03/02/2023

Introduzione

- Docente: Alessio Bechini
- Prende le presenze (ma solo per statistica non ai fini del titolo)

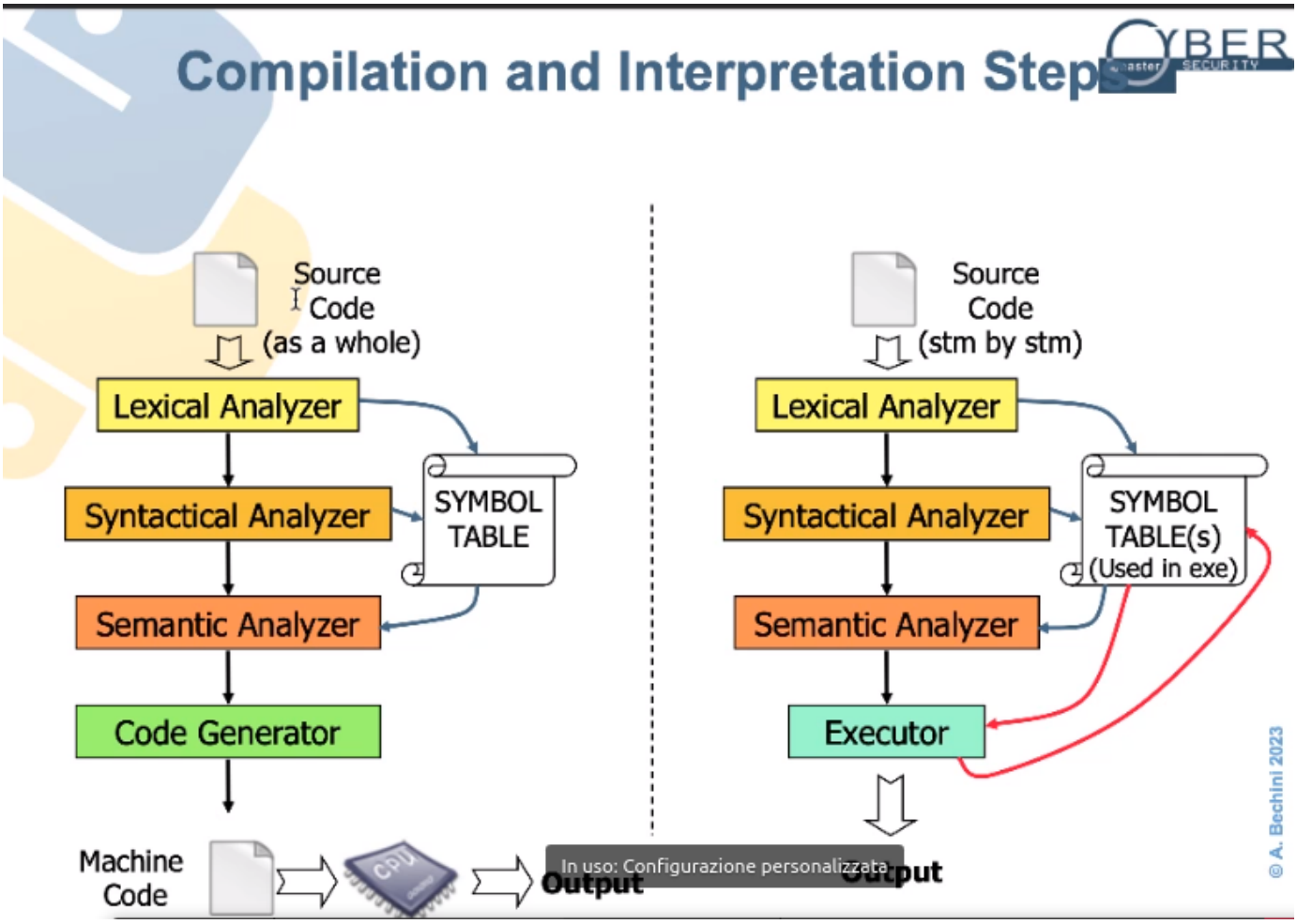
Compilation vs Interpretation

Compilation: trasformazione del programma di alto livello in un linguaggio di basso livello ovvero in istruzioni che la CPU è in grado di eseguire tramite l'utilizzo di compilatori. Per CPU diverse la compilazione deve essere diversa. Se voglio cambiare qualcosa a livello di codice devo ricompilare tutto.

