



**Università
degli Studi
di Ferrara**



**CYBER
CHALLENGE.IT**



Laboratorio Python Introduttivo

— CyberChallenge 2025 —

Giacomo Bettini - giacomo.bettini@unife.it

Lucia Ferrari - lucia03.ferrari@unife.it

Matteo Brina - matteo.brina@unife.it



**Università
degli Studi
di Ferrara**



**CYBER
CHALLENGE.IT**



Codice Classroom CC25: k3h7nms

— CyberChallenge 2025 —

Il Linguaggio Python

- Python è un linguaggio open source, ad oggetti e *interpretato* creato da Guido Van Rossum
- La prima versione risale al 1991
- Sono state rilasciate diverse versioni tra cui: 2, 2.7, 3, 3.6, 3.8, 3.12...
- Alcune versioni meno recenti come la 2.7 sono tuttora utilizzate, ma il loro supporto è terminato nel 2020
- In questo laboratorio seguiremo le **regole definite dalla versione 3.6+**, che rappresentano il riferimento più attuale

Ambiente di programmazione

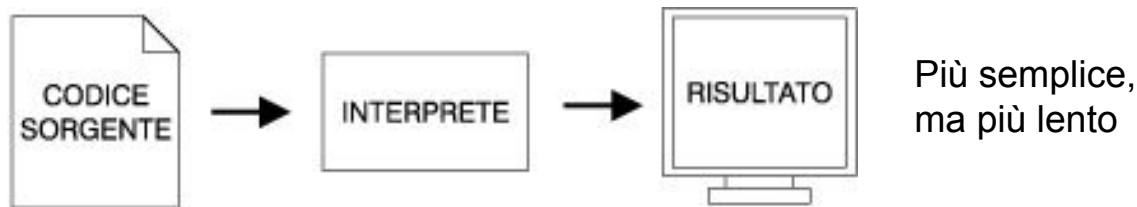
Due componenti principali:

- **Interprete** (*shell*): è la finestra che si apre all'avvio dell'ambiente di programmazione, e consente la valutazione di espressioni e l'esecuzione di istruzioni in modalità interattiva.
- **Editor di testo** (una o più finestre che si possono aprire attraverso la voce di menu *File > New window*): consente la scrittura in un *file* di testo di un programma, e successivamente di eseguirlo.

Interpretazione vs Compilazione

Prima di eseguire i programmi scritti in un linguaggio di alto livello, questi devono essere elaborati. Due tipi di elaborazione:

Interpretazione:



Compilazione:



Tipizzazione

Python è un linguaggio **dinamicamente tipizzato**. A differenza del C (che è **staticamente tipizzato**), quando una variabile viene definita non ne deve per forza essere specificato il tipo.

In questo tipo di tipizzazione, l'interprete Python controllerà *il tipo* solamente a ***tempo di esecuzione***.

Python usa il cosiddetto **duck typing**: *"If it walks like a duck, and it quacks like a duck, then it is a duck"*. Infatti, Python inferisce il tipo di una variabile guardandone i metodi e gli attributi.

```
>>> if False:
...     1 + "two"
...     # Non viene mai eseguito (e quindi mai interpretato).
...     # Non viene segnalato nessun TypeError.
... else:
...     1 + 2
...
3
>>> # Il risultato è 3.
>>>
>>> # Ora chiediamo direttamente di valutare 1 + "two", che
>>> # questa volta, dovendo eseguire, scatena un errore
>>> 1 + "two"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
```

Indentazione

A differenza di altri linguaggi che delimitano blocchi di codice con parentesi graffe (come C, C++ e Java), Python usa l'**indentazione** per identificare dei blocchi logici di codice da eseguire.

L'uso dell'indentazione permette anche di aumentare la leggibilità e quindi la comprensione del codice.

Linguaggio C

```
printf("Operazione 1\n");  
if (condizione) {  
    printf("Operazione 2\n");  
}  
printf("Operazione 3\n");
```

Linguaggio Python

```
print("Operazione 1")  
if condizione:  
    ...print("Operazione 2")  
print("Operazione 3")
```

Realizzazione di codici Python

I codici Python possono essere scritti all'interno di *file* che per convenzione hanno estensione **.py**.

I file possono essere creati tramite l'utilizzo di semplici editor di testo oppure tramite uso di ambienti di programmazione (IDE).

Ci sono diversi ambienti di programmazione, alcuni dei quali sono PyCharm (IDE pensato specificatamente per Python) oppure Visual Studio Code.

Nel corso di questo laboratorio si farà uso dell'IDE **Visual Studio Code**.

Installazione Strumenti

Installazione Python

In alcune distribuzioni Linux, la versione Python 3 deve essere installata manualmente.

Il procedimento esposto di seguito riguarda l'installazione di Python 3 tramite package manager per Ubuntu 20.04, Ubuntu 22.04 e successivi.

Per installare Python 3, inserire su terminale i seguenti comandi:

```
$ sudo apt update
```

```
$ sudo apt install python3
```

Installazione Python

Per verificare la corretta installazione, dare il comando `python3 --version` il quale fornirà la versione installata.

```
$ python3 --version
```

```
$ Python 3.x.x
```

Per semplificarne l'uso, è possibile definire un alias come segue:

```
$ alias python=python3
```

Oppure (consigliato) installare il pacchetto `python-is-python3`:

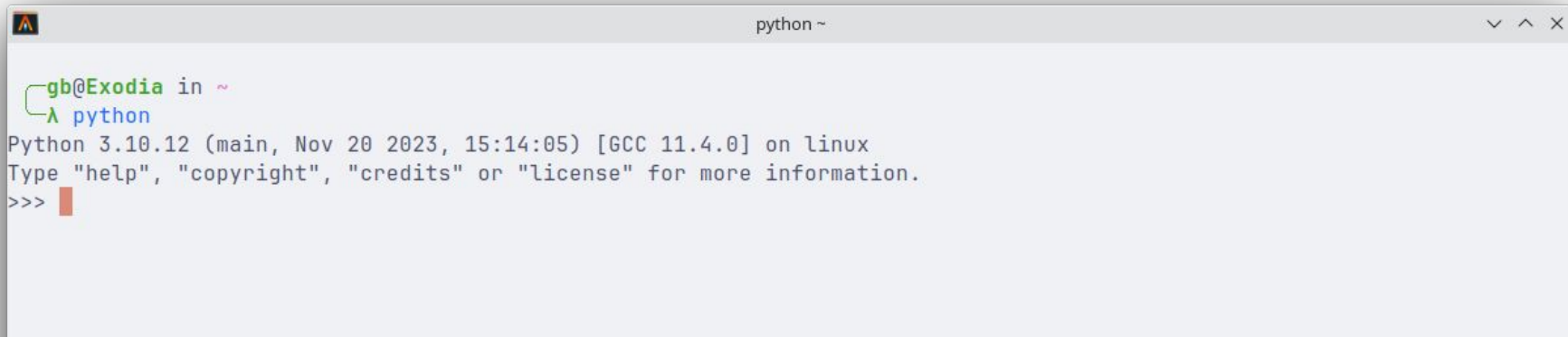
```
$ sudo apt install python-is-python3
```

Console Python

Una volta installato Python, è possibile utilizzare il suo interprete anche da linea di comando (Console Python).

Tra le varie possibilità che questa offre, permette di effettuare rapidamente test sull'utilizzo di parti di codice Python, senza dover per forza creare ed eseguire script.

È possibile eseguire l'*interprete Python* (se Python è stato installato correttamente), tramite il comando `python` (oppure `python3`)



```
python ~
gb@Exodia in ~
python
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

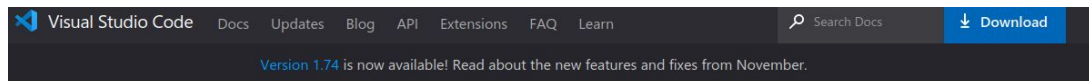
Console Python

Notare la tipizzazione
dinamica

