Corso di STATISTICA, INFORMATICA, ELABORAZIONE DELLE INFORMAZIONI

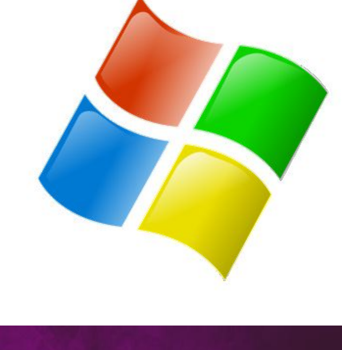
UNIVERSITÀ DEGLI STUDI DELLA BASILICATA

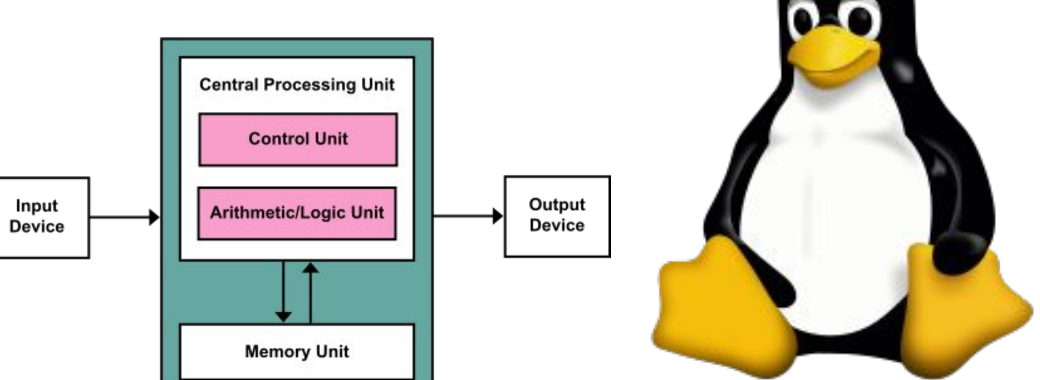
Modulo di Sistemi di Elaborazione delle Informazioni

Funzioni
parte 2

Docente:

Domenico Daniele Bloisi



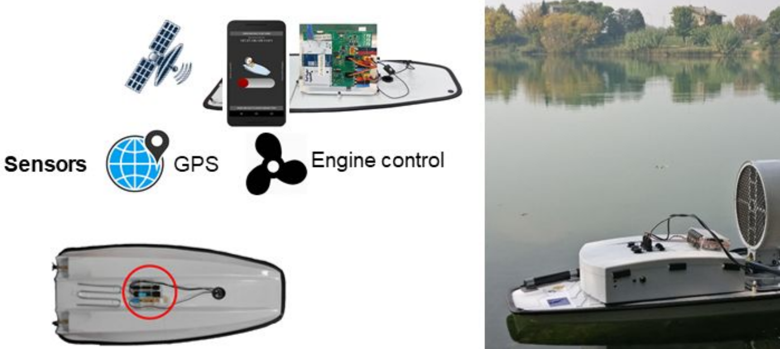


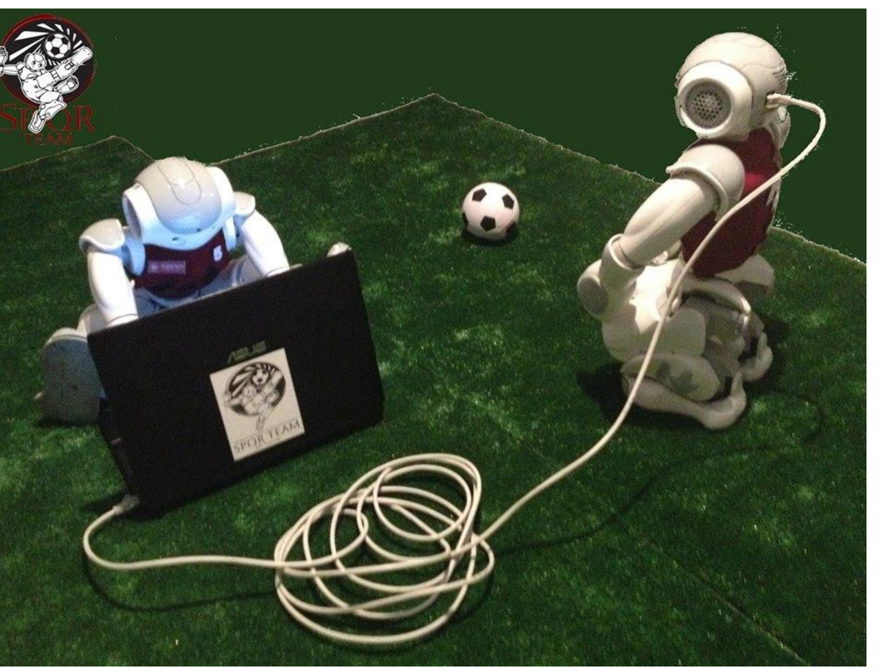
ubuntu

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



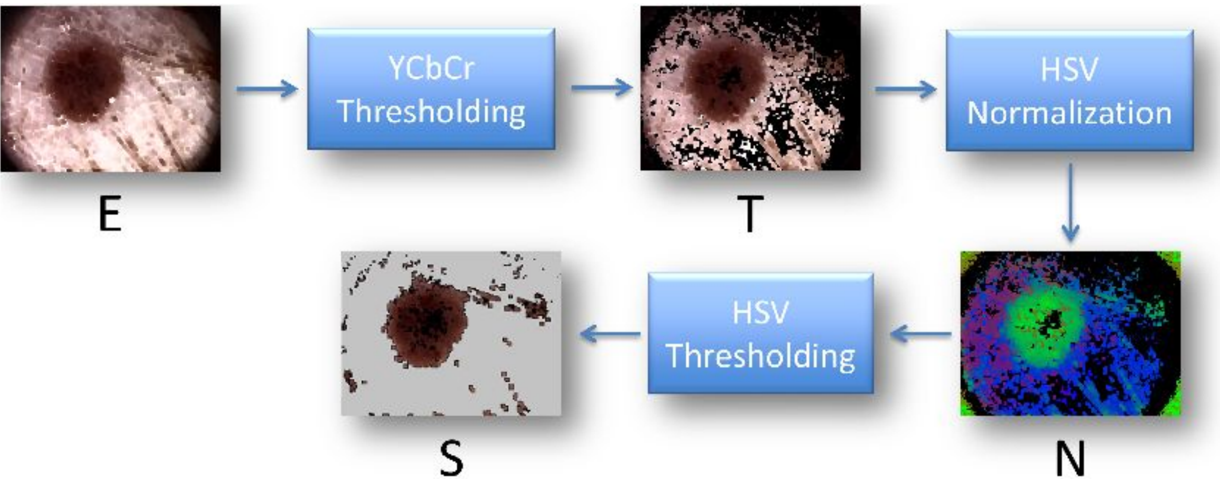


Interessi di ricerca

- Intelligent surveillance
- Robot vision
- Medical image analysis

https://youtu.be/2KHNZX7UIWQ

https://youtu.be/9a70Ucgbi_U



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, II 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

Recap

Definizione di nuove funzioni

La definizione di una nuova funzione è composta dai seguenti elementi:

- il nome della funzione
- il numero dei suoi argomenti
- la sequenza di istruzioni, detta corpo della funzione, che dovranno essere eseguite quando la funzione sarà chiamata

La definizione di una nuova funzione avviene attraverso l'uso della keyword `def`

def

Sintassi:

```
def nome_funzione (par1, ..., parn):
    corpo_della_funzione
```

- `nome_funzione` è un nome simbolico scelto dal programmatore, con gli stessi vincoli a cui sono soggetti i nomi delle variabili
- `par1, ..., parn` sono nomi (scelti dal programmatore) di variabili, dette parametri della funzione, alle quali l'interprete assegnerà i valori degli argomenti che verranno indicati nella chiamata della funzione
- `corpo_della_funzione` è una sequenza di una o più istruzioni qualsiasi, ciascuna scritta in una riga distinta, con un rientro di almeno un carattere, identico per tutte le istruzioni

La prima riga della definizione (contenente i nomi della funzione e dei parametri) è detta intestazione della funzione.

`return`

Per concludere l'esecuzione di una funzione e indicare il valore che la funzione dovrà restituire come risultato della sua chiamata si usa l’istruzione `return`.

Sintassi:

```
return espressione
```

dove `espressione` è un'espressione Python qualsiasi.

L’istruzione `return` può essere usata solo solo all’interno di una funzione.

Se una funzione non deve restituire alcun valore:

- l’istruzione `return` può essere usata, senza l’indicazione di alcuna espressione, per concludere l'esecuzione della funzione
- se non si usa l’istruzione `return`, l’esecuzione della funzione terminerà dopo l'esecuzione dell'ultima istruzione del corpo

Definizione e chiamata di una funzione

L'esecuzione dell'istruzione `def` non comporta l'esecuzione delle istruzioni della funzione: tali istruzioni verranno eseguite solo attraverso una chiamata della funzione.

L'istruzione `def` dovrà essere eseguita una sola volta, prima di qualsiasi chiamata della funzione. In caso contrario, il nome della funzione non sarà riconosciuto dall'inteprete e la chiamata produrrà un messaggio di errore.

Esecuzione della chiamata di funzione

L'interprete esegue la chiamata di una funzione nel modo seguente:

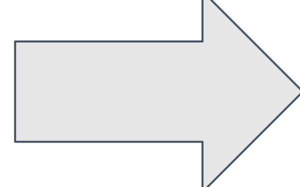
1. copia il valore di ciascun argomento nel parametro corrispondente (quindi tali variabili possiedono già un valore nel momento in cui inizia l'esecuzione della funzione)
2. esegue le istruzioni del corpo della funzione, fino all'istruzione `return` oppure fino all'ultima istruzione del corpo
3. se l'eventuale istruzione `return` è seguita da un'espressione, restituisce il valore di tale espressione come risultato della chiamata

Esercizio 11

Scrivere un programma che
- chieda all’utente di inserire giorno e mese corrente tramite tastiera
- utilizzi una funzione per calcolare la stagione corrente
- stampi il risultato

Possibile Soluzione Esercizio 11

```
def main():
  m = int(input("Inserire il mese corrente [1-12]: "))
  g = int(input("Inserisci il giorno corrente [1-31]: "))
  s = calcola_stagione(m, g)
  print("Siamo in", s+"!")
```



