

# Corso di **STATISTICA, INFORMATICA, ELABORAZIONE DELLE INFORMAZIONI**

Modulo di Sistemi di Elaborazione delle Informazioni

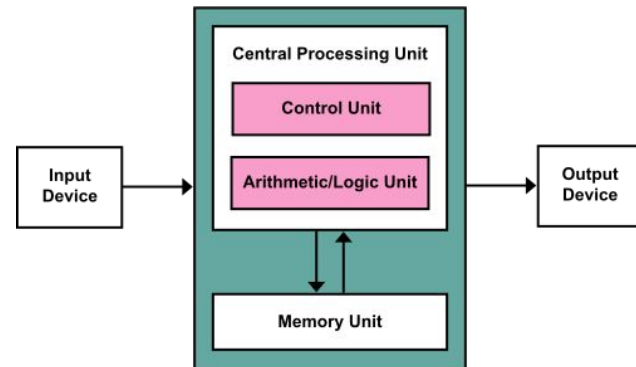
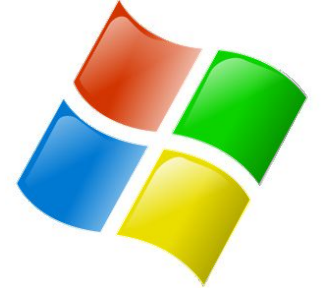
## Strutture Iterative



**UNIVERSITÀ DEGLI STUDI DELLA BASILICATA**

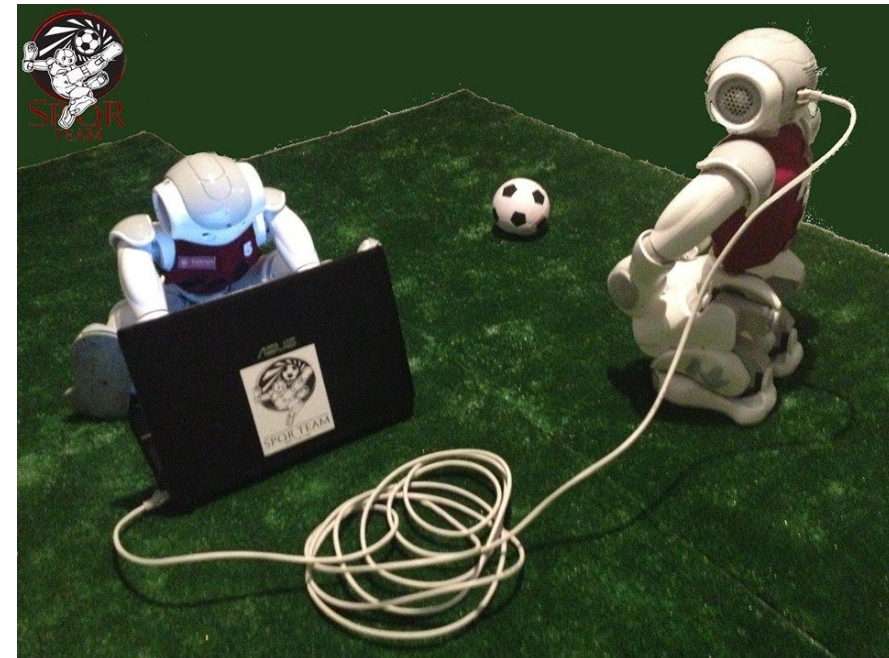
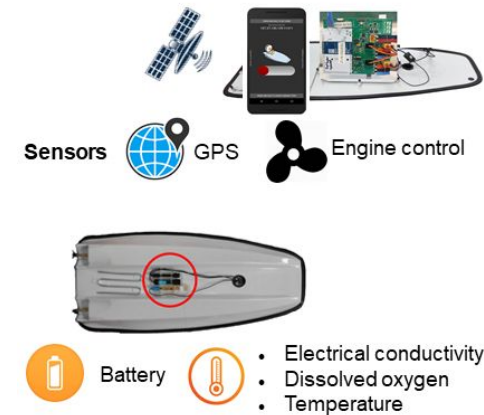


Docente:  
**Domenico Daniele Bloisi**



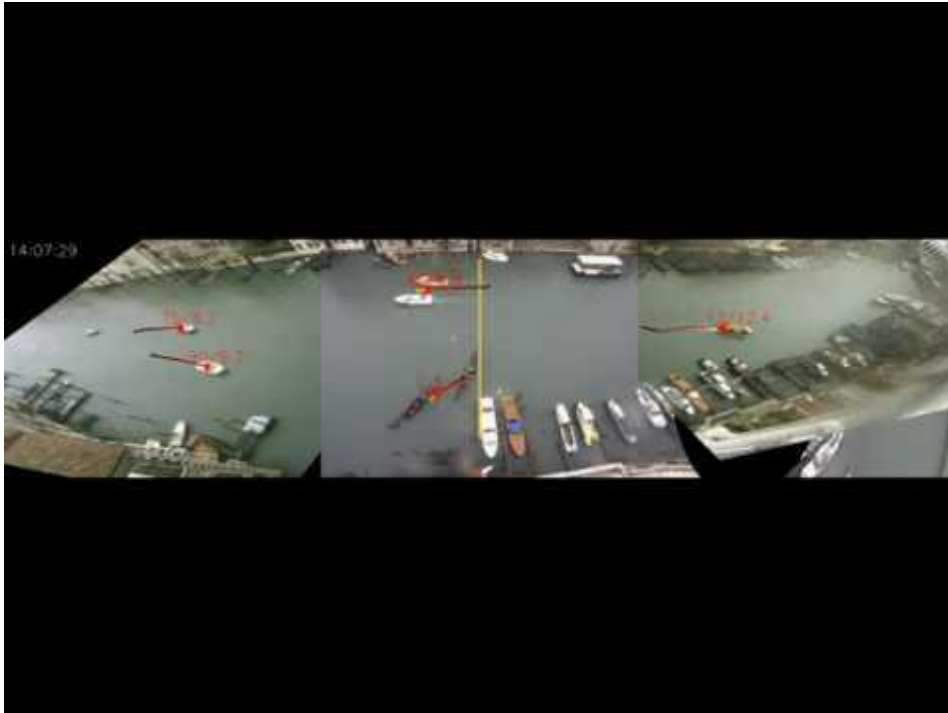
# Domenico Daniele Bloisi

- Professore Associato  
Dipartimento di Matematica, Informatica  
ed Economia  
Università degli studi della Basilicata  
<http://web.unibas.it/bloisi>
- SPQR Robot Soccer Team  
Dipartimento di Informatica, Automatica  
e Gestionale Università degli studi di  
Roma “La Sapienza”  
<http://spqr.diag.uniroma1.it>