```
python ~
gb@Exodia in ~
λ python
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 3
>>> b = 6
>>>
>>> c = a + b
>>>
>>> print(c)
9
>>>
```

Installazione VSCode

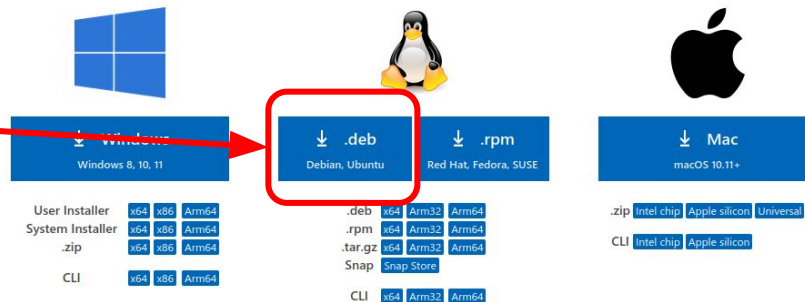
<https://code.visualstudio.com/download>



Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.

Scaricare il
file .deb



Installazione VSCode

1) Aprire un terminale, spostarsi nella cartella dei file scaricati (di solito *Scaricati* o *Downloads*)

2) Dare il comando:

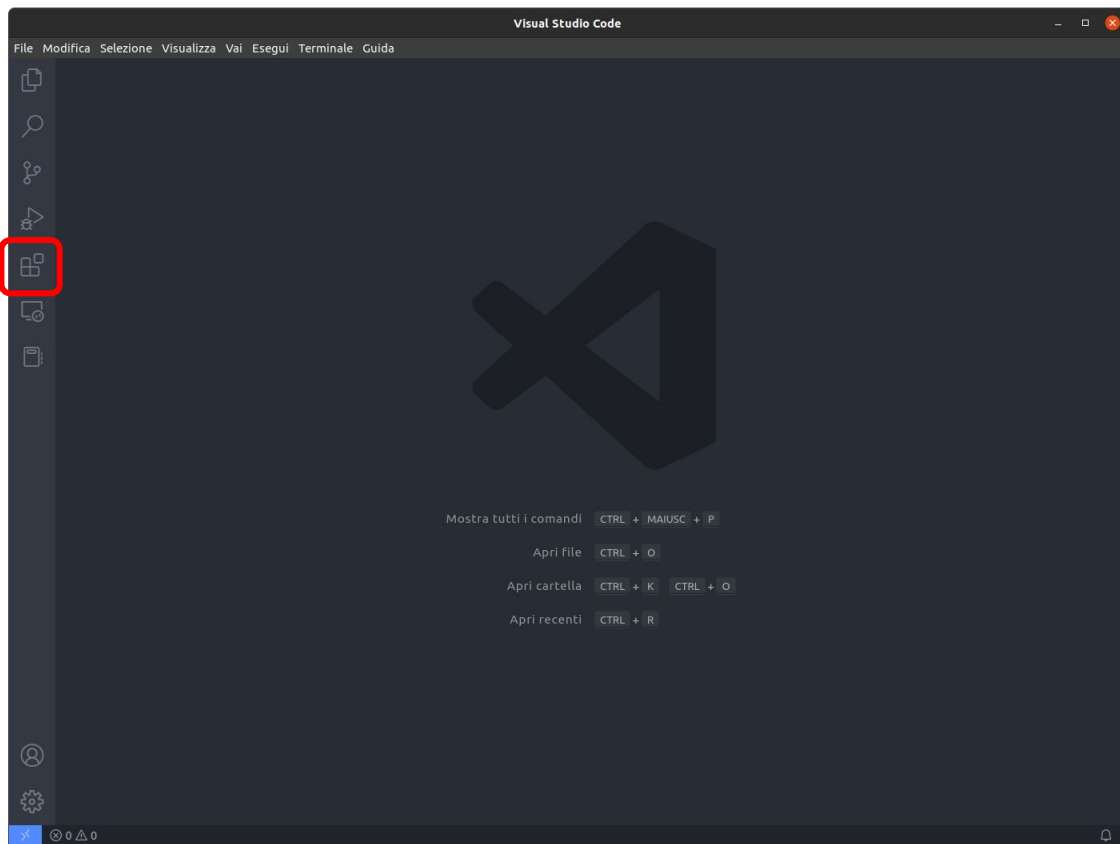
\$ sudo dpkg -i code*.deb

A screenshot of a terminal window titled '~/.Scaricati'. The window has a light gray background and a title bar with standard window controls. At the top, there are six large, colorful icons representing different file managers or applications: red, green, orange, blue, pink, and teal. Below these icons, the terminal shows two lines of commands and their output. The first line shows the user 'gb@Exodia' in the home directory '~' running 'cd Scaricati/'. The second line shows the user running 'sudo dpkg -i code_1.80.0-1688479026_amd64.deb' via 'v3.10.12'.

