

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

Modulo di Sistemi di Elaborazione delle Informazioni

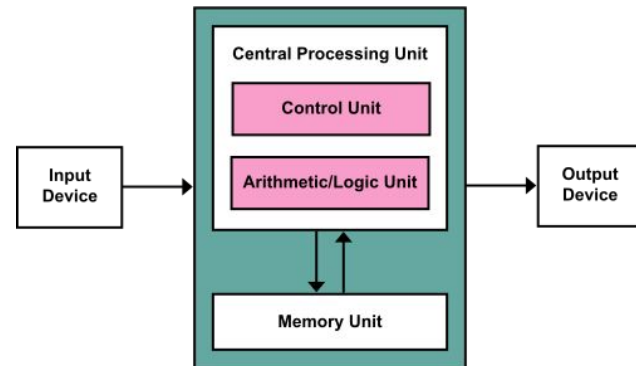
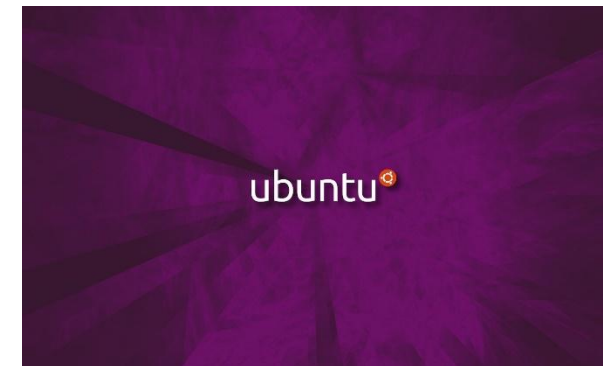
# Strutture Decisionali e Logica Booleana



**UNIVERSITÀ DEGLI STUDI DELLA BASILICATA**

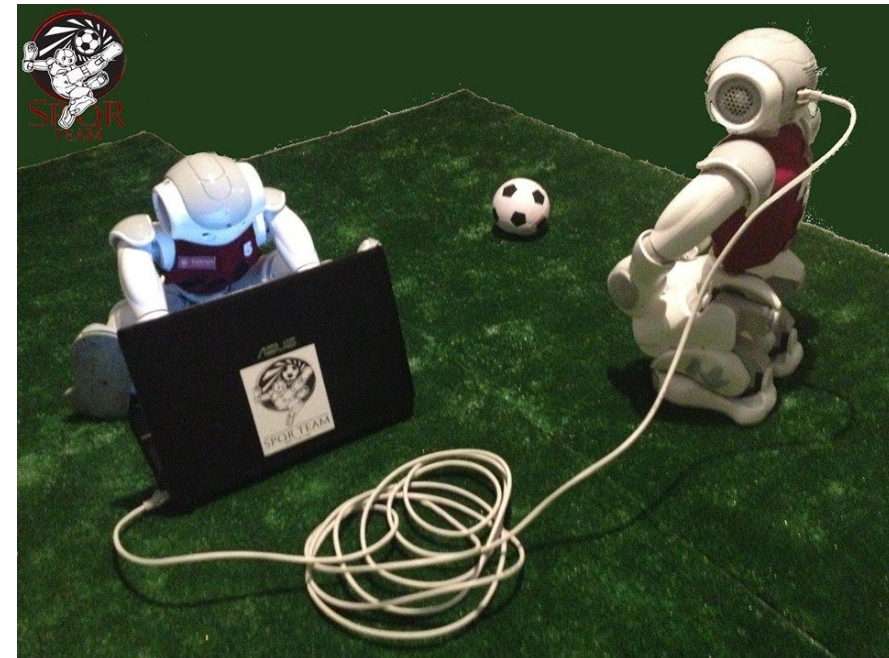
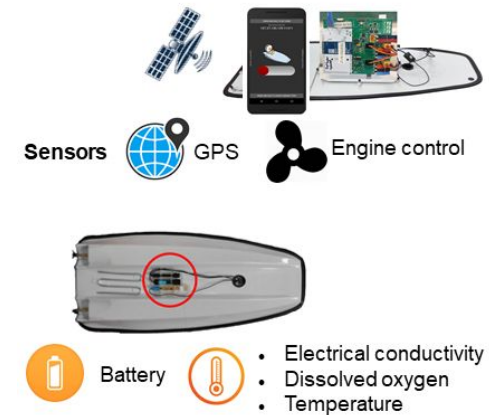