# Interessi di ricerca

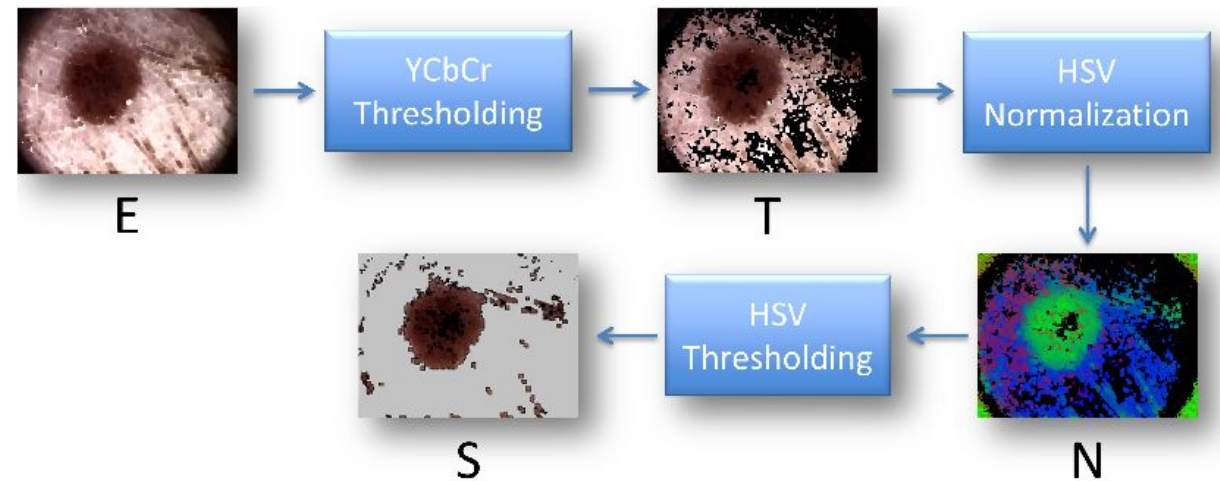
- Intelligent surveillance
- Robot vision
- Medical image analysis



[https://youtu.be/9a70Ucgbi\\_U](https://youtu.be/9a70Ucgbi_U)



<https://youtu.be/2KHNZX7UIWQ>



# UNIBAS Wolves <https://sites.google.com/unibas.it/wolves>



- UNIBAS WOLVES is the robot soccer team of the University of Basilicata. Established in 2019, it is focussed on developing software for NAO soccer robots participating in RoboCup competitions.

- UNIBAS WOLVES team is twinned with SPQR Team at Sapienza University of Rome



<https://youtu.be/ji0OmkaWh20>

# Informazioni sul corso

---

Il corso di STATISTICA, INFORMATICA, ELABORAZIONE DELLE INFORMAZIONI

- include 3 moduli:
  - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI  
(il martedì - docente: Domenico Bloisi)
  - INFORMATICA  
(il mercoledì - docente: Enzo Veltri)
  - PROBABILITA' E STATISTICA MATEMATICA  
(il giovedì - docente: Antonella Iuliano)
- Periodo: **I semestre** ottobre 2022 – gennaio 2023

# Informazioni sul modulo

---

- Home page del modulo:  
<https://web.unibas.it/bloisi/corsi/sei.html>
- Martedì dalle 11:30 alle 13:30

# Ricevimento Bloisi

---

- In presenza, durante il periodo delle lezioni:  
Lunedì dalle 17:00 alle 18:00  
presso Edificio 3D, Il piano, stanza 15  
**Si invitano gli studenti a controllare regolarmente la bacheca degli avvisi per eventuali variazioni**
- Tramite google Meet e al di fuori del periodo delle lezioni:  
da concordare con il docente tramite email

Per prenotare un appuntamento inviare  
una email a  
[domenico.bloisi@unibas.it](mailto:domenico.bloisi@unibas.it)



Recap



# Esercizio 7

---

Scrivere un programma che legga da tastiera 3 numeri interi e stampi a video il maggiore e il minore tra essi

# Possibile soluzione Esercizio 7

```
5 a
{x}
a = int(input("Inserire il primo numero (intero): "))
b = int(input("Inserire il secondo numero (intero): "))
c = int(input("Inserire il terzo numero (intero): "))

maggiore = a
minore = a

if a > b:
    if a > c:
        maggiore = a
    else:
        maggiore = c
else:
    if b > c:
        maggiore = b
    else:
        maggiore = c

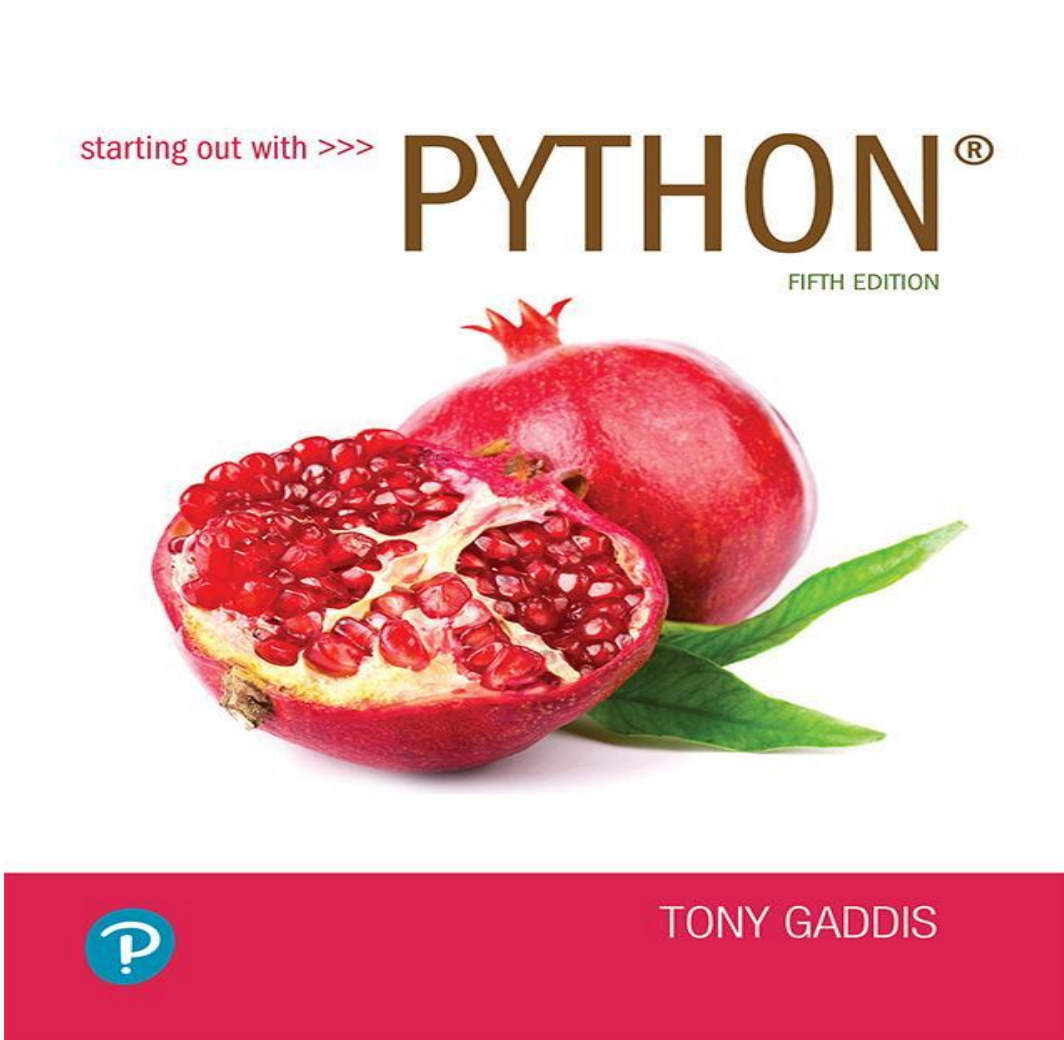
print("maggiore:", maggiore)

if a < b:
    if a < c:
        minore = a
    else:
        minore = c
else:
    if b < c:
        minore = b
    else:
        minore = c

print("minore:", minore)
```