```
gb@Exodia in ~  
λ cd Scaricati/  
  
gb@Exodia in ~/.Scaricati via v3.10.12  
λ sudo dpkg -i code_1.80.0-1688479026_amd64.deb
```

Installare le estensioni Python

Cliccare su
estensioni

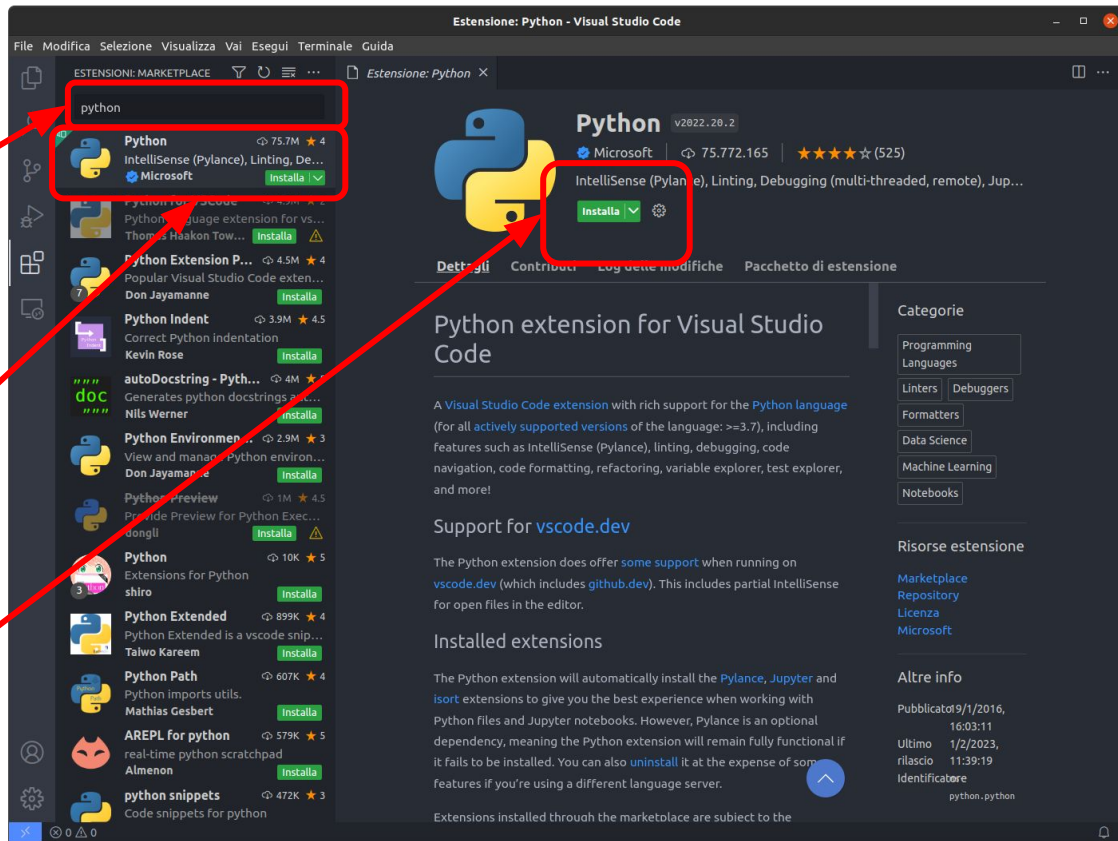


Installare le estensioni Python

1) Cercare "Python"

2) Cliccare sull'estensione "Python" di Microsoft

3) Cliccare "Installa"

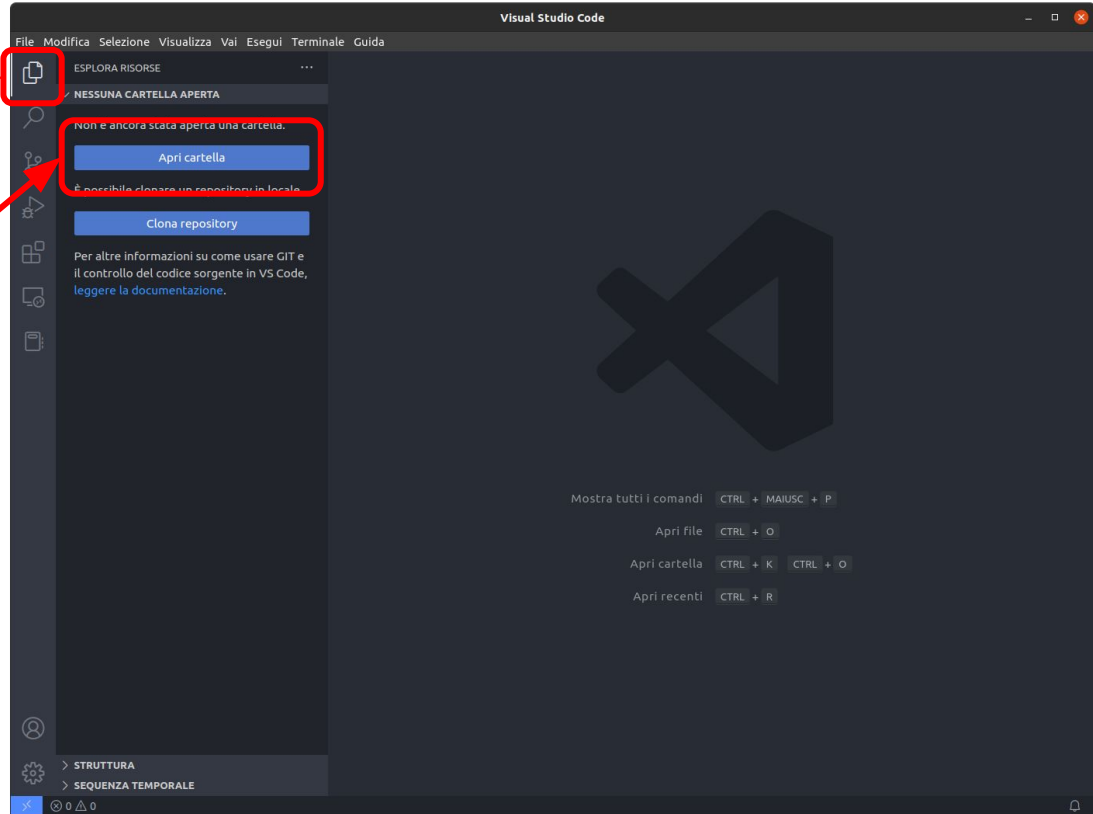


Creare ed eseguire script Python con VSCode

1) Cliccare su
"Esplora risorse"

2) Cliccare su
"Apri cartella"
e selezionare una
cartella a piacere

Se viene chiesto *"Si considerano attendibili gli autori dei file in questa cartella?"* rispondere *"Sì, mi fido degli autori"*

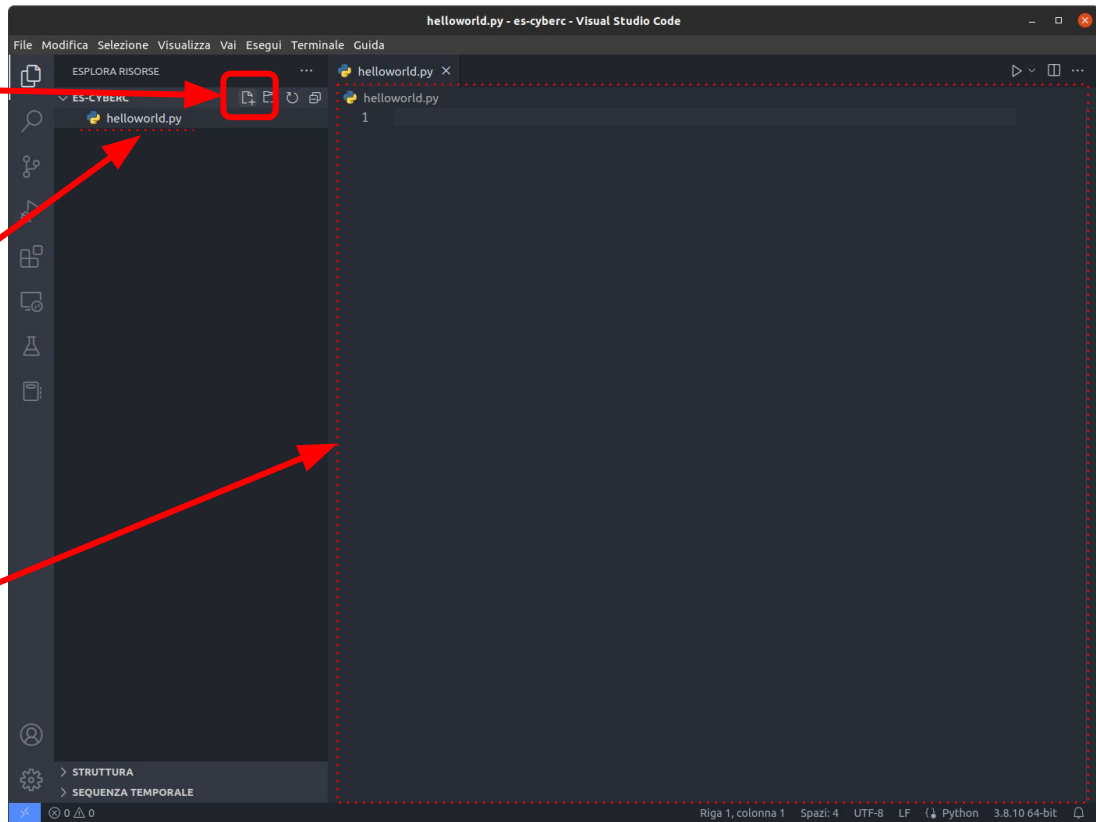


Creare ed eseguire script Python con VSCode

1) Cliccare su
"Nuovo file"

2) Dare un nome al file
con estensione .py (es.
helloworld.py)

Sulla destra potremo
scrivere il codice dello
script

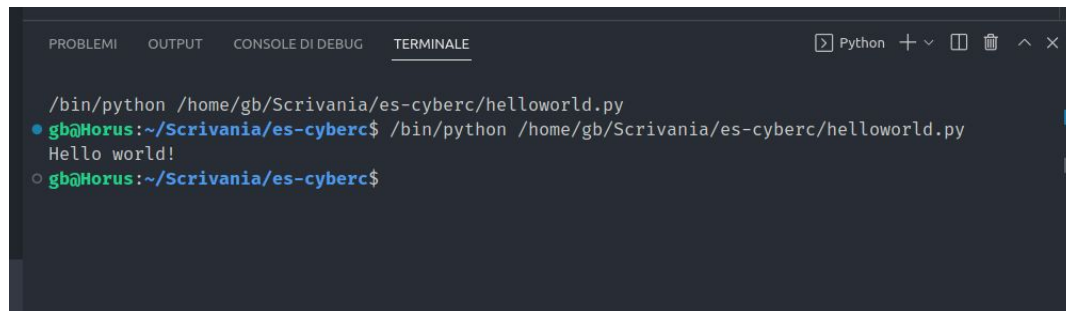
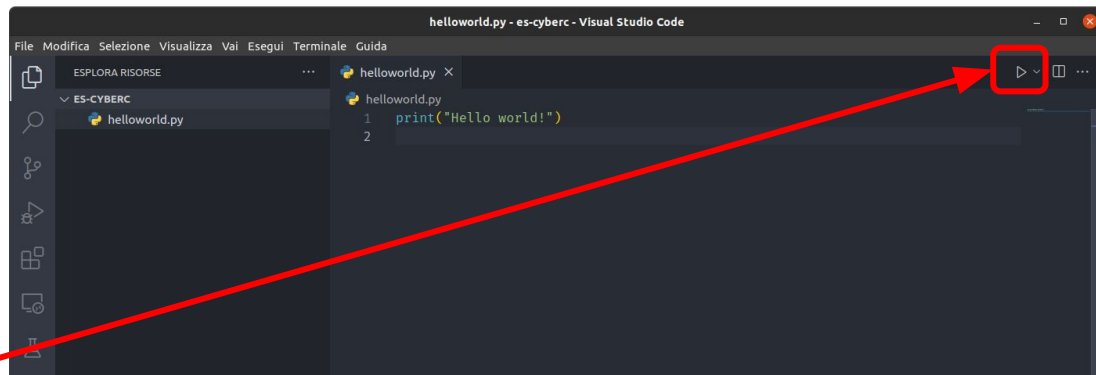


Creare ed eseguire script Python con VSCode

1) Scrivere il codice, ad esempio un semplice:
`print("Hello world!")`

2) Cliccare su "Esegui
il file Python"

3) Si aprirà un terminale
integrato a VSCode che
eseguirà lo script Python



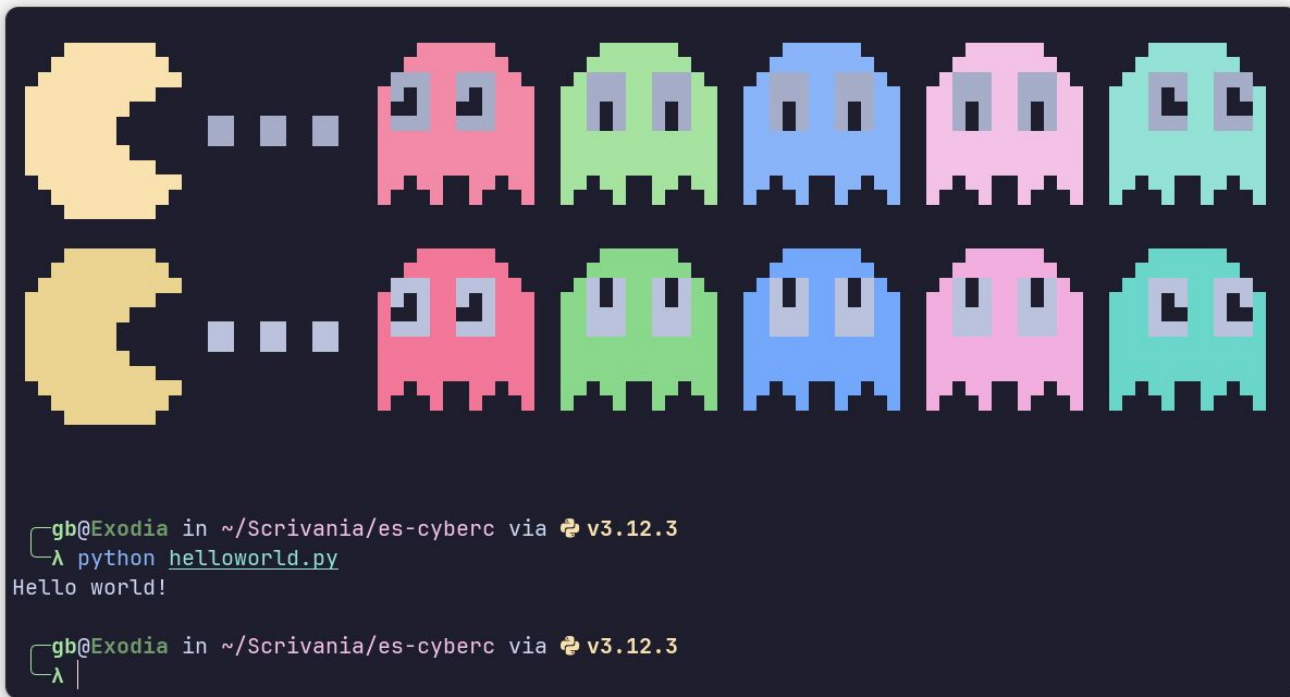
Creare ed eseguire script Python da terminale

Oppure, da terminale:

1) Spostarsi nella cartella
contenente lo script

2) `python helloworld.py`

In alcuni sistemi, e a seconda
della propria configurazione,
invece di **python** si dovrà
digitare **python3**



Elementi Base

Funzioni predefinite di Input/Output

Input:

- `input_utente = input(testo)`
stampa sullo schermo il messaggio `testo`, poi resta in attesa che l'utente scriva una *qualsiasi* sequenza di caratteri attraverso la tastiera, e dopo la pressione del tasto "Invio" restituisce tale sequenza all'interno di una stringa

Output:

- `print(espressione)`
stampa sullo schermo il *valore* di `espressione` (un'espressione *qualsiasi*)
- `print(f"Ciao {nome} {cognome}, benvenuto in {sito}. 2+3={2+3}")`
"f-string". Consente di stampare testo, variabili ed espressioni con una sintassi molto comoda.

Hello World in Python

file *helloworld.py*

```
def stampa() :  
    print("Hello World!")  
  
if __name__ == "__main__":  
    stampa()
```

Definizione di una
funzione `stampa()`

all'avvio del programma eseguo la
funzione `main()`

Non è obbligatorio definire una
funzione `main` per eseguire il `print`
di una stringa o più istruzioni in
sequenza.

Hello World in Python

file *helloworld.py*

```
def stampa():  
    print("Hello World!")  
  
if __name__ == "__main__":  
    stampa()
```

`__name__` è una variabile speciale di Python che viene inizializzata in base a come eseguo lo script. Contiene il valore `__main__` nel caso in cui si manda in esecuzione lo script. Conterrà il nome del file (in cui lo script è memorizzato) se importo lo script.

Tipi di Dato

Tipi di Dato	Nome	Descrizione	Esempio
Intero	int	Intero di dimensione arbitraria	-42, 0, 120, 99999
Reale	float	Numero a virgola mobile	3.14, -0.5, 0.009
Stringhe	str	Usata per rappresentare testo	"" , "stefano", "mele"
Booleano	bool	Per valori veri o falsi	True, False
Liste	list	Una sequenza mutabile di oggetti	[], [1,2,3], ["Hello", "World"], [1, "Hello", 2]
Tuple	tuple	Una sequenza immutabile di oggetti	(), (1,2,3), ("Hello", "World")
Dizionari	dict	Una struttura che associa chiavi a valori (mappe tra oggetti)	{}, {"nome":"Ezio", "cognome":"Auditore"}

Operatori base

Somma tra
due numeri
interi

