FONDAMENTI DI INFORMATICA

Alessandro Renda

Dipartimento di Ingegneria e Architettura, Università degli Studi di Trieste

LINGUAGGI DI PROGRAMMAZIONE

Anno Accademico 2024/2025

Prima di iniziare

- Se non diversamente specificato, le celle formattate in questo modo includono codice Python
- In generale, anche nel testo delle slide, per il codice Python è usato il font "monaco", che garantisce una spaziatura fissa dei caratteri

```
a, b = int(input()), int(input())
z = 0
w = a
while w > 0:
      z = z + b
      w = w - 1
print(z)
```

```
>>> x = 5
>>> print(x)
```

Algoritmi e programmi

- **Algoritmo**: descrizione di come si risolve un problema
  - Tipicamente, non si descrive codifica di dati, interazione con esecutore, e altri dettagli
  - Tipicamente, un algoritmo è destinato alla comunicazione tra esseri umani
- **Programma**: algoritmo scritto in modo da poter essere eseguito da un calcolatore
  - Direttamente, se si utilizza **linguaggio macchina**
  - Indirettamente, se si utilizza **linguaggio di programmazione** che richiede traduzione

Linguaggi di programmazione

- Ogni linguaggio di programmazione è caratterizzato da due componenti:
  - **Sintassi:** l'insieme delle regole che specificano come comporre istruzioni *ben formate*
  - **Semantica:** specifica il significato di ogni istruzione ben formata, e quindi la successione delle operazioni che vengono compiute allorché l'istruzione viene eseguita

Linguaggi di programmazione

```
>>> x = 5
>>> print(x)
```

Linguaggi di programmazione

```
>>> x = 5
>>> print x
```

```
SyntaxError: Missing parentheses in call to 'print'.
Did you mean print(...)?
```

Linguaggi di programmazione

```
>>> a = 10
>>> b = 2
>>> somma = a + b
>>> print(somma)
```

Linguaggi di programmazione

```
>>> a = 10
>>> b = "2"
>>> somma = a + b
>>> print(somma)
```

```
TypeError: unsupported operand type(s) for +: 'int'
and 'str'
```

Linguaggi di programmazione

- Classificazione in base al **livello di astrazione**
  - Linguaggi di basso livello
  - Linguaggi di alto livello

Linguaggi di programmazione

- **Linguaggi di basso livello**
  - I generazione: **linguaggio macchina**
    - Insieme di istruzioni del processore. Ciò che una CPU è in grado di interpretare ed eseguire direttamente
    - Le istruzioni e dati relativi al programma in esecuzione sono caricati in memoria in forma binaria
  - II generazione: **linguaggio di tipo assemblativo**
    - Istruzioni macchina sono sostituite da codici alfanumerici (codici mnemonici),
    - Mantenuta la corrispondenza univoca con istruzioni del linguaggio macchina
    - Per essere eseguito, il programma deve essere tradotto in linguaggio macchina
    - Traduzione eseguita da un componente detto *assemblatore*

Linguaggi di programmazione

- Istruzione per la somma di due operandi src1 e src2, contenuti nei registri R02 e R03, e il salvataggio del risultato in dest (R01)
  - Ogni istruzione è identificata con codice operativo
  - Gli operandi indicano gli indirizzi dove recuperare i dati su cui operare e dove copiare i risultati

Struttura istruzione	codice operativo	dest	src1	src2
Linguaggio assemblatore	add	R01	R02	R03
Linguaggio macchina	000000 00000 100000	00001	00010	00011

- Esistono CPU con *struttura fisica diversa* (ad es. da produttori diversi: Intel, AMD) ma della stessa classe (ad es. *x86-64* o *ARM*) e quindi tra loro *compatibili*: in grado di eseguire lo stesso insieme di istruzioni

Linguaggi di programmazione

assembly

```
section .text
    global main

main:
    mov edx, len
    mov ecx, msg
    mov ebx, 1
    mov eax, 4
    int 0x80
    mov eax, 1
    int 0x80

section .data
msg db 'Hello, World!', 0xa
len equ $ - msg
```

```
print("Hello, World!")
```

