Miscellaneous
Scripting in Python

Matteo PROTOPAPA
Cybersecurity National Lab

https://cybersecnatlab.it

License & Disclaimer

License Information

This presentation is licensed under the Creative Commons BY-NC License

To view a copy of the license, visit:

http://creativecommons.org/licenses/by-nc/3.0/legalcode

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Python

- Useremo spesso il linguaggio Python, poiché:
  - Esistono numerose librerie che permettono di astrarre i problemi da risolvere
  - Ha una sintassi facile
  - È molto diffuso e dovremo leggere codice scritto da altri
- Di solito, Python è già installato nelle distribuzioni Linux; se così non fosse, può essere installato via apt o tramite il sito ufficiale https://www.python.org/

Sintassi di base

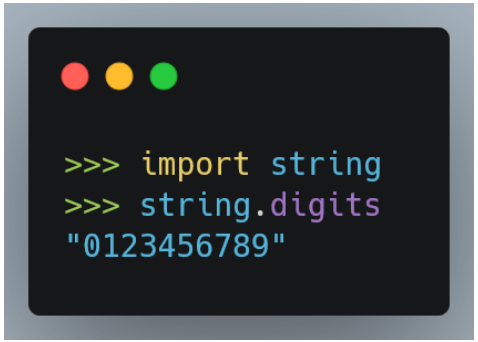

```
>>> a = 1                    # non serve dichiarare le variabili per assegnare un valore
>>> a = "uno"                # Python non è fortemente tipizzato, ogni variabile può contenere vari tipi di dato
>>> b = "ciao"
>>> c = " a tutti"
>>> b+c                      # in Python si possono sommare (concatenare) stringhe
"ciao a tutti"
>>> d = [None]*20            # creazione di una lista di 20 elementi
>>> d[19] = "ciao"           # assegna la stringa "ciao" alla lista
>>> d[18] = 20               # le liste non sono tipizzate, si possono inserire
                             # elementi di tipi diversi
>>> d[-1]                    # gli indici negativi partono dalla fine della lista, con -1 che corrisponde
                             # all'ultimo elemento
"ciao"
>>> e = {}                   # creazione di un dizionario vuoto
>>> e["chiave"] = "valore"   # inserisce la chiave e il valore nel dizionario
>>> e["chiave"]              # è possibile recuperare il valore usando la stessa chiave
"valore"
```

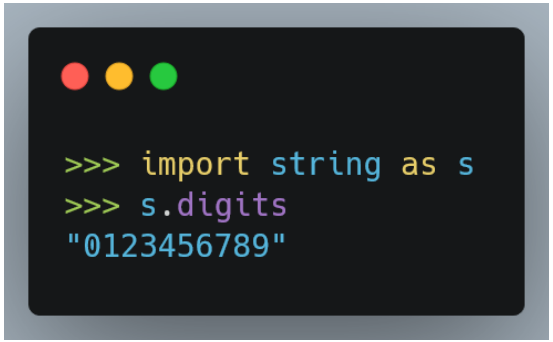
Moduli

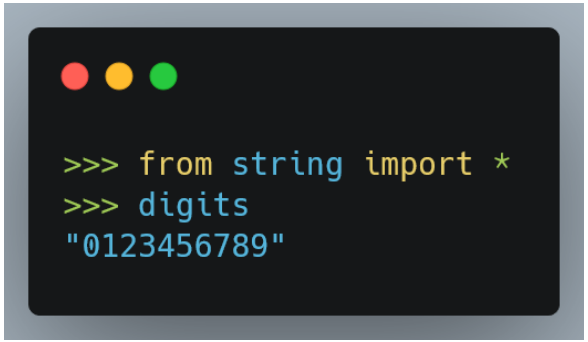
- In Python, le librerie si chiamano moduli
- Come per gli altri linguaggi, una libreria contiene codice già funzionante organizzato per funzioni, con cui possiamo facilmente risolvere determinati problemi
- Di solito, un modulo Python si installa tramite il comando pip3 install <nome_del_modulo>

Moduli

- È possibile importare un intero modulo, eventualmente con un alias, ed accedere ai suoi metodi
- È possibile importare metodi specifici (o * per importarli tutti)







Costrutti

- In Python è importante la formattazione del codice, poiché i suoi costrutti usano l'indentazione per determinare inizio e fine
- Riuscite a indovinare l'output di questo script?

