161  
con sentencias while , 51–52

método lower() , 101–102  
minúsculas, mostrando caracteres como, 101–102

METRO

función mainClock.tick() , 289–290, 332

Matemáticas

expresiones, 3–4  
enteros y flotantes, 2–3  
operadores, 2  
errores de sintaxis, 4–5  
trucos, 168–169

módulo de matemáticas , 180

métodos, llamada, 94–96  
Formato de archivo MIDI, 303, 322

módulos, importación, 24  
ratón

manejo de eventos, 292–293, 327  
haciendo invisible, 322

evento BOTÓN DEL RATÓN , 292  
Evento MOUSEBUTTONUP , 292, 296

Evento MOUSEMOTION , 292, 325, 327  
método move() , 329  
método move\_ip() , 329  
formato de archivo MP3, 303  
cadenas multilínea, 50–51  
asignación múltiple, 118–119 multiplicación  
(\*), 2  
asignación aumentada, 156  
música  
fondo, 322, 325  
repitiendo para siempre, 308  
configuración, 307–308

norte  
Error de nombre, 6, 16  
nombres, para variables, 20  
bucles anidados, 161, 226  
nueva línea (\n), 42, 43  
Ningún valor, 142  
no es igual a (!=) operador, 32, 36  
no operador, 53–54  
números. Ver también matemáticas  
valor absoluto de, 170  
negativo, en sistema de coordenadas cartesianas, 166–167 método  
sort() para ordenar, 157–158

objetos, 261  
operadores  
    Booleano, 52–55  
    comparación, 32, 33–34  
    matemáticas, 2  
u operador, 53  
orden de operaciones, 3–4  
origen, en sistema de coordenadas cartesianas, 167  
Otelo. Ver juego Reversegam  
salida, 37, 39

PAG

parámetros, para funciones, 57–58  
paréntesis [()], y orden de operación, 4  
programa de pausa, 283, 319–320  
porcentajes, 246–247  
objeto PixelArray , 269  
píxeles, 167, 260  
    colorear, 269–270  
texto plano, 198  
método play() , módulo pygame.mixer.music , 309  
Formato de archivo PNG, 302  
puntos, fuentes, 263  
función polygon() , módulo pygame.draw , 266–267  
función imprimir() , 18, 37  
    parámetro de palabra clave final , 43–44  
programas Ver también nombres de programas y juegos individuales

fin de, 19  
corriendo, 16  
ahorro, 15  
escritura, 13–15  
módulo pygame  
    círculo de funciones  
        de dibujo, 268  
        elipse, 268  
        línea, 267  
        polígono, 266–267  
        rectángulo, 268–269  
    eventos y bucle de juego, 270–271  
    salir de programas, 271  
    rellenar objeto de superficie con color, 266  
    Programa Hello World, 256–257  
        ejecución de muestra, 257  
        código fuente, 257–259  
    módulo de importación, 259  
    inicializando, 259, 321–322  
    instalando, 256  
    Valores de color RGB, 261–262  
    configuración de ventana, 260–261  
módulo pygame.display  
    función set\_caption(), 261, 277  
    función set\_mode(), 260, 261, 321  
    función actualizar(), 266, 270, 282  
módulo pygame.draw

función círculo() , 268  
función elipse() , 268  
función línea() , 267  
función polígono() , 266–267  
función rect() , 268–269

módulo pygame.event  
    Objeto de evento , 271, 312  
    función get() , 271, 292, 312

módulo pygame.font  
    Objeto de fuente , 312, 263  
    Función SysFont() , 263, 322

función pygame.image.load() , 307

función pygame.init() , 259, 321

módulo pygame.locals , 318

módulo pygame.mixer , 307  
    Función de sonido() , 308

módulo pygame.mixer.music , 307, 332  
    función de carga() , 308, 322  
    función reproducir() , 308, 325  
    función detener() , 308