```
int_1 = 1
int_2 = 2
print(int_1 + int_2)

# output: 3
```

Concatenazione
tra due stringhe

```
str_1 = "Hello"
str_2 = "World"
print(str_1 + " " + str_2)

# output: "Hello World"
```

Operatore	Descrizione	Esempio
+	addizione	$10 + 12 \rightarrow 22$
-	sottrazione	$5 - 1 \rightarrow 4$
*	moltiplicazione	$10 * 12 \rightarrow 120$
/	divisione	$9 / 4 \rightarrow 2.25$
//	divisione intera	$9 // 4 \rightarrow 2$
%	modulo (resto della divisione)	$9 \% 4 \rightarrow 1$
**	potenza	$2 ** 3 \rightarrow 8$

Operazioni su Liste

Le liste sono rappresentate come un *insieme di elementi racchiusi tra parentesi quadre*. Questi possono essere di un tipo qualsiasi e una lista può contenere anche elementi di tipologie differenti. Ad esempio:

```
a = [1, "hello", 3.4]
```

Così *come le stringhe*, è possibile accedere a un singolo valore della lista come segue:

```
a = [1, "hello", 3.4]
```

```
b = "ciao"
```

```
a[0]      # Output: 1
```

```
b[1]      # Output: "i"
```

```
a[-1]     # Output: [3.4]
```

```
b[-2]     # Output: "a"
```

Operazioni su Liste

Come per le *stringhe*, è possibile concatenare diverse liste tramite l'operatore '+'

Es:

- $[1,2] + [3] \Rightarrow [1,2,3]$

- $["a"] + ["b","c"] \Rightarrow ["a", "b", "c"]$

Slicing

L'operatore di **slicing** si applica a variabili che contengono *stringhe*, *liste* o *tuple* e restituisce una *sottosequenza di quella originale*

- **var[a:b]** restituisce la sottosequenza (stringa o lista) contenuta nella variabile **var**, avente indici da **a** a **b-1**
- **var[a:b:s]** restituisce la sottosequenza di indici **a**, **a+s**, **a+2s**, ..., fino all'elemento di indice **b** escluso

Es, considerando **x = ["a", "b", "c", "d"]**:

```
x[1:3]    # output ["b", "c"]
```

```
x[0:4:2]  # output ["a", "c"]
```

```
x[: -1]   # output ["a", "b", "c"]
```

```
y = x[:]  # copia della lista x
```

Metodi sulle liste

metodo	descrizione	esempio
append(x)	Aggiunge un elemento (x) alla fine della lista.	<pre>a = ["bee", "moth"] print(a) a.append("ant")</pre>
extend(iterable)	Estende la lista aggiungendo tutti gli elementi presenti in iterable. Con questo metodo è possibile unire due liste.	<pre>a = ["bee", "moth"] print(a) a.extend(["ant", "fly"])</pre>
insert(i, x)	Inserisce un elemento in una determinata posizione specificata da i.	<pre>a = ["bee", "moth"] a.insert(0, "ant") ["ant", "bee", "moth"] a.insert(2, "fly") ["ant", "bee", "fly", "moth"]</pre>

Metodi sulle liste

metodo	descrizione	esempio
<code>remove(x)</code>	Rimuove il primo elemento con valore x dalla lista. Ritorna un errore se la lista non contiene tale elemento.	<pre>a = ["bee", "moth", "ant"] a.remove("moth")</pre>
<code>pop([i])</code>	Rimuove l'elemento alla posizione specificata e lo ritorna. Se i non viene specificato, <code>pop()</code> rimuove e ritorna l'ultimo elemento della lista.	<pre>a = ["bee", "moth", "ant"] print(a) a.pop() print(a) a = ["bee", "moth", "ant"] print(a) a.pop(1) print(a)</pre>
<code>clear()</code>	Rimuove tutti gli elementi dalla lista.	<pre>a = ["bee", "ant", "moth", "ant"] a.clear()</pre>

Metodi sulle liste

metodo	descrizione	esempio
<code>index(x[, start[, end]])</code>	Ritorna la posizione del primo elemento della lista che ha valore x. Se la lista non contiene x viene lanciato un <code>ValueError</code> . L'argomento opzionale <i>end</i> viene utilizzato per limitare la ricerca in una porzione della lista.	<pre>a = ["bee", "ant", "moth", "ant"] print(a.index("ant")) print(a.index("ant", 2))</pre>
<code>count(x)</code>	Ritorna il numero di occorrenze di x all'interno della lista.	<pre>a = ["bee", "ant", "moth", "ant"] a.count("bee")</pre>
<code>sort(key=None, reverse=False)</code>	Ordina gli elementi della lista. Gli argomenti possono essere utilizzati per customizzare l'operazione di ordinamento. key Specifica una funzione di un argomento che viene utilizzata per estrarre una chiave di comparazione. Il default è None (gli elementi vengono confrontati direttamente). reverse Boolean. Se impostato a True, ogni confronto viene fatto al contrario (ovvero, la lista viene ordinata al contrario).	<pre>mylist = [1, 5, 2, 7, -1, 8, 2, 10] mylist.sort() print(mylist)</pre>

Operazioni su Dizionari

Dizionari: insiemi di elementi ordinati costituiti da coppie **chiave/valore**; ogni valore è identificato dalla chiave associata:

- **chiave (key):** stringa o numero (unici nel dizionario)
- **valore (value):** un valore qualsiasi
- sintassi: {**k_1**: **v_1**, ... , **k_n**: **v_n**}

- La coppia **chiave: valore** è detta "item"

```
import datetime

mydict = {
    "evento": "CyberChallenge",
    "anno": 2025,
    "data": datetime.datetime.now(),
    "docenti": ["Giacomo", "Lucia", "Matteo"],
    1: ["a", "b", "c"], # La chiave è un numero (int)
    3.1415: "pi greco", # La chiave è un numero (float)
}

print(mydict)
```

Operazioni su Dizionari

```
diz = {  
    "a": "ciao",  
    "b": "come",  
    "c": "stai"  
}
```

Ottenere un valore partendo dalla chiave

- `diz.get("a")`
> "ciao"
- `diz.get(1)`
(non restituisce niente)
- `diz["a"]`
> "ciao"
- `diz.pop("a")`
Restituisce ed elimina l'elemento "a": "ciao"

Inserire un nuovo elemento nel dizionario

- `diz["d"] = "tutto"`
- `diz["e"] = "bene"`

Ottenere tutte le chiavi

- `diz.keys()`

Ottenere tutti i valori

- `diz.values()`

Ottenere tutti gli elementi

- `diz.items()`

Funzioni e metodi predefiniti

		Built-in Functions		
<code>abs()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>all()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>any()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>ascii()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bin()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>bool()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	
<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>	

Operatori Logici e di Confronto

Operatori di Confronto

Operatore	Descrizione	Esempio
<code>==</code>	Uguale a	$9 == 9 \rightarrow \text{True}$ $3 == 5 \rightarrow \text{False}$
<code>!=</code>	Diverso da	$3 != 5 \rightarrow \text{True}$ $3 != 3 \rightarrow \text{False}$
<code><</code>	Minore di	$4 < 8 \rightarrow \text{True}$ $4 < 2 \rightarrow \text{False}$
<code><=</code>	Minore o uguale a	$4 <= 4 \rightarrow \text{True}$ $4 <= 2 \rightarrow \text{False}$
<code>></code>	Maggiore di	$2 > 1 \rightarrow \text{True}$ $1 > 3 \rightarrow \text{False}$
<code>>=</code>	Maggiore o uguale a	$5 >= 1 \rightarrow \text{True}$ $5 >= 7 \rightarrow \text{False}$

Operatori Logici (Booleani)

Nome	Operatore	Descrizione
Congiunzione	<code>A and B</code>	Restituisce True se entrambi gli operandi sono veri, altrimenti False
Disgiunzione	<code>A or B</code>	Ritorna True se almeno uno degli operandi è vero, altrimenti False
Negazione	<code>not A</code>	Ritorna False se l'operando è vero, True se invece è falso
in	<code>x in y</code>	Ritorna True se l'elemento x è parte della lista, stringa, set o tupla y; se non lo è ritorna False

Strutture per il controllo del flusso

1. **Condizionale** - if/elif/else
2. **Ciclo** - for ... in ...
3. **Ciclo** - while

Strutture per il controllo del flusso - if

```
if condizione1:  
    operazione1()  
elif condizione2:  
    operazione2()  
elif condizione3:  
    operazione3()  
else:  
    operazione4()
```

Strutture per il controllo del flusso - if

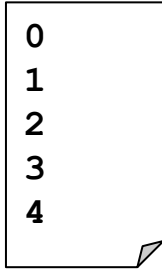
```
mylist = ["a", 1, "b", 2]
if "b" in mylist:
    print("Trovato 'b'")
else:
    print("'b' non trovato")

> Trovato 'b'
```

Funziona anche per
insiemi (set), stringhe,
tuple...

Strutture per il controllo del flusso - for

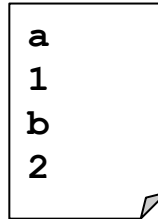
```
for i in range(0,5):  
    print(i)
```



0
1
2
3
4

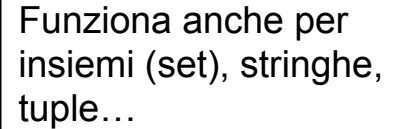
Iterazione "classica"

```
mylist = ["a", 1, "b", 2]  
for i in mylist:  
    print(i)
```



a
1
b
2

Iterazione "for each"



Funziona anche per
insiemi (set), stringhe,
tuple...

Strutture per il controllo del flusso - for (dict)

Per i dizionari, scrivere `for i in dizionario` itera sulle chiavi.