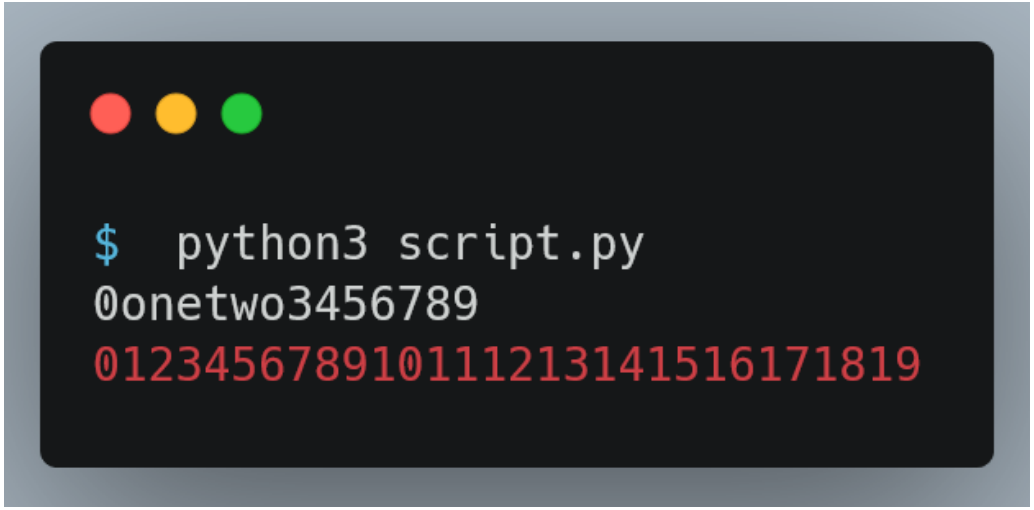
```
from string import digits

for d in digits:
        if d == "2":
                print("two", end="")
        elif d == "1":
                print("one", end="")
        else:
                print(d, end="")

print("")

for i in range(20):
        print(i, end="")
```

Costrutti



Costrutti

- Nello script, ci sono esempi di vari costrutti:
  - if – elif – else
  - for (può essere usato per iterare agevolmente sugli elementi di una lista o di una stringa)
  - range (la funzione range(a,b,c) permette di iterare su tutti gli interi da a a b-1 inclusi, con uno step c)
  - print

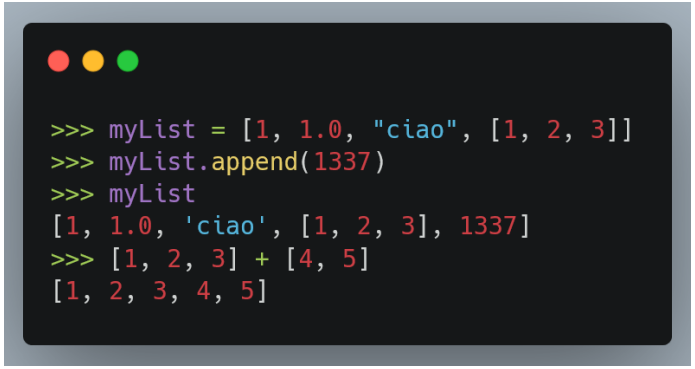
Costrutti

- Un ciclo while è molto simile al ciclo for

```
>>> n = 3
>>> while n < 5:
...     print(n)
...     n = n+1
...
3
4
```

Liste

- Come visto nel primo esempio, in Python le liste possono contenere dati di qualsiasi tipo
- Si possono aggiungere elementi alla fine di una lista con append
- Si possono concatenare più liste come se fosse una somma



Liste

- Attenzione! Nell'esempio viene creata la lista
  - d = [None] * 20
- Con questa sintassi, viene inserito lo stesso oggetto più volte, non una sua copia

```
>>> l = [[1,2,3]]*3
>>> l
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
>>> l[0].append(4)
>>> l
[[1, 2, 3, 4], [1, 2, 3, 4], [1, 2, 3, 4]]
```

```
>>> l = [[1,2,3] for _ in range(3)]
>>> l
[[1, 2, 3], [1, 2, 3], [1, 2, 3]]
>>> l[0].append(4)
>>> l
[[1, 2, 3, 4], [1, 2, 3], [1, 2, 3]]
```

Liste

- Come visto nell'esempio, per accedere ad una lista si usano le parentesi quadre e gli indici negativi partono dalla fine della lista
- La notazione myList[a:b:c] permette di selezionare gli indici da a a b-1 inclusi, con uno step c
- Di conseguenza, un modo facile per invertire l'ordine di una lista è myList[::-1]

```
>>> import string
>>> myList = list(string.digits)
>>> myList
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
>>> myList[0]
'0'
>>> myList[-2]
'8'
>>> myList[3:9:3]
['3', '6']
>>> myList[::-1]
['9', '8', '7', '6', '5', '4', '3', '2', '1', '0']
```

Dizionari

- Dal punto di vista dell'utilizzo, i dizionari sono come delle liste, ma con delle chiavi invece che degli indici

```
>>> d = {2: "stringa", "chiave": "valore"}
>>> d[2]
'stringa'
>>> d["flag"] = 7
>>> d
{2: 'stringa', 'chiave': 'valore', 'flag': 7}
>>> d.keys()
dict_keys([2, 'chiave', 'flag'])
>>> d.values()
dict_values(['stringa', 'valore', 7])
>>> for k in d:
...     print(str(k) + ": " + str(d[k]))
...
2: stringa
chiave: valore
flag: 7
```