Linguaggi di programmazione

- **Linguaggi di alto livello**
  - III generazione: linguaggi imperativi e procedurali di uso generale (es. Fortran)
  - IV generazione: linguaggi per specifici ambiti applicativi (es. Matlab)
  - V generazione: linguaggi di descrizione dei problemi e orientati alla risoluzione automatica (es. SQL)
- Caratteristiche
  - Risulta più semplice, rapido, efficiente lo sviluppo di applicazioni informatiche
  - Livello di astrazione più vicino all'algoritmo che alla macchina
  - Programma come sequenza di istruzioni, dette *statement*

Linguaggi di programmazione

- Il codice sorgente (source code) è la forma leggibile dall'uomo (human-readable) di un programma per computer
- Per essere eseguito, il programma deve essere **tradotto** in linguaggio macchina

- Due approcci: **interpretazione** e **compilazione**

Interpretazione

- L'**interprete** è un programma che permette l'esecuzione di un altro programma traducendolo **istruzione per istruzione**. Il suo funzionamento consiste nei seguenti passi
  - Preleva un'istruzione dalla memoria
  - La decodifica
  - La traduce in istruzioni del linguaggio macchina
  - Manda in esecuzione tali istruzioni
  - Quindi, passa all'istruzione successiva e ripete, fino a che viene raggiunta un'istruzione di terminazione
- Non viene prodotto un *file* eseguibile

Compilazione

- Il **compilatore** è un programma che permette l'esecuzione di un altro programma traducendo **l'intero codice sorgente**.
  - Prende in ingresso l'intero codice sorgente
  - Lo traduce in un nuovo programma, detto **eseguibile**, codificato per esecuzione su CPU

Interpretazione e Compilazione

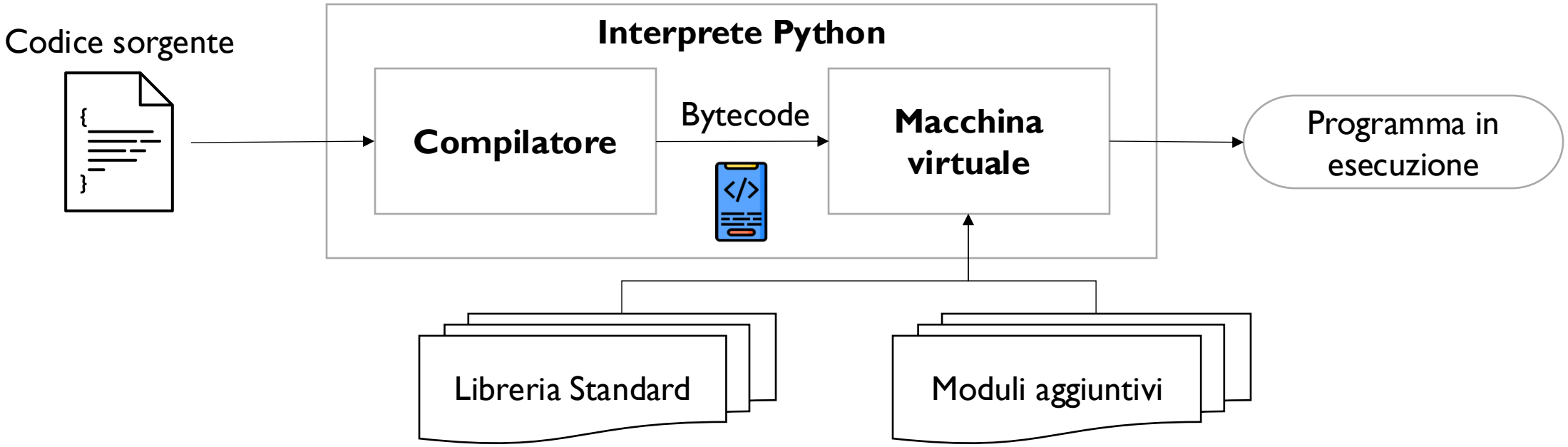
- Nella compilazione la traduzione avviene una sola volta, in una fase che precedere l'esecuzione, e in maniera globale
  - Processo di traduzione ottimizzato
  - In generale, *migliori* prestazioni nell'esecuzione
- Nella interpretazione la traduzione avviene *on-the-fly:*
  - In generale, *peggiori* prestazioni nell'esecuzione
- In caso di modifica del codice sorgente
  - La compilazione deve essere ripetuta completamente per rigenerare il codice eseguibile
  - L'interprete permette invece di eseguire immediatamente il codice aggiornato