```
diz = {"a":1,"b":2,"c":3}
for i in diz:
    print(i)
```

```
>a
>b
>c
```

Se vogliamo ciclare sui soli valori:

```
diz = {"a":1,"b":2,"c":3}
for i in diz.values():
    print(i)
```

```
>1
>2
>3
```

Strutture per il controllo del flusso - for (dict)

Alternativa migliore: metodo `items()`

```
diz = {"a":1,"b":2,"c":3}
for i in diz.items():
    print(i)
```

```
>('a', 1)
>('b', 2)
>('c', 3)
```

Possiamo anche fare:

```
for key,val in diz.items():
    print(f"Chiave: {key}, Valore: {val}")
```

Chiave: a, Valore: 1

Chiave: b, Valore: 2

Chiave: c, Valore: 3

Strutture per il controllo del flusso - while

```
while condizione:  
    operazione()
```

Caso d'uso tipico: **ciclo infinito**

```
while True:  
    operazione()
```

Interazione con file

Quando si programma, capita spesso di dover leggere e/o scrivere file. In altri linguaggi, ad esempio il C, è necessario prima **aprire** il file, poi si possono fare determinate operazioni, e infine tale file va **chiuso**.


In Python esiste la sintassi per effettuare tali operazioni (apertura/azioni/chiusura), ma il linguaggio mette anche a disposizione un costrutto speciale che si occupa **automaticamente** di **aprire** il file e **chiuderlo** alla fine del blocco di codice.

Vedremo solo la lettura e scrittura di file testuali.

Interazione con file

```
with open("miofile.txt") as f:  
    linea = f.readline()  
    while linea:  
        print(f"Ho letto: {linea}")  
        linea = f.readline()
```

Finché la linea letta
non è vuota...



Interazione con file - costruito più comodo

```
with open("miofile.txt") as f:  
    for linea in f:  
        print(f"Ho letto: {linea}")
```

Itera automaticamente su tutte le righe del file di testo

Interazione con file


Nota: i colori indicano la "utilità" dei vari modi

Come in altri linguaggi, si può specificare il **modo** con cui i file vengono aperti:

Read Only ('r')	Open text file for reading. (Metodo di default, se non se ne specifica nessuno)
Read and Write ('r+')	Open the file for reading and writing.
Write Only ('w')	Open the file for writing. For the existing files, the data is truncated and over-written.
Write and Read ('w+')	Open the file for reading and writing. For an existing file, data is truncated and over-written.
Append Only ('a')	Open the file for writing. The file is created if it does not exist. The <u>handle</u> is positioned <u>at the end of the file</u> . The data being written will be inserted at the end, after the existing data.
Append and Read ('a+')	Open the file for reading and writing. The file is created if it does not exist. The <u>handle</u> is positioned <u>at the end of the file</u> . The data being written will be inserted at the end, after the existing data.

Interazione con file

```
with open("miofile.txt", "rt") as f:  
    for linea in f:  
        print(f"Ho letto: {linea}")
```



Nota: 't' indica che leggiamo un file di testo (di default se non scriviamo nulla). Per i file binari scriveremmo 'b'.

Definire una funzione

```
def myfunc(a):  
    print(f"L'argomento 'a' passato è: {a}")
```

```
myfunc(5)
```

```
>L'argomento 'a' passato è: 5
```

```
myfunc("ciao")
```

```
>L'argomento 'a' passato è: ciao
```

Definire una funzione

```
def myfunc(a, b=1, c=0):  
    print(f"Il risultato è {a*b+c}")
```

```
myfunc(5)
```

```
>Il risultato è 5
```

```
myfunc(5,3)
```

```
>Il risultato è 15
```

```
myfunc(5,3,1)
```

```
>Il risultato è 16
```

Definire una funzione

```
def myfunc(a, b=1, c=0):  
    print(f"Il risultato è {a*b+c}")
```

Parametri opzionali (o di default), vanno messi dopo i parametri obbligatori

```
myfunc(5)
```

```
>Il risultato è 5
```

```
myfunc(5,3)
```

```
>Il risultato è 15
```

```
myfunc(5,3,1)
```

```
>Il risultato è 16
```

Definire una funzione

```
def myfunc(a, b, c):  
    return a*b+c
```

```
res = myfunc(5,3,1)  
print(res)  
>16
```

```
def myfunc(a, b, c):  
    pass
```

La funzione non fa niente (ad es. utile se la voglio solo definire, per poi implementarla in seguito)

Accenno alle classi

Python, come linguaggio a oggetti, consente di definire delle classi:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

```
p1 = Person("John", 36)
```

```
print(p1.name) # John
```

```
print(p1.age)  # 36
```

Accenno alle classi

Python, come linguaggio a oggetti, ci permette di definire classi:

Definizione della classe
con relativo nome (Person)

```
class Person:
```

```
    def __init__(self, name, age):  
        self.name = name  
        self.age = age
```

Definizione del metodo `__init__`, che è il costruttore (il primo parametro deve essere sempre `self`). `self` = "this" in Java, ovvero "questa istanza della classe"

```
p1 = Person("John", 36)
```

Istanziatura della classe (creo un oggetto di tipo Person)

```
print(p1.name) # John  
print(p1.age)  # 36
```

Accesso agli attributi dell'oggetto

Esercizi

Esercizi - 1

- È dato un file `dati.txt` caratterizzato dal seguente formato rappresentante delle coppie nome/età. Per esempio:

```
Carlo 23
Giacomo 25
Lucia 23
Matteo 15
```

Si legga tutto il file creando un dizionario le cui chiavi rappresentano le età. A ciascuna età viene associata una **lista** con i nomi di persone che hanno quell'età. Una volta realizzato il dizionario, lo si stampi a video.

Esercizi - 2

- Scrivere un codice Python che legga un *file di testo* e crei un dizionario che tenga traccia della frequenza di comparsa di ogni carattere letto dal file.

Ad esempio, se il testo è il seguente:

```
Nel mezzo del cammin di nostra vita  
mi ritrovai per una selva oscura,  
ché la diritta via era smarrita.
```

Lo script Python dovrebbe stampare:

```
freq_dict: {'N': 1, 'e': 6, 'l': 4, ' ': 16, 'm': 5, 'z': 2, 'o': 4, 'd':  
3, 'c': 3, 'a': 13, 'i': 10, 'n': 3, 's': 4, 't': 6, 'r': 9, 'v': 4, 'p':  
1, 'u': 2, ',': 1, 'h': 1, 'é': 1, '.': 1}
```

Esercizi - 3

- Il file `voti.txt` contiene diverse righe, ognuna delle quali è così composta:

`<nome_alunno> <voto1> <voto2> ... <votoN>`

Realizzare uno script Python che legga riga per riga il file e salvi all'interno di una lista di tuple, dove ogni tupla è composta dal nome dello studente e un'ulteriore lista contenente i suoi relativi voti. Una volta realizzato questo, per ogni studente registrato, calcolare e stampare la media dei suoi voti.

Ad esempio, se il file `voti.txt` è:

```
Matteo 30 30 30 30 29
Giacomo 31 31 31 31
Alessandro 18 18 25 30
Lucia 29 31 30 29 31 30
Carlo 19 26 24 30 22 18 28
```

L'output dovrebbe essere:

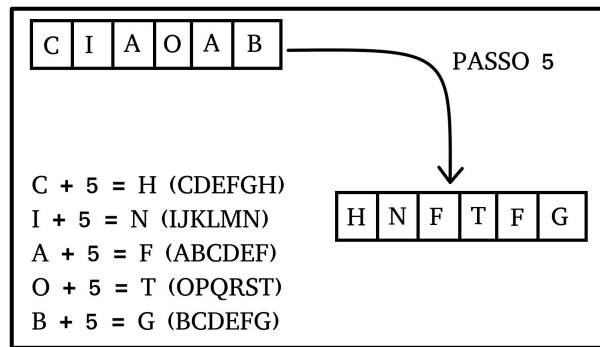
```
Studente: Matteo, Media voti: 29.8
Studente: Giacomo, Media voti: 31.0
Studente: Alessandro, Media voti: 22.75
Studente: Lucia, Media voti: 30.0
Studente: Carlo, Media voti: 23.85...
```

Esercizio Bonus

- Realizzare uno script che implementi il Cifrario di Cesare.

Esso è un cifrario a *sostituzione monoalfabetica*, in cui ogni lettera del testo in chiaro è sostituita, nel testo cifrato, dalla lettera che si trova un certo numero di posizioni dopo nell'alfabeto. Realizzare uno script Python, il quale mediante le funzioni `chr()` e `ord()` implementi le seguenti due funzioni:

- `cifra_messaggio()`**: questa funzione deve richiedere all'utente di inserire il percorso del file che si vuole cifrare e di inserire il valore del passo da usare per effettuare la cifratura. Una volta ricevuti questi due valori, la funzione deve cifrare il file indicato e salvarlo in un nuovo file
- `decifra_messaggio()`**: funzione che, come dice il nome, deve poter decifrare il file cifrato precedentemente. Anche in questo caso deve richiedere all'utente il passo da usare per la decifratura e il percorso del file da decifrare



Librerie

Librerie

La community Python rende disponibile una mole di **librerie** con le quali è possibile svolgere una vastissima serie di compiti, quali la creazione di grafici, driver per dispositivi, computer vision, machine/deep learning...

Per gestire le librerie nel proprio sistema si usa il tool **pip**.

Pip

Pip (Pip Installs Packages) è un tool che permette di cercare, scaricare e installare librerie Python che si trovano sul Python Package Index (PyPI). Consente inoltre di gestire le librerie già scaricate, permettendo di aggiornarle o rimuoverle.

Per **installare una nuova libreria** con pip, la sintassi è la seguente:

```
$ pip install <nome_libreria>
```

Pip

Per **aggiornare** una libreria:

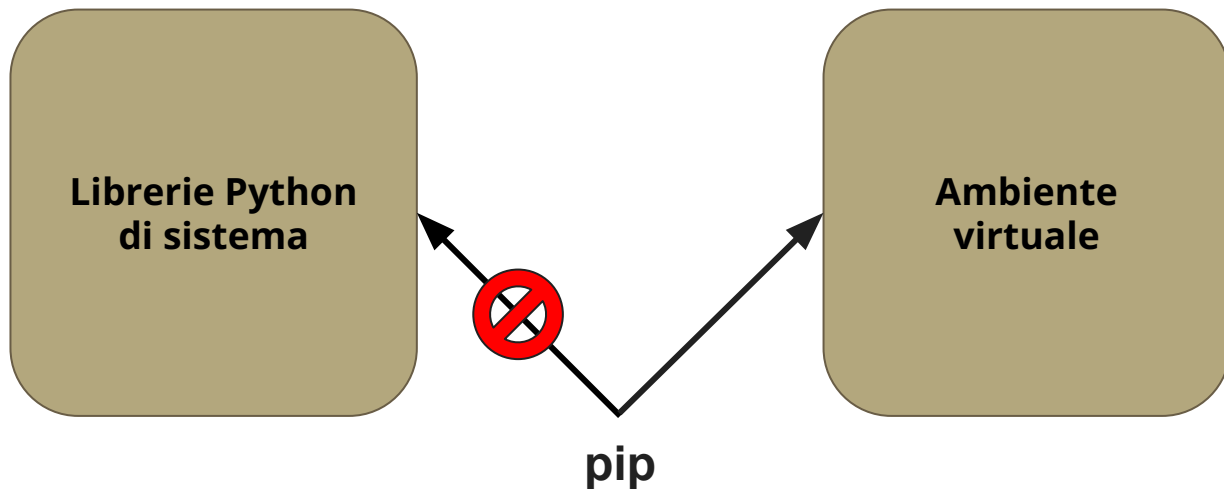
```
$ pip install --upgrade <nome_libreria>
```

Per **disinstallare** una libreria:

```
$ pip uninstall <nome_libreria>
```


Ambiente virtuale (venv)

Ad oggi, le nuove versioni di Python **impediscono** di installare nuove librerie a livello di sistema per evitare problemi con il funzionamento del sistema operativo. Si consiglia quindi di creare un **ambiente virtuale** (virtual environment, **venv**).



Ambiente virtuale (venv)

Infatti, se proviamo a installare una libreria senza ambiente virtuale...

```
gb@Exodia in ~  
λ pip install matplotlib  
error: externally-managed-environment
```

× This environment is externally managed

↳ To install Python packages system-wide, try apt install python3-xyz, where xyz is the package you are trying to install.

If you wish to install a non-Debian-packaged Python package, create a virtual environment using `python3 -m venv path/to/venv`. Then use `path/to/venv/bin/python` and `path/to/venv/bin/pip`. Make sure you have `python3-full` installed.

Ambiente virtuale (venv)

In Python ci sono più modi di creare ambienti isolati in cui installare le librerie.

Il più immediato è usare **venv**, un modulo di Python. Installarlo con:

```
sudo apt install python3-venv
```

Ora si può creare un ambiente virtuale chiamato "venv" nella cartella corrente con:

```
python -m venv venv
```




Nome dell'ambiente
virtuale

Ambiente virtuale (venv)

Una volta creato, l'ambiente virtuale va **attivato**. Attivare l'ambiente virtuale significa che con i comandi `python` e `pip` andremo a utilizzare gli strumenti dell'ambiente virtuale, e non quelli di sistema.

```
source venv/bin/activate
```

```
(venv) gb@Exodia:~/Scrivania/cc25/lab-python-introducttivo/esercizi_librerie$
```



Nome dell'ambiente virtuale,
ci conferma che è attivo

Se ora installiamo un pacchetto con `pip`, verrà installato **all'interno dell'ambiente virtuale**, e non fra le librerie di sistema.

Ambiente virtuale (venv)

```
(venv) gb@Exodia:~/Scrivania/cc25/lab-python-introductivo/esercizi_librerie$ pip install matplotlib
Collecting matplotlib
  Using cached matplotlib-3.10.0-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (11 kB)
Collecting contourpy<=1.0.1 (from matplotlib)
  Using cached contourpy-1.3.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (5.4 kB)
Using cached six-1.17.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: six, pyparsing, pillow, packaging, numpy, kiwisolver, fonttools, cyclor, python-dateutil,
  contourpy, matplotlib
Successfully installed contourpy-1.3.1 cyclor-0.12.1 fonttools-4.55.8 kiwisolver-1.4.8 matplotlib-3.10.0 numpy-2.2.2 pac
kaging-24.2 pillow-11.1.0 pyparsing-3.2.1 python-dateutil-2.9.0.post0 six-1.17.0
```

Se ora installiamo un pacchetto con pip, verrà installato **all'interno dell'ambiente virtuale**, e non fra le librerie di sistema.

Se chiudiamo il terminale, l'ambiente virtuale dovrà essere **riattivato** con lo stesso comando di prima.

Altri modi per creare ambienti virtuali

Conda

virtualenv

pyenv

poetry

uv

eccetera eccetera...

Come usare le librerie

Le librerie vanno **importate** per essere usate. Due sintassi possibili:

```
import <libreria>
```

oppure

```
from <libreria> import <funz1/classe1>, <funz2/classe2>, ...
```

Questi si mettono tipicamente all'**inizio dello script**.

Come usare le librerie

Ad esempio, per usare la funzione `floor` della libreria `math`:

```
import math
```

```
a = math.floor(3.1415)
```

```
print(a)
```



⌊3.1415⌋

>3

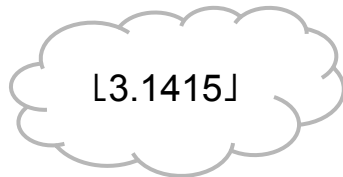
Come usare le librerie

Ad esempio, per usare la funzione `floor` della libreria `math` (alternativa):

```
from math import floor
```

```
a = floor(3.1415)
```

```
print(a)
```



>3

Come usare le librerie

Oppure, per importare tutte le funzioni della libreria `math`:

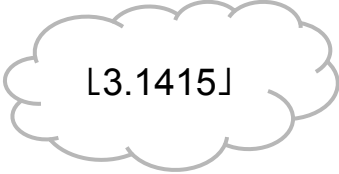
```
from math import *
```

```
a = floor(pi)
```

```
print(a)
```

```
>3
```

`math` mette a disposizione il valore di `pi` greco (`math.pi`, oppure `pi` se si è usato `from math import ...`)



3.1415

Librerie standard utili: sys e os

Due librerie standard molto utili sono **sys** e **os**.

<code>sys.argv</code>	Lista degli argomenti da riga di comando. Il numero dei parametri si ottiene con <code>len(sys.argv)</code>
<code>sys.exit()</code>	Chiude il programma
<code>os.getcwd()</code>	Restituisce la cartella in cui è in esecuzione lo script
<code>os.chdir(path)</code>	Modifica la cartella corrente in <i>path</i> (come il comando <code>cd</code> in UNIX)
<code>os.chmod(path, mode)</code> / <code>os.chown(path, mode)</code>	Come <code>chmod</code> e <code>chown</code> in UNIX
<code>os.mkdir(path)</code> / <code>os.makedirs(path)</code>	Crea (solo) la cartella specificata dal <i>path</i> / e anche tutte le ulteriori cartelle "padre" necessarie se non esistono
<code>os.listdir(path)</code>	Restituisce una lista contenente i file e le cartelle all'interno di <i>path</i> (molto simile a <code>ls</code> in UNIX)
<code>os.remove(path)</code> / <code>os.rmdir(path)</code>	Elimina il file / la cartella specificati da <i>path</i>
<code>os.rename(src, dst)</code>	Rinomina il file o la cartella <i>src</i> in <i>dst</i>

Librerie utili per le CTF

1. **pwntools** - CTF framework and exploit development library
 - a. Inviare payload a servizi Web, interazione con processi, ...
2. **requests** - Inviare richieste HTTP
3. **pycryptodome** - Per la crittografia
 - a. Metodi per cifrare e decifrare dati (es. AES), creazione e verifica delle signature, creazione di digest (es. SHA-256), creazione e gestione di chiavi pubbliche e private...

```
$ pip install pwntools requests pycryptodome
```

Pwntools

Pwntools può essere usato per connettersi e interagire con servizi Web remoti e processi, creare un "listener", eccetera.

Come da documentazione, il primo passaggio consiste nell'importare l'intera suite di funzionalità:

```
from pwn import *
```

Pwntools

Esempio di utilizzo: creazione di un **"echo server"** e di un **client** (entrambi locali). L'echo server si mette in ascolto di connessioni, riceve un messaggio e lo reinvia al client, ad esempio scrivendo "Mi hai inviato: <messaggio>".

Si noti che lo scambio di informazioni avviene sotto forma di byte, quindi utilizzeremo i metodi `encode()` e `decode()` per trasformare le stringhe in/da byte.

Volendo specificare che questo encoding/decoding venga fatto tramite encoding UTF-8, sarà sufficiente usare i metodi `encode('utf-8')` e `decode('utf-8')`.

Pwntools - Echo server

```
from pwn import *

l = listen(54321)

conn = l.wait_for_connection()

while True:

    line = conn.recv().decode('utf-8')

    if line:

        print(f"Messaggio ricevuto: {line}")

        tosend = f"Mi hai inviato: {line}".encode('utf-8')

        conn.send(tosend)
```

Pwntools - Client

```
from pwn import *

conn = remote("localhost", 54321)

while True:

    i = input("Inserisci il messaggio da inviare: ")
    tosend = i.encode('utf-8')
    conn.send(tosend)

    line = conn.recv().decode('utf-8')
    print(f"Ricevuto dal server:\n{line}")
```


Pwntools - Interazione con processi

Esempio di utilizzo: Interazione con un processo locale (adattato dalla documentazione).