función pygame.quit() , 271, 295

pygame.Rect tipo de datos, 264–265

función pygame.Rect() , 264, 290, 328

pygame.Objeto de superficie , 261, 312. Véase también Objeto de superficie

módulo pygame.time

Reloj() función, 289–290

Objeto reloj , 298, 313, 321

módulo pygame.transform , 307

escala() función, 309

Teorema de Pitágoras, 187–188

Python, instalación, xxv–xxvi

## q

SALIR evento, 271, 292

comillas, para cadenas, 12, 42–43

## R

módulo aleatorio

función elección() , 115–117

importar, 23–24 función

randint() , 24–26

función aleatoria() , 154–155, 232

algoritmo de movimiento aleatorio, 248, 250

contra el mejor algoritmo de esquina, 252–253

función range() , 26, 98

función rect() , 268–269

Recto objeto, 309, 312

función colisionar () , 320

referencias, a listas, 132–135

método remove() , 189–190

método render() , objeto de fuente , 263, 321

renderizar fuentes, 263

declaración de devolución , 55–56  
valor de retorno, 18  
método inverso() , 95  
juego de reversa  
simulación de IA  
comparación de algoritmos de IA, 247–254  
computadora jugando contra sí misma, 240–247  
comprobación de coordenadas válidas, 225–227  
comprobación de movimientos válidos comprobación de  
ocho direcciones, 223–224  
determinar fichas para voltear, 224–225  
constantes, 220  
estrategia de movimientos de esquina, 232  
estructuras de datos  
copiar, 229–230  
creación de tablero nuevo, 222  
dibujo en pantalla, 221–222  
determinar si el espacio es esquina, 230  
determinar quién va primero, 228–229  
bucle de juego, 237–238  
obtener la jugada de la computadora, 232–233  
obtener una lista con todas las jugadas válidas, 226  
obtener la jugada del jugador, 230–232  
obtener la elección de mosaico del jugador, 228  
obtener puntuación, 227–228  
pistas, 214, 226  
cómo jugar, 210–213

módulos de importación, 220 que \_\_\_\_\_  
enumeran los movimientos con la puntuación más alta, 233  
colocación de baldosas, 229  
jugando de nuevo, 238  
 impresión de partituras en pantalla, 234  
 turno de la computadora en ejecución, 236–237  
 ejecución de muestra, 213–215  
 código fuente, 215–220  
 juego inicial, 234–237  
Regresé. Ver juego Reversegam  
valores de color RGB, 261  
triángulo rectángulo, teorema de Pitágoras y, 187–188 función round() ,  
247  
ejecutar programas, 16  
errores de ejecución, 64

## S

guardar programas, 15  
función scale() , módulo pygame.transform , 307, 309  
alcance, global y local, 56–57  
errores semánticos, 64  
función set\_caption() , módulo pygame.display, 261 , 277  
función set\_mode() , módulo pygame.display, 260 , 261, 321  
evaluación de cortocircuito, 139–141  
función shuffle() , módulo aleatorio , 154–155, 232  
mezclar un conjunto único de dígitos, 154–155  
simulación