Docente:  
**Domenico Daniele Bloisi**



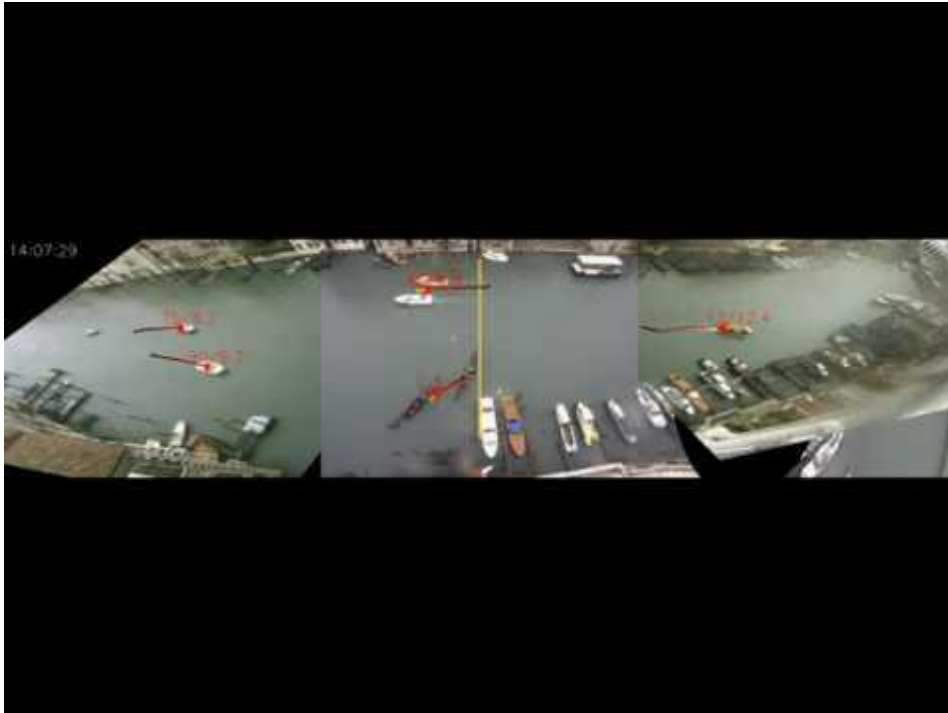
# Domenico Daniele Bloisi

- 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

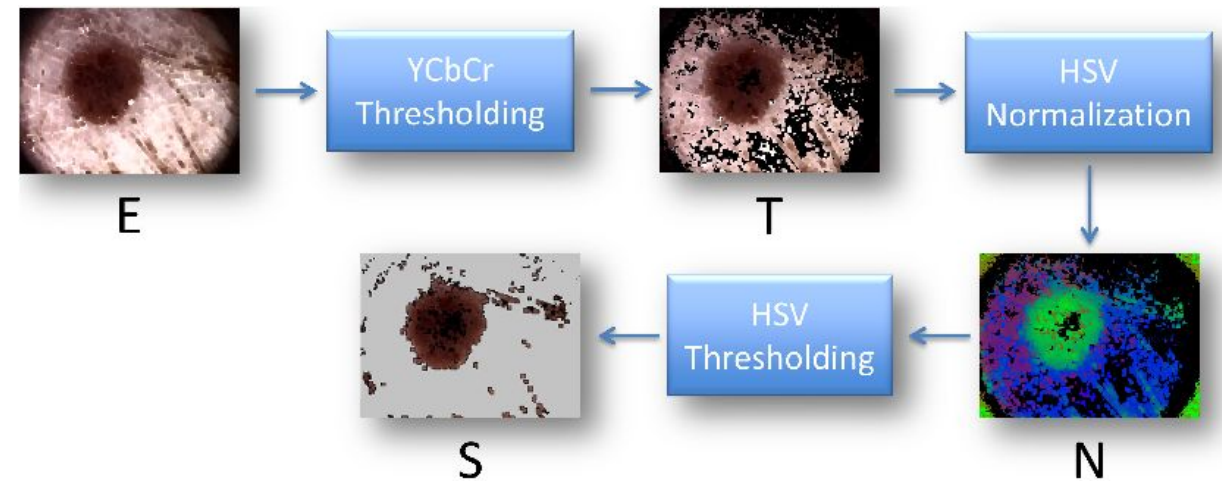
- Intelligent surveillance
- Robot vision
- Medical image analysis



[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

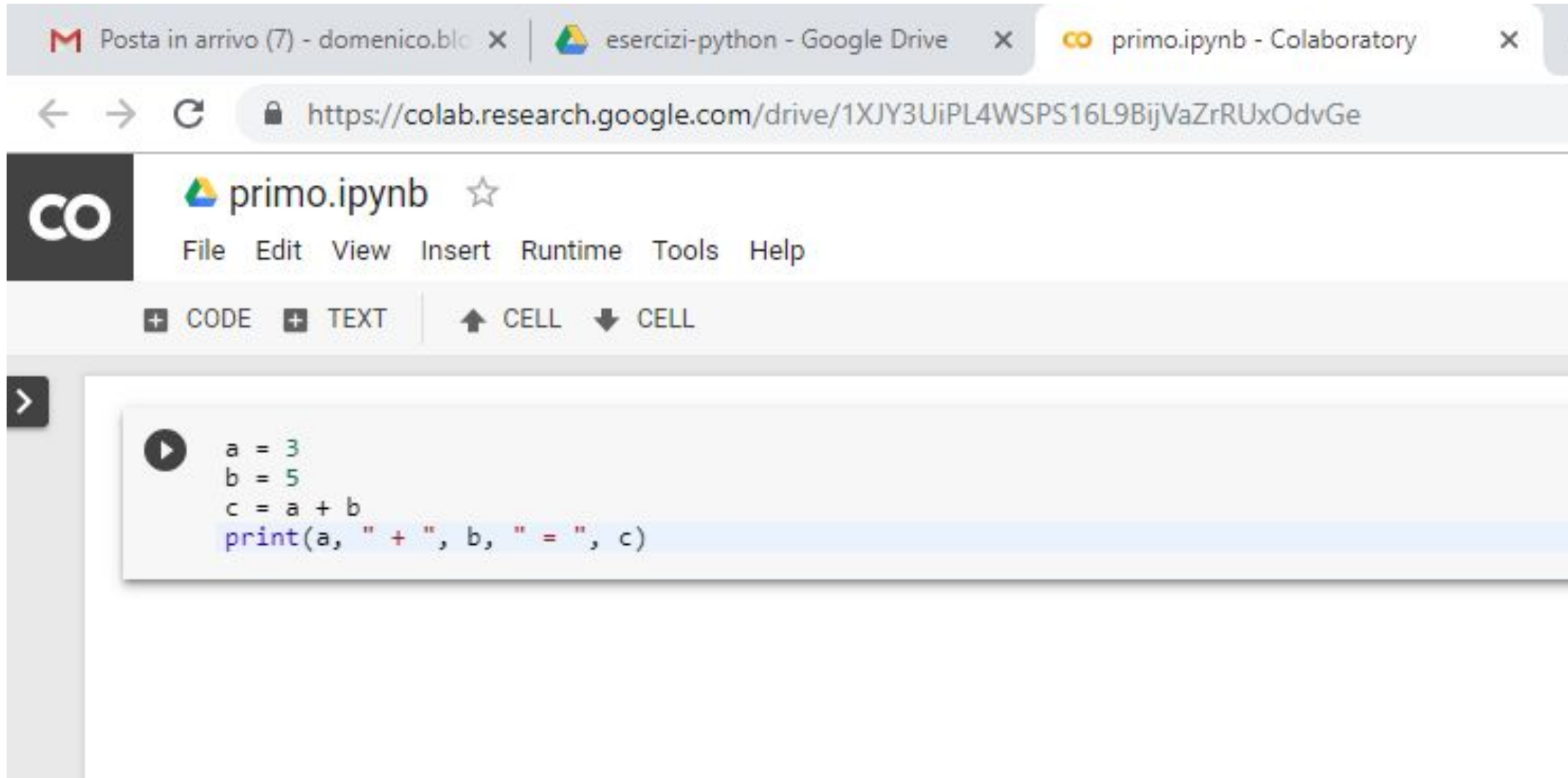
Per prenotare un appuntamento inviare  
una email a  
[domenico.bloisi@unibas.it](mailto:domenico.bloisi@unibas.it)