L’interprete Python

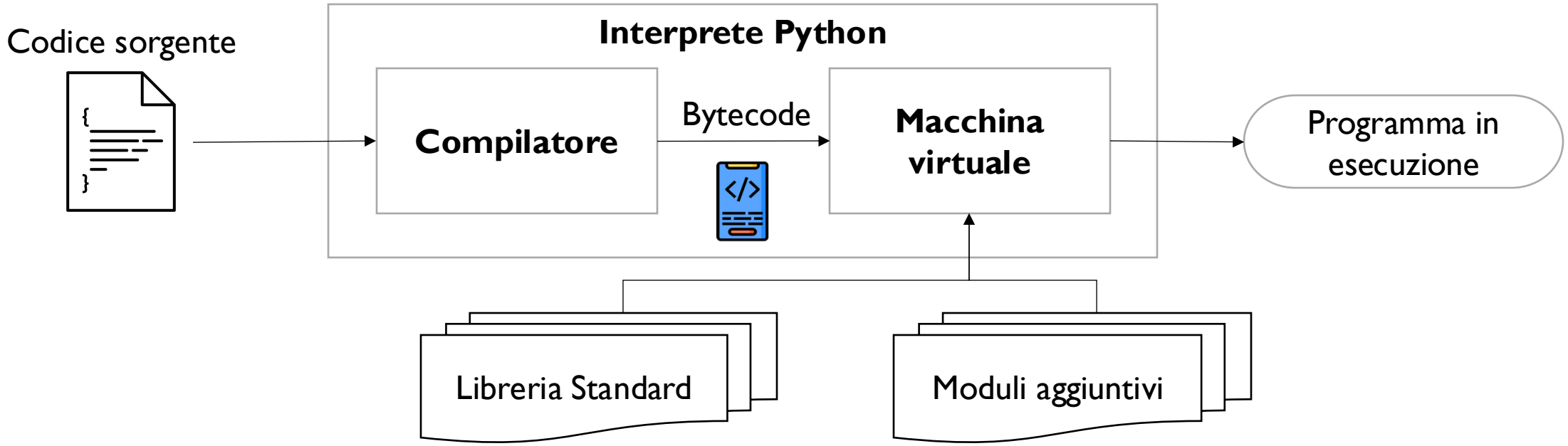
- In prima approssimazione, Python è definito *interpretato*
  - Legge il programma ed esegue le istruzioni, una dopo l’altra