Possibile Soluzione Esercizio 11

	STAGIONI ASTRONOMICHE	STAGIONI METEOROLOGICHE
	21 marzo – 20 giugno	1 marzo – 31 maggio
	21 giugno – 22 settembre	1 giugno – 31 agosto
	23 settembre – 21 dicembre	1 settembre – 30 novembre
	22 dicembre – 21 marzo	1 dicembre – 28 (29) febbraio

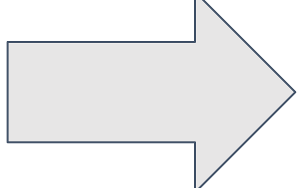
https://www.focus.it/scienza/scienze/stagioni-astronomiche-e-meteorologiche

Possibile Soluzione Esercizio 11

```
def calcola_stagione(mese, giorno):
  if mese >= 3 and mese <= 6:
    if mese == 3:
      if giorno >= 21:
        stagione = "primavera"
      else:
        stagione = "inverno"
    elif mese == 6:
      if giorno <= 20:
        stagione = "primavera"
      else:
        stagione = "estate"
    else:
      stagione = "primavera"
```

Possibile Soluzione Esercizio 11

```
elif mese >= 7 and mese <= 9:
  if mese == 9:
    if giorno >= 23:
      stagione = "autunno"
    else:
      stagione = "primavera"
  else:
    stagione = "estate"
```



Possibile Soluzione Esercizio 11

```
    elif mese >= 10 and mese <= 12:
        if mese == 12:
            if giorno >= 22:
                stagione = "inverno"
            else:
                stagione = "autunno"
        else:
            stagione = "autunno"
    else:
        stagione = "inverno"
    return stagione

main()
```

Possibile Soluzione Esercizio 11

```
        stagione = "primavera"
    else:
      stagione = "estate"
  elif mese >= 10 and mese <= 12:
    if mese == 12:
      if giorno >= 22:
        stagione = "inverno"
      else:
        stagione = "autunno"
    else:
      stagione = "autunno"
  else:
    stagione = "inverno"
  return stagione

main()
```

```
Inserire il mese corrente [1-12]: 12
Inserisci il giorno corrente [1-31]: 25
Siamo in inverno!
```

Using the `pass` Keyword

- You can use the `pass` keyword to create empty functions
- The `pass` keyword is ignored by the Python interpreter
- This can be helpful when designing a program

```
def step1():
    pass

def step2():
    pass
```

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Using the `pass` Keyword

```
def main():
  codice_fiscale = input("inserire cf: ")
  invia_email(codice_fiscale)
  print("ok inviata")

def invia_email(cf):
  pass

main()
```

```
inserire cf: BLSDNC
ok inviata
```

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Local Variables (1 of 2)

- Local variable: variable that is assigned a value inside a function
  - Belongs to the function in which it was created
    - Only statements inside that function can access it, error will occur if another function tries to access the variable
- Scope: the part of a program in which a variable may be accessed
  - For local variable: function in which created

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Variabili locali

I parametri di una funzione e le eventuali altre variabili alle quali viene assegnato un valore all'interno di essa sono dette locali, cioè vengono create dall’interprete nel momento in cui la funzione viene eseguita (con una chiamata) e vengono distrutte quando l'esecuzione della funzione termina.

Variabili locali: esempio

```python
def stampa_quadrato(x):
  x_quadro = x ** 2
  print(x_quadro)

def main():
  a = 5
  stampa_quadrato(a)
  print(a)

main()
```

Cosa stampa questo programma?

Variabili locali: esempio

```python
def stampa_quadrato(x):
  x_quadro = x ** 2
  print(x_quadro)

def main():
  a = 5
  stampa_quadrato(a)
  print(a)

main()
```

```
25
5
```

Variabili locali: esempio

```
def stampa_quadrato(x):
  a = x ** 2
  print(a)

def main():
  a = 5
  stampa_quadrato(a)
  print(a)

main()
```

e questo?

Variabili locali: esempio

```
def stampa_quadrato(x):
  a = x ** 2
  print(a)

def main():
  a = 5
  stampa_quadrato(a)
  print(a)

main()
```

```
25
5
```

Local Variables (2 of 2)

- Local variable cannot be accessed by statements inside its function which precede its creation
- Different functions may have local variables with the same name
  - Each function does not see the other function’s local variables, so no confusion

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Passing Arguments to Functions (1 of 4)

- Argument: piece of data that is sent into a function
  - Function can use argument in calculations
  - When calling the function, the argument is placed in parentheses following the function name

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Passing Arguments to Functions (2 of 4)

```
def main():
    value = 5
    show_double(value)

def show_double(number):
    result = number * 2
    print(result)
```

Figure 5-13 The `value` variable is passed as an argument

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Passing Arguments to Functions (3 of 4)

- Parameter variable: variable that is assigned the value of an argument when the function is called
  - The parameter and the argument reference the same value
  - General format:

```
def function_name(parameter):
```

  - Scope of a parameter: the function in which the parameter is used

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Passing Arguments to Functions (4 of 4)

```python
def main():
    value = 5
    show_double(value)

def show_double(number):
    result = number * 2
    print(result)
```

value

5

number

Figure 5-14 The `value` variable and the `number` parameter reference the same value

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Passing Multiple Arguments (1 of 2)

- Python allows writing a function that accepts multiple arguments
  - Parameter list replaces single parameter
    - Parameter list items separated by comma
- Arguments are passed *by position* to corresponding parameters
  - First parameter receives value of first argument, second parameter receives value of second argument, etc.