AI. Consulte Reversegam, simulación de IA de lanzamientos de monedas, 73–75

comillas simples ('), 42–43

función sleep() , módulo de tiempo , 58, 283

listas de

corte, 98–99

operador para, 330

sóñar, 171

Juego Sonar Treasure Hunt

comprobar la pérdida de un jugador, 195

comprobando la victoria del jugador, 194

creación de tablero de juego, 180–181

creación de cofres del tesoro aleatorios, 184–185 diseño, 180

determinar si el movimiento es válido, 185

mostrar el estado del juego para el jugador, 193

dibujar tablero de juego, 181–184 dibujar

océano, 183–184 encontrar el cofre del tesoro

más cercano, 186–189

encontrar cofre del tesoro hundido, 194

obtener el movimiento del jugador, 190–191, 193–194 adivinar

la ubicación, 189 colocar el movimiento en el tablero, 185–191

imprimir las instrucciones del juego, 191–192

salir, 190 eliminar

valores de listas, 189–190 ejecución de muestra, 173–

código fuente, 175–178  
juego inicial, 192–195  
terminación del programa, 195  
variables, 193  
método sort() , para listas, 157–158  
formatos de archivo de sonido, 303  
Función Sound() , módulo pygame.mixer , 308  
Objeto sonoro , 322  
sonidos  
sumando, 308  
para game over, 332  
configuración, 307–308  
activar y desactivar, 308  
código fuente, 14  
Programa AISim1.py , 241–242  
programa AISim2.py , 244  
Programa AISim3.py , 248–249  
Programa de animación, 274–276  
Juego de deducción de bagels, 151–152  
Programa de cifrado César, 201–202  
Programa de detección de colisiones, 287–289  
Juego de los Dodgers, 313–317  
Juego Dragon Realm, 47–48  
Juego Adivina el Número, 22–23  
Juego del ahorcado, 88–91  
programa de chistes, 40  
reutilizar, 24

Juego inverso, 215–220  
Juego Sonar Treasure Hunt, 175–178  
espacios en, xxiv–xxv, 23  
Juego de sprites y sonidos, 304–306  
Juego de tres en raya, 123–126  
espacios  
en código fuente, xxiv–xxv, 23  
entre valores y operadores, 3  
caracteres especiales, impresión, 42  
método split(), 95–96, 190–191  
duendes, 302  
sumando, 307  
cambios de tamaño, 307, 309  
Juego de Sprites y Sonidos  
comprobación de colisión con cerezas, 309 jugador  
de dibujo en la ventana, 309 ejecución de muestra,  
303  
código fuente, 304–306  
configuración de ventanas y estructuras de datos, 306–  
307 función sqrt(), módulo matemático , 180, 188  
corchetes ([]), para índices, 92  
raíz cuadrada, 187  
método comienza con(), 105  
sentencias, 5. Ver también sentencias de control de flujo  
continuar, 161  
definición, 49, 50

del, 117  
fin=", 97  
para, 26, 28–29, 37  
importar, 24  
estadísticas, 246  
recorriendo el programa con el depurador, 65, 67–70  
en código, 68  
fuera, 69  
función stop() , módulo pygame.mixer.music , 332  
función str() , 31, 35  
cuerdas, 12  
comprobando solo números, 158–159  
operadores de comparación y, 33–34  
concatenación, 13, 26  
convertir a entero, 30–31 encontrar,  
204–205 interpolación, 159–160  
  
saltos de linea, 51  
multilínea, 50–51  
comillas para, 42–43  
método split() , 95–96  
resta (-), 2  
asignación aumentada, 155  
Objeto de superficie , 261, 312  
creando, 307  
dibujando en pantalla, 270