Pro: prestazioni elevate
Contro bassa flessibilità di utilizzo a supporto della compilazione

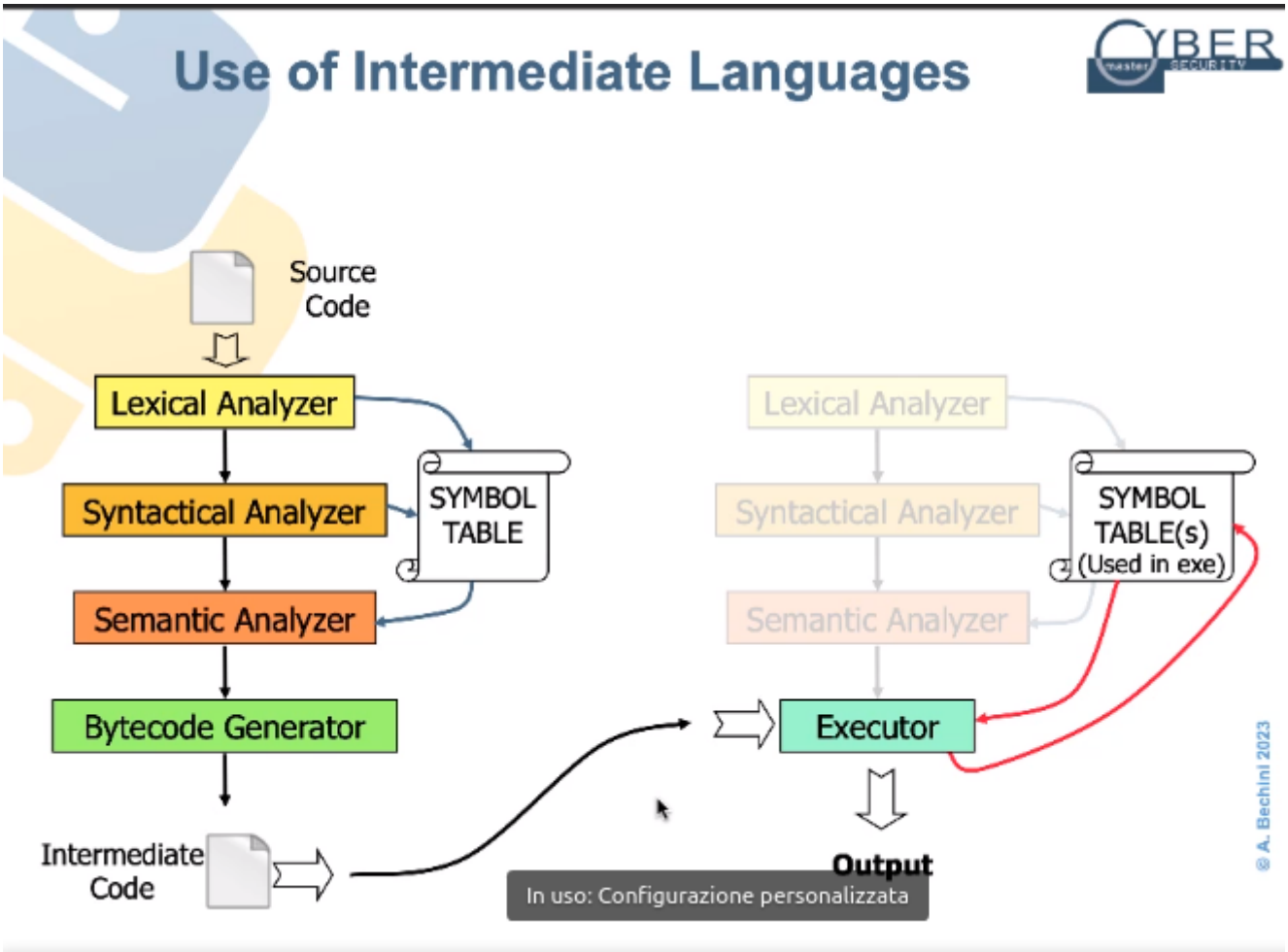
Interpretation: Per colmare questo gap si è pensato di sviluppare un programma in grado di leggere un programma scritto in alto livello ed eseguirlo in termini di istruzioni da dare alla CPU. L'esecuzione del programma stesso sfrutta un intermedio ovvero l'interprete. Le prestazioni saranno ovviamente più basse.