# Starting out with Python

Fifth Edition



## Chapter 4 Repetition Structures

# Topics

- Introduction to Repetition Structures
- The `while` Loop: a Condition-Controlled Loop
- The `for` Loop: a Count-Controlled Loop
- Calculating a Running Total
- Sentinels
- Input Validation Loops
- Nested Loops
- Turtle Graphics: Using Loops to Draw Designs

# Introduction to Repetition Structures

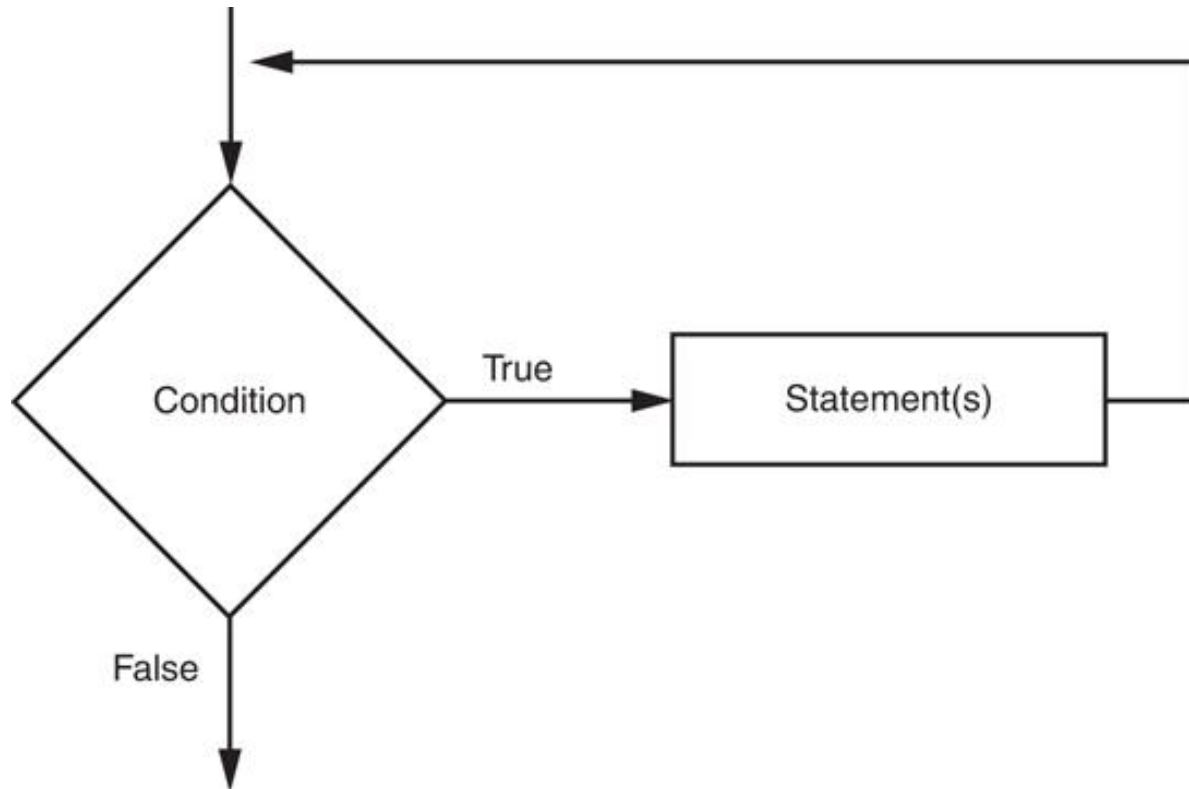
- Often have to write code that performs the same task multiple times
  - Disadvantages to duplicating code
    - Makes program large
    - Time consuming
    - May need to be corrected in many places
- Repetition structure: makes computer repeat included code as necessary
  - Includes condition-controlled loops and count-controlled loops

# The `while` Loop: a Condition-Controlled Loop (1 of 3)

- `while` loop: while condition is true, do something
  - Two parts:
    - Condition tested for true or false value
    - Statements repeated as long as condition is true
  - In flow chart, line goes back to previous part
  - General format:

```
while condition:  
    statements
```

# The `while` Loop: a Condition-Controlled Loop (2 of 3)



**Figure 4-1** The logic of a `while` loop

# The `while` Loop: a Condition-Controlled Loop (3 of 3)

- In order for a loop to stop executing, something has to happen inside the loop to make the condition false
- Iteration: one execution of the body of a loop
- `while` loop is known as a *pretest* loop
  - Tests condition before performing an iteration
    - Will never execute if condition is false to start with
    - Requires performing some steps prior to the loop



# while

---

Sintassi:

```
while espr_cond:  
    sequenza_di_istruzioni
```

- la keyword `while` deve essere scritta senza rientri
- `espr_cond` è una espressione condizionale qualsiasi
- `sequenza_di_istruzioni` consiste in una o più istruzioni qualsiasi. Ciascuna di tali istruzioni deve essere scritta in una riga distinta, con un rientro di almeno un carattere. Il rientro deve essere identico per tutte le istruzioni della sequenza

