Capítulo 5: Jugar con conjuntos y probabilidad

### **Retos de programación**

A continuación, tienes que resolver algunos retos de programación que te darán la oportunidad de aplicar lo que has aprendido en este capítulo.

#### ***#nº 1: Utilizar diagramas de Venn para visualizar relaciones entre conjuntos***

Un *diagrama* de Venn es una forma sencilla de ver gráficamente la relación entre conjuntos. Nos dice cuántos elementos son comunes entre los dos conjuntos, cuántos elementos están sólo en un conjunto y cuántos elementos no están en ninguno de los dos conjuntos. Consideremos un conjunto, *A*, que representa el conjunto de los números impares positivos menores que 20 -es decir, *A* = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19}- y consideremos otro conjunto, *B*, que representa el conjunto de los números primos menores que 20 -es decir, *B* = {2, 3, 5, 7, 11, 13, 17, 19}. Podemos dibujar diagramas de Venn con Python utilizando el paquete matplotlib\_venn (consulta el [Apéndice A](app01.html#app01) para ver las instrucciones de instalación de este paquete). Una vez instalado, puedes dibujar el diagrama de Venn como se indica a continuación:

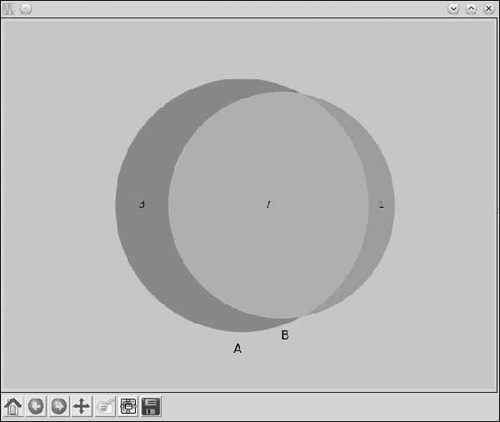
'''  
Draw a Venn diagram for two sets  
'''  
  
from matplotlib\_venn import venn2  
import matplotlib.pyplot as plt  
from sympy import FiniteSet  
  
def draw\_venn(sets):  
  
venn2(subsets=sets)  
plt.show()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
s1 = FiniteSet(1, 3, 5, 7, 9, 11, 13, 15, 17, 19)  
s2 = FiniteSet(2, 3, 5, 7, 11, 13, 17, 19)  
  
draw\_venn([s1, s2])

Una vez que importamos todos los módulos y funciones necesarios (la función venn2(), matplotlib.pyplot, y la clase FiniteSet ), todo lo que tenemos que hacer es crear los dos conjuntos y luego llamar a la función venn2(), utilizando el argumento subsets palabra clave para especificar los conjuntos como una tupla.

[La Figura 5-4](ch05.html#ch5fig4) muestra el diagrama de Venn creado por el programa anterior. Los conjuntos *A* y *B* comparten siete elementos comunes, por lo que se escribe 7 en el área común. Cada uno de los conjuntos también tiene elementos únicos, por lo que el número de elementos únicos -3 y 1, respectivamente- se escribe en las áreas individuales. Las etiquetas debajo de los dos conjuntos se muestran como *A* y *B*. Puedes especificar tus propias etiquetas utilizando el argumento de la palabra clave set\_labels:

>>> venn2(subsets=(a,b), set\_labels=('S', 'T'))

Esto cambiaría las etiquetas de los conjuntos a S y T.

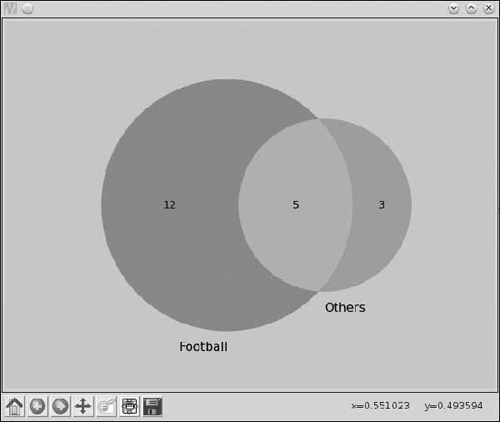


*Figura 5-4: Diagrama de Venn que muestra la relación entre dos* conjuntos, A *y* B

Para tu reto, imagina que has creado un cuestionario online en el que planteas a tus compañeros la siguiente pregunta: *¿Practicas fútbol, otro deporte o ningún deporte?* Una vez que tengas los resultados, crea un archivo CSV, *deportes.csv*, como el siguiente

StudentID,Football,Others  
1,1,0  
2,1,1  
3,0,1  
--snip--

Crea 20 filas de este tipo para los 20 alumnos de tu clase. La primera columna es el ID del alumno (la encuesta no es anónima), la segunda columna tiene un 1 si el alumno ha marcado "fútbol" como el deporte que más le gusta practicar, y la tercera columna tiene un 1 si el alumno practica cualquier otro deporte o ninguno. Escribe un programa para crear un diagrama de Venn que represente los resultados resumidos de la encuesta, como se muestra en la [Figura 5-5](ch05.html#ch5fig5).