llenar de color, 266

errores de sintaxis, 64

en expresiones matemáticas, 4–5

módulo de sistema , 180

## T

tecla TAB (\t), 42

terminar programa, 19

Tecla ESC para, 295, 326

texto. Ver también cadenas

dibujo a ventana, 320–321

entrada por usuario, 18

fuentes para, 262–263

ubicación del escenario, 264–265

Juego de tres en raya

argumentos, 143

inteligencia artificial para, 128–129 comprobación

de tablero completo, 145

Comprobación de los espacios de las esquinas, el centro y los lados,

144–145 Comprobación del movimiento de la computadora para ganar,

143–144 Comprobación del espacio libre a bordo, 138 Comprobación de

la victoria, 135–137

en un solo movimiento, 144

elegir movimiento de la lista, 141–142 elegir la

marca del jugador, 146 decidir quién va primero,

131

diseño con diagramas de flujo, 127–129

duplicar datos del tablero, 137–138 permitir \_\_\_\_\_  
que el jugador elija X u O, 130–131 \_\_\_\_\_  
colocación de marca en el tablero, 131–135 \_\_\_\_\_  
movimiento del jugador, 138–139 \_\_\_\_\_  
jugando de nuevo, 148 \_\_\_\_\_  
tablero de impresión en la pantalla, 129–130 \_\_\_\_\_  
turno de la computadora en funcionamiento, 147–148 \_\_\_\_\_  
turno del jugador que corre, 146–147 \_\_\_\_\_  
ejecución de muestra, 122–\_\_\_\_\_  
123 código fuente, 123–126 \_\_\_\_\_  
juego inicial, 145–146 \_\_\_\_\_  
método tick() , del objeto Clock , 289–290, 298, 313, 332 \_\_\_\_\_  
módulo de tiempo  
importar, 48 \_\_\_\_\_  
función dormir() , 58, 283 \_\_\_\_\_  
rastrear, 64 \_\_\_\_\_  
Valor booleano verdadero , 31 \_\_\_\_\_  
para tipos de datos, 227 \_\_\_\_\_  
while palabra clave y, 51 \_\_\_\_\_  
mesa de la verdad  
para no operador, 54 \_\_\_\_\_  
para y operador, 52–53 \_\_\_\_\_  
para u operador, 53 \_\_\_\_\_  
tuplas, 260 \_\_\_\_\_  
para colores RGB, 261 \_\_\_\_\_  
errores de escritura, 6 \_\_\_\_\_

EN

método update() , módulo pygame.display , 266, 270, 282

método superior() , 101–102

mayúsculas, mostrando caracteres como, 101–102

usuarios

función de entrada() , 18–19

entrada con ratón y teclado, 292–295

EN

Error de valor, 139

valores, 3

asignando nuevo a variable, 7

conversión de tipo de datos, 29–31

almacenar en variables, 5–8

método de valores() , 114–115

Variables

asignación múltiple, 118–119

constante, 91

nombres para, 20

sobreescritura, 7

para referencias a diccionarios, 134

alcance, 56–57, 67, 68

lista de almacenamiento en, 98

almacenar cadenas en, 102

almacenar valores en, 5–8

En

Teclas WASD, 293, 294  
formato de archivo WAV, 303, 322  
declaración while , 51–52, 60  
bucle infinito, 160–161  
bloque de salida, 103  
objeto de superficie de ventana  
método blit() , 269–270, 331  
atributos centerx y centery , 264  
algoritmo del peor movimiento, 248, 250  
contra el mejor algoritmo de esquina, 252  
escribir programas, 13–15

## X

eje x, 165, 167  
coordenadas x, 165  
para objeto PixelArray , 269

## Y

eje y, 165, 167  
coordenadas y, 165 para  
el objeto PixelArray , 269

## DE

índices de base cero, 92, 96

## RECURSOS

Visite <https://www.nostarch.com/inventwithpython/> para obtener recursos, erratas y más información.

Más libros serios de NO STARCH PRESS



### PATIO DE JUEGO DE PROGRAMACIÓN SCRATCH

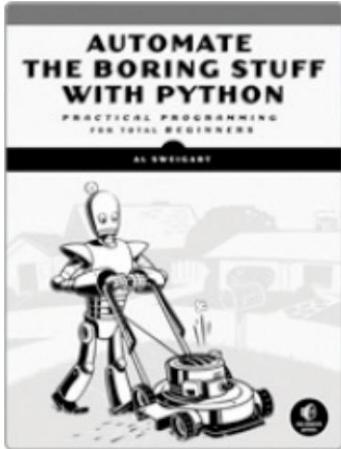
Aprende a programar haciendo juegos geniales de AL

SWEIGART

SEPTIEMBRE 2016, 288 PP., \$24.95

ISBN 978-1-59327-762-8

a todo color



### AUTOMATIZA LAS COSAS ABURRIDAS CON PYTHON

Programación práctica para principiantes totales

por AL SWEIGART ABRIL DE 2015, 504 PP.,

\$29.95

ISBN 978-1-59327-599-0



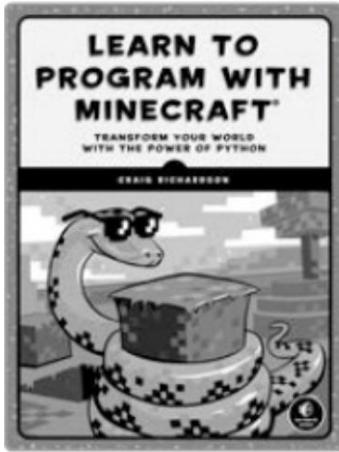
### LA LÍNEA DE COMANDO DE LINUX

Una introducción completa

por WILLIAM E. SHOTTS, JR.

