Politecnico di Milano

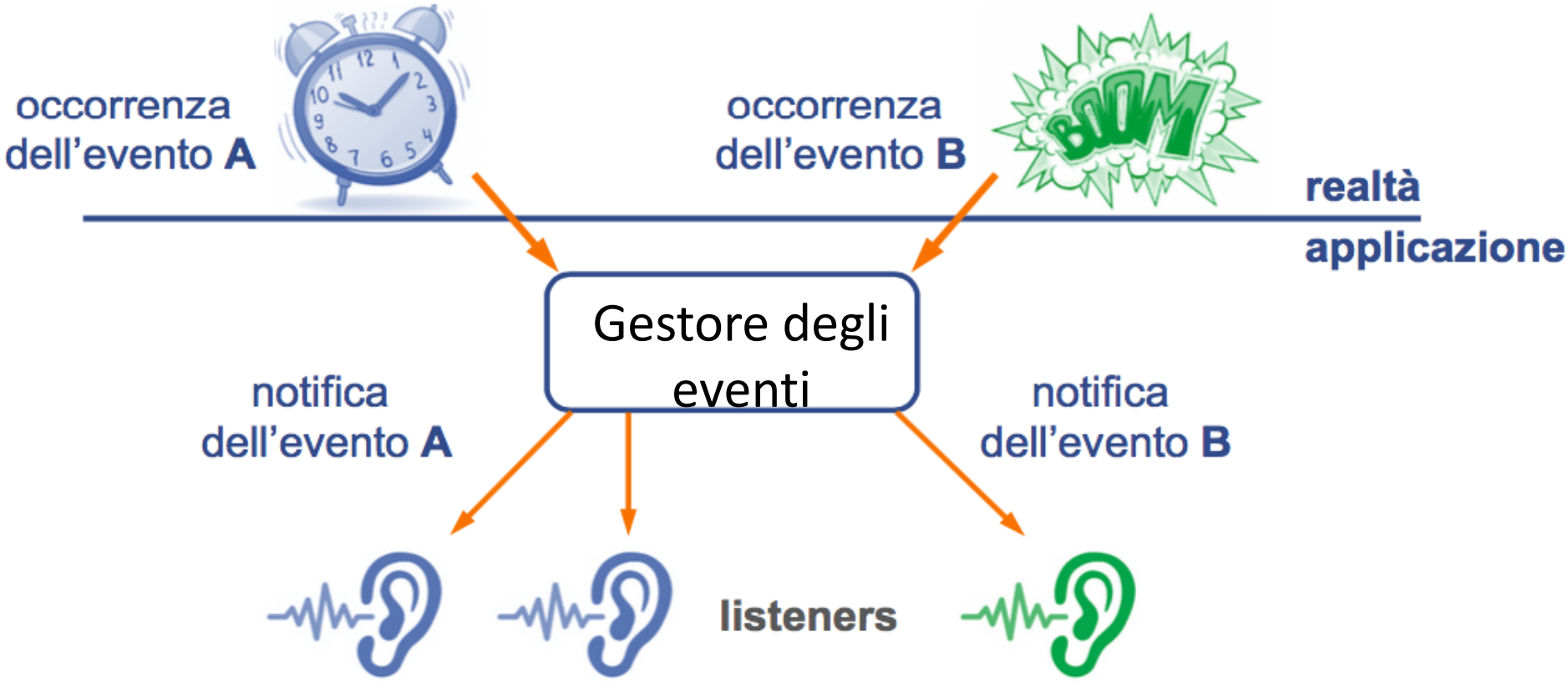
Coding in Python

10-14 Giugno 2018

Lezione 4

PROGRAMMAZIONE A EVENTI

Modello a eventi



Tipi di applicazioni a eventi

Sistemi per smart home

Sistemi per il controllo del funzionamento di un auto

Giochi

GUI = Graphical User Interfaces

...

Tipico gestore degli eventi per una GUI o per un gioco

Termina solo quando si compie un'azione particolare

```
while not quit:
    for event in eventList:
        if event.type == QUIT:
            quit = True
        if event.type == mousePressed:
            actionA()
        if event.type == keyPressed:
            actionB()
        ....
    redrawGUI()
```

Tipico gestore degli eventi per una GUI o per un gioco

Estrae, a uno a uno, tutti gli eventi che aspettano di essere considerati

```
while not quit:
    for event in eventList:
        if event.type == QUIT:
            quit = True
        if event.type == mousePressed:
            actionA()
        if event.type == keyPressed:
            actionB()
        ....
    redrawGUI()
```

Tipico gestore degli eventi per una GUI o per un gioco

```
while not quit:
    for event in eventList:
        if event.type == QUIT:
            quit = True
        if event.type == mousePressed:
            actionA()
        if event.type == keyPressed:
            actionB()
        ....
    redrawGUI()
```

In base al tipo di evento, compie un'azione specifica

Tipico gestore degli eventi per una GUI o per un gioco

```
while not quit:
    for event in eventList:
        if event.type == QUIT:
            quit = True
        if event.type == mousePressed:
            actionA()
        if event.type == keyPressed:
            actionB()
        ....
    redrawGUI()
```

Ridisegna l'interfaccia in modo da rendere visibile qualsiasi cambiamento

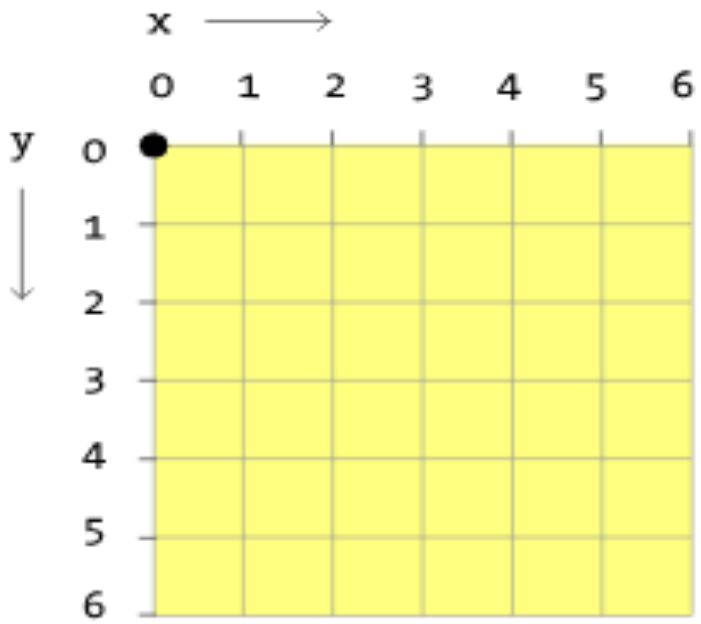
PROCESSING.PY

https://processing.org/

Cos'è Processing

- Software gratuito
- Programmi interattivi con grafica 2D e 3D
- Multipiattaforma (Linux, Windows, Mac, Android)
- Supporta diversi linguaggi di programmazione (Java, Python, Javascript, …)
- La documentazione per Python è disponibile qui https://py.processing.org/