*Figura 5-5: Diagrama de Venn que muestra el número de alumnos a los que les gusta jugar al fútbol y el número de alumnos a los que les gusta practicar otros deportes*

Dependiendo de los datos del archivo *sports.csv* que hayas creado, los números de cada conjunto variarán. La siguiente función lee un archivo CSV y devuelve dos listas correspondientes a los ID de los alumnos que practican fútbol y otros deportes:

def read\_csv(filename):  
football = []  
others = []  
with open(filename) as f:  
reader = csv.reader(f)  
next(reader)  
for row in reader:  
if row[1] == '1':  
football.append(row[0])  
if row[2] == '1':  
others.append(row[0])  
  
return football, others

#### ***#2: Ley de los grandes números***

Nos hemos referido a la tirada de un dado y al lanzamiento de una moneda como dos ejemplos de sucesos aleatorios que podemos simular utilizando números aleatorios. Hemos utilizado el término *suceso* para referirnos a la aparición de un número determinado en la tirada de un dado o a la aparición de cara o cruz en el lanzamiento de una moneda, y cada suceso tiene un valor de probabilidad asociado. En probabilidad, una *variable aleatoria -denotada normalmente*por *X- describe*un suceso. Por ejemplo, *X* = 1 describe el suceso de que aparezca 1 en la tirada de un dado, y *P(X* = 1) describe la probabilidad asociada. Hay dos tipos de variables aleatorias: (1) las variables aleatorias *discretas*, que sólo toman valores integrales y son el único tipo de variables aleatorias que veremos en este capítulo, y (2) las variables aleatorias *continuas*, que -como su nombre indica- pueden tomar cualquier valor real.

La *expectativa*, *E*, de una variable aleatoria discreta es el equivalente de la media o promedio que conocimos en el [Capítulo 3](ch03.html#ch03). La expectativa se puede calcular de la siguiente manera:

*E* = *x1P(*x1) + *x2P(*x2) + *x3P(*x3) + ... + *xnP*(*xn*)

Así, para un dado de seis caras, el *valor esperado* de una tirada puede calcularse así:

>>> e = 1\*(1/6) + 2\*(1/6) + 3\*(1/6) + 4\*(1/6) + 5\*(1/6) + 6\*(1/6)  
>>> e  
3.5

Según la *ley de los grandes números*, el valor medio de los resultados en múltiples ensayos se aproxima al valor esperado a medida que aumenta el número de ensayos. Tu reto en esta tarea es verificar esta ley al lanzar un dado de seis caras para el siguiente número de ensayos 100, 1000, 10000, 100000 y 500000. Aquí tienes un ejemplo de ejecución de tu programa completo:

Expected value: 3.5  
Trials: 100 Trial average 3.39  
Trials: 1000 Trial average 3.576  
Trials: 10000 Trial average 3.5054  
Trials: 100000 Trial average 3.50201  
Trials: 500000 Trial average 3.495568

#### ***#3: ¿Cuántas tiradas antes de quedarte sin dinero?***

Consideremos un juego sencillo jugado con un lanzamiento de moneda justo. Un jugador gana 1$ por cara y pierde 1,50$ por cruz. El juego termina cuando el saldo del jugador llega a 0 $. Dada una determinada cantidad inicial especificada por el usuario como entrada, tu reto es escribir un programa que simule este juego. Supón que el ordenador -tu oponente aquí- tiene una reserva de dinero ilimitada. He aquí una posible sesión de juego:

Enter your starting amount: 10  
Tails! Current amount: 8.5  
Tails! Current amount: 7.0  
Tails! Current amount: 5.5  
Tails! Current amount: 4.0  
Tails! Current amount: 2.5  
Heads! Current amount: 3.5  
Tails! Current amount: 2.0  
Tails! Current amount: 0.5  
Tails! Current amount: -1.0  
Game over :( Current amount: -1.0. Coin tosses: 9

#### ***#4: Barajar una baraja de cartas***

Considera una baraja estándar de 52 cartas. Tu reto aquí es escribir un programa para simular el barajado de esta baraja. Para simplificar la implementación, te sugiero que utilices los números enteros 1, 2, 3, ..., 52 para representar la baraja. Cada vez que ejecutes el programa, debe salir una baraja barajada, en este caso, una lista de enteros barajados.