ENERO 2012, 480 PP., \$39.95

ISBN 978-1-59327-389-7



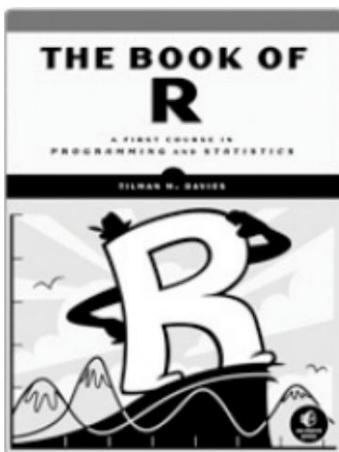
## APRENDE A PROGRAMAR CON MINECRAFT

Transforme su mundo con el poder de Python por CRAIG

RICHARDSON DICIEMBRE DE 2015, 320 PP., \$29.95

ISBN 978-1-59327-670-6

a todo color

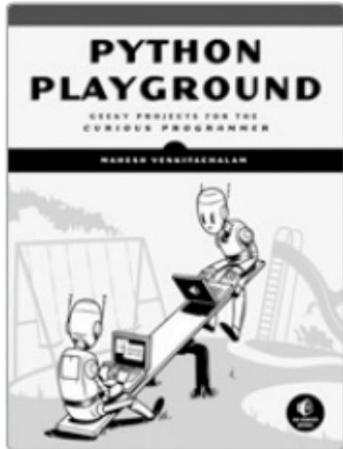


## EL LIBRO DE R

Un Primer Curso de Programación y Estadística por TILMAN

M. DAVIES JULIO 2016, 832 PP., \$49.95

ISBN 978-1-59327-651-5



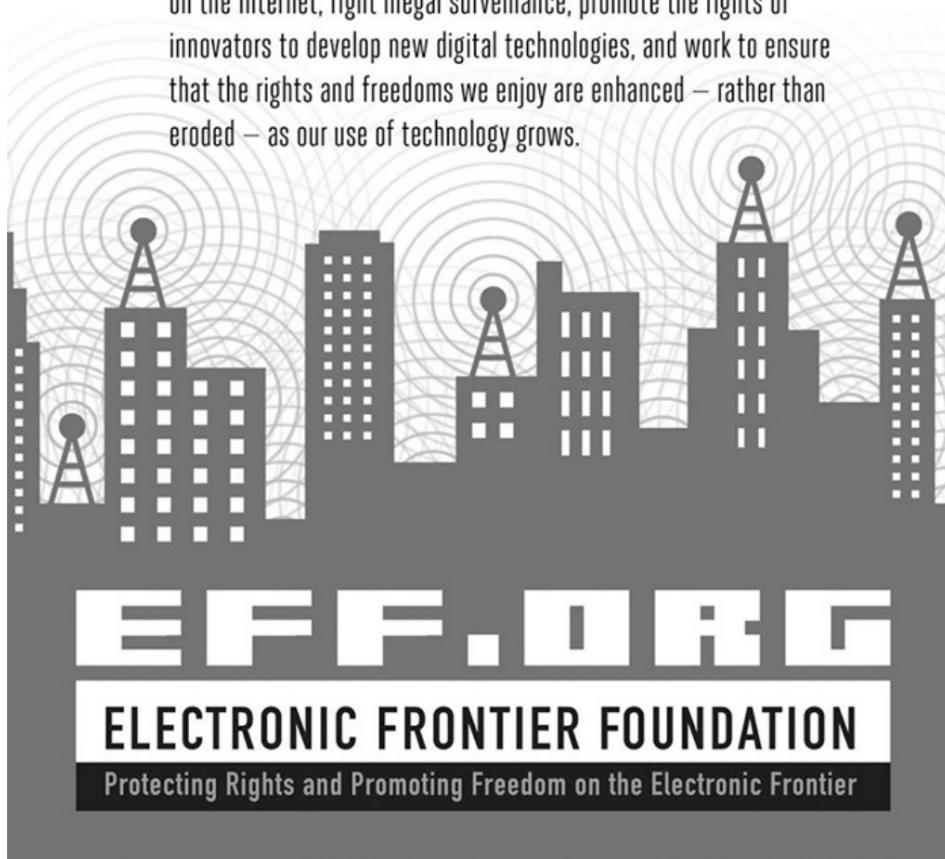
## PATIO DE JUEGOS DE PITONES

Geeky Projects for the Curious Programmer  
por MAHESH VENKITACHALAM OCTUBRE  
DE 2015, 352 PP., \$29.95  
ISBN 978-1-59327-604-1

