Generación y almacenamiento de contraseñas

Introducción a Python

Natalia Cadavid Aguilar
Mario Alberto Moctezuma Salazar

Julio 2020

Un curso de EPA dictado por el seminario de Ninja-Pythonistas

Módulo `secrets` — Genere números aleatorios seguros para administrar secretos

El módulo **`secrets`** de Python es usado para generar datos aleatorios seguros (criptográficamente fuertes) para administrar contraseñas, autenticación de cuentas o tokens de seguridad.

En particular, **`secrets`** debe usarse con preferencia al generador de números pseudoaleatorio predeterminado en el módulo **`random`**, que está diseñado para modelar y simular, no para seguridad o criptografía.

Módulo `secrets` — Genere números aleatorios seguros para administrar secretos

`secrets` proporciona acceso a la fuente de aleatoriedad más segura que proporciona tu sistema operativo.

Empezamos por importar a nuestro notebook el código que contiene este módulo:

```
import secrets
```

En **`secrets`** encontramos la clase **`random.SystemRandom`**, la cual genera números aleatorios utilizando las fuentes de mayor calidad proporcionadas por el sistema operativo. En particular usaremos el método **`choice`** de la clase **`random.SystemRandom`**, este nos permitirá escoger de forma aleatoria un elemento de una secuencia (lista, string, etc.) no vacía.

Módulo `secrets` — Genere números aleatorios seguros para administrar secretos

Por ejemplo si tenemos una lista de cosas para hacer

```
ToDo = ['sacar el perro', 'estudiar python',
        'bañarme', 'ejercicio']
```

y asignamos a la variable **Do** la escogencia aletoria

```
Do = secrets.choice(ToDo)
```

entonces

```
print('Lo que debo hacer es:', Do)
'Lo que debo hacer es: estudiar python'
```

Módulo `secrets` — Genere números aleatorios seguros para administrar secretos

Intenta aplicar **`secrets.SystemRandom().sample()`** ingresando como primer parámetro la lista **`ToDo`** y como segundo parámetro un número entero entre 1 y 4. [1]

[1] Tal vez intentar: **`secrets.SystemRandom().sample(ToDo, 2)`**

Módulo strings

El módulo **string** contiene algunas constantes, funciones y clases para manipular caracteres.

Al ser un módulo propio de Python, hay que importarlo antes de usar sus constantes o clases.

```
import string
```

Veamos algunas constantes

```
print(string.ascii_letters)
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

print(string.digits)
0123456789
```

Módulo string

```
print(string.ascii_lowercase)
abcdefghijklmnopqrstuvwxyz

print(string.ascii_uppercase)
ABCDEFGHIJKLMNOPQRSTUVWXYZ

print(string.hexdigits)
0123456789abcdefABCDEF

print(string.punctuation)
"!#$%&'()*+,-./:;?@[\]^_`{|}~
```

Módulo strings

Este módulo tiene una función **capwords(s)**. Esta función convierte en mayúscula la primer letra de cada palabra.

```
frase = 'Bienvenido al módulo string'
print(string.capwords(frase))
'Bienvenido Al Módulo String'
```

ASCII

ASCII es el acrónimo de ”American Standard Code for Information Interchange”, es un código de caracteres basado en el alfabeto latino, tal como se usa en inglés moderno para comunicación electrónica. El código **ASCII** representa texto en computadoras, equipo de telecomuniaciones y otros equipos.

Muchos códigos de caracteres se basan en **ASCII**, aunque pueden incluir otros caracteres.

Python bytes()

El método **bytes()** regresa un objeto en bytes inicializado con el tamaño y los datos dados, los cuales pueden ser un entero, una cadena, una lista, entre otros.

```
frase = "Python es interesante."
# Cadena codificada con ascii
arr = bytes(frase, 'ascii')
print(arr)
b'Python es interesante.'
```

Python `bytes()`

Intenta aplicar **`bytes()`** ingresando una lista sencilla, por ejemplo **`[1,2,3,4]`**, en este caso NO es necesario especificar la codificación ('ascii').