Recap



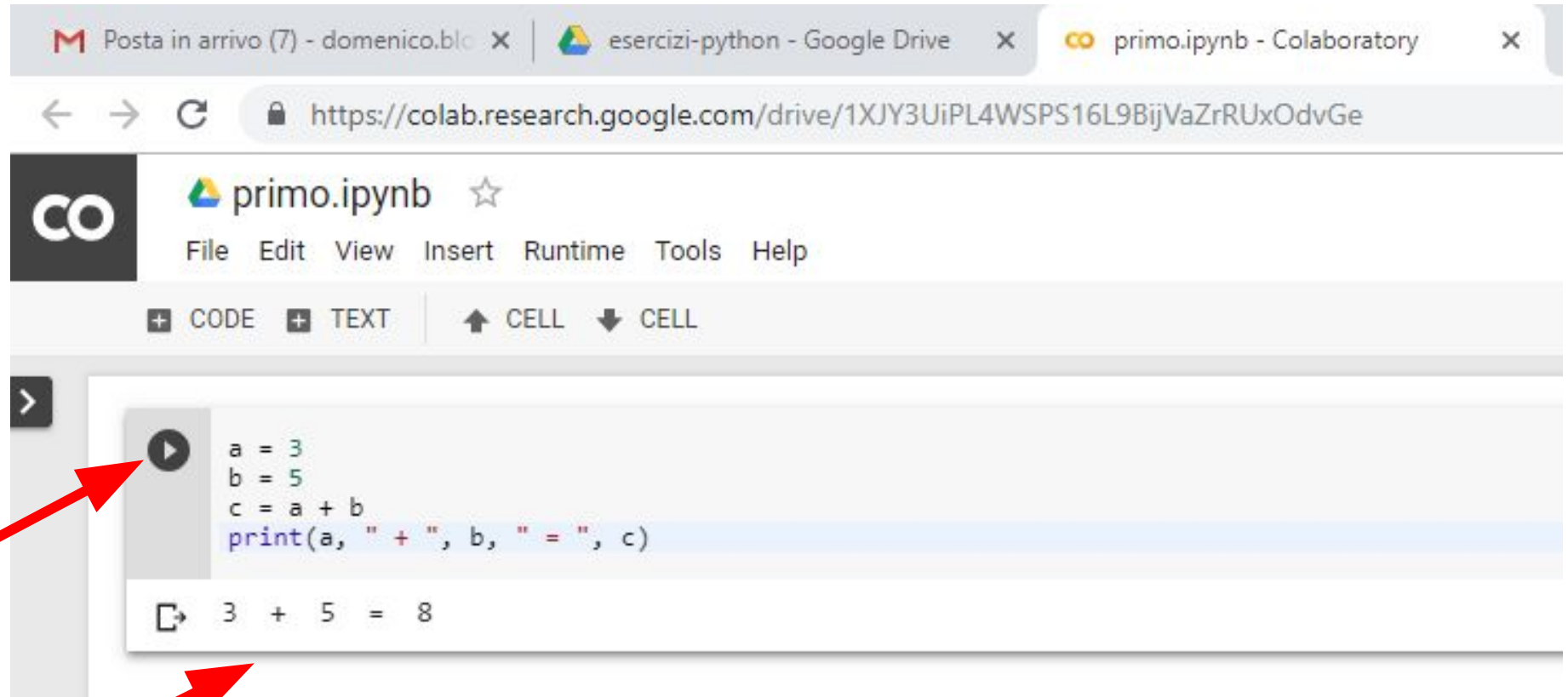
# Scriviamo il primo programma Colab



The screenshot shows a web browser with three tabs: "Posta in arrivo (7) - domenico.bl...", "esercizi-python - Google Drive", and "primo.ipynb - Colaboratory". The address bar shows the URL "https://colab.research.google.com/drive/1XJY3UiPL4WSPS16L9BijVaZrRUxOdvGe". The notebook interface includes a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". Below the menu bar are buttons for "+ CODE", "+ TEXT", "↑ CELL", and "↓ CELL". The main area contains a code cell with a play button icon and the following Python code:

```
a = 3
b = 5
c = a + b
print(a, " + ", b, " = ", c)
```

# Esecuzione del primo programma Colab



The screenshot shows a web browser with three tabs: 'Posta in arrivo (7) - domenico.blo', 'esercizi-python - Google Drive', and 'primo.ipynb - Colaboratory'. The address bar shows the URL 'https://colab.research.google.com/drive/1XJY3UiPL4WSPS16L9BijVaZrRUxOdvGe'. The notebook interface includes a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu are buttons for '+ CODE', '+ TEXT', '↑ CELL', and '↓ CELL'. A code cell is selected, containing the following Python code:

```
a = 3
b = 5
c = a + b
print(a, " + ", b, " = ", c)
```