Pro: debugging più semplice e flessibilità maggiore
Contro: prestazioni più basse



All'interno dell'interprete posso avere un codice sorgente ma viene analizzato comando per comando ed eseguito comando per comando. Anche in questo caso devo tenere traccia di simboli usati nel programma (ad esempio le variabili). La differenza fondamentale sta nell'utilizzo della tabella dei simboli perchè viene utilizzata una volta soltanto ai fini della generazione del codice nel caso compilato mentre nel caso interpretato dovrà contenere anche tutte le informazioni per supportare l'esecuzione del programma e quindi è a carico dell'ultimo modulo (Executor) ad esempio andrà a leggere dalla tabella dei simboli in quale punto del programma ci troviamo quindi i valori memorizzati fino a quel punto e avrà anche un aggiornamento del contenuto della tabella dei simboli. Inoltre le strutture dati a supporto del programma stesso possono essere molteplici.

Si è pensato di migliorare la situazione e prendere il meglio dei due. Un possibile modo di migliorare il tutto è ottenere una soluzione ibrida in cui l'interprete esegue comandi la cui complessità è intermedia tra i comandi del linguaggio di alto livello e il codice macchina. Avrà bisogno della sua tabella dei simboli che leggerà ed andrà poi a scrivere ed aggiornare. Per la sua esecuzione dovrà avere il codice intermedio che rappresenta il programma espresso non più in linguaggio di alto livello ma in liguaggio intermedio (bytecode). Per ottenerlo posso usare un approccio a compilazione che mi restituisce il bytecode che va in ingresso a un interprete che sarà un po più veloce. L'esecutore è un programma che dovrà essere fatto girare su delle CPU che possono essere di vario tipo. Ma con OS e hardware diversi posso sviluppare un esecutore che sia in grado di avere in input sempre lo stesso codice intermedio. Quindi ci permette di evitare lo svantaggio dell'approccio compilativo che era specifico per ogni CPU.



Comunque abbiamo uno step di compilazione ed uno di esecuzione. Ci sono vari modi con cui questi step possono essere portati avanti. Alcuni linguaggi adottano un organizzazione di un tipo altri di altro tipo. JAVA adotta un approccio di questo genere. L'executor è la java virtual machine JVM. L'insieme di questi blocchi fa parte di javac, compilare del bytecode.

Python adotta un approccio di questo genere con alcune differenze rispetto a JAVA in quanto la generazione del codice intermedio è fatta partire dall'interprete ovvero la generazione del codice intermedio viene fatta al momento senza che il programmatore provveda alla compilazione del programma. È il sistema stesso che si occupa di tutto del processo.
Abbiamo una tabella dei simboli sia sulla destra che sulla sinistra.

Python è un...

Linguaggio di scripting: il loro scopo è quello di controllare le applicazioni.

i linguaggi di programmazione invece hanno come scopo lo sviluppo di vere e proprie applicazioni.

Gli script in genere sono interpretati in altri casi invece vengono compilati su rappresentazioni intermedie o bytecode. E poi il bytecode a sua volta viene interpretato.

IDE

L'ide che verrà utilizzato durante il corso è Spyder.

Una distribuzione è un insieme di strumenti che comprendono sia il linguaggio stesso sia un insieme di altri tool.
Inoltre verranno visti i Jupiter Notebooks (data analysis) che sono lo strumento in cui il linguaggio viene più frequentemente usato. All'interno della distribuzione Anaconda è possibile gestire le librerie da installare e se vi è la necessità di usare linguaggi e librerie in ambienti diversi è possibile usare ambienti diversi con versioni e librerie differenti del linguaggio.

Le versioni di python a cui il corso fa riferimento sono superiori alla **3.0**. L'ultima versione precedente alla 3.0 è la **2.7**.