La finestra di disegno



Disegnare in Processing

size(width, height): definisce la dimensione della finestra in pixel

background(color): definisce il colore o l'immagine utilizzata come sfondo

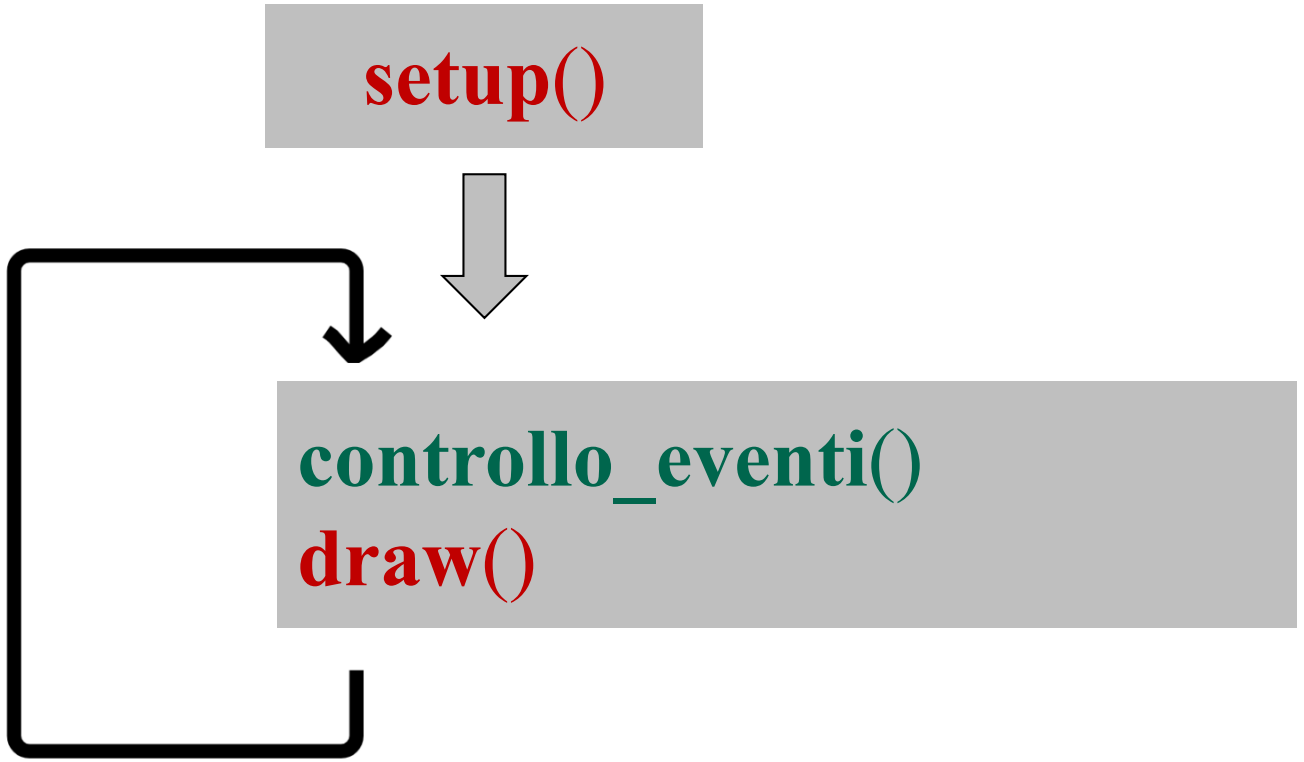
point(x, y): disegna un punto nelle coordinate x, y

line(x0, y0, x1, y1): disegna una linea dal punto (x0, y0) fino al punto (x1, y1)

rect(x, y, w, h): disegna un rettangolo a partire dal punto (x, y) con larghezza w e altezza h

ellipse(x, y, w, h) disegna un'ellisse in (x, y) con larghezza w e altezza h

Main Loop



La parte di controllo degli eventi è opzionale

setup()

Processing chiamerà la nostra funzione setup all'inizio dell'esecuzione del nostro programma.

All'interno di solito si definiscono la larghezza e l'altezza della finestra che verrà aperta, tramite la funzione size(larghezza, altezza).

Esempio:

```
def setup():
        size(800, 600)
        background(255)
        ellipse(50, 50, 100, 200)
```

draw()

Processing chiamerà la nostra funzione draw ad ogni fotogramma (60 fps = 60 volte al secondo).

Le operazioni all'interno di draw vengono quindi ripetute continuamente (è come se draw() fosse all'interno di un loop).

Per fare delle animazioni definiremo draw e all'interno disegneremo sullo schermo (come i fotogrammi di un video o un film)

draw() – esempio 1


```
def setup():
        size(800, 600)


def draw():
        ellipse(width/2, height/2, 50, 50)
```

Height e width sono variabili di sistema che immagazzinano le dimensioni della finestra definite dalla size

draw() – esempio 2

```
x = 0
def setup():
        size(800, 600)


def draw():
        global x

        ellipse(x, 100, 50, 50)
        x = x + 1
```

Necessario se voglio usare una variabile definita fuori dalla funzione (in questo caso x) all'interno della mia funzione

Mouse

Le coordinate del cursore del mouse sono sempre immagazzinate all'interno delle variabili mouseX e mouseY.

Esempio:

```
def draw():
        background(255)
        ellipse(mouseX, mouseY, 100, 100)
```

Colori

In Processing i colori sono rappresentati da dei numeri interi.

Bianco e nero

255

150

0

Colori RGB

255, 0, 0

255, 255, 0

150, 10, 50

Colori

background(colore)

fill(colore): scelta del colore di riempimento

stroke(colore): scelta del colore di linea

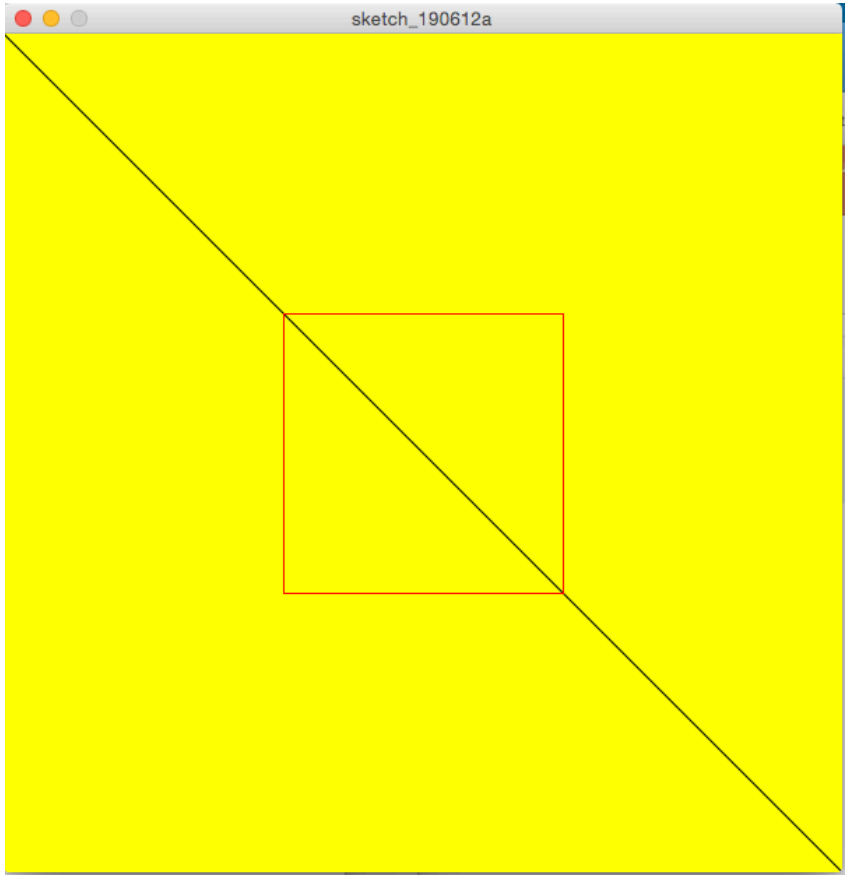
noFill(): nessun riempimento (trasparente)

noStroke(): nessuna linea (trasparente)