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Passing Multiple Arguments (2 of 2)

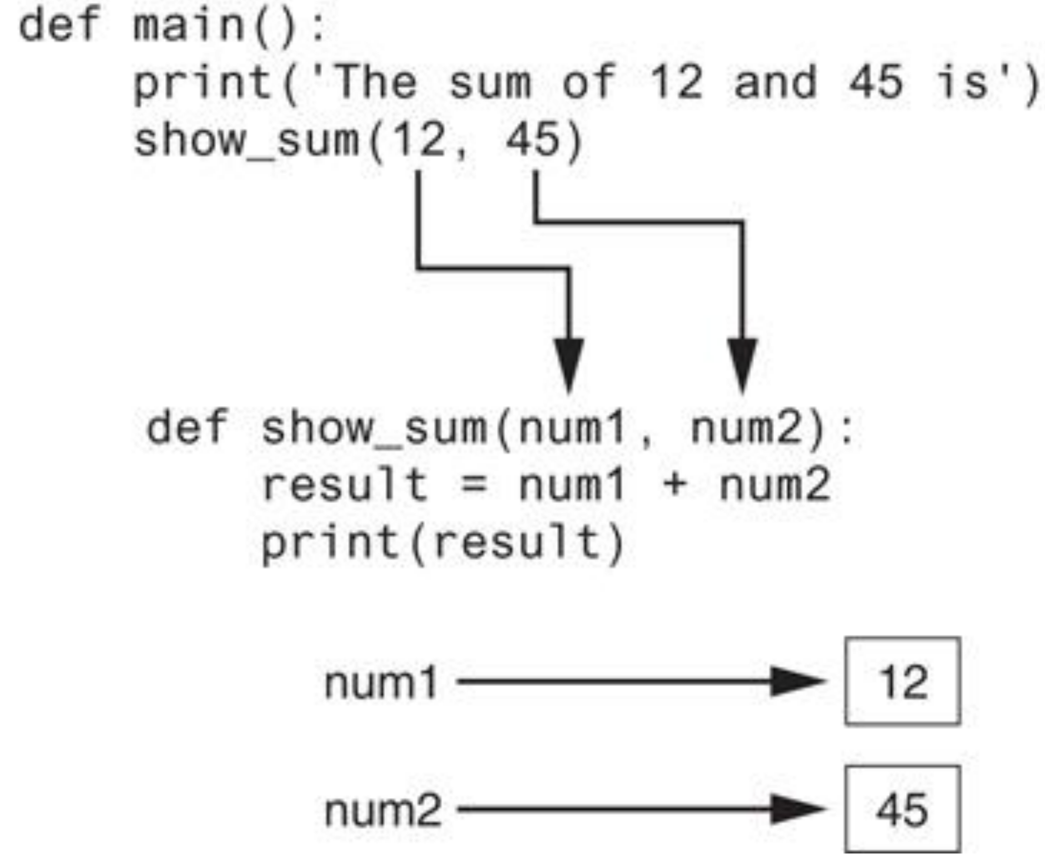


Figure 5-16 Two arguments passed to two parameters

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Making Changes to Parameters (1 of 3)

- Changes made to a parameter value within the function do not affect the argument
  - Known as *pass by value*
  - Provides a way for unidirectional communication between one function and another function
    - Calling function can communicate with called function

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Making Changes to Parameters (2 of 3)

```
def main():
    value = 99
    print(f'The value is {value}.')
    change_me(value)
    print(f'Back in main the value is {value}.')

def change_me(arg):
    print('I am changing the value.')
    arg = 0
    print(f'Now the value is {arg}.')
```


Figure 5-17 The `value` variable is passed to the change_me function

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Making Changes to Parameters (3 of 3)

- Figure 5-18
  - The `value` variable passed to the `change_me` function cannot be changed by it

```
def main():
    value = 99
    print(f'The value is {value}.')
    change_me(value)
    print(f'Back in main the value is {value}.')

def change_me(arg):
    print('I am changing the value.')
    arg = 0
    print(f'Now the value is {arg}.')
```


Figure 5-18 The `value` variable is passed to the change_me function

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Keyword Arguments

- Keyword argument: argument that specifies which parameter the value should be passed to
  - Position when calling function is irrelevant
  - General Format:
  - `function_name(parameter=value)`
- Possible to mix keyword and positional arguments when calling a function
  - Positional arguments must appear first

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Keyword Arguments

```
[6] def stampa_data(giorno, mese, anno):
      print("La data di oggi è",
            "giorno:", giorno,
            "mese:", mese,
            "anno:", anno)

    stampa_data(6,12,2022)
```

```
La data di oggi è giorno: 6 mese: 12 anno: 2022
```

```
[7] stampa_data(12,6,2022)
```

```
La data di oggi è giorno: 12 mese: 6 anno: 2022
```

```
stampa_data(mese=12, giorno=6, anno=2022)
```

```
La data di oggi è giorno: 6 mese: 12 anno: 2022
```