Aquí tienes una posible salida de tu programa:

[3, 9, 21, 50, 32, 4, 20, 52, 7, 13, 41, 25, 49, 36, 23, 45, 1, 22, 40, 19, 2,  
35, 28, 30, 39, 44, 29, 38, 48, 16, 15, 18, 46, 31, 14, 33, 10, 6, 24, 5, 43,  
47, 11, 34, 37, 27, 8, 17, 51, 12, 42, 26]

El módulo random de la biblioteca estándar de Python tiene una función, shuffle(), para esta misma operación:

>>> import random  
>>> x = [1, 2, 3, 4]  
➊ >>> random.shuffle(x)  
>>> x  
[4, 2, 1, 3]

Crea una lista, x, formada por los números [1, 2, 3, 4]. A continuación, llama a la función shuffle() ➊, pasando esta lista como argumento. Verás que los números de x se han barajado. Observa que la lista se baraja "en su sitio". Es decir, se pierde el orden original.

Pero, ¿y si quisieras utilizar este programa en un juego de cartas? En ese caso, no basta con obtener la lista de enteros barajada. También necesitarás una forma de volver a asignar los enteros al palo y rango específicos de cada carta. Una forma de hacerlo es crear una clase Python que represente una sola carta:

class Card:  
def \_\_init\_\_(self, suit, rank):  
self.suit = suit  
self.rank = rank

Para representar el as de tréboles, crea un objeto tarjeta, card1 = Card('clubs', 'ace'). A continuación, haz lo mismo con las demás cartas. A continuación, crea una lista formada por cada uno de los objetos carta y baraja esta lista. El resultado será una baraja de cartas barajada de la que también sabrás el palo y el rango de cada carta. El resultado del programa debería ser algo parecido a esto:

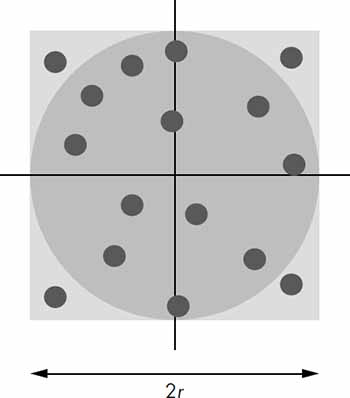
10 of spades  
6 of clubs  
jack of spades  
9 of spades

#### ***#5: Estimación del área de un círculo***

Considera una diana con un círculo de radio *r* inscrito en un cuadrado de lado *2r*. Supongamos que empiezas a lanzarle un gran número de dardos. Algunos de ellos caerán en la diana dentro del círculo -digamos, *N- y*otros fuera de él -digamos, *M*. Si consideramos la fracción de dardos que caen dentro del círculo,

image

entonces el valor de *f* × *A*, donde *A* es el área del cuadrado, sería aproximadamente igual al área del círculo (ver [Figura 5-6](ch05.html#ch5fig6)). Los dardos están representados por los pequeños puntos circulares de la figura. Denominaremos área estimada al valor de *f* × *A*. El área real es, por supuesto, πr2.



*Figura 5-6: Un círculo de radio* r *inscrito en un tablero cuadrado de lado 2r. Los puntos representan dardos lanzados al azar contra el tablero.*

Como parte de este reto, escribe un programa que encuentre el área estimada de un círculo, dado cualquier radio, utilizando este enfoque. El programa debe imprimir el área estimada del círculo para tres valores diferentes del número de dardos:103,105 y106. ¡Son muchos dardos! Verás que al aumentar el número de dardos el área estimada se aproxima al área real. Aquí tienes un ejemplo de salida de la solución completada:

Radius: 2  
Area: 12.566370614359172, Estimated (1000 darts): 12.576  
Area: 12.566370614359172, Estimated (100000 darts): 12.58176  
Area: 12.566370614359172, Estimated (1000000 darts): 12.560128

El lanzamiento de dardos puede simularse mediante una llamada a la función random.uniform(a, b), que devolverá un número aleatorio entre *a* y *b*. En este caso, utiliza los valores *a* = 0, *b* = *2r* (el lado del cuadrado).

##### **Estimación del valor de Pi**

Considera de nuevo [la Figura 5-6](ch05.html#ch5fig6). El área del cuadrado es 4r2, y el área del círculo inscrito es πr2. Si dividimos el área del círculo por el área del cuadrado, obtenemos *π/4*. La fracción *f* que hemos calculado antes

image

es por tanto una aproximación de *π/4*, lo que a su vez significa que el valor de

image

debe ser próximo al valor de *π*. Tu siguiente reto es escribir un programa que estime el valor de *π* suponiendo cualquier valor para el radio. A medida que aumentes el número de dardos, el valor estimado de *π* debería acercarse al valor conocido de la constante.

[anterior](ch05_4.html)[Subtema 5 de 5: (Ver todo)](ch05.html)