Colori - esempio

```
def setup():
  size(600, 600)


def draw():
  background(255, 255, 0)
  stroke(0)
  line(0, 0, 600, 600)
  stroke(255, 0, 0)
  noFill()
  rect(200, 200, 200, 200)
```



Inserimento di un testo


```
def setup():
  size(600, 600)


def draw():
  background(255, 255, 0)
  stroke(0)
  line(0, 0, 600, 600)
  stroke(255, 0, 0)
  noFill()
  rect(200, 200, 200, 200)
  textSize(26)
  fill(0)
  text('Prova testo', 300, 300)
```

Esempio: cerchi colorati


```
def setup():
    size(800, 600)
    background(255)

def draw():
    background(255)
    noStroke()

    fill(240, 10, 10)
    ellipse(mouseX, mouseY, 50, 50)

    fill(10, 240, 10)
    ellipse(width - mouseX, mouseY, 100, 100)
```

Esercizio: Palla rimbalzante

Scrivere un programma in cui una palla si muove di una certa velocità (costante) e rimbalza quando tocca i bordi della finestra.

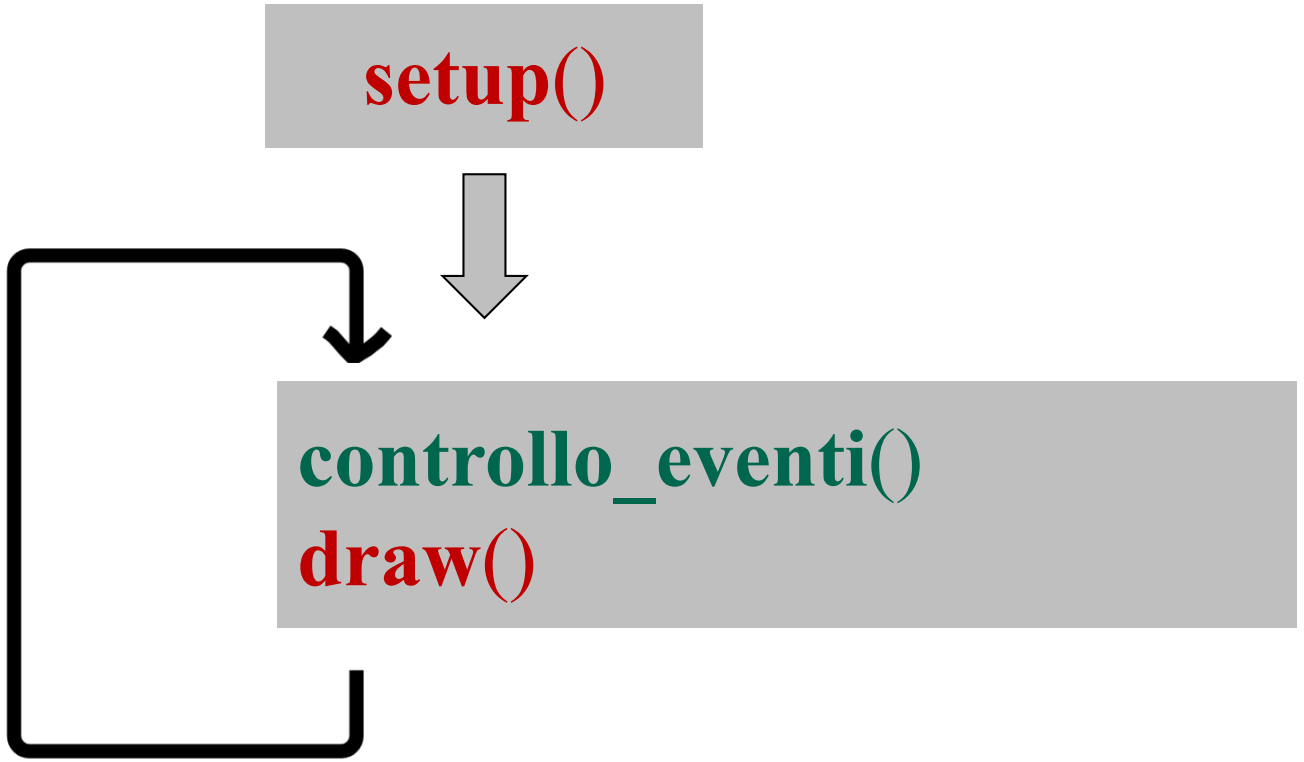
Palla rimbalzante – soluzione parziale


```
x = 0
y = 0
ballSize = 50
moveForward = True

def setup():
  global y
  size(600, 600)
  y = height/2
```

