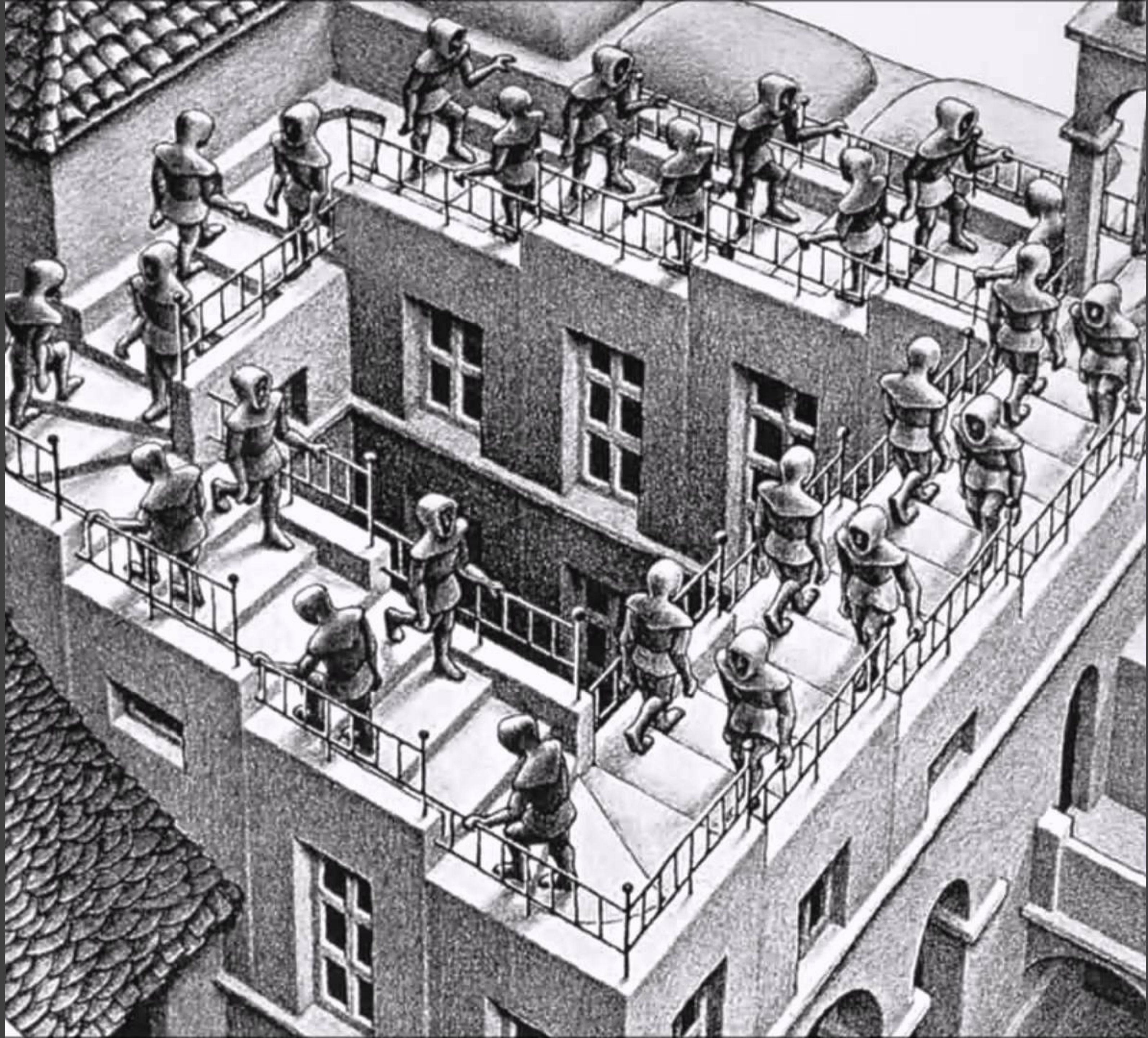
Linguaggio Python

Università Roma Tre – Dipartimento di Matematica e Fisica
M. Liverani, Laboratorio di ottimizzazione combinatoria



Il linguaggio

- Python è un linguaggio di programmazione **imperativo/procedurale a oggetti** (multi-paradigma) sviluppato nel 1991 da **Guido van Rossum**; il nome del linguaggio è un omaggio ai Monty Python
- È un **linguaggio interpretato**: non dispone quindi di un compilatore in grado di produrre una volta per tutte un eseguibile in linguaggio macchina, ma richiede la presenza del **programma interprete** per poter tradurre ed eseguire il programma in formato sorgente
- Il linguaggio è adatto a diversi scopi:
  - **didattica della programmazione**: il codice sorgente è molto chiaro e assai vicino allo pseudo-codice utilizzato per formalizzare gli algoritmi
  - **rapid prototyping**: è un linguaggio con cui lo sviluppo del codice è molto rapido e si presta bene allo sviluppo di prototipi di programmi che potranno essere poi riscritti in modo più efficiente in altri linguaggi
  - **analisi dati**: è un linguaggio che si presta bene all'elaborazione di grandi moli di dati o dati numerici enormi, senza obbligare il programmatore ad occuparsi di problematiche di allocazione della memoria
- Python è tuttavia piuttosto lento: non è adatto al calcolo numerico o allo sviluppo di videogame interattivi
- È disponibile gratuitamente (*open source*) su Internet sul sito **https://www.python.org**
- Esistono due versioni del linguaggio: la 2.x e la 3.x (oggi 3.11.2); le due versioni hanno alcune importanti differenze concettuali e sintattiche: nel corso faremo riferimento alla **versione 3.x**