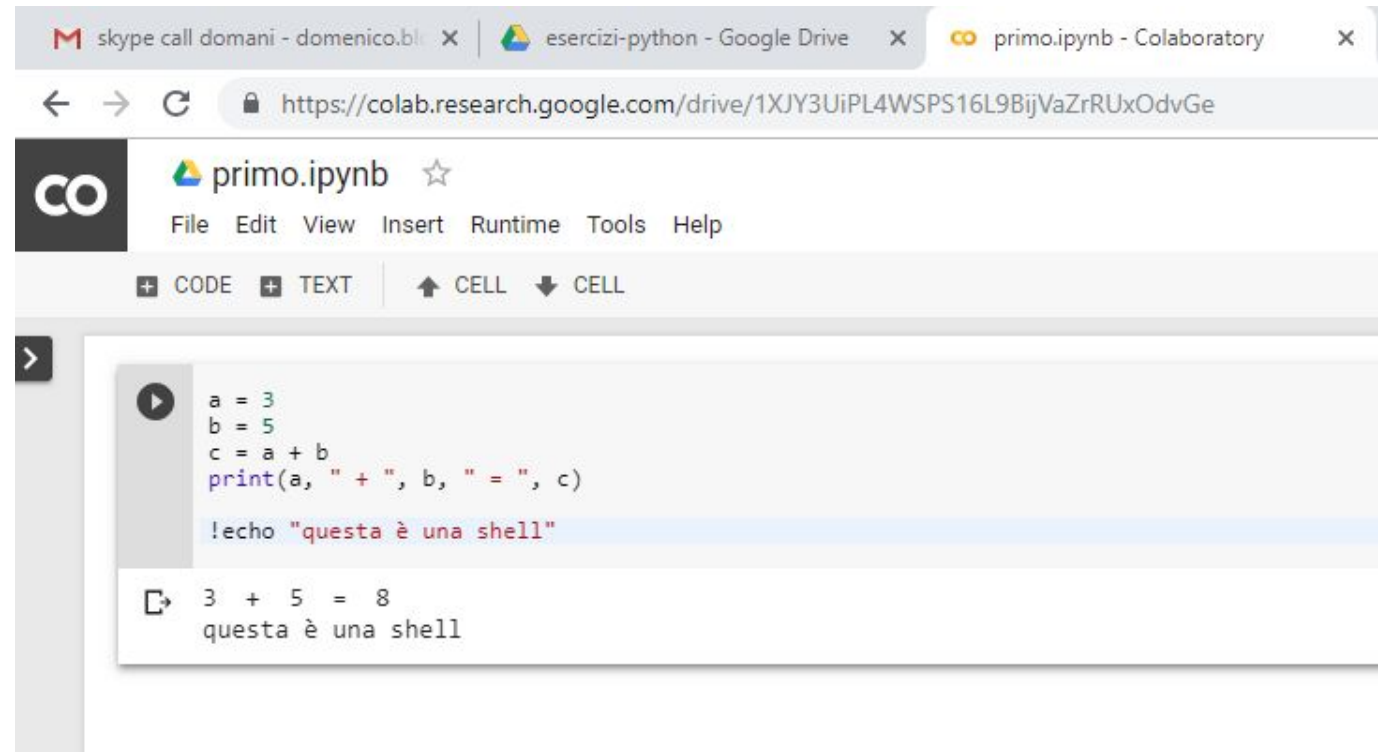
The output of the cell is displayed below the code: '3 + 5 = 8'. Two red arrows point to the 'Run' button (a play icon) and the output text.

Selezionando 'Run cell' possiamo eseguire il nostro codice e ottenere l'output

# Comandi di shell in Colab

Possiamo utilizzare comandi di shell facendoli precedere dal simbolo ' ! '

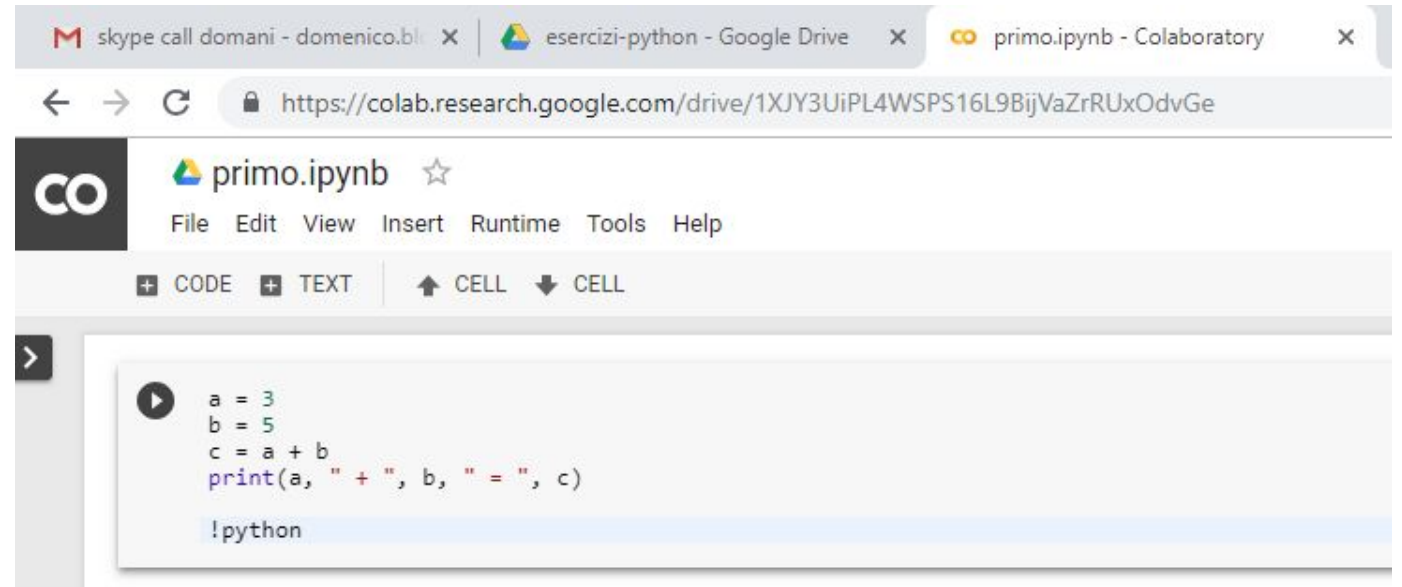
Per esempio, per lanciare il comando `echo` possiamo inserire il simbolo ' ! ' e poi procedere come se avessimo a disposizione una shell reale



```
skype call domani - domenico.bl x | esercizi-python - Google Drive x | primo.ipynb - Colaboratory x
https://colab.research.google.com/drive/1XJY3UiPL4WSPS16L9BijVaZrRUxOdvGe
primo.ipynb
File Edit View Insert Runtime Tools Help
CODE TEXT CELL CELL
a = 3
b = 5
c = a + b
print(a, " + ", b, " = ", c)
!echo "questa è una shell"
3 + 5 = 8
questa è una shell
```