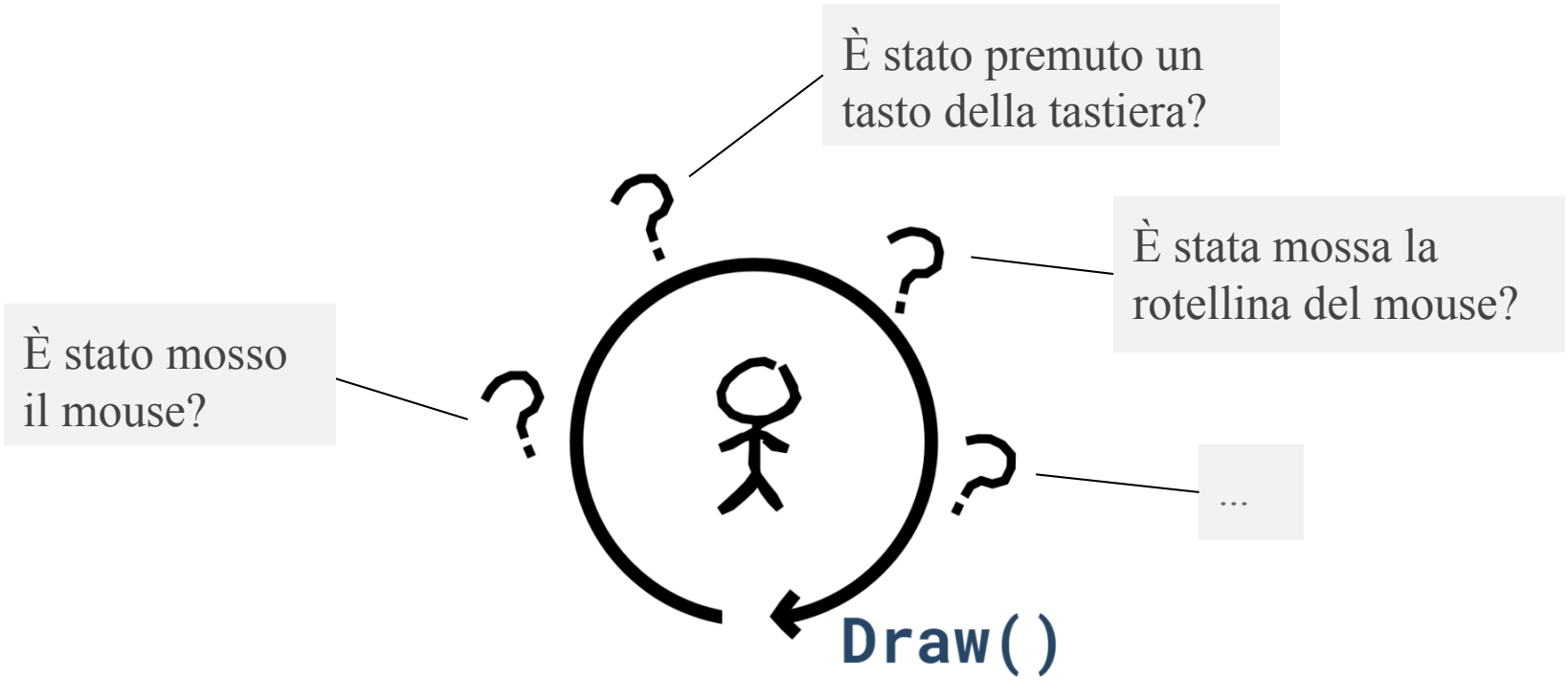
```
def draw():
    global x, y, ballSize, moveForward
    background(255)
    stroke(255, 0, 0)
    fill(255, 0, 0)
    ellipse(x, y, ballSize, ballSize)
    if moveForward and x < (width - ballSize/2):
        x = x+1
    elif moveForward and x >= (width - ballSize/2):
        x = x-1
        moveForward = False
    elif moveForward == False and x > ballSize/2:
        x = x-1
    elif moveForward == False and x <= ballSize/2:
        moveForward = True
        x = x+1
```

Main Loop



La parte di controllo degli eventi è opzionale

Main Loop - Eventi



Funzioni di gestione degli eventi

mousePressed()

mouseReleased()

mouseClicked()

mouseDragged()

mouseMoved()

keyPressed()

keyReleased()

keyTyped()

Esempio: disegnare col mouse


```
def mouseDragged():
        noStroke()
        fill(255, 0, 0)
        ellipse(mouseX, mouseY, 5, 5)
```

Esempio: colore sfondo


```
from random import randint
def mouseClicked():
    background(randint(0, 255))
```

Tastiera

Possiamo sapere quale tasto della tastiera è stato premuto utilizzando `key`.

Possiamo definire keyPressed() per reagire all'evento di pressione del tasto

Esempio: movimento con WASD


```
x = 0
y = 0

def setup():
        size(800, 600)

def draw():
    global x, y
    background(255)
    ellipse(x, y, 50, 50)
```

```
def keyPressed():
    global x, y
    if key == 'w':
        y = y - 10
    elif key == 's':
        y = y + 10
    elif key == 'd':
        x = x + 10
    elif key == 'a':
        x = x - 10
```

Costruzione di una stringa (1)

```
discovered = '__________'
def setup():
    size(600, 600)
def draw():
    global discovered
    background(255)
    textSize(26)
    fill(0)
    text(discovered, width/2, height/2)
```

Costruzione di una stringa (2)


```
def keyPressed():
    global discovered
    pos = 0
    while pos < len(discovered) and discovered[pos] != '_':
        pos = pos + 1
    if pos < len(discovered):
        discovered = discovered[:pos] + key + discovered[pos+1:]
```

Immagini

```
def setup():
    global img
    size(600, 600)
    img = loadImage('icon.png')

def draw():
     global img
     background(255)
     image(img, 10, 10, 0.10*width, 0.10*height)
```

L'immagine viene caricata da file e assegnata a img

Attenzione: nel folder dove si trova il programma, ci deve essere un folder data. L'immagine deve trovarsi in questo folder

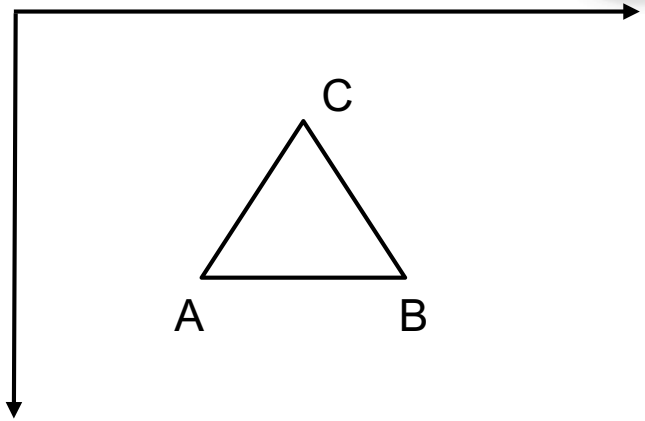
L'immagine viene resa visibile nella posizione (10, 10).
Il quarto e quinto parametro consentono di definire la dimensione dell'immagine

Esercizio

- Estendere il programma dell'immagine per fare in modo che quest'ultima possa essere spostata con i tasti WASD

Costruzione e uso di una nuova figura geometrica

```
def triangolo(xA, yA, l):
    xB = xA+l
    yB = yA
    xC = xA + (xB - xA)/2
    yC = yA - l*sin(radians(60))
    beginShape();
    vertex(xA, yA)
    vertex(xB, yB)
    vertex(xC, yC)
    endShape(CLOSE)
```




```
def setup():
    size(300, 300)
    background(255)

    noFill()
    triangolo(50, 75, 50)
    triangolo(170, 75, 40)

    fill(255, 204, 255)
    stroke(128, 0, 128)
    triangolo(50, 180, 60)
```

RICORSIONE

Chiamata ricorsiva di funzioni

- Possibilità per una funzione P di richiamare se stessa
  - in modo diretto (P chiama P), oppure...
  - indiretto (P chiama Q che chiama P)
- Si applica alla soluzione di problemi per i quali:
  - Esiste una soluzione per uno o più *casi base*
  - Il caso generale si risolve utilizzando la soluzione ricavata per un *sottocaso* che a sua volta può essere risolto allo stesso modo, fino a convergere verso il caso base.