- Ma non è proprio così

L'interprete Python

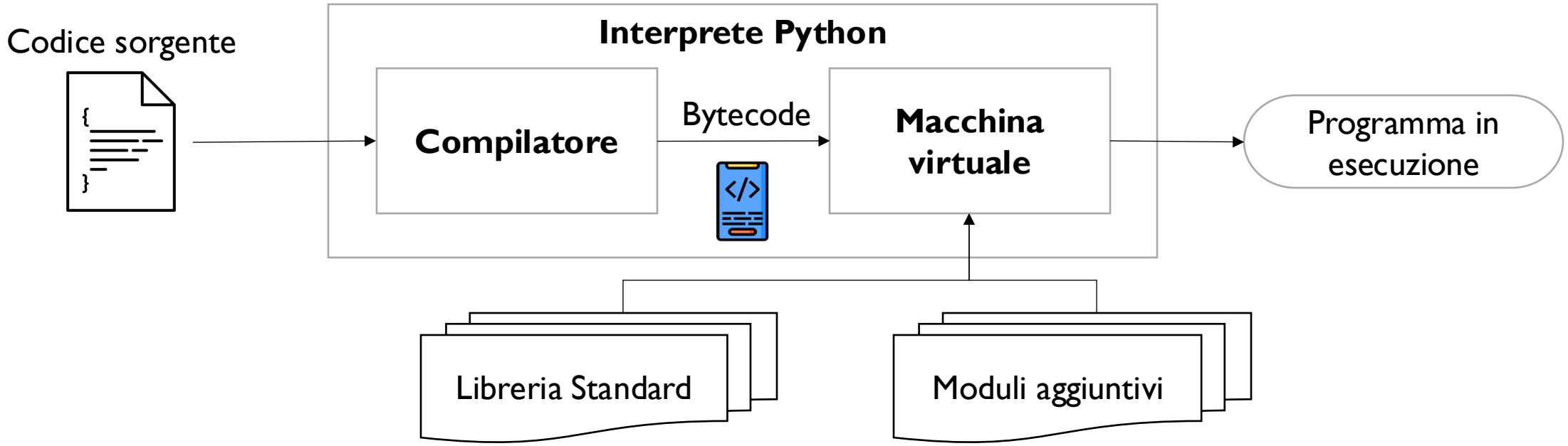


L'interprete Python



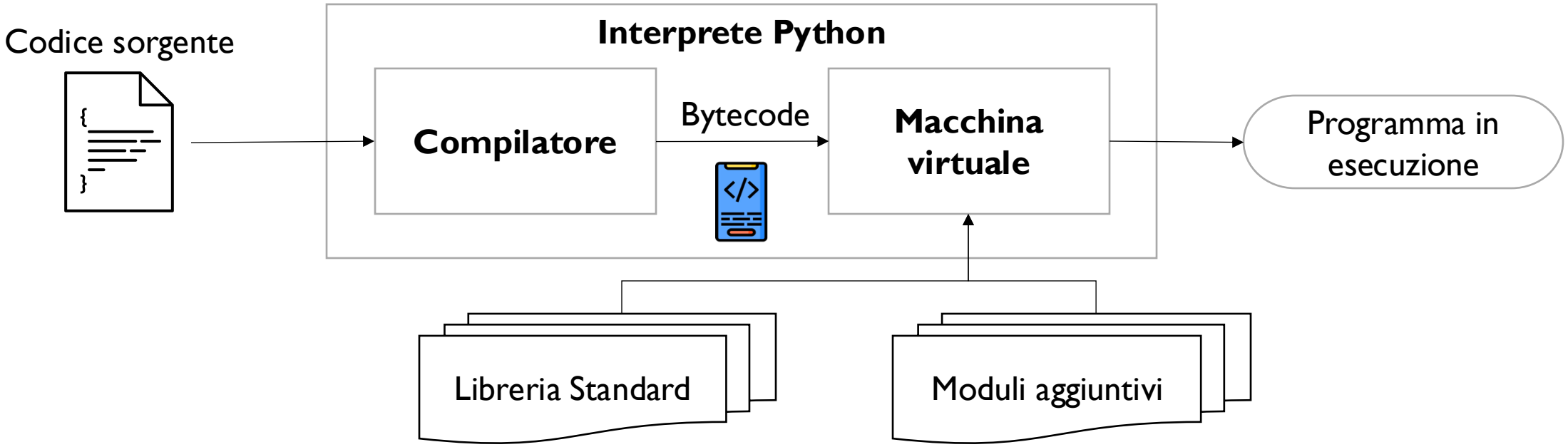
- Il codice sorgente (tipicamente un file .py) viene fornito all'**interprete Python**
- L'interprete è costituito da due componenti: compilatore e macchina virtuale