Keyword Arguments

```
[9] stampa_data(12, giorno=6, anno=2022)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-c7afba6e508f> in <module>
----> 1 stampa_data(12, giorno=6, anno=2022)

TypeError: stampa_data() got multiple values for argument 'giorno'

SEARCH STACK OVERFLOW
```

```
[10] stampa_data(giorno=6, 12, anno=2022)

  File "<ipython-input-10-dcde78be8518>", line 1
    stampa_data(giorno=6, 12, anno=2022)
                         ^
SyntaxError: positional argument follows keyword argument

SEARCH STACK OVERFLOW
```

```
stampa_data(6, mese=12, anno=2022)

La data di oggi è giorno: 6 mese: 12 anno: 2022
```

ghts Reserved

Global Variables and Global Constants (1 of 2)

- Global variable: created by assignment statement written outside all the functions
  - Can be accessed by any statement in the program file, including from within a function
  - If a function needs to assign a value to the global variable, the global variable must be redeclared within the function
    - General format: `global` *`variable_name`*

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Variabili globali

Se invece all’interno di una funzione il nome di una variabile (che non sia uno dei parametri) compare in una espressione senza che in precedenza nella funzione sia stato assegnato a essa alcun valore, tale variabile è considerata globale, cioè l’interprete assume che il suo valore sia stato definito nelle istruzioni precedenti la chiamata della funzione.

In questo modo, le istruzioni di una funzione possono accedere al valore di variabile definita nel programma chiamante (se tale variabile non esiste si ottiene un messaggio di errore).

Variabili globali

```
# Crea una variabile globale.
my_value = 10

# La funzione show_value stampa
# il valore della variabile globale.
def show_value():
  print(my_value)

# Chiama la funzione show_value.
show_value()
print(my_value)
```

```
10
10
```

Variabili globali

```
# Crea una variabile globale.
my_value = 10

# La funzione show_value stampa
# il valore della variabile globale.
def show_value():
  my_value = 15
  print(my_value)

# Chiama la funzione show_value.
show_value()
print(my_value)
```

```
15
10
```

Variabili globali

```
# Crea una variabile globale.
my_value = 10

# La funzione show_value stampa
# il valore della variabile globale.
def show_value():
  global my_value
  my_value = 15
  print(my_value)

# Chiama la funzione show_value.
show_value()
print(my_value)
```

```
15
15
```

Global Variables Are Bad

http://wiki.c2.com/?GlobalVariablesAreBad

In generale, è preferibile evitare l'uso di variabili globali nelle funzioni, poiché la loro presenza rende più difficile assicurare la correttezza di un programma.

Global Variables and Global Constants (2 of 2)

- Reasons to avoid using global variables:
  - Global variables making debugging difficult
    - Many locations in the code could be causing a wrong variable value
  - Functions that use global variables are usually dependent on those variables
    - Makes function hard to transfer to another program
  - Global variables make a program hard to understand

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Global Constants

- Global constant: global name that references a value that cannot be changed
  - Permissible to use global constants in a program
  - To simulate global constant in Python, create global variable and do not re-declare it within functions

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Global Constants

```
MAGGIORE_ETA = 18

def is_adulto(eta):
  if eta >= MAGGIORE_ETA:
    return True
  else:
    return False

valore = int(input("inserisci la tua età: "))
if is_adulto(valore):
  print("Sei maggiorenne")
else:
  print("Sei minorenne")
```

```
inserisci la tua età: 24
Sei maggiorenne
```

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Categoria	*Pressione arteriosa in mm Hg*	
	Sistolica	**Diastolica**
Ottimale	< 120	< 80
Normale	< 130	< 85
Normale – alta	130 – 139	85 – 89
Ipertensione di Grado 1 borderline	140 – 149	90 – 94
Ipertensione di Grado 1 lieve	150 – 159	95 – 99
Ipertensione di Grado 2 moderata	160 – 179	100 – 109
Ipertensione di Grado 3 grave	≥ 180	≥ 110
Ipertensione sistolica isolata borderline	140 – 149	< 90
Ipertensione sistolica isolata	≥ 150	< 90

Standard Library Functions and the `import` Statement (1 of 3)

- Standard library: library of pre-written functions that comes with Python
  - *Library functions* perform tasks that programmers commonly need
    - Example: `print`, `input`, `range`
      - Viewed by programmers as a “black box”
- Some library functions built into Python interpreter
  - To use, just call the function

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Esempi di funzioni built-in

`len(stringa)`
restituisce il numero di caratteri di una stringa

`abs(numero)`
restituisce il valore assoluto di un numero

`str(espressione)`
restituisce una stringa composta dalla sequenza di caratteri corrispondenti alla rappresentazione del valore di `espressione` (che può essere di un qualsiasi tipo: numero, stringa, valore logico, ecc.)