Esempi di problemi ricorsivi

- Fattoriale di un numero naturale n
  - Se n=0, n! = 1
  - Se n=1, n!=1*1
  - Se n=2, n!=2*1*1,
  - Se n=3, n!=3*2!=3*2*1!=3*2*1*0!
  - Generalizzando, se n≠0, n! = n*(n-1)!
- Definizione di numero naturale pari
  - 0 è pari
  - n è pari se n-2 è pari

Caso base

Utilizzo la soluzione ottenuta per un sottocaso.
Si converge verso il caso base

Caso base

Sottocaso

Esempio di sottoprogramma ricorsivo: il fattoriale di un numero


```
def factRicorsivo(n):
    if n==0 or n == 1:
        return 1
    elif n>1:
        return n*factRicorsivo(n-1)
```

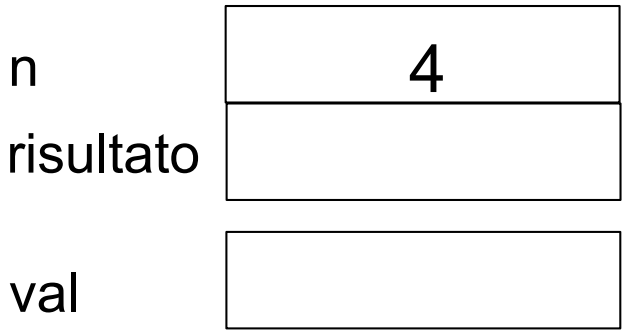
Il modello di esecuzione dei sottoprogrammi ricorsivi

- Ogni chiamata attiva del sottoprogramma usa un'area di memoria distinta dalle altre (*record di attivazione o ambiente di esecuzione*)
- … in cui conserva una copia distinta dei parametri e delle variabili locali
- Il record di attivazione rimane in memoria per tutta la durata dell'esecuzione della chiamata corrispondente

Il modello di esecuzione: l'esempio di factRicorsivo

Supponiamo che factRicorsivo venga richiamato così

>> val = factRicorsivo(4)



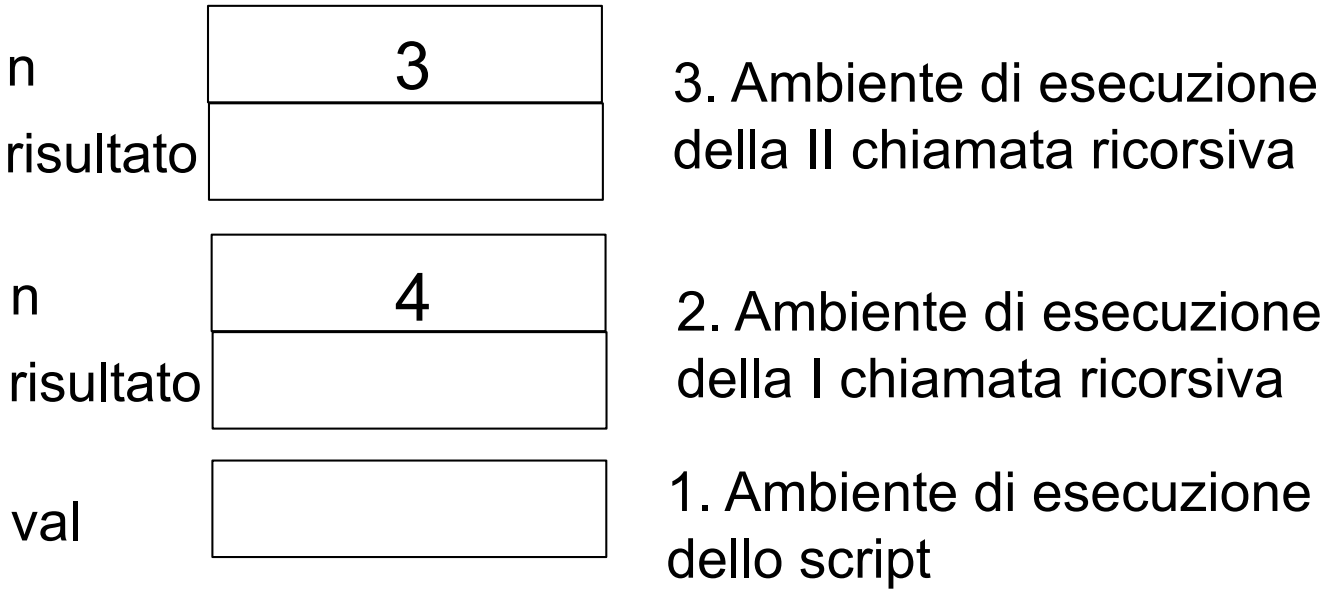
2. Ambiente di esecuzione della I chiamata ricorsiva

1. Ambiente di esecuzione dello script

Il modello di esecuzione: l'esempio di factRicorsivo

Supponiamo che factRicorsivo venga richiamato così

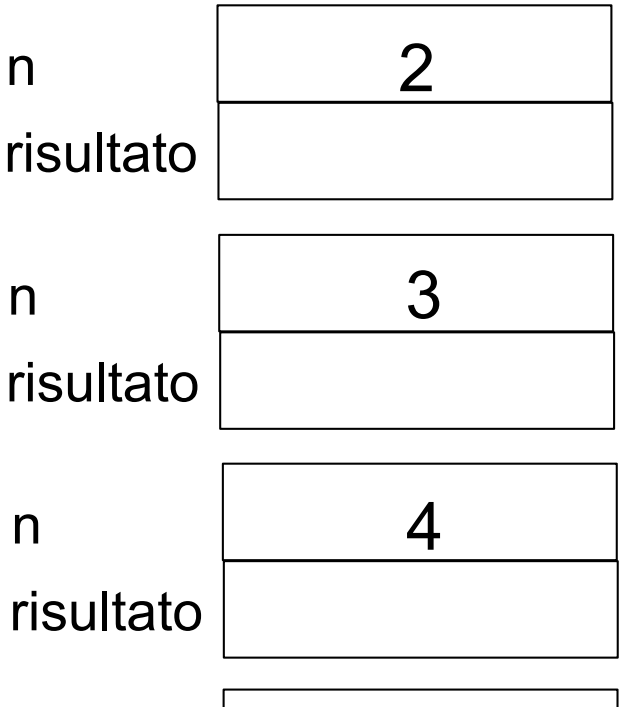
>> val = factRicorsivo(4)



Il modello di esecuzione: l'esempio di factRicorsivo

Supponiamo che factRicorsivo venga richiamato così

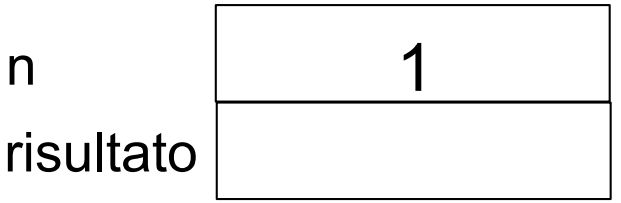
>> val = factRicorsivo(4)



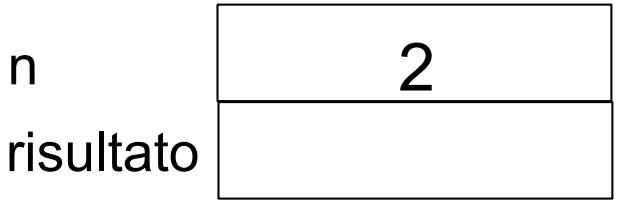
Il modello di esecuzione: l'esempio di factRicorsivo