```
from pwn import *  
  
sh = process("/bin/sh")  
  
sh.sendline(b"sleep 3; echo hello world;")  
  
out1 = sh.recvline(timeout=1)  
  
out5 = sh.recvline(timeout=5)  
  
print(f"Ricevuto dopo 1 secondo: {out1}")  
  
print(f"Ricevuto dopo altri 5 secondi: {out5}")  
  
sh.close()
```

Nota: si può aprire una shell interattiva con `sh.interactive()`

Pwntools

E molto di più:

- Connessioni ssh
- Operazioni sui bit
- Assemblamento di codice ASM e disassemblamento di binari
- ...

Pwntools

Dubbi su come usare i vari metodi?

Fate riferimento alla documentazione!!!

<https://docs.pwntools.com/en/latest/>

<https://github.com/Gallopsled/pwntools-tutorial#readme>

Requests

La libreria requests consente di effettuare richieste HTTP in maniera semplice.

Ad esempio, per recuperare la pagina <http://www.example.com> (richiesta GET):

```
import requests

res = requests.get("http://www.example.com")

print(f"Stato HTTP: {res.status_code}")
print(f"Risposta:\n{res.text}")
```

Otteniamo l'intera pagina Web (HTML).

Requests

Per inviare una richiesta POST alla pagina <http://www.example.com>, inserendo nel body della richiesta "prova123":

```
import requests

res = requests.post("http://www.example.com", data="prova123")

print(f"Stato HTTP: {res.status_code}")
print(f"Risposta:\n{res.text}")
```

Requests

Metodi principali messi a disposizione da requests:

`requests.get` → HTTP GET

`requests.post` → HTTP POST

`requests.put` → HTTP PUT

`requests.delete` → HTTP DELETE

Ma anche `head`, `patch`, `options`, ...

Requests

Esempio di utilizzo (da <https://pypi.org/project/requests/>):

```
>>> import requests
>>> r = requests.get('https://httpbin.org/basic-auth/user/pass', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
'{"authenticated": true, ... '
>>> r.json()
{'authenticated': True, ...}
```

Requests

Esempio di utilizzo di richiesta POST (inserimento di un JSON in un sito online):

A dark-themed code editor window with three colored window control buttons (red, yellow, green) in the top-left corner. It contains Python code for sending a POST request with a JSON body to a specific URL and printing the response status and text.

```
data = {  
    'title': 'Python Requests',  
    'body': 'Requests are awesome',  
    'userId': 1  
}  
  
response = requests.post('https://jsonplaceholder.typicode.com/posts', data)  
  
print(response.status_code)  
  
print(response.text)
```


Requests

Il codice della slide precedente stampa quanto segue:

201 ← response.status_code

{

"title": "Python Requests",

"body": "Requests are awesome",

"userId": "1",

"id": 101

}

← response.text

Stato HTTP 201: "Created"

Nel body della risposta il sito
manda una copia di quanto ha
inserito nel proprio database

Requests

Dubbi su come usare i vari metodi?

Fate riferimento alla documentazione!!!

<https://requests.readthedocs.io/en/latest/api/>

```
requests.post(url, data=None, json=None, **kwargs)
```

[source]

Sends a POST request.

- Parameters:**
- **url** - URL for the new **Request** object.
 - **data** - (optional) Dictionary, list of tuples, bytes, or file-like object to send in the body of the **Request**.
 - **json** - (optional) json data to send in the body of the **Request**.
 - ****kwargs** - Optional arguments that **request** takes.

Returns: **Response** object

Return type: **requests.Response**

Pycryptodome

Pycryptodome fornisce numerose primitive crittografiche, ad esempio per la cifratura simmetrica (es. AES), generazione di chiavi (es. RSA), cifratura asimmetrica (es. RSA), funzioni di hash (es. SHA-256)...

Vedremo alcuni esempi semplici di utilizzo.

Pycryptodome - Cifrare dati con AES

```
from Crypto.Cipher import AES

from Crypto.Random import get_random_bytes

data = b'testo molto segreto'

key = get_random_bytes(16)

cipher = AES.new(key, AES.MODE_EAX)

ciphertext, tag = cipher.encrypt_and_digest(data)

file_out = open("encrypted.bin", "wb")

[ file_out.write(x) for x in (cipher.nonce, tag, ciphertext) ]

file_out.close()
```

Pycryptodome - Cifrare dati con AES

```
from Crypto.Cipher import AES
```

b"stringa" converte
la stringa in bytes

```
from Crypto.Random import get_random_bytes
```

```
data = b'testo molto segreto'
```

EAX è una modalità di cifratura
autenticata (authenticated
encryption) - vedrete nelle lezioni

```
key = get_random_bytes(16)
```

```
cipher = AES.new(key, AES.MODE_EAX)
```

```
ciphertext, tag = cipher.encrypt_and_digest(data)
```

In una sola funzione effettua:

- **encrypt()** - cifratura
- **digest()** - calcolo del Message Authentication Code (MAC) per verifica dell'autenticità e integrità

```
file_out = open("encrypted.bin", "wb")
```

```
[ file_out.write(x) for x in (cipher.nonce, tag, ciphertext) ]
```

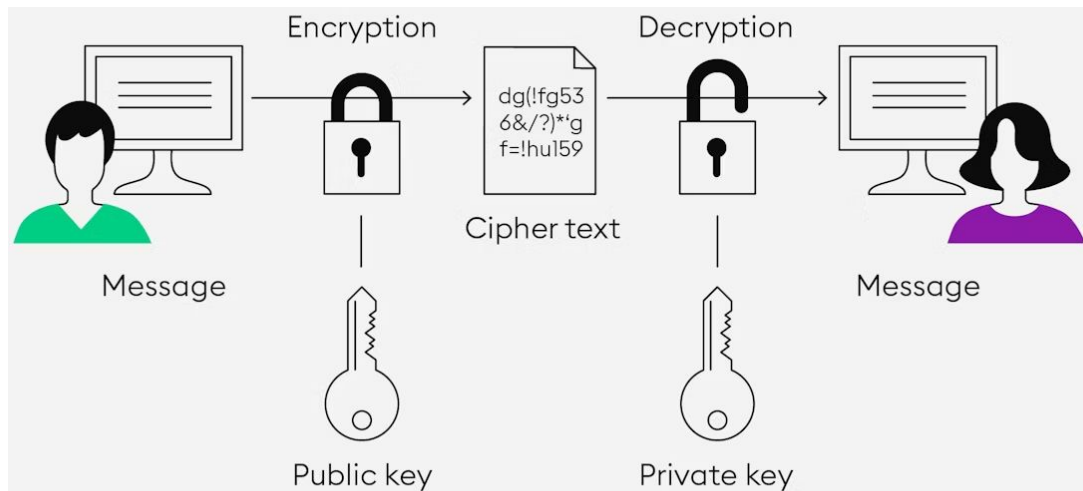
```
file_out.close()
```

Scrive un
codice
univoco, il tag
MAC e il
cifrato su un
file binario

Pycryptodome - Cifrare dati con RSA

Esempio semplice: Alice vuole inviare un messaggio a Bob. Per fare ciò Alice deve cifrare il messaggio usando la chiave **pubblica** di Bob; successivamente Bob potrà decifrare il messaggio ricevuto usando la sua chiave **privata**.

Bob è l'unico che può decifrare il messaggio perché è l'unico in possesso della sua chiave privata.



Pycryptodome - Cifrare dati con RSA

Primo step: generazione della coppia di chiavi di Bob

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

messaggio = b"Messaggio segreto!"

# Generazione chiave pubblica e privata di Bob
key_bob = RSA.generate(2048)
private_key_bob = key_bob.export_key()
public_key_bob = key_bob.publickey().export_key()
```

Algoritmo di
cifratura a chiave
asimmetrica

Pycryptodome - Cifrare dati con RSA

Continua: cifratura del messaggio usando la chiave pubblica di Bob, Bob lo decifra con la sua chiave privata

```
# Alice cifra il messaggio per Bob, usando la chiave pubblica di Bob
cipher = PKCS1_OAEP.new(key_bob)
msg_cifrato = cipher.encrypt(messaggio)
print(f"Messaggio cifrato per Bob:\n{msg_cifrato}")

# Bob decifra il messaggio, usando la propria chiave privata
msg_decifrato = cipher.decrypt(msg_cifrato)
print(f"Messaggio decifrato da Bob:\n{msg_decifrato}")
```


Pycryptodome - Cifrare dati con RSA

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

messaggio = b"Messaggio segreto!"

key_bob = RSA.generate(2048)
private_key_bob = key_bob.export_key()
public_key_bob = key_bob.publickey().export_key()

cipher = PKCS1_OAEP.new(key_bob)
msg_cifrato = cipher.encrypt(messaggio)
print(f"Messaggio cifrato per Bob:\n{msg_cifrato}")