# Moltiplicazione con while

---

```
#acquisizione input
fattore_1 = int(input("Inserisci il primo fattore: "))
fattore_2 = int(input("Inserisci il secondo fattore: "))

contatore = 0
prodotto = 0
while contatore < fattore_2:
    prodotto = prodotto + fattore_1
    contatore = contatore + 1

print("Il prodotto tra", fattore_1, "e",
      fattore_2, "è", prodotto)
```

```
Inserisci il primo fattore: 24
Inserisci il secondo fattore: 5
Il prodotto tra 24 e 5 è 120
```

---

# Massimo comun divisore

---

Calcolo del massimo comun divisore con l'algoritmo di Euclide

Dati due numeri naturali  $a$  e  $b$ , entrambi maggiori di 1 con  $a > b$ :

- se  $b$  è un divisore esatto di  $a$ ,  $b$  è ovviamente il massimo comun divisore tra i due numeri;
- altrimenti, detto  $r$  il resto della divisione tra  $a$  e  $b$ , il MCD tra  $a$  e  $b$  è uguale al MCD tra  $b$  e  $r$ , cioè  $\text{MCD}(a,b) = \text{MCD}(b,r)$

# Massimo comun divisore: esempio

---

```
▶ #acquisizione input
a = int(input("Inserire il primo intero: "))
b = int(input("Inserire il secondo intero: "))

print("il massimo comun divisore tra", a, "e", b, "è:")

#algoritmo di Euclide
while a != b:
    if a < b:
        b = b - a
    else:
        a = a - b
MCD = a

#stampa risultato
print(MCD)
```

```
↳ Inserire il primo intero: 32
Inserire il secondo intero: 28
il massimo comun divisore tra 32 e 28 è:
4
```

# Infinite Loops

- Loops must contain within themselves a way to terminate
  - Something inside a `while` loop must eventually make the condition false
- Infinite loop: loop that does not have a way of stopping
  - Repeats until program is interrupted
  - Occurs when programmer forgets to include stopping code in the loop



# The `for` Loop: a Count-Controlled Loop (1 of 2)

- Count-Controlled loop: iterates a specific number of times
  - Use a `for` statement to write count-controlled loop
    - Designed to work with sequence of data items
      - Iterates once for each item in the sequence
    - General format:  

```
for variable in [val1, val2, etc]:  
    statements
```
    - Target variable: the variable which is the target of the assignment at the beginning of each iteration

## esempio FOR

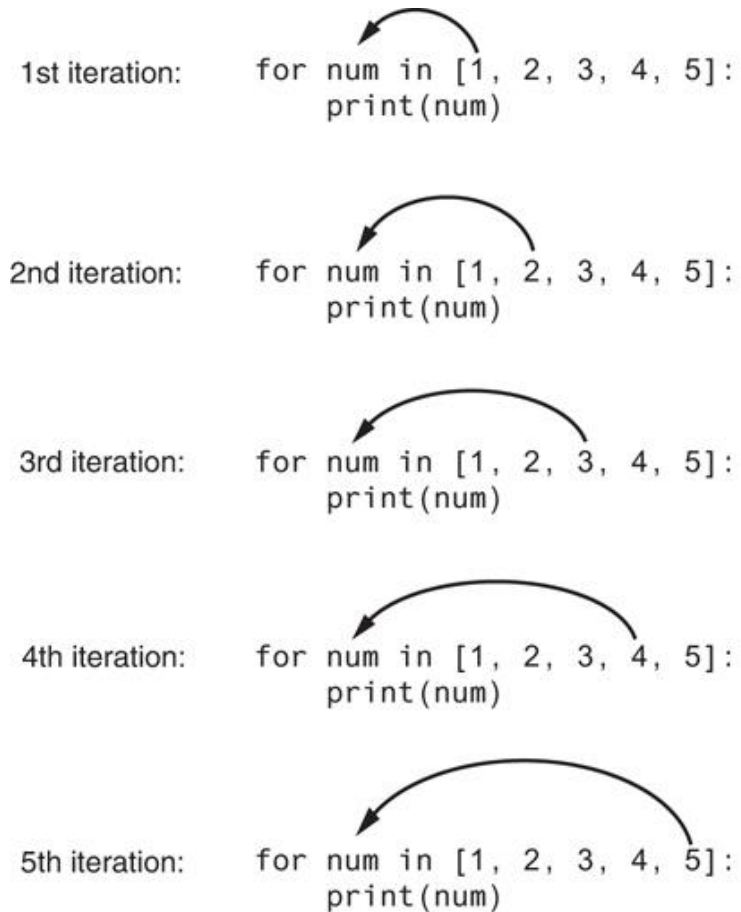
✓  
0 s

```
[9] for num in [1,2,3,4,5]:  
    print(num)
```

1  
2  
3  
4  
5



# The `for` Loop: a Count-Controlled Loop (2 of 2)



**Figure 4-4** The `for` loop

# The range function

The range type represents an immutable sequence of numbers and is commonly used for looping a specific number of times in for loops.

- `range(stop)`
- `range(start, stop[, step])`

```
▶ a = list(range(10))
print(a)

b = list(range(1, 11))
print(b)

c = list(range(0, 30, 5))
print(c)

d = list(range(0, 10, 3))
print(d)

e = list(range(0, -10, -1))
print(e)

f = list(range(0))
print(f)

g = list(range(1, 0))
print(g)
```

```
↳ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[0, 5, 10, 15, 20, 25]
[0, 3, 6, 9]
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
[]
[]
```

## The range function

# Using the `range` function with the `for` Loop

- The `range` function simplifies the process of writing a `for` loop
  - `range` returns an iterable object
    - Iterable: contains a sequence of values that can be iterated over
- `range` characteristics:
  - One argument: used as ending limit
  - Two arguments: starting value and ending limit
  - Three arguments: third argument is step value

# Using the Target Variable Inside the Loop

- Purpose of target variable is to reference each item in a sequence as the loop iterates
- Target variable can be used in calculations or tasks in the body of the loop
  - Example: calculate square root of each number in a range

```
▶ # Questo programma utilizza un ciclo per visualizzare
# una tabella contenente i numeri da 1 a 10
# unitamente ai relativi quadrati.

# Stampa le intestazioni della tabella.
print('Numero \t Quadrato')
print('-----')

# Stampa i numeri da 1 a 10
# e i loro quadrati.
for number in range(1, 11):
    square = number**2
    print(number, "\t", square)
```

```
↳ Numero  Quadrato
-----
1         1
2         4
3         9
4        16
5        25
6        36
7        49
8        64
9        81
10       100
```

## Using the Target Variable Inside the Loop

# Letting the User Control the Loop Iterations

- Sometimes the programmer does not know exactly how many times the loop will execute
- Can receive range inputs from the user, place them in variables, and call the `range` function in the for clause using these variables
  - Be sure to consider the end cases: `range` does not include the ending limit