# Interprete Python in Colab

Per lanciare l'interprete Python su Colab possiamo scrivere `!python`



The screenshot shows a web browser window with three tabs: "skype call domani - domenico.bl...", "esercizi-python - Google Drive", and "primo.ipynb - Colaboratory". The address bar shows the URL "https://colab.research.google.com/drive/1XJY3UiPL4WSPS16L9BijVaZrRUxOdvGe". The notebook interface includes a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". Below the menu bar are buttons for "CODE", "TEXT", "CELL", and "CELL". The main content area shows a code cell with the following code:

```
a = 3
b = 5
c = a + b
print(a, " + ", b, " = ", c)

!python
```



The screenshot shows the output of the code cell from the previous image. It displays the Python code and the output of the shell command:

```
a = 3
b = 5
c = a + b
print(a, " + ", b, " = ", c)

!python
```

3 + 5 = 8  
Python 3.6.7 (default, Oct 22 2018, 11:32:17)  
[GCC 8.2.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print('interprete')  
interprete  
>>> exit()

# ! vs. %

---

You cannot use `!cd` to navigate the filesystem:

```
In [11]: !pwd
/home/jake/projects/myproject

In [12]: !cd ..

In [13]: !pwd
/home/jake/projects/myproject
```

The reason is that shell commands in the notebook are executed in a temporary subshell. If you would like to change the working directory in a more enduring way, you can use the `%cd` magic command:

```
In [14]: %cd ..
/home/jake/projects
```

# Variabili e istruzione di assegnamento: sintassi

---

Sintassi:

```
variabile = espressione
```

- non devono esserci spazi prima del nome della variabile
- `variabile` deve essere un nome simbolico scelto dal programmatore (con le limitazioni descritte più avanti)
- `espressione` indica il valore (per es., un numero) che si vuole associare alla variabile

Esempi:

```
x = -3.2
```

```
messaggio = "Buongiorno"
```

```
y = x + 1
```

# Limitazioni sui nomi delle variabili

---

- possono essere composti da uno o più dei seguenti caratteri:
  - lettere minuscole e maiuscole (NB: «A» e «a» sono considerate lettere diverse)
  - cifre
  - il carattere `_` (underscore)

esempi: `x`, `somma`, `Massimo_Comun_Divisore`, `y_1`

- non devono iniziare con una cifra

esempio: `12abc` non è un nome ammesso

- non devono contenere simboli matematici (`+`, `-`, `/`, `*`, parentesi)
- non devono coincidere con le keywords del linguaggio, che sono le seguenti:

<code>and</code>	<code>as</code>	<code>assert</code>	<code>break</code>	<code>class</code>
<code>continue</code>	<code>def</code>	<code>del</code>	<code>elif</code>	<code>else</code>
<code>except</code>	<code>exec</code>	<code>if</code>	<code>import</code>	<code>in</code>
<code>is</code>	<code>lambda</code>	<code>not</code>	<code>or</code>	<code>pass</code>
<code>print</code>	<code>raise</code>	<code>return</code>	<code>try</code>	<code>while</code>
<code>yield</code>	<code>True</code>	<code>False</code>	<code>None</code>	

Un nome di variabile può avere fino ad un massimo di 256 caratteri totali

# Tipi di dato ed espressioni

---

I **tipi di dato base** che possono essere rappresentati ed elaborati dai programmi Python, sono:

- numeri interi
- numeri frazionari (*floating-point*)
- stringhe di caratteri
- booleani

Le espressioni Python che producono valori appartenenti a tali tipi di dati, e che possono contenere opportuni **operatori**, sono le seguenti:

- espressioni aritmetiche
- espressioni stringa



# Espressioni aritmetiche

---

La più semplice espressione aritmetica è un singolo numero.

I numeri vengono rappresentati nelle istruzioni Python con diverse notazioni, simili a quelle matematiche, espressi in base dieci.

Python distingue tra due tipi di dati numerici:

- numeri interi, codificati nei calcolatori in complemento a due; esempi: 12, -9
- numeri frazionari ( floating point ), codificati in virgola mobile, e rappresentati nei programmi:
  - come parte intera e frazionaria, separate da un punto (notazione anglosassone)  
esempi: 3.14, -45.2, 1.0
  - in notazione esponenziale,  $m \times b^e$ , con base b pari a dieci ed esponente introdotto dal carattere E oppure e  
esempi: 1.99E33 ( $1,99 \times 10^{33}$ ), -42.3e-4 ( $-42,3 \times 10^{-4}$ ), 2E3 ( $2 \times 10^3$ )

**NOTA: i numeri espressi in notazione esponenziale sono sempre considerati numeri frazionari**

# Operatori aritmetici

---

Espressioni aritmetiche più complesse si ottengono combinando numeri attraverso operatori (addizione, divisione, ecc.), e usando le parentesi tonde per definire la precedenza tra gli operatori.

