Introducción al scripting en Python

Términos relacionados:

- Scripting, Python, programming, scrapping, pwntools, remote, requests.

Referencias:

- Modulo pwntools Python 3: https://github.com/Gallopsled/pwntools
- Python 3 Cheat Sheet
  https://github.com/ehmatthes/pcc/releases/download/v1.0.0/beginners_python_cheat_sheet_pcc_all.pdf
  https://github.com/gto76/python-cheatsheet

Sobre esta guia

Para resolver la mayoría de las prácticas de esta materia, va a ser necesario “ensuciarse las manos” programando, por lo que esta práctica es una introducción al scripting. El lenguaje podría ser cualquiera, pero para unificar, la cátedra recomienda Python3.

Por qué Python?

- Lenguaje de programación open source
- Multiplataforma (Linux, Windows, Mac, Android)
- Simple y fácil de aprender
- Libre y gratuito
- Poderoso lenguaje de alto nivel
- Mundialmente usado (Google, Nasa, etc)
- Portable
- Una cantidad innumerable de librerías

Pero lo maś importante del por qué, es que Python tiene un uso intensivo en la industria de la seguridad informática y seguridad de la información:

- Desarrollo de exploits
- Redes
- Debugging
- Criptografía
- Ingeniería inversa
- Fuzzing
- Web
- Análisis forense
- Análisis de malware

¿Python2 o Python3?:

- Vamos a usar Python3 - https://wiki.python.org/moin/Python2orPython3

Python

Python es un lenguaje dinámico e interpretado. No hay declaraciones de tipos en variables, parámetros, funciones o métodos. Esto hace el código corto y flexible. El chequeo de tipos lo realiza el intérprete en tiempo de ejecución y el tipo de una variable puede cambiar a lo largo de su tiempo de vida.

Una buena manera de ver cómo funciona Python es utilizando su intérprete y ejecutando código directamente en él.

```
$ python3
Python 3.7.4 (default, Jul 11 2019, 10:43:21)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 6        # seteo una variable en esta sesión.
>>> a            # ingresando una expresión, se imprime su valor.
6
>>> a + 2
8
>>> a = 'hola'   # la variable 'a' puede también almacenar un string
>>> a
'hola'
>>> len(a)       # llamo a la función len() con un string
4                # devuelve un entero
>>> a + len(a)   # intento algo que no funciona, tipos incompatibles
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Los archivos de código fuente Python usan la extensión “.py” y son llamados módulos. La manera mas fácil de ejecutar, por ejemplo, “hola.py” es con el comando:

```
$ python3 hola.py Pedro
```

el cual llama al intérprete de Python para ejecutar el código contenido en “hola.py”, pasandole el argumento “Pedro”.

Ejemplo:

```
#!/usr/bin/env python      # shebang, indica al SO el interprete de comandos

# esto es un comentario de una línea

/* esto
es un comentario
de varias líneas */

# importo los módulos que voy a usar
import sys

# los argumentos pasados están en sys.argv[1], sys.argv[2] ...
print ('Hola', sys.argv[1])
```

Ejecuto el programa:

```
$ python hola.py Pedro
Hola Pedro
$ ./hola.py Alicia
Hola Alicia
```

Basics

Enteros (int)

```
>>> puertoHTTP = 80
>>> subred = 24
```

Flotante

```
>>> 8.8 / 4
2.2
```

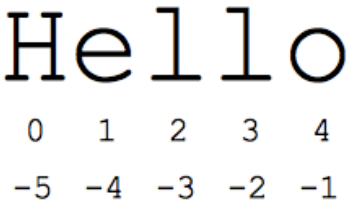
Strings (str)

```
>>> s = "¡Esta es una variable string"
# longitud del string
>>> len(s)
27

# concateno dos strings con '+'
>>> s + '!!!!'
'¡Esta es una variable string!!!!'
```

Slices

Útil para referirse a subpartes de secuencias, típicamente strings y listas. El slice **s[inicio:fin]** incluye los elementos comenzando por *inicio* y extendiéndose hasta *fin* pero sin incluirlo.