Supponiamo che factRicorsivo venga richiamato così

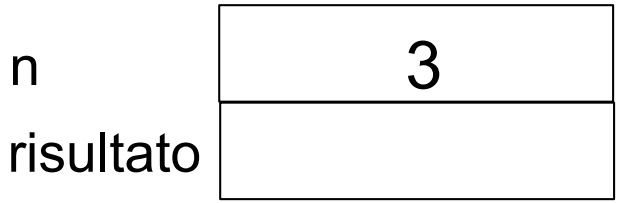
```
>> val = factRicorsivo(4)
```



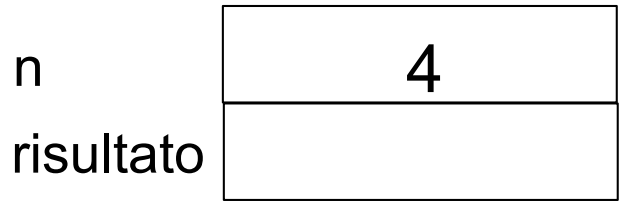
5. Ambiente di esecuzione della IV chiamata ricorsiva



4. Ambiente di esecuzione della III chiamata ricorsiva



3. Ambiente di esecuzione della II chiamata ricorsiva



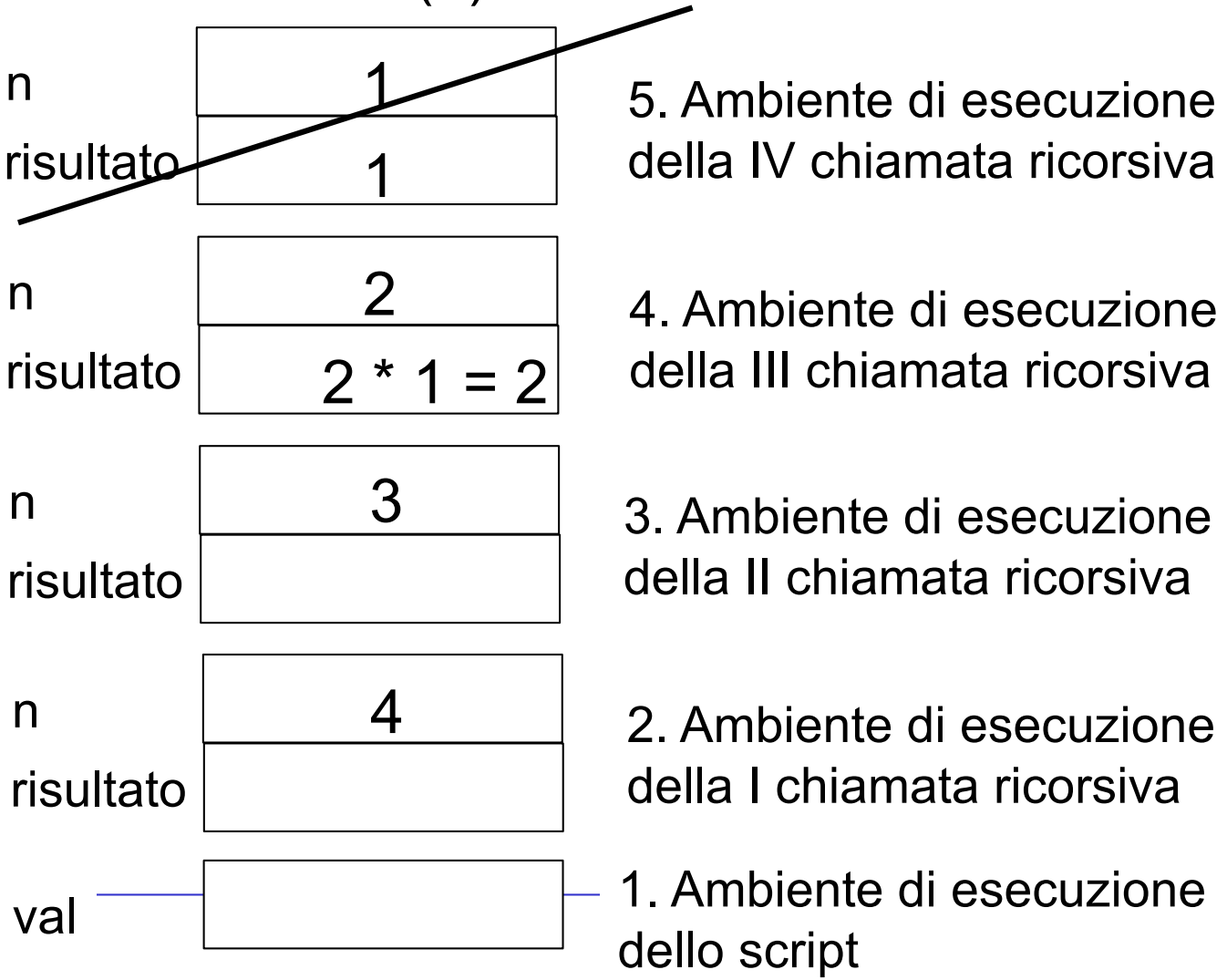
2. Ambiente di esecuzione della I chiamata ricorsiva

1. Ambiente di esecuzione dello script

Il modello di esecuzione: l'esempio di factRicorsivo

Supponiamo che factRicorsivo venga richiamato così

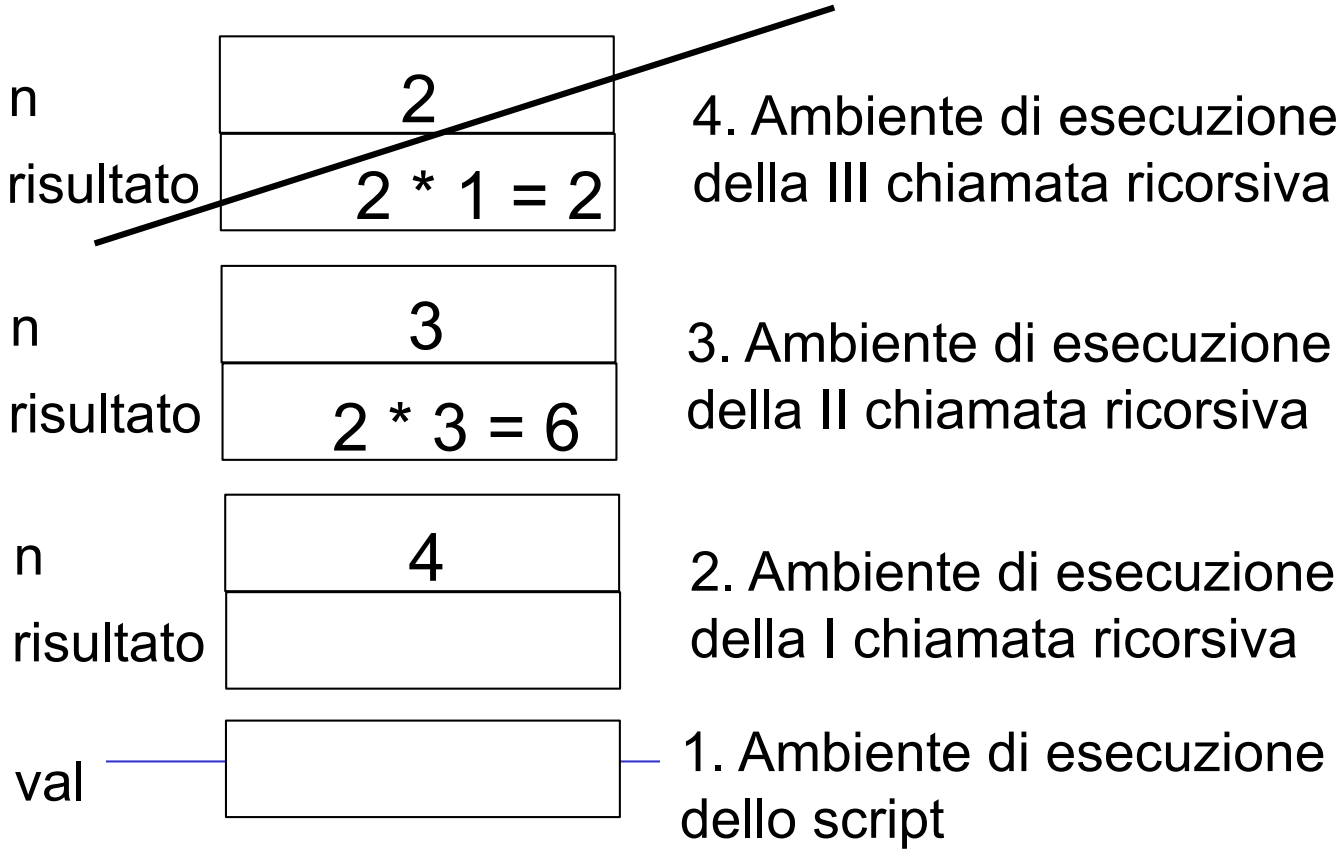
>> val = factRicorsivo(4)