L'interprete e l'ambiente di sviluppo

- Occorre installare e configurare sul proprio computer:
  - un **interprete Python** ver. 3.x (https://www.python.org/downloads/)
  - un **editor di testo** (es.: Atom, https://atom.io)
  - per lanciare l'interprete dalla linea di comando è sufficiente digitare il comando «`python`» (o «`python3`»)
  - per eseguire un programma in Python bisogna eseguire l'interprete seguito dal nome del programma da eseguire; ad esempio: «`python3 esercizio.py`»

- Il programma **pip** (o pip3) è il **Package Installer for Python** e serve per scaricare da Internet ed installare librerie di funzioni aggiuntive o per aggiornare quelle esistenti
  - ad esempio, per installare «*modulo*»: `python3 -m pip install` *modulo*

- In alternativa è possibile utilizzare un ambiente di sviluppo on-line, come:
  - **replit** (https://repl.it)
  - **jupyter** (https://jupyter.org)

Sintassi generale

- I programmi vanno scritti in un file di testo in formato ASCII standard, riportando ogni istruzione su una riga a se stante
- Il linguaggio è ***case sensitive***: attenzione alle differenze tra lettere maiuscole e minuscole
- Non si deve usare un terminatore dell'istruzione come il «`;`» utilizzato in C o in Java (è opzionale)
- Non si usano le parentesi graffe per **delimitare i blocchi di istruzioni** (es.: nelle strutture di controllo iterative o condizionali): in Python si usa l'**indentazione** delle righe di codice, che quindi non è facoltativa, ma ha uno scopo sintattico ben preciso
- Il linguaggio dispone di un'infinità di **moduli e librerie** che possono essere importati nel programma per aggiungere funzioni e classi di oggetti (con i relativi metodi)
  - le **funzioni** operano sugli argomenti specificati tra parentesi tonde (es.: «`print(a)`»)
  - per operare sugli **oggetti** si usano i **metodi**, specificati dopo un punto che segue il nome dell'oggetto e che può avere degli argomenti specificati tra parentesi (es.: «`np.sin(x)`»)
- È possibile definire nuove funzioni all'interno del programma, utilizzando la parola chiave «`def`»

Tipi di dato

- Il linguaggio prevede **numerosi tipi di dato** (`int`, `float`, `bool`, `long`, `complex`, `str`), ma le variabili non devono essere dichiarate esplicitamente; il tipo viene definito assegnandogli un valore
  - es.: «`a = 3`», «`b = 7/3`», «`c = ’Marco’`», «`flag = True`»
- Le parole chiave che identificano i tipi di dato possono essere specificate per convertire un valore da un tipo ad un altro
  - es.: «`int(2/3)`» oppure «`str(2**150)`»
- Gli operatori aritmetici sono gli stessi già definiti in altri linguaggi di programmazione: somma (`+`), sottrazione (`-`), prodotto (`*`), divisione (`/`), elevamento a potenza (`**`) e il modulo (`%`)
  - es.: «`a**n`» indica a^n, «`5%2`» restituisce il valore 1
- Come in C e in Java sono disponibili gli operatori di incremento e decremento compatti `+=` e `-=`
  - es.: «`i += 1`» equivale a «`i = i+1`»
- Rispetto a linguaggi di livello più basso (come il C) Python mette a disposizione numerose **strutture dati** definite come oggetti, con specifici metodi per operare su di esse:

  `list` (lista o array di dimensione variabile), `tuple` (lista di dimensione prefissata), `set` (insieme), `frozenset` (insieme non modificabile), `dict` (dizionario con coppie chiave-valore), `file` (file su memoria di massa)