# Esercizio 8

---

Si scriva un codice Python che riceva come input da tastiera un intero  $n$  e disegni sullo schermo un numero di caratteri `'*'` pari ad  $n$

```
valore intero n: 5
*****
finito
```

```
valore intero n: 21
*****
finito
```



# Esercizio 8

---

✓  
3 s

```
▶ n = int(input("valore intero n: "))  
  
for i in range(n):  
    print("*")  
  
print("finito")
```

```
valore intero n: 5  
*  
*  
*  
*  
*  
finito
```

```
valore intero n: 5  
*****  
finito
```

# Esercizio 8

---



✓  
3 s



```
n = int(input("valore intero n: "))

for i in range(n):
    if(i < n-1):
        print("*", end="")
    else:
        print("*")

print("finito")
```

```
valore intero n: 5
*****
finito
```

```
valore intero n: 5
*****
finito
```

# Generating an Iterable Sequence that Ranges from Highest to Lowest

- The `range` function can be used to generate a sequence with numbers in descending order
  - Make sure starting number is larger than end limit, and step value is negative
  - Example: `range(10, 0, -1)`

```
▶ for num in range(10, 0, -1):  
    print(num)
```

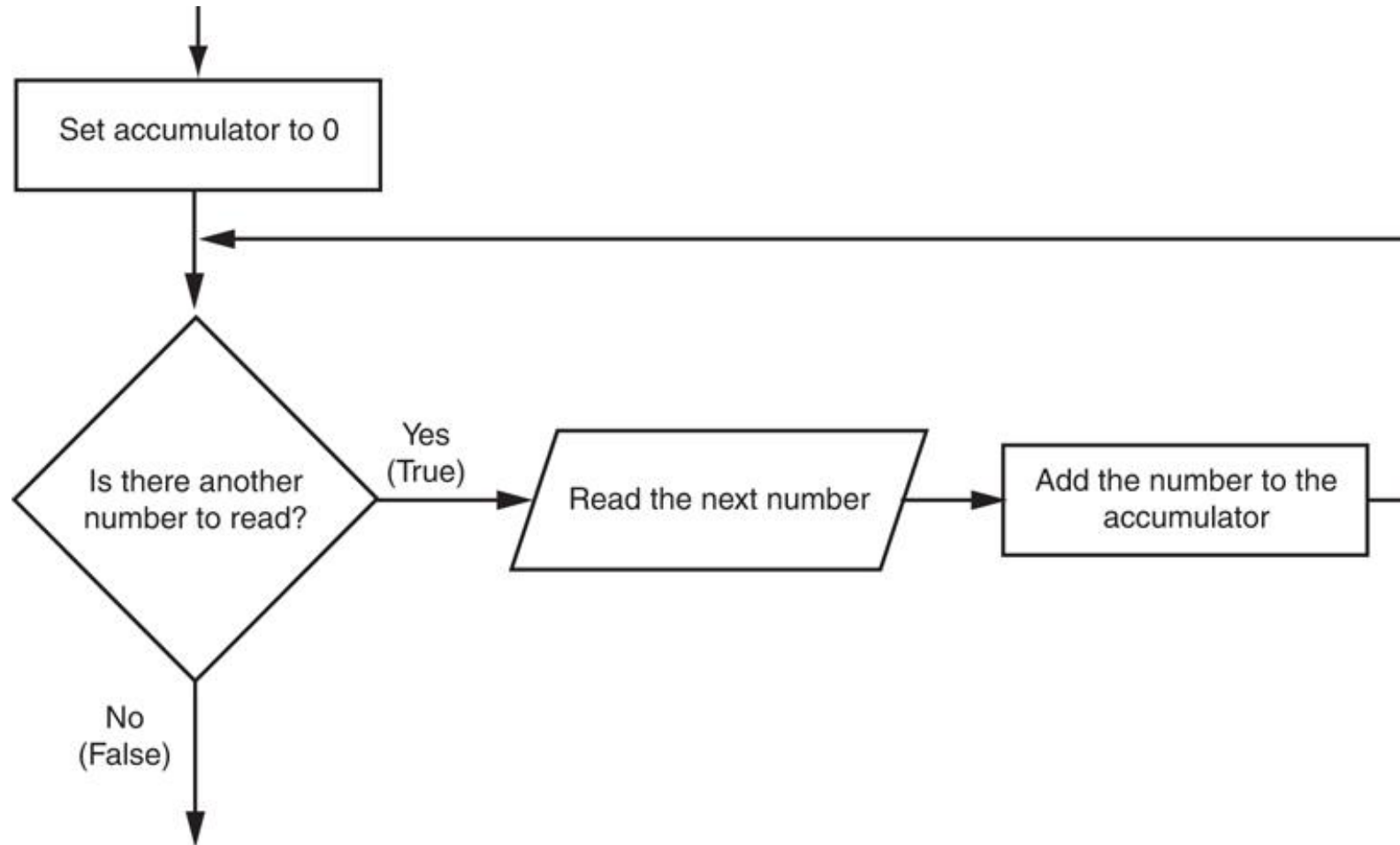
```
↳ 10  
   9  
   8  
   7  
   6  
   5  
   4  
   3  
   2  
   1
```

**Generating an  
Iterable  
Sequence that  
Ranges from  
Highest to  
Lowest**

# Calculating a Running Total (1 of 2)

- Programs often need to calculate a total of a series of numbers
  - Typically include two elements:
    - A loop that reads each number in series
    - An *accumulator* variable
  - Known as program that keeps a running total: accumulates total and reads in series
  - At end of loop, accumulator will reference the total

# Calculating a Running Total (2 of 2)



**Figure 4-6** Logic for calculating a running total

# Calculating a Running Total

```
▶ # Questo programma calcola la somma di una serie
# di numeri inseriti dall'utente.

MAX = 5 # Il numero massimo

# Inizializza una variabile accumulatore.
total = 0.0

# Spiega che cosa stiamo facendo.
print('Questo programma calcola la somma di')
print(f'{MAX} numeri inseriti.')

# Ottiene i numeri e li accumula.
for counter in range(MAX):
    number = int(input('Inserisci un numero: '))
    total = total + number

# Visualizza il totale dei numeri.
print(f'Il totale è {total}.')
```

```
↳ Questo programma calcola la somma di
5 numeri inseriti.
Inserisci un numero: 1
Inserisci un numero: 2
Inserisci un numero: 5
Inserisci un numero: 78
Inserisci un numero: 34
Il totale è 120.0.
```

# The Augmented Assignment Operators (1 of 2)

- In many assignment statements, the variable on the left side of the = operator also appears on the right side of the = operator
- Augmented assignment operators: special set of operators designed for this type of job
  - Shorthand operators