Esempi di funzioni built-in

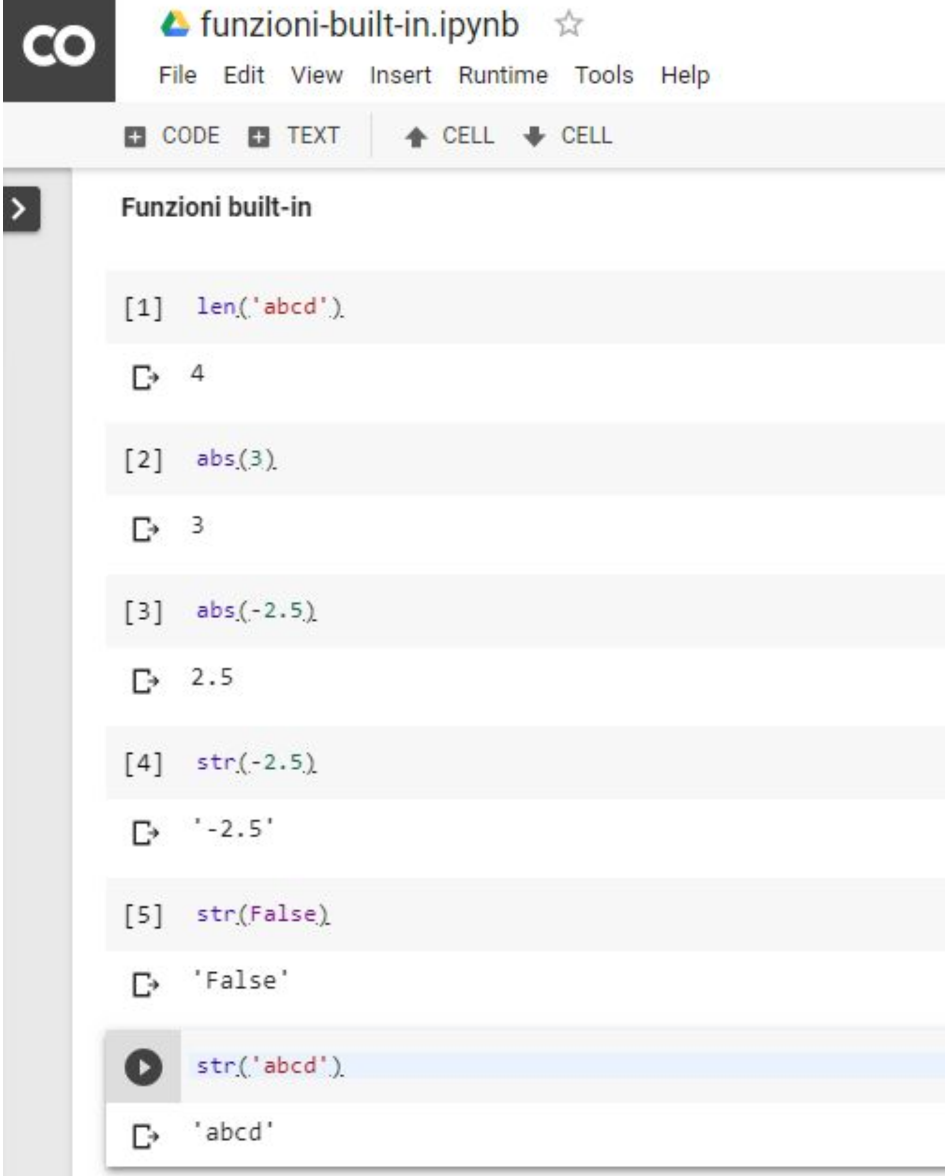
`int(numero)`
restituisce la parte intera di un numero

`float(numero)`
restituisce il valore di numero come numero frazionario (floating point); può essere usata per evitare che la divisione tra interi produca la sola parte intera del quoziente, per es.: `float(2) / 3`

`int(stringa)`
Se stringa contiene la rappresentazione di un numero intero, restituisce il numero corrispondente a tale valore; in caso contrario produce un errore

`float(stringa)`
Se stringa contiene la rappresentazione di un numero qualsiasi (sia intero che frazionario), restituisce il suo valore espresso come numero frazionario; in caso contrario produce un errore

Esempi funzioni built-in



Esempi funzioni built-in

funzioni-built-in.ipynb

File Edit View Insert Runtime Tools Help

CODE TEXT CELL CELL

```
[7] int(-3.25)
-3

[8] float(6)
6.0

[9] 2 / 3
0.6666666666666666

[10] 2 // 3
0

int('2.5')
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-51dfdf260f91> in <module>()
----> 1 int('2.5')

ValueError: invalid literal for int() with base 10: '2.5'
```

SEARCH STACK OVERFLOW

Standard Library Functions and the `import` Statement (2 of 3)

- <u>Modules</u>: files that stores functions of the standard library
  - Help organize library functions not built into the interpreter
  - Copied to computer when you install Python
- To call a function stored in a module, need to write an `import` statement
  - Written at the top of the program
  - Format: `import` *`module_name`*

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

La libreria `random`

funzione	descrizione
`random()`	genera un numero reale nell'intervallo [0, 1), da una distribuzione di probabilità **uniforme** (cioè, ogni valore di tale intervallo ha la stessa probabilità di essere "estratto")
`uniform(a,b)`	come sopra, nell'intervallo [*a*, *b*) (gli argomenti sono numeri qualsiasi)
`randint(a,b)`	genera un numero intero nell'insieme {*a*, . . . , *b*}, da una distribuzione di probabilità **uniforme** (gli argomenti devono essere numeri interi)

Ogni chiamata di tali funzioni produce un numero pseudo-casuale, indipendente (in teoria) dai valori prodotti dalle chiamate precedenti.