Liste / array

- Una lista è un oggetto e può essere definita esplicitamente elencandone gli elementi tra parentesi quadre, separati da virgole, oppure mediante il «costruttore» `list()` o l'enumerazione di elementi con la funzione `range(`n`)`
  - es.:
    ```
    a = [10, 7, 4, 2, 9]
    b = list()
    c = range(10)
    ```
- Gli elementi della lista sono identificati con indici numerici; il primo elemento ha indice 0
  - es.: `a[3] = 17`
- Si opera sulle liste mediante metodi:
  - `append`: aggiunge un elemento in fondo alla lista; es.: `a.append(17)`
  - `clear`: svuota la lista; es.: `a.clear()`
  - `copy`: copia il contenuto di una lista in un'altra; es.: `b = a.copy()`
  - `count`: conta il numero di occorrenze di un determinato valore nella lista; es.: `a.count(17)`
  - `index`: restituisce l'indice della prima occorrenza di un determinato valore nella lista; es.: `x = a.index(12)`
  - `insert`: inserisce un elemento in una determinata posizione; es.: `a.insert(3, 27)`
  - `pop`: estrae un elemento da una determina posizione; es.: `x = a.pop(3)`; `y = a.pop()`
  - `remove`: rimuove dalla lista la prima occorrenza di un determinato valore; es.: `a.remove(12)`
  - `sort`: ordina la lista; es.: `a.sort()`
- La funzione `len` restituisce il numero di elementi della lista; es.: `len(a)`

Input / output

- Per eseguire operazioni di input dalla tastiera dell’utente (*standard input*) si utilizza la funzione `input`:
  - `nome = input(‘Come ti chiami?’)`
  - `n = int(input(‘Numero di elementi:’))`

- Per eseguire operazioni di output visualizzando informazioni sul video del terminale dell’utente (*standard output*) si utilizza la funzione `print`:
  - `print("Ciao!")`
  - `print("Il risultato è", x)`
  - `a = [10, 20, 30]`
  - `print("Elementi dell’array", a)`

Strutture di controllo

- Struttura condizionale:

```
if condizione:
    istruzioni
    istruzioni
else:
    istruzioni
    istruzioni
```

Es.: dati due numeri visualizza il maggiore dei due:

```
a = int(input("Primo numero:"))
b = int(input("Secondo numero:"))
if a>b:
    print("il massimo è", a)
else:
    print("il massimo è", b)
```

Condizioni ed espressioni logiche

- Le condizioni sono formulate come espressioni logiche booleane che eseguono confronti tra variabili o valori costanti
- Gli operatori di confronto sono gli stessi utilizzati in altri linguaggi di programmazione:
  `>` (maggiore), `<` (minore), `>=` (maggiore o uguale), `<=` (minore o uguale), `==` (uguale), `!=` (diverso)
- Le espressioni booleane sono costruite utilizzando anche gli operatori logici `and`, `or` e `not` ed eventualmente le parentesi, per definire diverse priorità tra gli operatori

```
a > b and not (b != 15 or c % 2 == 0)
```

- Il valore di un'espressione logica è `True` o `False`

Strutture di controllo

- Struttura di controllo iterativa con istruzione **for**:

```
for variabile in lista:
    istruzione1
    istruzione2...
```

- Struttura di controllo iterativa con istruzione **while**:

```
while condizione:
    istruzione1
    istruzione2...
```

Es.: dato $n > 0$ stampare i naturali da 1 a n:

```
n = int(input("Inserisci un intero positivo:"))
for x in range(1,n+1):
    print(x)
```

Es.: dato $k > 0$ stampare i primi 10 multipli di k:

```
k = int(input("Inserisci un intero positivo:"))
i = 1
while i<=10:
    print(k*i)
    i = i+1
```

Minimo Comune Multiplo

- Dati due numeri interi x e y calcola il minimo comune multiplo (mcm):

Algoritmo 1 MCM(x,y)

1: siano $m_x := x$ e $m_y := y$
2: se $m_x < m_y$ allora $m_x := m_x + x$ altrimenti se $m_y < m_x$ allora $m_y := m_y + y$
3: se $m_x \neq m_y$ allora vai al passo 2
4: il minimo comune multiplo tra x e y è m_x

```
x = int(input("Primo intero:"))
y = int(input("Secondo intero:"))
mx = x
my = y
while mx != my:
    if mx < my:
        mx = mx+x
    else:
        my = my+y
print("mcm(", x, ",", y, ") = ", mx)
```

Definizione di funzioni

- È possibile definire funzioni con la parola chiave **def**:

```
def funzione(parametro_1, ..., parametro_n):
    istruzione1
    istruzione2...
    return(valore)
```

Es.: dati n numeri ne visualizza la media aritmetica:

```
def media(a, n):
    s = 0
    for i in range(n):
        s = s+a[i]
    return(float(s/n))

n = int(input("Numero di elementi: "))
a = list()
for i in range(n):
    a.append(int(input(">")))
print("media:", media(a, n))
```

Funzioni ricorsive

- È possibile definire funzioni che richiamano se stesse (dette «*funzioni ricorsive*»):
- Ad esempio il fattoriale di un intero $n>0$ si può calcolare iterativamente come
  $n! = 1 \times 2 \times 3 \times \ldots \times (n-1) \times n$
  oppure ricorsivamente come
  $n! = 1$ se $n = 1$
  altrimenti $n! = n \times (n-1)!$

```
def fattoriale(n):
    if n==1: f = 1
    else: f = n * fattoriale(n-1)
    return(f)

n = int(input("Intero positivo:"))
print("fattoriale:", fattoriale(n))
```