Lecture 4

Libraries

- Random
- Statistics
- Command-Line Arguments
- slice
- Packages
- APIs
- Making Your Own Libraries
- Summing Up

Libraries

En general, las bibliotecas son fragmentos de código escritos por ti o por otros que puedes usar en tu programa. Python te permite compartir funciones o características con otros como "módulos". Si copias y pegas código de un proyecto anterior, es probable que puedas crear un módulo o biblioteca que podrías traer a tu nuevo proyecto.

Random

`random` es una biblioteca que viene con Python que puedes importar a tu propio proyecto. Es más fácil como programador apoyarse en el trabajo previo de otros programadores. Entonces, ¿cómo se carga un módulo en tu propio programa? Puedes usar la palabra `import` en tu programa. Dentro del módulo `random`, hay una función incorporada llamada `random.choice(seq)`. `random` es el módulo que estás importando. Dentro de ese módulo, está la función `choice`. Esa función toma una secuencia `seq` que es una lista.

En tu ventana de terminal, escribe `code generate.py`. En tu editor de texto, escribe el siguiente código:

```
import random

coin = random.choice(["heads", "tails"])
print(coin)
```

Nota que la lista dentro de `choice` tiene corchetes, comillas y una coma. Dado que has pasado dos elementos, Python hace los cálculos y da un 50% de probabilidad para "heads" y "tails". Al ejecutar tu código, notarás que este código, de hecho, funciona bien.

Podemos mejorar nuestro código. `from` nos permite ser muy específicos sobre lo que nos gustaría importar. Anteriormente, nuestra línea de importación estaba trayendo todo el contenido de las funciones de `random`. Sin embargo, ¿y si solo queremos cargar una pequeña parte de un módulo? Modifica tu código como sigue:

```
from random import choice

coin = choice(["heads", "tails"])
print(coin)
```

Nota que ahora podemos importar solo la función `choice` de `random`. A partir de ese momento, ya no necesitamos codificar `random.choice`. Ahora solo podemos codificar `choice` por sí sola. `choice` se carga explícitamente en nuestro programa. ¡Esto ahorra recursos del sistema y potencialmente puede hacer que nuestro código se ejecute más rápido!

Sigamos con la función `random.randint(a, b)`. Esta función generará un número aleatorio entre `a` y `b`. Modifica tu código como sigue:

```
import random

number = random.randint(1, 10)
print(number)
```

Nota que nuestro código generará aleatoriamente un número entre 1 y 10.

Podemos introducir en nuestro código `random.shuffle(x)` donde barajará una lista en un orden aleatorio.

```
import random

cards = ["jack", "queen", "king"]
random.shuffle(cards)
for card in cards:
    print(card)
```

Nota que `random.shuffle` barajará las cartas en su lugar. A diferencia de otras funciones, no devolverá un valor. En su lugar, tomará la lista `cards` y la barajará dentro de esa lista. Ejecuta tu código varias veces para ver cómo funciona.

Ahora tenemos estas tres formas de generar información aleatoria.

Puedes aprender más en la documentación de random de Python.

Statistics

Python viene con una biblioteca de estadísticas incorporada. ¿Cómo podríamos usar este módulo? `average` es una función de esta biblioteca que es bastante útil. En tu ventana de terminal, escribe `code average.py`. En la ventana del editor de texto, modifica tu código como sigue:

```
import statistics

print(statistics.mean([100, 90]))
```

Nota que importamos una biblioteca diferente llamada `statistics`. La función `mean` toma una lista de valores. Esto imprimirá el promedio de estos valores. En tu ventana de terminal, escribe `python average.py`.

Considera las posibilidades de usar el módulo `statistics` en tus propios programas. Puedes aprender más en la documentación de statistics de Python.