Generating Random Numbers (1 of 5)

- Random number are useful in a lot of programming tasks
- `random` module: includes library functions for working with random numbers
- Dot notation: notation for calling a function belonging to a module
  - Format: `module_name.function_name()`

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Generating Random Numbers (2 of 5)

- `randint` function: generates a random number in the range provided by the arguments
  - Returns the random number to part of program that called the function
  - Returned integer can be used anywhere that an integer would be used
  - You can experiment with the function in interactive mode

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

[19]
```
randint(1,100)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-19-9aa34682caf0> in <module>
----> 1 randint(1,100)

NameError: name 'randint' is not defined
```

SEARCH STACK OVERFLOW

```
import random

random.randint(1,100)
```

```
35
```

Generating Random Numbers (3 of 5)

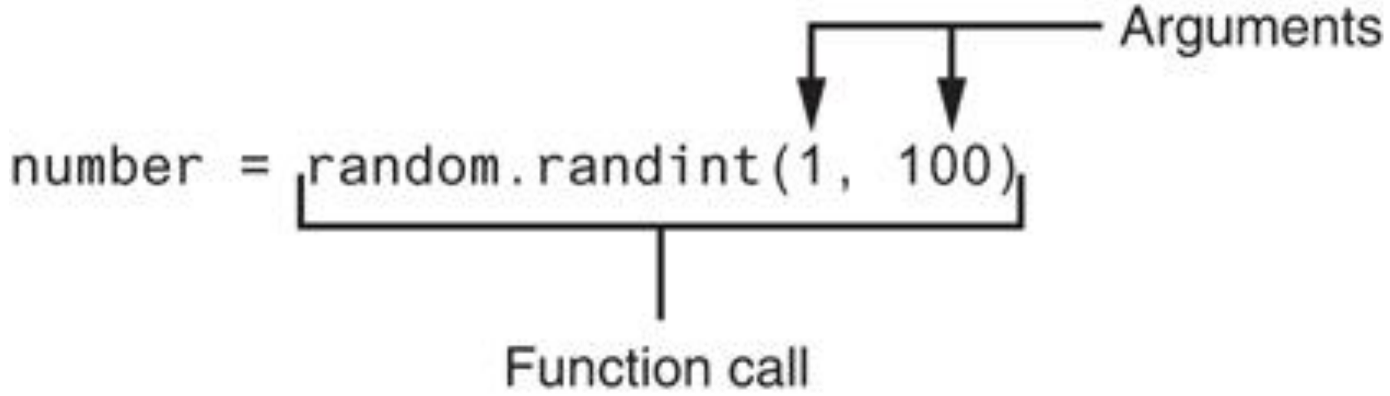


Figure 5-20 A statement that calls the random function

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Standard Library Functions and the `import` Statement (3 of 3)

Figure 5-19 A library function viewed as a black box

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

`from import`

Per poter chiamare una funzione di librerie come `math` e `random` è necessario utilizzare la combinazione `from import`

Sintassi:

```
from nome_libreria import nome_funzione
```

- `nome_libreria` è il nome simbolico di una libreria
- `nome_funzione` può essere:
  - il nome di una specifica funzione di tale libreria (questo consentirà di usare solo tale funzione)
  - il simbolo `*` indicante tutte le funzioni di tale libreria

Se la combinazione `from import` non viene usata correttamente, la chiamata di funzione produrrà un errore, come mostrato negli esempi seguenti.

from import

```
from random import randint

randint(1,100)
```

```
35
```

Esempi `random`

```
[2] from random import *
    random()

    0.6589012566357493

[3] random()

    0.8476015372012984

[4] random()

    0.09743656069098616

[5] uniform(-2, 2)

    -1.8376847371489315

[6] uniform(-2, 2)

    -1.9511529669296759

[7] randint(1, 10)

    7

    randint(1, 10)

    6
```

Writing Your Own Value-Returning Functions (1 of 2)

- To write a value-returning function, you write a simple function and add one or more `return` statements
  - Format: `return` *`expression`*
    - The value for *`expression`* will be returned to the part of the program that called the function
  - The expression in the `return` statement can be a complex expression, such as a sum of two variables or the result of another value-returning function

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Writing Your Own Value-Returning Functions (2 of 2)

Figure 5-23 Parts of the function

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

How to Use Value-Returning Functions

- Value-returning function can be useful in specific situations
  - Example: have function prompt user for input and return the user’s input
  - Simplify mathematical expressions
  - Complex calculations that need to be repeated throughout the program
- Use the returned value
  - Assign it to a variable or use as an argument in another function

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Returning Strings

- You can write functions that return strings
- For example:

```
def get_name():
    # Get the user’s name.
    name = input(‘Enter your name:’)
    # Return the name.
    return name
```


Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Returning Boolean Values

- Boolean function: returns either `True` or `False`
  - Use to test a condition such as for decision and repetition structures
    - Common calculations, such as whether a number is even, can be easily repeated by calling a function
  - Use to simplify complex input validation code

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Returning Multiple Values

- In Python, a function can return multiple values
  - Specified after the `return` statement separated by commas
    - Format: `return` *`expression1,`*
      `expression2, etc.`
  - When you call such a function in an assignment statement, you need a separate variable on the left side of the = operator to receive each returned value

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Returning `None` From a Function

- The special value `None` means “no value”
- Sometimes it is useful to return `None` from a function to indicate that an error has occurred

```
def divide(num1, num2):
    if num2 == 0:
        result = None
    else:
        result = num1 / num2
    return result
```

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

The `math` Module (1 of 3)

- <u>`math` module</u>: part of standard library that contains functions that are useful for performing mathematical calculations
  - Typically accept one or more values as arguments, perform mathematical operation, and return the result
  - Use of module requires an `import math` statement