```
>>> s = 'Hello'
>>> s[1:4]
'ell'
>>> s[1:]
'ello'
>>> s[:]
'Hello'
>>> s[-1]
'o'
>>> s[:-3]
'He'
>>> s[-3:]
'llo'
```

Jugando con strings

```
>>> logfile = "/var/log/messages"
>>> logfile[0]
'/'
>>> logfile[1:4]
'var'
>>> logfile[-8:]
'messages'
# split() divide un string generando una lista
>>> logfile.split("/")
['', 'var', 'log', 'messages']
>>> logfile.split("/")[1]
'var'
```

Listas

Una lista es una estructura de datos y un tipo de dato en python con características especiales. Lo especial de las listas en Python es que nos permiten almacenar cualquier tipo de valor como enteros, cadenas y hasta otras funciones.

```
# Creo una lista vacía
>>> l = []
>>> l
[]

# Creo una lista con múltiples valores
>>> l = ['Pedro','Alicia','Roberto']
>>> l
['Pedro', 'Alicia', 'Roberto']

# Puedo tener listas con valores de distintos tipos
>>> l = ['Pedro',20,'Alicia',400]
>>> l
['Pedro', 20, 'Alicia', 400]

# Agrego elemento a la lista
>>> l.append('Roberto')
>>> l
['Pedro', 20, 'Alicia', 400, 'Roberto']

# Puedo utilizar slices también en listas
>>> l[:2]
['Pedro', 20]

# Accedo a la primer letra del primer elemento de la lista
>>> l[0][0]
'P'

# Puedo hacer lo mismo con el segundo elemento de la lista?
>>> l[1][0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not subscriptable
# Error de tipo, es un entero, el intérprete lo detecta en ejecución

>>> l[1]
20
```

Diccionarios

Un Diccionario es una estructura de datos y un tipo de dato en Python con características especiales que nos permite almacenar cualquier tipo de valor como enteros, cadenas, listas e incluso otras funciones. Estos diccionarios nos permiten además identificar cada elemento por una clave (Key).

```
# Creo un diccionario vacío
```

```
>>> d = {}

# Creo un diccionario con múltiples valores
>>> d = {'nombre' : 'Pedro', 'edad' : 22, 'direccion' : '1 y 50'}

# Accedo a un elemento mediante la clave de éste
>>> d['nombre']
'Pedro'

# Agrego un nuevo par clave-valor
>>> d['tel'] = 123456
>>> d
{'nombre': 'Pedro', 'edad': 22, 'direccion': '1 y 50', 'tel': 123456}
```

Representación numérica de un carácter Unicode

```
>>> ord("a")
97
>>> ord("A")
65
```

Inverso al anterior, pasar entero a su representación Unicode

```
>>> chr(97)
'a'
>>> chr(65)
'A'
```

String a bytes

```
>>> data = "String"
>>> data.encode()
b'String'
# Otra forma
>>> bytes = b"string"
```

Bytes a string

```
>>> bytes = b"Estos son bytes en diferentes \x66\x6f\x72\x6d\x61\x74\x6f\x73"
>>> bytes.decode()
'Estos son bytes en diferentes formatos'
```

String a bytes a hexadecimal

```
>>> data = "Esto es una prueba"
>>> bytes = data.encode()
b'Esto es una prueba'
>>> bytes.hex()
'4573746f20657320756e6120707275656261'
```

Módulo pwn

Para resolver algunos ejercicios de la práctica, van a necesitar interactuar con distintos servicios accesibles vía red. Para ello, conviene desarrollar un script para que se conecte a dicho servicio. Para alojar los servicios vamos a utilizar el host: “ic.catedras.linti.unlp.edu.ar” y dependiendo el ejercicio al que queramos acceder, el puerto que debemos utilizar.

Por ejemplo, el servicio alojado en el puerto 10001, es un servicio para empezar a probar:

```
$ nc ic.catedras.linti.unlp.edu.ar 10001


  ____       __
   /  _/___  / /__________
   / // __ \/ __/ ___/ __ \
 _/ // / / / /_/ /  / /_/ /
/___/_/ /_/\__/_/   \____/


   _______ __                                        _     __           __
  / ____(_) /_  ___  _____________  ____ ___  _____(_)___/ /___ _____/ /
 / /   / / __ \/ _ \/ ___/ ___/ _ \/ __ `/ / / / ___/ / __  / __ `/ __  /
/ /___/ / /_/ /  __/ /  (__  )  __/ /_/ / /_/ / /  / / /_/ / /_/ / /_/ /
\____/_/_.___/\___/_/  /____/\___/\__, /\__,_/_/  /_/\__,_/\__,_/\__,_/
                                 /____/

Bienvenidos! Tienen un segundo para resolver esta cuenta:
412 + 135
```

Este servicio nos pide resolver una cuenta, pero en una cantidad limitada de tiempo. Pasado el tiempo, la conexión se cierra.

Para resolverlo, utilizaremos la librería **pwntools** de Python 3. Si bien también es posible resolverlo con otras librerías como socket o telnet, en IC usaremos **pwntools** porque servirá en futuras temáticas, en particular sobre binary exploiting.

Para instalar en Linux, utilizamos:

```
sudo apt-get update
sudo apt-get install python3 python3-pip python3-dev git libssl-dev libffi-dev build-essential
python3 -m pip install --upgrade pip --user
python3 -m pip install --upgrade pwntools --user
```

Usando pwntools

```
from pwn import *
# Para debug del socket utilizamos:
# context.log_level = 'debug'
```

```python
# Nos conectamos utilizando remote
con = remote("ic.catedras.linti.unlp.edu.ar", 10001)

# para quitar el texto que no nos interesa (banner),
# leemos hasta justo antes de la cuenta, es decir, hasta ":\n"
con.readuntil("resolver esta cuenta:\n")

# Leemos hasta el salto de línea, la cuenta deseada
cuenta = con.readline()

print(type(cuenta))
print(cuenta)

# Pasamos los bytes a string, para poder realizar la cuenta
cuenta = cuenta.decode()

# Ahora toca resolver la operación

~~#resultado = str(eval(cuenta))~~

# No nos tentemos de usar eval para resolver la operación, es muy poderoso pero
# también peligroso si aceptamos y evaluamos cadenas que provienen de una fuente
# que no es de confianza. Por ejemplo si recibimos el string "2+3", eval lo
# convertirá en una expresión y la ejecutará devolviendo el resultado correcto,
# pero si recibimos "__import__('os').system(...)" eval también la ejecutará,
# logrando así el atacante lanzar comandos en nuestro sistema operativo.

# Para evitar usar eval podríamos parsear la información.
# Split convierte una cadena de texto en una lista, utilizando como separador los
# espacios en blanco
cuenta = cuenta.split() # ['297', '+', '155']

# Convierto a entero los operandos
op1 = int(cuenta[0])
op2 = int(cuenta[2])
operador = cuenta[1]

# Sumo multiplico o resto según el operador
if operador == '+':
    resultado = op1 + op2
elif operador == '*':
    resultado = op1 * op2
else:
    resultado = op1 - op2

# Enviamos la respuesta de la cuenta, como bytes:
con.send((str(resultado) + "\n").encode())

# Imprimimos toda la respuesta del servidor
print(con.readall())
```

Ejercicio 01

Resolver el reto mostrado anteriormente, alojado en el puerto **10001** del sitio **ic.catedras.linti.unlp.edu.ar**

Ejercicio 02

Resolver el reto alojado en el puerto **10002** del sitio **ic.catedras.linti.unlp.edu.ar**