Command-Line Arguments

Hasta ahora, hemos proporcionado todos los valores dentro del programa que hemos creado. ¿Qué pasa si queremos tomar la entrada desde la línea de comandos? Por ejemplo, en lugar de escribir `python average.py` en la terminal, ¿qué pasa si queremos escribir `python average.py 100 90` y obtener el promedio entre 100 y 90? `sys` es un módulo que nos permite tomar argumentos en la línea de comandos. `argv` es una función dentro del módulo `sys` que nos permite conocer lo que el usuario escribió en la línea de comandos. Nota cómo se utiliza `sys.argv` en el código a continuación. En la ventana de terminal, escribe `code name.py`. En el editor de texto, escribe el siguiente código:

```
import sys

print("hello, my name is", sys.argv[1])
```

Nota que el programa va a mirar lo que el usuario escribió en la línea de comandos. Actualmente, si escribes `python name.py David` en la ventana de la terminal, verás `hello, my name is David`. Nota que `sys.argv[1]` es donde se almacena “David”. ¿Por qué es eso? Bueno, en lecciones anteriores, podrías recordar que las listas comienzan en el elemento 0. ¿Qué crees que se almacena actualmente en `sys.argv[0]? Si adivinastename.py`, estarías en lo correcto.

Hay un pequeño problema con nuestro programa tal como está. ¿Qué pasa si el usuario no escribe el nombre en la línea de comandos? Pruébalo tú mismo. Escribe `python name.py`

en la ventana de la terminal. Se presentará un error `list index out of range`. La razón de esto es que no hay nada en `sys.argv[1]` porque no se escribió nada. Aquí está cómo podemos proteger nuestro programa de este tipo de error:

```
import sys

try:
    print("hello, my name is", sys.argv[1])
except IndexError:
    print("Too few arguments")
```

Nota que ahora se le dará al usuario una pista útil sobre cómo hacer que el programa funcione si se olvidan de escribir un nombre. Sin embargo, ¿podríamos ser más defensivos para asegurar que el usuario ingrese los valores correctos?

Nuestro programa puede mejorarse como sigue:

```
import sys

if len(sys.argv) < 2:
    print("Too few arguments")
elif len(sys.argv) > 2:
    print("Too many arguments")
else:
    print("hello, my name is", sys.argv[1])
```

Nota que si pruebas tu código, verás cómo se manejan estas excepciones, proporcionando al usuario consejos más refinados. Incluso si el usuario escribe demasiados o muy pocos argumentos, se le proporciona al usuario instrucciones claras sobre cómo solucionar el problema.

En este momento, nuestro código es lógicamente correcto. Sin embargo, hay algo muy agradable en mantener nuestra comprobación de errores separada del resto de nuestro código. ¿Cómo podríamos separar nuestra gestión de errores? Modifica tu código como sigue:

```
import sys

if len(sys.argv) < 2:
    sys.exit("Too few arguments")
elif len(sys.argv) > 2:
    sys.exit("Too many arguments")

print("hello, my name is", sys.argv[1])
```

Nota que estamos utilizando una función incorporada de `sys` llamada `exit` que nos permite salir del programa si se introduce un error por parte del usuario. Podemos estar seguros ahora de que el programa nunca ejecutará la última línea de código y no activará un error. Por lo tanto, `sys.argv` proporciona una forma por la cual los usuarios pueden introducir información desde la línea de comandos. `sys.exit` proporciona un medio por el cual el programa puede salir si surge un error.

Puedes aprender más en la documentación de sys de Python.

slice

`slice` es un comando que nos permite tomar una lista y decirle al compilador dónde queremos que considere el inicio y el final de la lista. Por ejemplo, modifica tu código como sigue:

```
import sys

if len(sys.argv) < 2:
    sys.exit("Too few arguments")

for arg in sys.argv:
    print("hello, my name is", arg)
```

Nota que si escribes `python name.py David Carter Rongxin` en la ventana de la terminal, el compilador no solo producirá la salida de los nombres, sino también `hello, my name is name.py`. Entonces, ¿cómo podríamos asegurarnos de que el compilador ignore el primer elemento de la lista donde actualmente se almacena `name.py`?

`slice` se puede emplear en nuestro código para comenzar la lista en otro lugar. Modifica tu código como sigue:

```
import sys

if len(sys.argv) < 2:
    sys.exit("Too few arguments")

for arg in sys.argv[1:]:
    print("hello, my name is", arg)
```

Nota que en lugar de comenzar la lista en 0, usamos corchetes para decirle al compilador que comience en 1 y vaya hasta el final usando el argumento `1:`. Al ejecutar este código, notarás que podemos mejorar nuestro código usando una sintaxis relativamente simple.

Packages

Una de las razones por las que Python es tan popular es que hay numerosas bibliotecas de terceros poderosas que agregan funcionalidad. Llamamos a estas bibliotecas de terceros, implementadas como una carpeta