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

The `math` Module (2 of 3)

Table 5-2 Many of the functions in the `math` module

`math` **Module Function**	**Description**
`acos(x)`	Returns the arc cosine of `x`, in radians.
`asin(x)`	Returns the arc sine of `x`, in radians.
`atan(x)`	Returns the arc tangent of `x`, in radians.
`ceil(x)`	Returns the smallest integer that is greater than or equal to `x`.
`cos(x)`	Returns the cosine of `x` in radians.
`degrees(x)`	Assuming `x` is an angle in radians, the function returns the angle converted to degrees.
`exp(x)`	Returns e^x
`floor(x)`	Returns the largest integer that is less than or equal to `x`.
`hypot(x, y)`	Returns the length of a hypotenuse that extends from (0, 0) to (`x, y`).
`log(x)`	Returns the natural logarithm of `x`.
`log10(x)`	Returns the base-10 logarithm of `x`.
`radians(x)`	Assuming `x` is an angle in degrees, the function returns the angle converted to radians.
`sin(x)`	Returns the sine of `x` in radians.
`sqrt(x)`	Returns the square root of `x`.
`tan(x)`	Returns the tangent of `x` in radians.

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Costanti matematiche

Oltre a varie funzioni, nella libreria `math` sono definite due variabili che contengono il valore (approssimato) delle costanti matematiche π (3,14. . .) ed *e* (la base dei logaritmi naturali: 2,71. . .):

- `pi`
- `e`

Per usare queste costanti è necessaria la combinazione `from import`, in una delle due versioni:

- `from math import *`
- `from math import nome_variabile`

dove `nome_variabile` dovrà essere `pi` oppure `e`

The `math` Module (3 of 3)

- The `math` module defines variables `pi` and `e`, which are assigned the mathematical values for *pi* and *e*
  - Can be used in equations that require these values, to get more accurate results
- Variables must also be called using the dot notation
  - Example:

```
circle_area = math.pi * radius**2
```

Copyright © 2021, 2018, 2015 Pearson Education, Inc. All Rights Reserved

Esempi `math`

```
[1] cos(pi / 2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-1c3767ed60fe> in <module>()
----> 1 cos(pi / 2)

NameError: name 'cos' is not defined
```

SEARCH STACK OVERFLOW

```
[2] from math import cos
    from math import pi

    cos(pi / 2)
```

```
6.123233995736766e-17
```

```
[6] from math import cos
    from math import pi

    if cos(pi / 2) == 0:
      print('OK')
```

```
from math import cos
from math import pi
import sys

if cos(pi / 2) < sys.float_info.epsilon:
  print('OK')
```

```
OK
```

Esercizio 12

Scrivere un programma che

- chieda all’utente di inserire un numero
- utilizzi una funzione per calcolare se il numero è pari o dispari
- stampi la stringa “è pari” oppure “è dispari”

Esercizio 13

Scrivere un programma che
- chieda all’utente di inserire un numero
- utilizzi una funzione che calcoli il fattoriale di tale numero
- stampi il risultato

Esercizio 14

Scrivere un programma che

- chieda all’utente di inserire una stringa
- utilizzi la funzione len per calcolare la lunghezza
- stampi il risultato
- ripeta le operazioni precedenti finchè l’utente non digiti “ora basta”

Corso di STATISTICA, INFORMATICA, ELABORAZIONE DELLE INFORMAZIONI

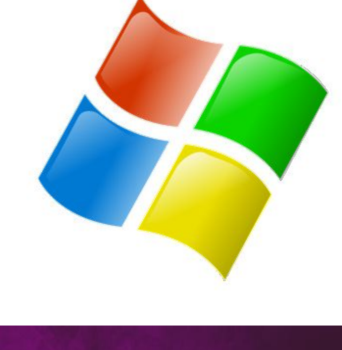
UNIVERSITÀ DEGLI STUDI DELLA BASILICATA

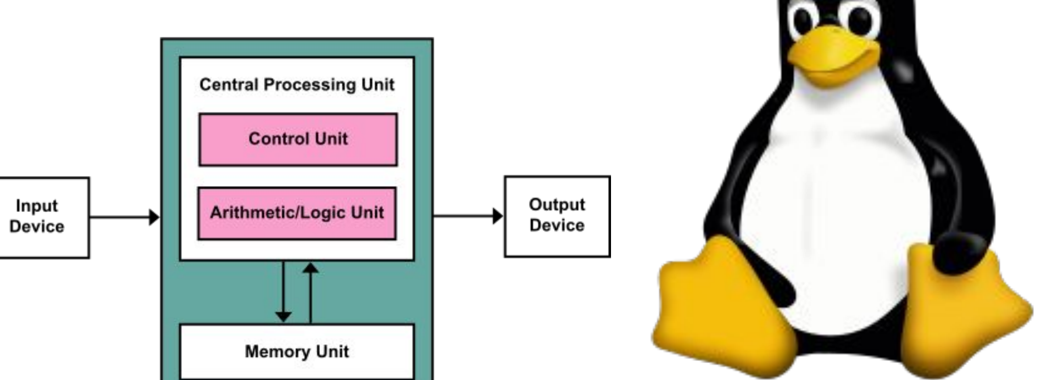
Modulo di Sistemi di Elaborazione delle Informazioni

Funzioni
parte 2

Docente:

Domenico Daniele Bloisi





ubuntu