Gli operatori disponibili nel linguaggio Python sono i seguenti:

## **simbolo**    **operatore**

+	somma
-	sottrazione
*	moltiplicazione
/	divisione
//	divisione (quoziente intero)
%	modulo (resto di una divisione)
**	elevamento a potenza

# Espressioni aritmetiche: esempi

---

Alcuni esempi di istruzioni di assegnazione contenenti espressioni aritmetiche.  
Il valore per ogni espressione è indicato in grassetto sulla destra.

```
x = -5           -5
y = 1 + 1        2
z = (1 + 2) * 3  9
circonferenza = 2 * 3.14 * 3  18.84
q1 = 6 / 2       3
q2 = 7.5 / 3     2.5
q3 = 5 // 2      2
resto = 10 % 2   0
```

# Rappresentazione del risultato

---

Se entrambi gli operandi di  $+$ ,  $-$  e  $*$  sono interi, il risultato è rappresentato come intero (senza parte frazionaria), altrimenti è rappresentato come numero frazionario.

Esempi:  $1 + 1 \rightarrow 2$ ,  $2 - 3.1 \rightarrow -1.1$ ,  $3.2 * 5 \rightarrow 16.0$

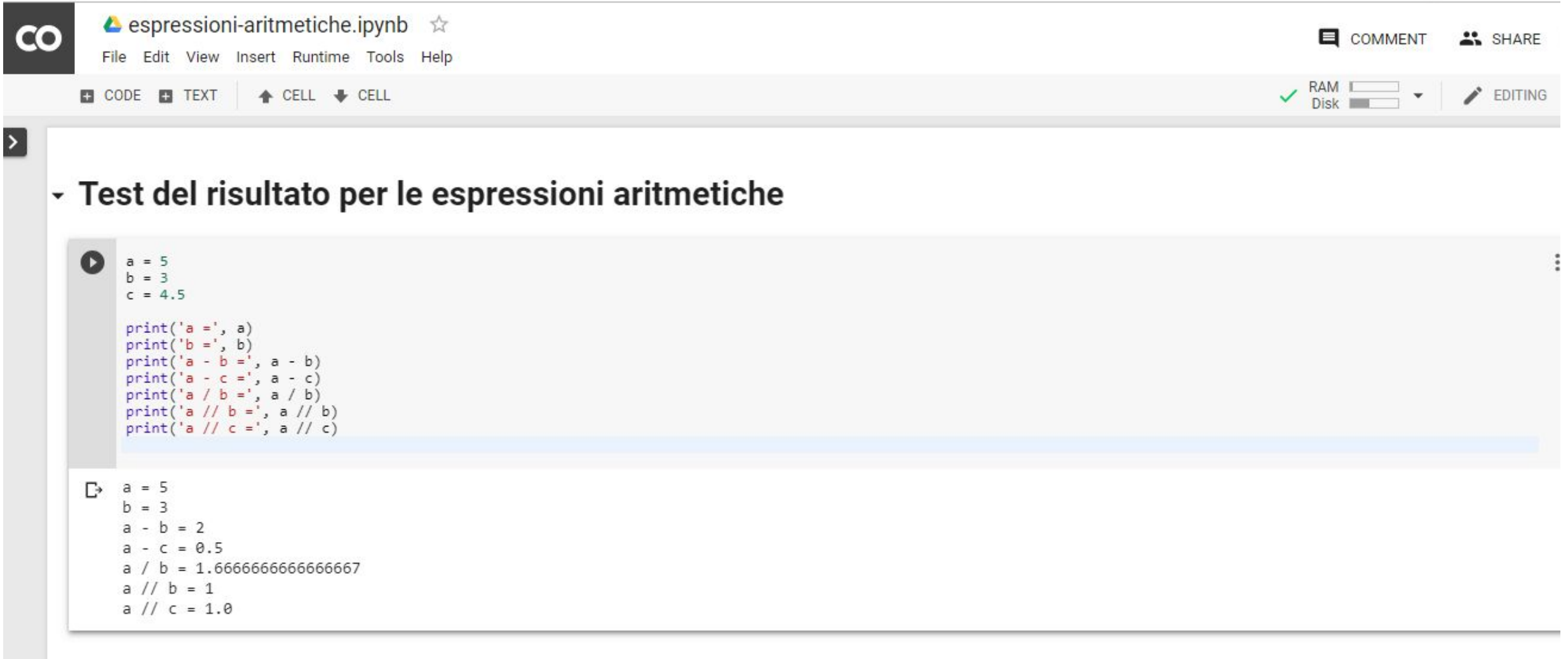
Anche se entrambi gli operandi di  $/$  sono interi, il risultato è invece sempre frazionario, come del resto accade se uno o entrambi gli operandi non sono interi.

Esempi:  $6 / 2 \rightarrow 3.0$ ,  $6.0 / 2 \rightarrow 3.0$ ,  $2 / 5 \rightarrow 0.4$ ,  
 $2 / 5.0 \rightarrow 0.4$ ,  $-2 / 3 \rightarrow -0.66666666666666666666$

L'operatore  $//$  produce sempre il più grande intero non maggiore del quoziente, rappresentato come intero se entrambi gli operandi sono interi, altrimenti come numero frazionario.

Esempi:  $6 // 2 \rightarrow 3$ ,  $6.0 // 2 \rightarrow 3.0$ ,  $2 // 5 \rightarrow 0$ ,  
 $2 // 5.0 \rightarrow 0.0$ ,  $-2 // 3 \rightarrow -1$

# Prova su Colab



The screenshot shows a Google Colab notebook titled "espressioni-aritmetiche.ipynb". The interface includes a top navigation bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help" menus. On the right, there are "COMMENT" and "SHARE" buttons. Below the navigation bar, there are controls for adding code or text cells and a status bar showing RAM and Disk usage, along with an "EDITING" indicator.