msg_decifrato = cipher.decrypt(msg_cifrato)
print(f"Messaggio decifrato da Bob:\n{msg_decifrato}")
```

Notate che da nessuna parte nel codice compaiono riferimenti ad Alice? Cifrando in questo modo, non c'è nessuna informazione su chi ha cifrato/inviato il messaggio!

Pycryptodome

Per poter verificare l'autenticità dei messaggi si possono usare in maniera combinata le **funzioni di hash** (ad es. SHA-256) e il cosiddetto **processo di firma** (che usa le chiavi pubbliche e private, ad es. quelle RSA).

Le **funzioni di hash** trasformano deterministicamente e univocamente un input in una nuova stringa di lunghezza fissa.

`ciao a tutti` → `SHA256` →

`876f9f754020c6dce2ea9bb517d52d6252a51d7a2406eb012b5217040757c8c2`

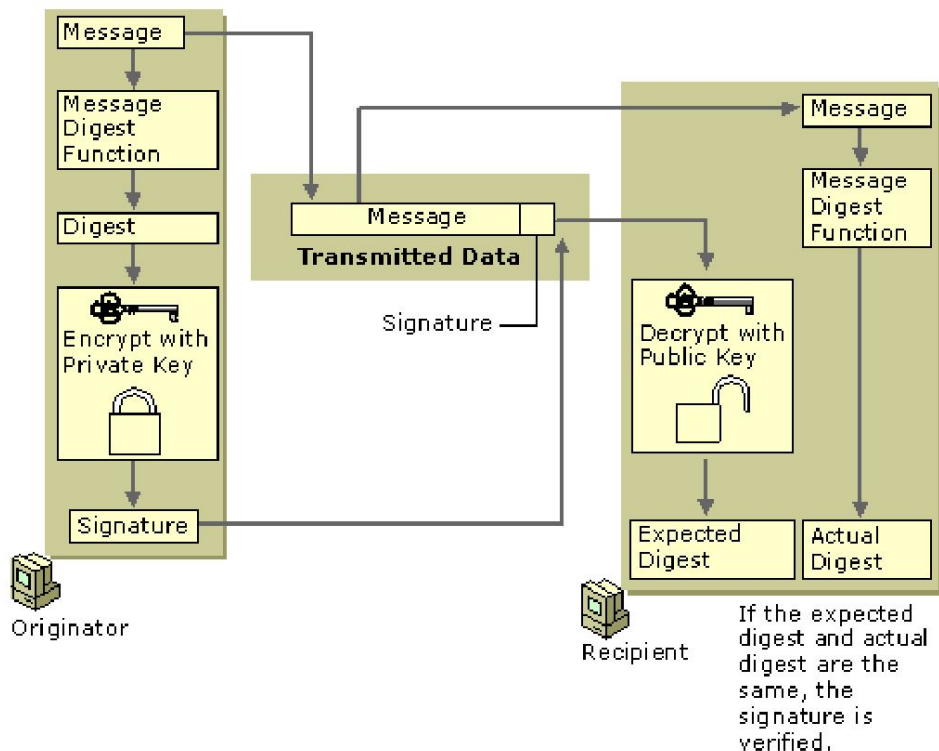
`come state` → `SHA256` →

`cfe1c9e8cd7e540adac3bbb3cb24394dd1b2e5f5a60ff6bd9a2cc6d8b0dae15c`

Pycryptodome

Questo è il processo di **verifica della firma**.

Noi vogliamo trasmettere un messaggio cifrato, quindi oltre a verificare la firma (e quindi l'autenticità del messaggio) dovremo **decifrare** il messaggio.



Pycryptodome

1. Alice cifra il messaggio per Bob utilizzando la chiave pubblica di Bob
2. Alice calcola l'hash SHA-256 del messaggio cifrato
3. Alice firma l'hash SHA-256 con la propria chiave privata, ottenendo una signature
4. Alice trasmette a Bob il messaggio cifrato e la signature

1. Bob calcola l'hash SHA-256 del messaggio cifrato
2. Bob verifica la firma (usa la chiave pubblica di Alice per "decifrare" la sua signature, se corrisponde con l'hash SHA-256 che ha appena calcolato allora la firma di Alice è valida)
3. Bob decifra il messaggio cifrato con la sua chiave privata

In questo modo Bob ha verificato che il messaggio cifrato che ha ricevuto gli è stato effettivamente inviato da Alice

Pycryptodome

Come possiamo eseguire questo processo con Pycryptodome?

Innanzitutto importiamo le librerie e definiamo il messaggio.

```
from Crypto.PublicKey import RSA
```

```
from Crypto.Cipher import PKCS1_OAEP
```

```
from Crypto.Signature import PKCS1_v1_5
```

```
from Crypto.Hash import SHA256
```

```
messaggio = "Questo è un messaggio segreto!".encode('utf-8')
```

Pycryptodome

Creiamo le coppie di chiavi di Alice e Bob (come visto prima).

```
key_alice = RSA.generate(2048)
```

```
private_key_alice = key_alice.export_key()
```

```
public_key_alice = key_alice.publickey().export_key()
```

```
key_bob = RSA.generate(2048)
```

```
private_key_bob = key_bob.export_key()
```

```
public_key_bob = key_bob.publickey().export_key()
```

Pycryptodome

```
# Alice cifra il messaggio per Bob, usando la chiave PUBBLICA di Bob
```

```
cipher_alice = PKCS1_OAEP.new(key_bob)
```

```
msg_cifrato = cipher_alice.encrypt(messaggio)
```

```
# Alice fa l'hash SHA256 del messaggio cifrato
```

```
hash_msg_cifrato = SHA256.new(msg_cifrato)
```

```
# Alice firma l'hash del messaggio cifrato usando la propria chiave PRIVATA
```

```
msg_firmato = PKCS1_v1_5.new(key_alice).sign(hash_msg_cifrato)
```

```
print(f"Messaggio cifrato per Bob:\n{msg_cifrato.hex()}\n")
```

```
print(f"Hash SHA-256 messaggio cifrato per Bob:\n{hash_msg_cifrato.digest().hex()}\n")
```

```
print(f"Messaggio firmato per Bob:\n{msg_firmato.hex()}\n")
```

Pycryptodome

```
# Alice trasmette a Bob il messaggio cifrato (msg_cifrato), e il messaggio firmato (msg_firmato)

# Bob calcola l'hash del messaggio cifrato

hash_msg_cifrato_bob = SHA256.new(msg_cifrato)

print(f"Hash SHA-256 calcolato da Bob:\n{hash_msg_cifrato_bob.digest().hex()}\n")

# Bob verifica la firma del messaggio firmato inviatogli da Alice usando la chiave pubblica di Alice

try:

    pub_key_alice = RSA.import_key(public_key_alice)

    PKCS1_v1_5.new(pub_key_alice).verify(hash_msg_cifrato_bob, msg_firmato)

    print("La firma del messaggio è valida, quindi è verificato che il mittente è Alice.\n")

except (ValueError, TypeError):

    print("La firma non è valida, non posso verificare che il mittente sia Alice.\n")
```


Pycryptodome

```
# Infine, Bob decifra il messaggio, usando la propria chiave PRIVATA

# (a questo punto Bob è sicuro che il messaggio gli sia stato inviato da Alice).

cipher_bob = PKCS1_OAEP.new(key_bob)

msg_decifrato = cipher_bob.decrypt(msg_cifrato)

print(f"Messaggio decifrato da Bob:\n{msg_decifrato.decode('utf-8')}")
```

Nella prossima slide il codice completo senza commenti.

Pycryptodome

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA256
messaggio = "Questo è un messaggio segreto!".encode('utf-8')
key_alice = RSA.generate(2048)
private_key_alice = key_alice.export_key()
public_key_alice = key_alice.publickey().export_key()
key_bob = RSA.generate(2048)
private_key_bob = key_bob.export_key()
public_key_bob = key_bob.publickey().export_key()
cipher_alice = PKCS1_OAEP.new(key_bob)
msg_cifrato = cipher_alice.encrypt(messaggio)
hash_msg_cifrato = SHA256.new(msg_cifrato)
msg_firmato = PKCS1_v1_5.new(key_alice).sign(hash_msg_cifrato)
print(f"Messaggio cifrato per Bob:\n{msg_cifrato.hex()}\n")
print(f"Hash SHA-256 messaggio cifrato per Bob:\n{hash_msg_cifrato.digest().hex()}\n")
print(f"Messaggio firmato per Bob:\n{msg_firmato.hex()}\n")
hash_msg_cifrato_bob = SHA256.new(msg_cifrato)
print(f"Hash SHA-256 calcolato da Bob:\n{hash_msg_cifrato_bob.digest().hex()}\n")
try:
    pub_key_alice = RSA.import_key(public_key_alice)
    PKCS1_v1_5.new(pub_key_alice).verify(hash_msg_cifrato_bob, msg_firmato)
    print("La firma del messaggio è valida, quindi è verificato che il mittente è Alice.\n")
except (ValueError, TypeError):
    print("La firma non è valida, non posso verificare che il mittente sia Alice.\n")
cipher_bob = PKCS1_OAEP.new(key_bob)
msg_decifrato = cipher_bob.decrypt(msg_cifrato)
print(f"Messaggio decifrato da Bob:\n{msg_decifrato.decode('utf-8')}")
```

Esercizi

Esercizi - Pwntools

Partendo da quanto visto negli esempi precedenti:

1. Creare uno script Python che, usando Pwntools, rimane in attesa di connessioni da client che invieranno comandi (ad esempio: ls, whoami, echo "ciao"...). Una volta ricevuto un comando, lo invia a un processo shell (quale /bin/bash) e invia l'output del comando al client che lo ha richiesto.
2. Creare il client che chiede all'utente di inserire da tastiera un comando, lo invia al server creato al punto 1 e stampa a video l'output ricevuto.

Esercizi - Requests

```
data = {  
    "userId": 12345,  
    "title": "Esercizio Requests",  
    "completed": True  
}
```

Partendo da quanto visto negli esempi precedenti:

1. Creare uno script Python che utilizza la libreria requests per:
 - Recuperare la pagina www.example.com (richiesta GET)
 - Scrivere il contenuto della pagina in un file HTML
2. Il sito jsonplaceholder.typicode.com simula una ReST API che utilizza JSON come formato delle informazioni. Contattando la pagina jsonplaceholder.typicode.com/todos è possibile vedere una lista di *todo* ("cose da fare"), inoltre è possibile aggiungerli, eliminarli o modificarli.
 - Tramite una richiesta POST all'URL jsonplaceholder.typicode.com/todos, inserire un nuovo *todo*, ad esempio quello mostrato in alto a destra in questa slide
 - Il sito fornisce come risposta una copia di ciò che ha inserito nel suo database. Recuperare l'ID del todo inserito dalla risposta e inviare una richiesta DELETE a jsonplaceholder.typicode.com/todos/<ID_TODO> per eliminarlo

Esercizi - Pycryptodome

Partendo da quanto visto negli esempi precedenti, realizzare questi passaggi:

1. Generazione di chiavi RSA di Alice e Bob
 2. Alice calcola e firma lo SHA-256 di un messaggio (in chiaro, senza averlo cifrato) con la sua chiave privata, ottenendo una signature (firma)
 3. Alice cifra con AES in modalità CBC la firma (per questo passaggio usare una chiave qualsiasi di 16 caratteri)
 4. Alice "trasmette" a Bob il messaggio in chiaro e la firma cifrata (come negli esempi, in realtà non simuliamo questa trasmissione, ma riutilizziamo le stesse variabili)
-
1. Bob decifra la firma usando la chiave simmetrica AES definita in precedenza
 2. Bob calcola lo SHA-256 del messaggio in chiaro ricevuto da Alice
 3. Bob verifica la firma ricevuta da Alice



CYBER
CHALLENGE.IT



Università
degli Studi
di Ferrara

Grazie a tutti e a presto!

Giacomo Bettini - giacomo.bettini@unife.it

Lucia Ferrari - lucia03.ferrari@unife.it

Matteo Brina - matteo.brina@unife.it