L'interprete Python



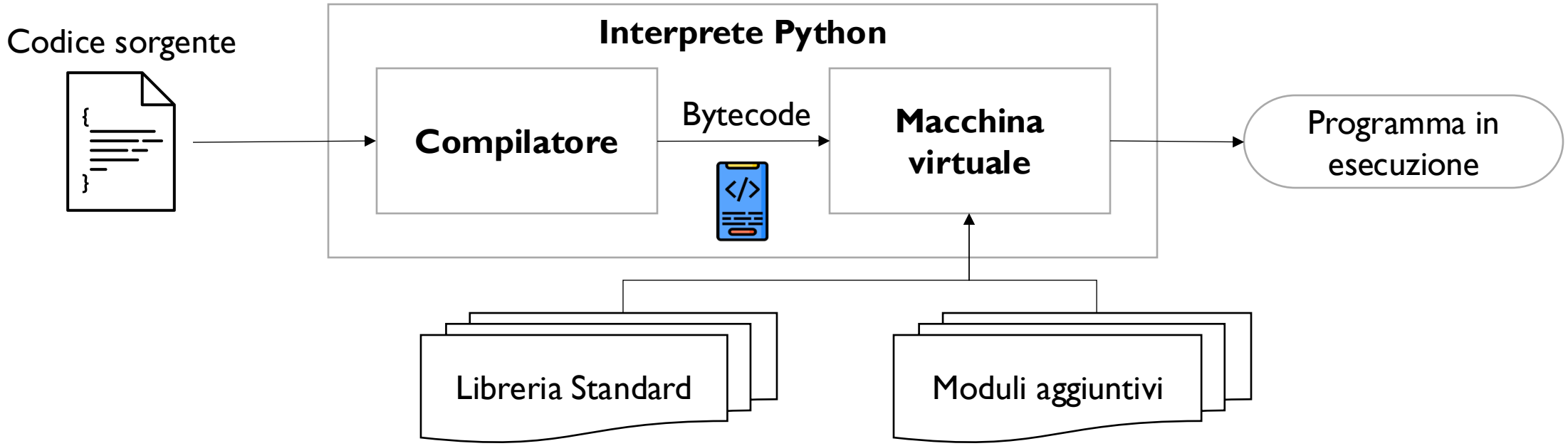
- Il **compilatore** traduce il codice sorgente in una rappresentazione intermedia e indipendente dalla piattaforma detta *bytecode*
- Il bytecode è costituito da istruzioni elementari ma non è codice macchina
- Potrebbe essere memorizzato in un file .pyc nella stessa cartella o in "__pycache__"

L'interprete Python



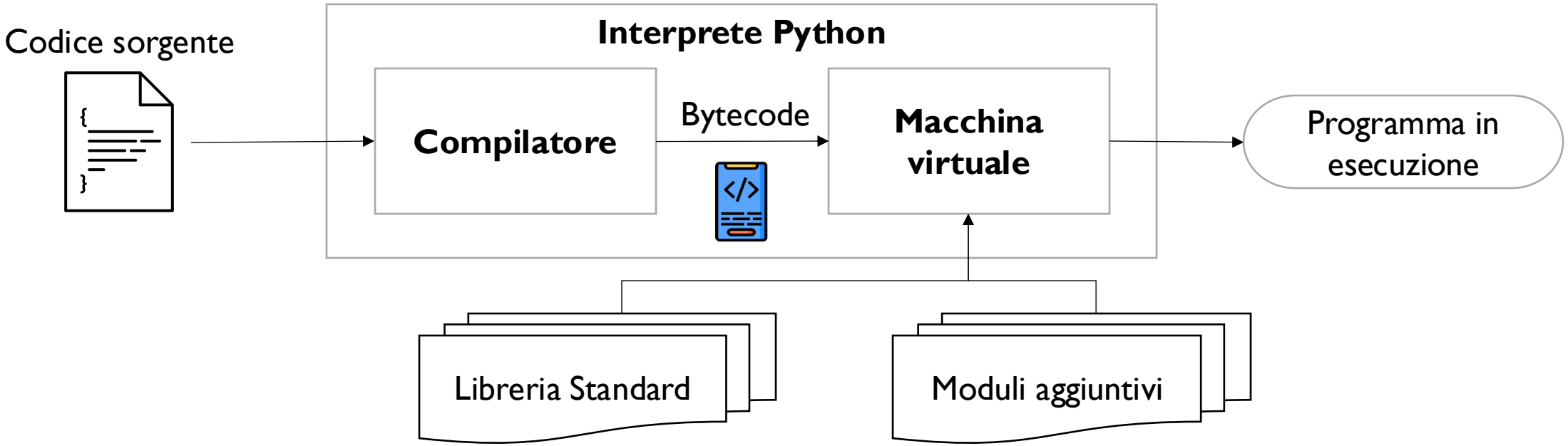
- Il bytecode viene passato ad una **macchina virtuale**: ha il compito di eseguire il programma
- La macchina virtuale interpreta il bytecode e lo esegue istruzione per istruzione