## Test del risultato per le espressioni aritmetiche

```
a = 5
b = 3
c = 4.5

print('a =', a)
print('b =', b)
print('a - b =', a - b)
print('a - c =', a - c)
print('a / b =', a / b)
print('a // b =', a // b)
print('a // c =', a // c)
```

```
a = 5
b = 3
a - b = 2
a - c = 0.5
a / b = 1.6666666666666667
a // b = 1
a // c = 1.0
```

# Espressioni: stringhe

---

I programmi Python possono elaborare testi rappresentati come sequenze di caratteri (lettere, numeri, segni di punteggiatura) racchiuse tra apici singoli o doppi, dette stringhe.

Esempi:

```
"Esempio di stringa"  
'Esempio di stringa'  
"A"  
'qwerty, 123456'  
"" (una stringa vuota)
```

È possibile assegnare una stringa a una variabile

Esempi:

```
testo = "Esempio di stringa"  
carattere = "a"  
messaggio = "Premere un tasto per continuare."  
t = ""
```

# Concatenazione di stringhe

---

Il linguaggio Python prevede alcuni operatori anche per il tipo di dato stringa. Uno di questi è l'operatore di concatenazione, che si rappresenta con il simbolo + (lo stesso dell'addizione tra numeri) e produce come risultato una nuova stringa ottenuta concatenando due altre stringhe.

Esempi:

```
parola = "mappa" + "mondo"
```

assegna alla variabile parola la stringa "mappamondo"

```
testo = "Due" + " " + "parole"
```

assegna alla variabile testo la stringa "Due parole" (si noti che la stringa " " contiene solo un carattere di spaziatura)

# Sequenze di escape

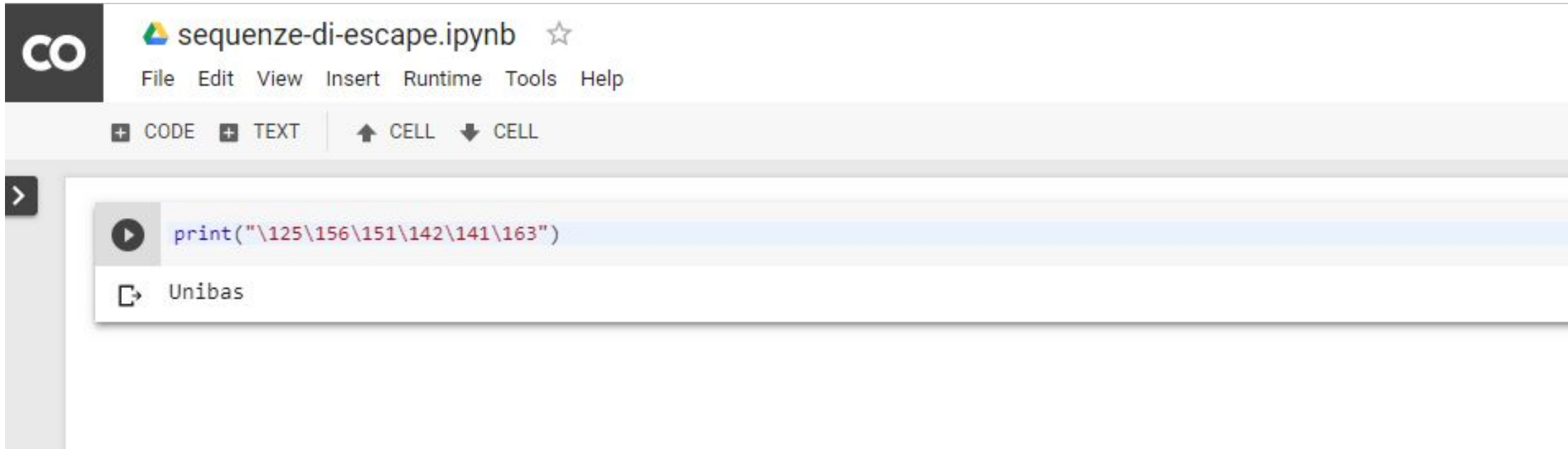
---

Sequenza di escape	Significato
\\	Backslash (\)
\'	Carattere di quotatura singola (')
\"	Carattere di quotatura doppia (")
\a	ASCII Bell (BEL)
\b	ASCII Backspace (BS)
\f	ASCII Formfeed (FF)
\n	ASCII Linefeed (LF)
\N{name}	Carattere chiamato name nel database Unicode (solamente in Unicode)
\r	ASCII Carriage Return (CR)
\t	ASCII Tab orizzontale (TAB)
\uxxxx	Carattere con valore esadecimale a 16 bit xxxx (valido solo per Unicode)
\Uxxxxxxxx	Carattere con valore esadecimale a 32 bit xxxxxxxx (valido solo per Unicode)
\v	ASCII Tab verticale (VT)
\ooo	Carattere ASCII con valore ottale ooo
\xhh	Carattere ASCII con valore esadecimale hh



# Sequenze di escape: esempio Colab

---



The image shows a Google Colab notebook interface. At the top left is the Colab logo (CO). The notebook title is "sequenze-di-escape.ipynb" with a star icon. Below the title is a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". A toolbar contains buttons for "+ CODE", "+ TEXT", "↑ CELL", and "↓ CELL". A code cell is active, containing the Python code `print("\125\156\151\142\141\163")`. Below the code cell, the output is displayed as "Unibas".

```
print("\125\156\151\142\141\163")
```

Unibas

# Espressioni contenenti nomi di variabili

---

Una espressione contenente il nome di una variabile alla quale non sia stato ancora assegnato alcun valore è [sintatticamente errata](#).