# The Augmented Assignment Operators (2 of 2)

**Table 4-2** Augmented assignment operators

Operator	Example Usage	Equivalent To
<code>+=</code>	<code>x += 5</code>	<code>x = x + 5</code>
<code>-=</code>	<code>y -= 2</code>	<code>y = y - 2</code>
<code>*=</code>	<code>z *= 10</code>	<code>z = z * 10</code>
<code>/=</code>	<code>a /= b</code>	<code>a = a / b</code>
<code>%=</code>	<code>c %= 3</code>	<code>c = c % 3</code>
<code>//=</code>	<code>x //= 3</code>	<code>x = x // 3</code>
<code>**=</code>	<code>y **= 2</code>	<code>y = y**2</code>

# Sentinels

- Sentinel: special value that marks the end of a sequence of items
  - When program reaches a sentinel, it knows that the end of the sequence of items was reached, and the loop terminates
  - Must be distinctive enough so as not to be mistaken for a regular value in the sequence
  - Example: when reading an input file, empty line can be used as a sentinel

# Sentinels

```
▶ import random

sentinel = input("vuoi che stampi un numero tra 0 e 100 (Y/N): ")

while sentinel == "Y":
    print(random.randint(0, 100))
    sentinel = input("vuoi che stampi un numero tra 0 e 100 (Y/N): ")
```

```
↳ vuoi che stampi un numero tra 0 e 100 (Y/N): Y
27
vuoi che stampi un numero tra 0 e 100 (Y/N): Y
0
vuoi che stampi un numero tra 0 e 100 (Y/N): Y
76
vuoi che stampi un numero tra 0 e 100 (Y/N): N
```

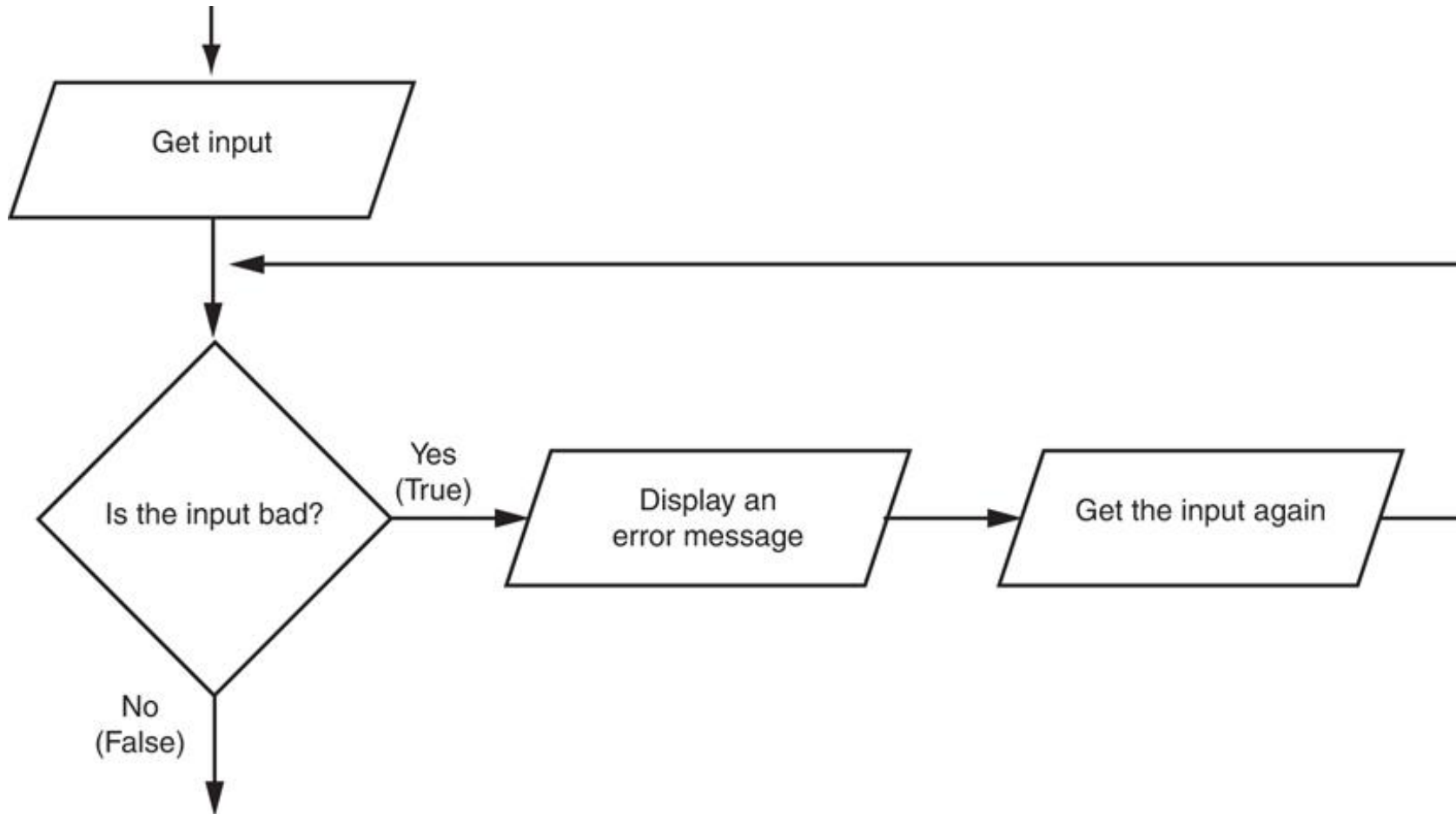
# Input Validation Loops (1 of 3)

- Computer cannot tell the difference between good data and bad data
  - If user provides bad input, program will produce bad output
  - GIGO: garbage in, garbage out
  - It is important to design program such that bad input is never accepted

# Input Validation Loops (2 of 3)

- Input validation: inspecting input before it is processed by the program
  - If input is invalid, prompt user to enter correct data
  - Commonly accomplished using a `while` loop which repeats as long as the input is bad
    - If input is bad, display error message and receive another set of data
    - If input is good, continue to process the input

# Input Validation Loops (3 of 3)



**Figure 4-7** Logic containing an input validation loop

# Input Validation Loops

```
▶ import random

sentinel = input("vuoi che stampi un numero tra 0 e 100 (Y/N): ")
while sentinel != "Y" and sentinel != "N":
    sentinel = input("vuoi che stampi un numero tra 0 e 100 (Y/N): ")

while sentinel == "Y":
    print(random.randint(0, 100))
    sentinel = input("vuoi che stampi un numero tra 0 e 100 (Y/N): ")
```