L'interprete Python



- Il codice sorgente non contiene tutte le informazioni di cui ha bisogno la VM.
  Ad esempio non contiene la implementazione della funzione print
- La VM recupera queste informazioni da altre fonti (libreria standard e moduli aggiuntivi)

L'interprete Python



- Soluzione ibrida: compromesso favorevole tra **portabilità**, **flessibilità**, **prestazioni**

Paradigmi di programmazione

- Classificazione in base al **paradigma di programmazione**
  - Imperativo-procedurale
  - Orientato agli oggetti
  - Funzionale
  - Dichiarativo

- Python è multi-paradigma:
  - Supporta diversi stili di programmazione
  - I principali sono imperativo-procedurale e orientato agli oggetti

Paradigma imperativo-procedurale

- Idea generale:
  - *Procedurale*: descrivere esplicitamente, mediante successioni di istruzioni, la procedura necessaria per risolvere il problema
  - *Imperativo*: ogni istruzione corrisponde all'invocazione di una funzionalità (*un ordine*) del calcolatore
- Molto diffuso:
  - Esempi: Fortran, Basic, Pascal, C
  - Python supporta *anche* questo paradigma

Paradigma imperativo-procedurale

```
a, b = int(input()), int(input())
z = 0
w = a
while w > 0:
    z = z + b
    w = w - 1
print(z)
```

- Tipi di istruzioni:
  - Ingresso / Uscita
  - Aritmetico-logiche
  - Di controllo

Paradigma orientato agli oggetti (*object-oriented*)

- Idea generale: imitare le interazioni tipiche del mondo reale
  - Programma come successione di interazioni tra oggetti dotati di un comportamento
  - Oggetti reagiscono a interazioni risolvendo la loro parte di problema
  - È possibile usare oggetti senza conoscere struttura e funzionamento interni
- Molto diffuso
  - Esempi: C++, Java
  - Python supporta *anche* questo paradigma
- Ideale per gestire programmi complessi
- Ne parleremo più avanti

Paradigma orientato agli oggetti (*object-oriented*)

```
class Telaio():
      def __init__(self, materiale):
            self.materiale = materiale

class Bicicletta():
      def __init__(self, marca, telaio):
      self.marca = marca
      self.telaio = telaio

      def descrivi(self):
            return f"Bici {self.marca}, telaio in {self.telaio.materiale}"

telaio_alluminio = Telaio("alluminio")
bici = Bicicletta("Bianchi", telaio_alluminio)
print(bici.descrivi())
```

Paradigma funzionale

- Idea generale: flusso del programma come una serie di valutazioni di funzioni
  - Le istruzioni vengono rappresentate come strutture dati "eseguibili"
  - Python ha alcune caratteristiche che permettono di usare uno stile funzionale (tuttavia non è *puramente* funzionale)

```
def quadrato(x):
      return x * x

numeri = [1, 2, 3, 4, 5]
quadrati = list(map(quadrato, numeri))
```

Paradigma dichiarativo

- Idea generale: il problema viene specificato formalizzando (sotto forma di *dichiarazioni*) le caratteristiche della soluzione anziché il procedimento per ottenerla
  - Viene descritto *cosa* mi aspetto dal programma, non *come* ottenerlo
  - La risoluzione del problema è delegata al traduttore
  - Difficile generalizzare, quindi limitato a specifici settori applicativi
  - Esempio: SQL

SQL

```
SELECT nome_calciatore, gol
FROM calciatori
WHERE gol > 10
ORDER BY gol DESC;
```