Il modello di esecuzione: l'esempio di factRicorsivo

Supponiamo che factRicorsivo venga richiamato così

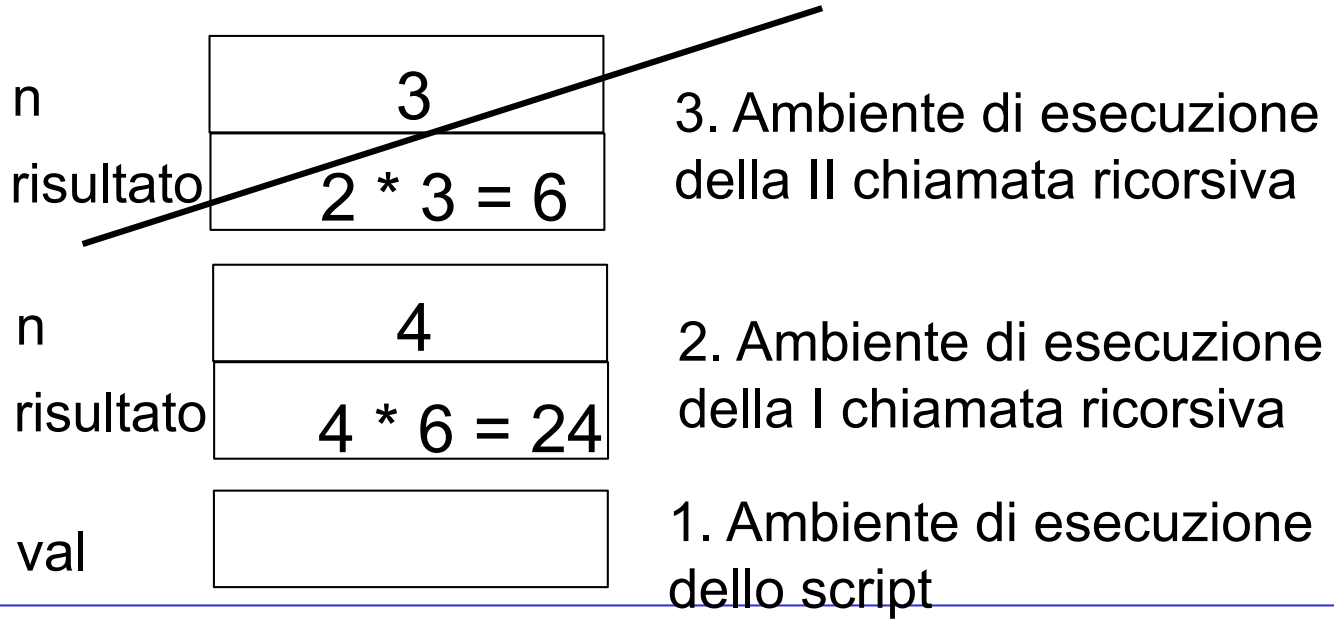
>> val = factRicorsivo(4)



Il modello di esecuzione: l'esempio di factRicorsivo

Supponiamo che factRicorsivo venga richiamato così

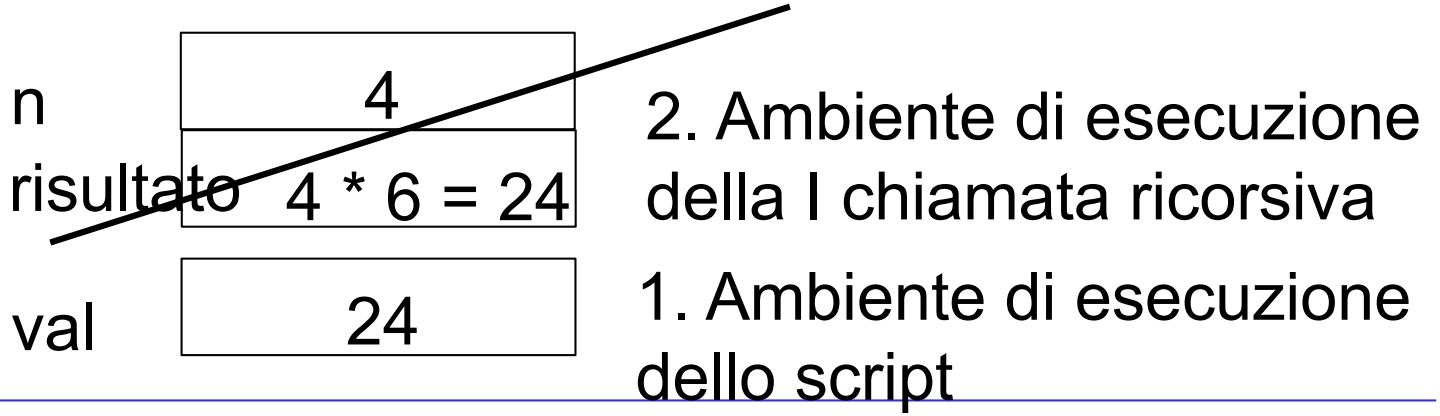
>> val = factRicorsivo(4)



Il modello di esecuzione: l'esempio di factRicorsivo

Supponiamo che factRicorsivo venga richiamato così

>> val = factRicorsivo(4)



Fattoriali a confronto

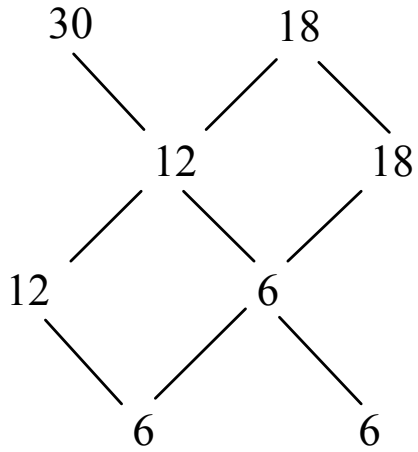

```
def factRicorsivo(n) :
    if n==0 or n == 1:
        return 1
    elif n>1:
        return n*factRicorsivo(n-1)
```

```
def fact(n):
    f=1
    for k  in range(n):
        f=f*(k+1)
    return f
```