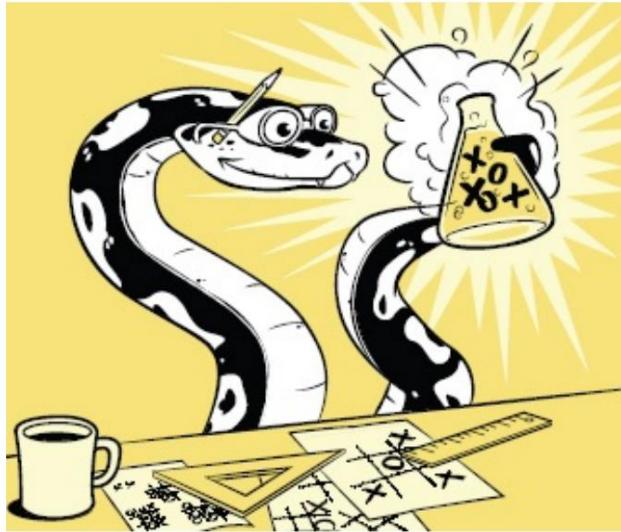
1.800.420.7240 O 1.415.863.9900 | [VENTAS@NOSTARCH.COM](mailto:VENTAS@NOSTARCH.COM) |  
[WWW.NOSTARCH.COM](http://WWW.NOSTARCH.COM)



The Electronic Frontier Foundation (EFF) is the leading organization defending civil liberties in the digital world. We defend free speech on the Internet, fight illegal surveillance, promote the rights of innovators to develop new digital technologies, and work to ensure that the rights and freedoms we enjoy are enhanced – rather than eroded – as our use of technology grows.



## ¡NO JUEGUES SÓLO JUEGOS, HÁGALOS!



Inventa tus propios juegos de computadora con Python te enseñará cómo hacer juegos de computadora usando el popular lenguaje de programación Python, ¡incluso si nunca has programado antes!

Comience creando juegos clásicos como Hangman, Guess the Number y Tic-Tac-Toe, y luego avance hacia juegos más avanzados, como un juego de búsqueda de tesoros basado en texto y un juego animado de esquivar colisiones con efectos de sonido. En el camino, aprenderá conceptos clave de programación y matemáticas que lo ayudarán a llevar su programación de juegos al siguiente nivel.

Aprender como:

- Combine bucles, variables y sentencias de control de flujo en programas de trabajo reales
- Elija las estructuras de datos adecuadas para el trabajo, como listas,

diccionarios y tuplas

- Agregue gráficos y animaciones a sus juegos con pygame módulo
- Manejar la entrada del teclado y el mouse
- Programa inteligencia artificial simple para que puedas jugar contra la computadora
- Usar criptografía para convertir mensajes de texto en código secreto
- Depure sus programas y encuentre errores comunes

A medida que trabaje en cada juego, construirá una base sólida en Python y una comprensión de los fundamentos de la informática.

¿Qué nuevo juego crearás con el poder de Python?

## SOBRE EL AUTOR

Al Sweigart es un desarrollador de software que enseña programación a niños y adultos. Sus tutoriales de programación se pueden encontrar en <https://inventwithpython.com/>. Es el autor más vendido de Automatizar las cosas aburridas con Python y Scratch Programming Playground.

CUBIERTAS PYTHON 3.X



LO MEJOR EN ENTRETENIMIENTO GEEK™

[www.nostarch.com](http://www.nostarch.com)

"YO ME ACOSTO PLANO".

Este libro utiliza una encuadernación duradera que no se cierra de golpe.

a caro