Stringhe

- Anche le stringhe si usano in modo simile alle liste, ma sono immutabili, quindi non si può cambiare il valore di una parte di una stringa

```
>>> s = "stringa bellissima"
>>> s[3]
'i'
>>> s[-3:]
'ima'
>>> s[0] = 'x'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Stringhe

- Python fornisce molti metodi utili per la manipolazione di stringhe

```
>>> s = "abbiamo una stringa"
>>> s.split(' ')
['abbiamo', 'una', 'stringa']
>>> s.strip('a')
'bbiamo una string'
>>> s.replace("bbiamo", "vete")
'avete una stringa'
>>> s.rjust(30, ' ')
'          abbiamo una stringa'
```

Stringhe vs Bytes

- Alcune funzioni utilizzano variabili di tipo *bytes*, altre di tipo *string*; in Python 2, questi tipi erano fondamentalmente la stessa cosa, con Python 3 sono tipi diversi e l'interprete può generare errori
- Per convertire da *string* a *bytes*, esiste il metodo .encode(); per l'operazione inversa si usa .decode()

Codifiche

- ASCII
- Binario
- Decimale
- Esadecimale
- Base64

Codifiche

ASCII	Binario	Decimale	Esadecimale	Base64
A	01000001	65	41	QQ==
a	01100001	97	61	YQ==
0	00110000	48	30	MA==
}	01111101	125	7d	fQ==

Codifiche

- Esistono vari modi di passare da una codifica ad un'altra
- Ovviamente è possibile usare una codifica intermedia se non esistono metodi che convertono direttamente

```
>>> import base64
>>> ord('A')
65
>>> chr(97)
'a'
>>> int('01111101', 2)
125
>>> bin(48)
'0b110000'
>>> hex(65)
'0x41'
>>> base64.b64encode(b'a')
b'YQ=='
>>> base64.b64decode(b'MA==')
b'0'
>>> bytes.fromhex('7d')
b'}'
>>> b'A'.hex()
'41'
```

requests

- *requests* è un modulo che consente di comunicare con un server tramite HTTP

```
>>> import requests
>>> response = requests.get("http://www.google.com")
>>> response
<Response [200]>
>>> response.text[:100]
'<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="it"><head><meta
content'
>>> response.headers
{'Date': 'Wed, 18 Jan 2023 00:00:41 GMT', 'Expires': '-1', 'Cache-Control': 'private, max-
age=0', 'Content-Type': 'text/html; charset=ISO-8859-1', 'Cross-Origin-Opener-Policy-Report-
Only': 'same-origin-allow-popups; report-to="gws"', 'Report-To':
'{"group":"gws","max_age":2592000,"endpoints":[{"url":"https://csp.withgoogle.com/csp/report-
to/gws/other"}]}', 'Content-Encoding': 'gzip', 'Server': 'gws', 'Content-Length': '6230', 'X-
XSS-Protection': '0', 'X-Frame-Options': 'SAMEORIGIN', 'Set-Cookie':
'AEC=ARSKqsJTR8MiFHUBn3G6IZHc38S0g0meGtsBW8jjiQZSwBH8UsXrf7PlEz0; expires=Mon, 17-Jul-2023
00:00:41 GMT; path=/; domain=.google.com; Secure; HttpOnly; SameSite=lax'}
```

requests

- Con *requests* è possibile specificare cookie e header nelle richieste

```
>>> import requests
>>> cookies = {'myCookie': 'c'}
>>> headers = {'myHeader': 'h'}
>>> requests.get("http://www.google.com", cookies=cookies, headers=headers)
<Response [200]>
```

requests

- È anche possibile gestire automaticamente le sessioni, di modo che sia il modulo a gestire i cookie necessari al normale funzionamento della pagina

```
>>> import requests
>>> s = requests.Session()
>>> s.get("http://www.google.com")
<Response [200]>
>>> s.cookies.get_dict()
{'AEC': 'ARSKqsKSlbkeVT2g2y_S1MXjgCrDsDOUJ6XdnRbNBRQKygyTP419M5YzwlE'}
```

pwntools

- Il modulo *pwntools* permette invece di comunicare tramite socket oppure con un programma in locale

```
$ ./helloworld
Hello world!
$ python3
Python 3.8.10 (default, Nov 14 2022, 12:59:47)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pwn
>>> p = pwn.process("./helloworld")
[x] Starting local process './helloworld'
[+] Starting local process './helloworld': pid 133
>>> p.recvline()
[*] Process './helloworld' stopped with exit code 0 (pid 133)
b'Hello world!\n'
```

pwntools

- Si apre la connessione con process(proc) o con remote(ip, port)
- Si riceve l'output con i metodi .recv(n), .recvline(s), .recvlines(n), .recvuntil(stop)
- Si invia l'input con i metodi .send(s), .sendline(s)
- Si chiude la connessione con .quit()
- Attenzione: *pwntools* usa il tipo *bytes*!

pwntools

- Il modulo *pwntools* offre un'infinità di altre funzionalità, alcune saranno presentate nei prossimi giorni

Miscellaneous
Scripting in Python

Matteo PROTOPAPA
Cybersecurity National Lab

https://cybersecnatlab.it