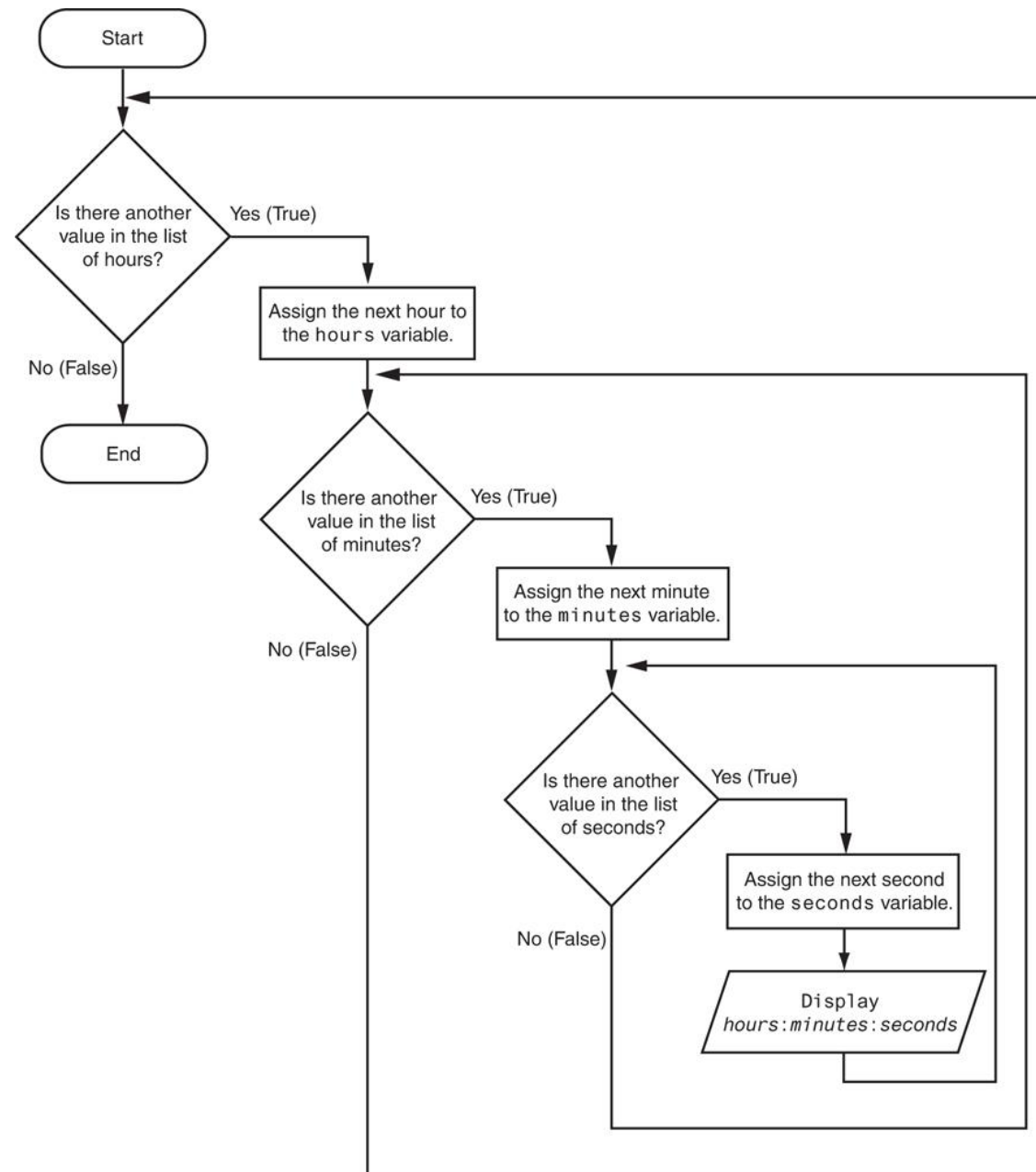
```
↳ vuoi che stampi un numero tra 0 e 100 (Y/N): C
vuoi che stampi un numero tra 0 e 100 (Y/N): 12
vuoi che stampi un numero tra 0 e 100 (Y/N): pippo
vuoi che stampi un numero tra 0 e 100 (Y/N): Y
60
vuoi che stampi un numero tra 0 e 100 (Y/N): Y
78
vuoi che stampi un numero tra 0 e 100 (Y/N):
```

# Nested Loops (1 of 3)

- Nested loop: loop that is contained inside another loop
  - Example: analog clock works like a nested loop
    - Hours hand moves once for every twelve movements of the minutes hand: for each iteration of the “hours,” do twelve iterations of “minutes”
    - Seconds hand moves 60 times for each movement of the minutes hand: for each iteration of “minutes,” do 60 iterations of “seconds”



# Nested Loops (2 of 3)



**Figure 4-8** Flowchart for a clock simulator

## Nested Loops (3 of 3)

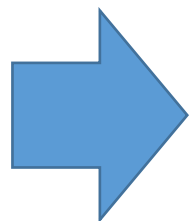
- Key points about nested loops:
  - Inner loop goes through all of its iterations for each iteration of outer loop
  - Inner loops complete their iterations faster than outer loops
  - Total number of iterations in nested loop:

*number of iterations of inner loop X number of iterations of outer loop*

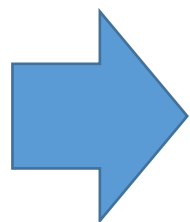
# Esercizio 9

---

Si scriva un codice che riceva come input da tastiera due interi  $a$  e  $b$  e disegni sullo schermo un rettangolo di dimensioni  $a \times b$  usando il carattere '\*', così come mostrato negli esempi



```
Lato a: 5
Lato b: 7
* * * * *
*       *
*       *
*       *
*       *
*       *
*       *
* * * * *
```



```
Lato a: 8
Lato b: 3
* * * * * * * *
*               *
* * * * * * * *
```