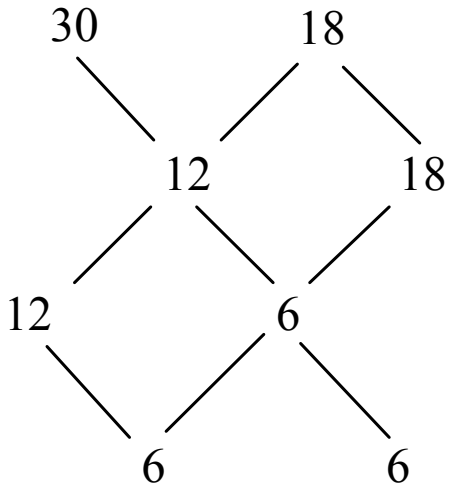
- In factRicorsivo maggior uso della memoria
  - Per il calcolo di 4! si allocano 9 aree di memoria di tipo numerico contro le tre della versione iterativa
- Se il problema ha una soluzione ricorsiva, è molto più facile ed elegante scrivere l'algoritmo ricorsivo che quello iterativo
  - Se è necessario ridurre il consumo di memoria, si può successivamente ottimizzare la soluzione trasformandola nella versione iterativa.

Massimo Comun Divisore, alg. Euclide (1)

- se m = n, MCD(m, n) = m (caso base)
- se m > n, MCD(m, n) = MCD(m-n, n) (caso risorsivo)
- se m < n MCD(m, n) = MCD(m, n-m) (caso risorsivo)
- Il ragionamento può essere ripetuto
- Prima o poi si arriva a una coppia di numeri uguali



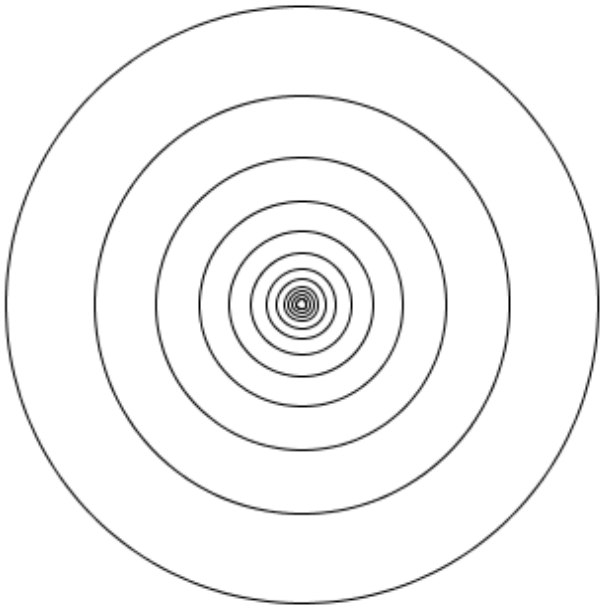
Massimo Comun Divisore, alg. Euclide (2)




```
Versione ricorsiva
def MCDeuclidRic(m,n):
  if m==n:
      M=m
  elif m>n:
      M = MCDeuclidRic(m-n,n)
  else:
      M = MCDeuclidRic(m,n-m)
  return M
```

```
Versione iterativa
def MCDeuclid(m,n):
  while m != n:
      if m>n:
          m=m-n
      else:
          n=n-m
  return m
```

Ricorsione in processing, es frattale 1 (cerchi concentrici)

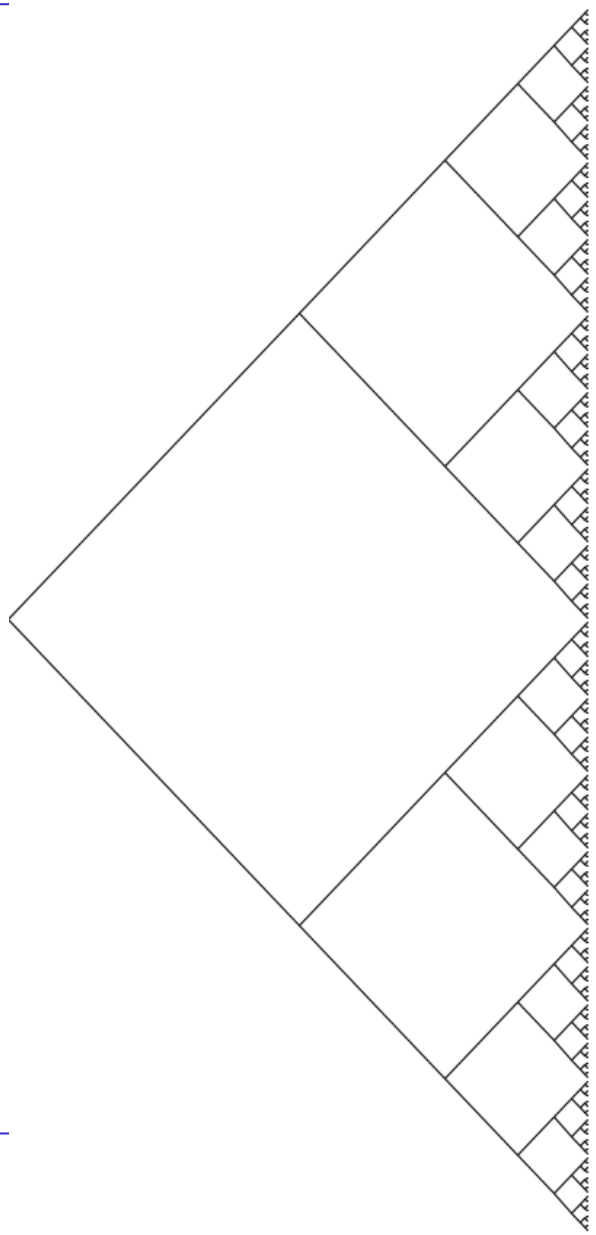


Esempio frattale 1


```
def frattale1(x, y, grandezza):
    ellipse(x, y, grandezza, grandezza)
    if grandezza > 5:
        frattale1(x, y, grandezza * 0.7)

def setup():
    size(600, 400)
    background(255)
    frattale1(width/2, height/2, 300)
```

Esempio frattale 2



Esempio frattale 2


```
def frattale(x, y, lunghezza):
    line(x, y, x+lunghezza, y+lunghezza)
    line(x, y, x+lunghezza, y-lunghezza)

    if lunghezza > 3:
        frattale(x+lunghezza, y+lunghezza, lunghezza*0.7)
        frattale(x+lunghezza, y-lunghezza, lunghezza*0.7)

def setup():
    size(600, 800)
    background(255)
    frattale(0, height/2, 100)
```

Acknowledgements / Contributions

Part of these slides are by Michele Lambertucci, student at Politecnico di Milano

Adaptation and extensions by Elisabetta Di Nitto, Politecnico di Milano

Some examples have been derived from the ones available on the Processing website