Clase `cryptography.fernet.Fernet`

`Fernet` garantiza que un mensaje encriptado no se puede leer o manipular sin la "llave" (key). Esta clase permite encriptar y desencriptar.

```
from cryptography.fernet import Fernet
```

Primero se debe generar la "llave". ¡Es importante mantenerla en un lugar seguro! (se puede guardar en una memoria por ejemplo). Si se pierde ya no serás capaz de desencriptar los mensajes. Si alguien más tiene acceso a esta, entonces podrá desencriptar tus mensajes.

```
mi_llave = Fernet.generate_key()
```

¡Cada vez que se ejecuta el paso anterior se genera una nueva llave!

Clase `cryptography.fernet.Fernet`

Antes de encriptar o desencriptar hay que proporcionar la llave como

```
Fernet_llave = Fernet(mi_llave)
```

El resultado de la encriptación se conoce como “token de Fernet” y tiene garantías de seguridad y autenticación fuertes.

```
token = Fernet_llave.encrypt(b"Mi contraseña secreta")
```

`.encrypt()` sólo acepta datos en **`bytes`**.

Para desencriptar sólo hay que introducir el “token”

```
Fernet_llave.decrypt(token)
"Mi contraseña secreta"
```

Clase `cryptography.fernet.Fernet`

Crea tu propia llave, recuerda guardarla bien!

Para guardar tu llave puedes usar lo siguiente

```
Llave = open('llave.txt', 'wb+')
Llave.write(mi_llave)
```

y para volverla a usar puedes hacer lo siguiente

```
Llave = open('llave.txt', 'rb+')
mi_llave = Llave.read()
```

Si se guardó el archivo en una memoria o en otro lugar, al momento de abrir se debe especificar la ruta. **¡Ahora intenta encriptar un mensaje!**.

¿ Qué tan segura es mi contraseña?

Ingresa a la página web https://howsecureismypassword.net/ y ve qué tan seguras son las siguientes contraseñas:

1. 12345
2. g^\j5~pcZ^
3. 7\#E7-fD=d

Un programa sencillo, pero ingenuo para generar contraseñas

```
longitud=12
contrasenha=''
for numero in range(longitud):
  contrasenha += secrets.choice(cadena)
print(contrasenha)
```

¿Qué características debe reunir una contraseña para considerarse segura?

Una contraseña es considerada segura si cumple las siguientes condiciones

1. Tiene una longitud mayor o igual a 12
2. Contiene al menos una letra en minúscula
3. Contiene al menos una letra en mayúscula
4. Contiene al menos un número
5. Contiene al menos un símbolo de puntuación
6. No contiene caracteres iguales consecutivos

¿Un programa sencillo que cumpla las condiciones anteriores?

Antes de escribir un programa sencillo que cumpla con los requerimientos anteriores, creemos una función en python (similar a los métodos **isupper()**, **islower()** e **isdigit()**) para detectar elementos en **string.puntuation**:

```
def ispunctuation(caracter):
    if caracter in string.punctuation:
        return True
```

¿Un programa sencillo que cumpla las condiciones anteriores?

```
cadena = string.ascii_letters + string.digits +
         string.punctuation
longitud = int(input('Ingrese la longitud que desea
para su contraseña (solo enteros mayores que 12):'))
while True:
    contrasena = ''.join(secrets.choice(Cadena)
                 for i in range(longitud1))
    if (any(c.islower() for c in contrasena)
        and any(c.isupper() for c in contrasena)
        and sum(c.isdigit() for c in contrasena) >= 1
        and any(ispunctuation(c) for c in contrasena)
        and all(contrasena[k] != contrasena[k-1]
        for k in range(1, longitud1 - 1))):
        break
print(contrasena)
```

¿Cómo mejorar el programa anterior?

Podrías crear una función, digamos **`def contrasena(llave)`** la cual, además de preguntar la longitud, también pregunte la cuenta, y el usuario para cuál se creerá la contraseña. Y que además almacene, de forma encriptada, en un archivo: la cuenta, el usuario y la contraseña generada.

¿Podemos mejorar aún más el programa?

Que tal crear una función, digamos **`def contrasenas(llave)`** que pregunte el número de contraseñas a crear, use la función **`def contrasena()`** (con una leve modificación y esta vez sin recibir la llave como entrada) para almacenar, de forma encriptada, en un archivo: las cuentas con el respectivo usuario y contraseña generada.