# Esercizio 10

---

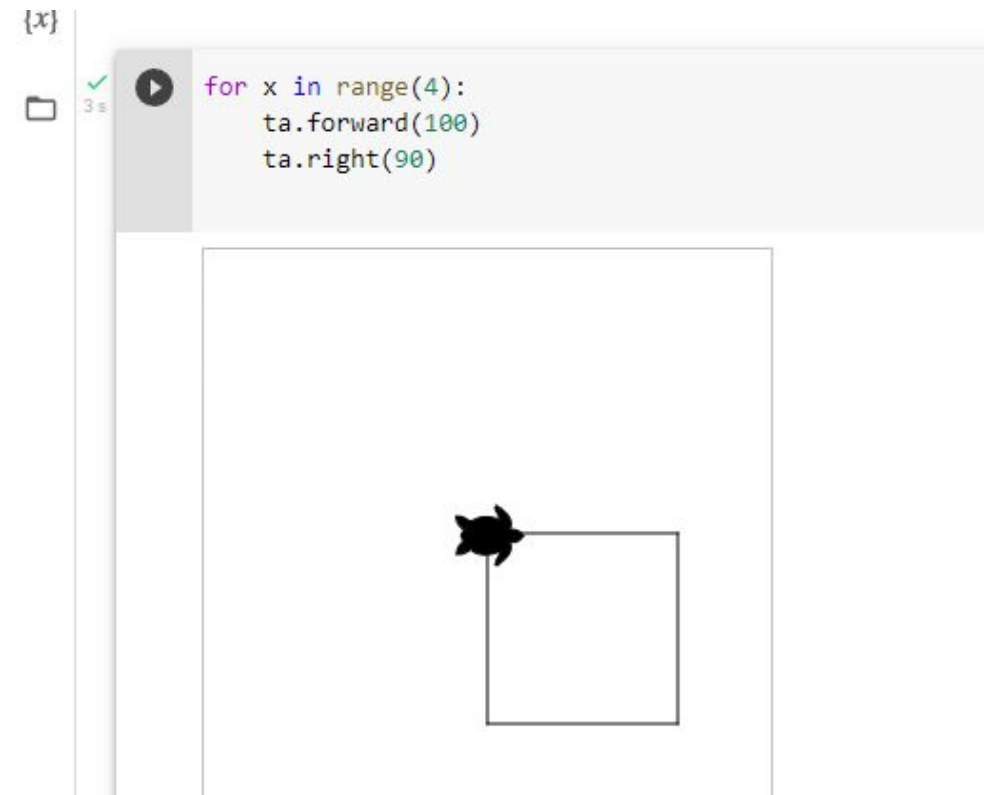
Scrivere un programma che legga da tastiera un intero  $h$  compreso tra 1 e 9 e stampi una piramide di numeri di altezza  $h$

```
Altezza: 5
  1
 121
12321
1234321
123454321
```

# Turtle Graphics: Using Loops to Draw Designs (1 of 4)

- You can use loops with the turtle to draw both simple shapes and elaborate designs. For example, the following for loop iterates four times to draw a square that is 100 pixels wide:

```
for x in range(4):  
    ta.forward(100)  
    ta.right(90)
```

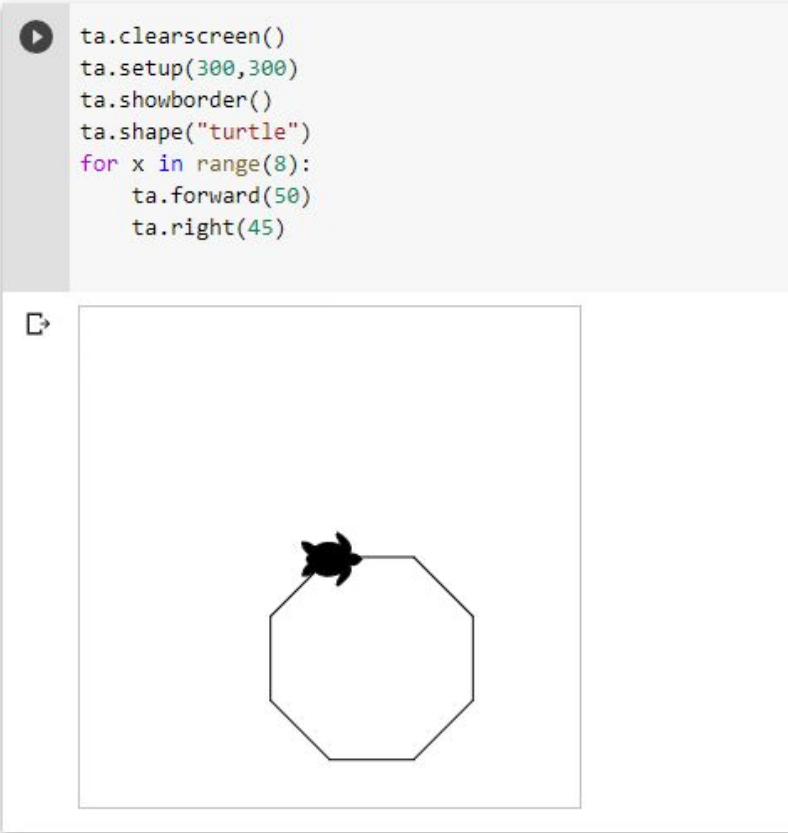


# Turtle Graphics: Using Loops to Draw Designs (2 of 4)

- This `for` loop iterates eight times to draw the octagon:

```
for x in range(8):  
    ta.forward(50)  
    ta.right(45)
```

```
3 s ✓ ▶ ta.clearscreen()  
ta.setup(300,300)  
ta.showborder()  
ta.shape("turtle")  
for x in range(8):  
    ta.forward(50)  
    ta.right(45)
```



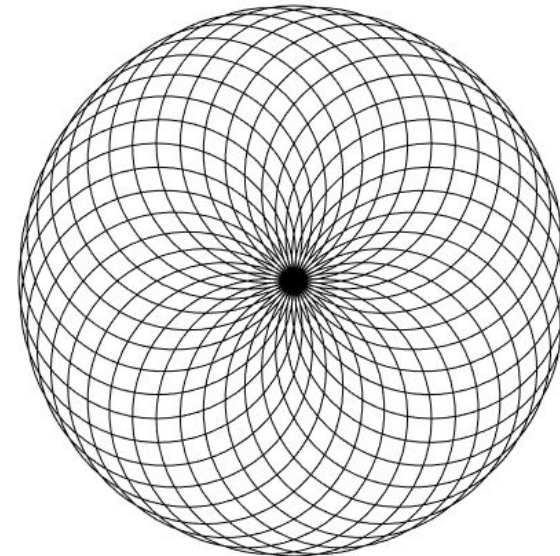
# Turtle Graphics: Using Loops to Draw Designs (3 of 4)

- You can create interesting designs by repeatedly drawing a simple shape, with the turtle tilted at a slightly different angle each time it draws the shape.

```
ta.clearscreen()
ta.speed(100)

NUM_CIRCLES = 36    # Number of circles to draw
RADIUS = 100       # Radius of each circle
ANGLE = 10         # Angle to turn

for x in range(NUM_CIRCLES):
    ta.circle(RADIUS)
    ta.left(ANGLE)
```



# Summary

- This chapter covered:
  - Repetition structures, including:
    - Condition-controlled loops
    - Count-controlled loops
    - Nested loops
  - Infinite loops and how they can be avoided
  - `range` function as used in `for` loops
  - Calculating a running total and augmented assignment operators
  - Use of sentinels to terminate loops
  - Using loops to draw turtle graphic designs



# Corso di **STATISTICA, INFORMATICA, ELABORAZIONE DELLE INFORMAZIONI**

Modulo di Sistemi di Elaborazione delle Informazioni

## Strutture Iterative



**UNIVERSITÀ DEGLI STUDI DELLA BASILICATA**



Docente:  
**Domenico Daniele Bloisi**