Per esempio, assumendo che alla variabile `h` non sia ancora stato assegnato alcun valore, l'istruzione

```
x = h + 1
```

genererà il seguente messaggio di errore:

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

# La funzione `print`

---

Sintassi:

```
print (espressione)
```

- non devono esserci spazi prima di `print`
- `espressione` deve essere una espressione valida del linguaggio Python

Semantica:

viene mostrato sullo standard output il valore di `espressione`

È anche possibile mostrare con una sola istruzione `print` i valori di un numero qualsiasi di espressioni, con la seguente sintassi:

```
print (espressione1, espressione2, ...)
```

In questo caso, i valori delle espressioni vengono mostrati su una stessa riga, separati da un carattere di spaziatura.

# Commenti

---

È sempre utile documentare i programmi inserendo commenti che indichino quale operazione viene svolta dal programma, quali sono i dati di ingresso e i risultati, qual è il significato delle variabili o di alcune sequenze di istruzioni.

Nei programmi Python i commenti possono essere inseriti in qualsiasi riga, preceduti dal carattere # (“cancelletto”).

Tutti i caratteri che seguono il cancelletto, fino al termine della riga, saranno considerati commenti e verranno trascurati dall'interprete.

# Commenti su più linee

---



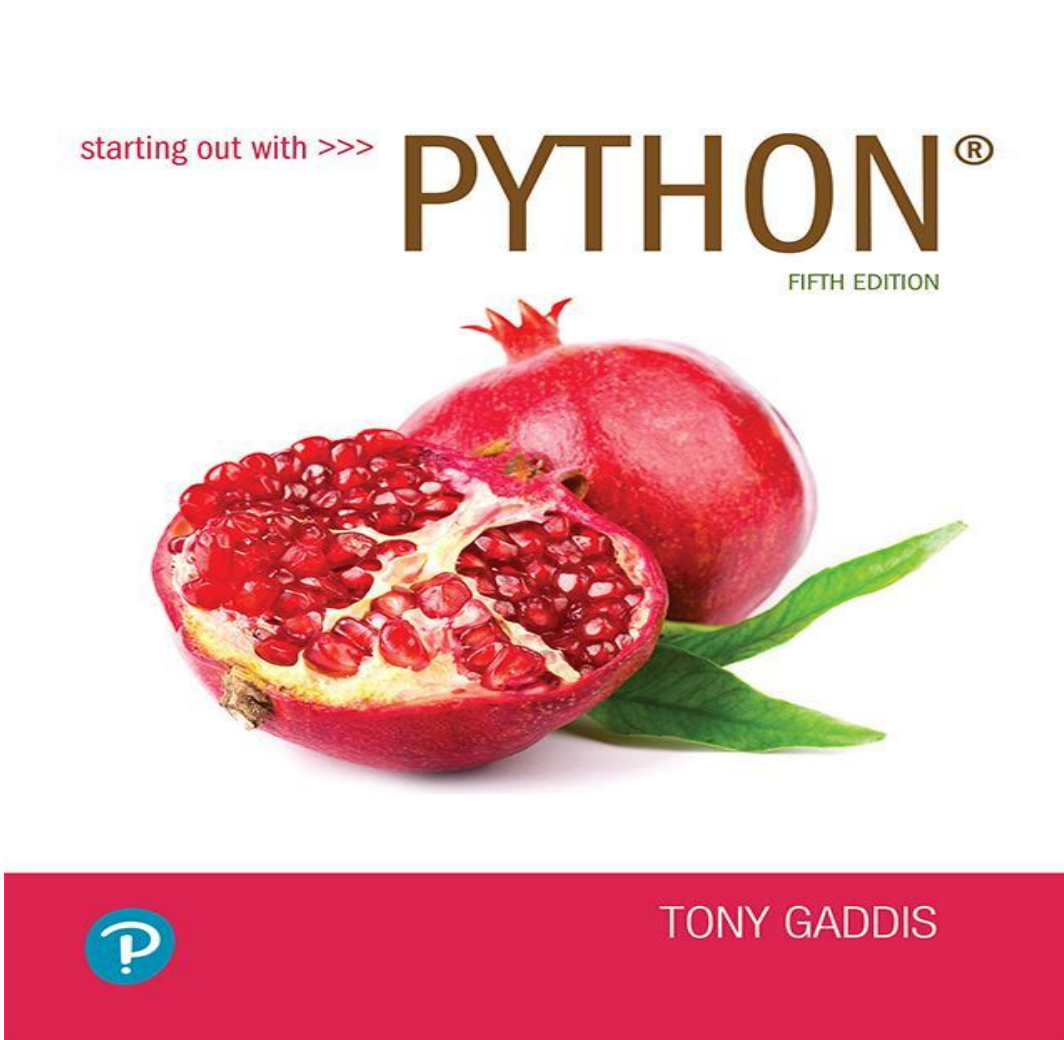
The screenshot shows a Jupyter Notebook interface. At the top left is the 'CO' logo. The browser address bar shows 'commenti.ipynb' with a star icon. Below the address bar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. A toolbar contains '+ CODE', '+ TEXT', '↑ CELL', and '↓ CELL'. The notebook title is 'Commenti su più linee'. A code cell is selected, containing the following code:

```
print('commenti su più linee')  
  
#queste linee verranno  
#ignorate
```

Below the code cell, a tooltip displays the text 'commenti su più linee' next to a copy icon.

# Starting out with Python

Fifth Edition



## Chapter 3

### Decision Structures and Boolean Logic

# Topics

- The `if` Statement
- The `if-else` Statement
- Comparing Strings
- Nested Decision Structures and the `if-elif-else` Statement
- Logical Operators
- Boolean Variables
- Turtle Graphics: Determining the State of the Turtle

# The `if` Statement (1 of 4)

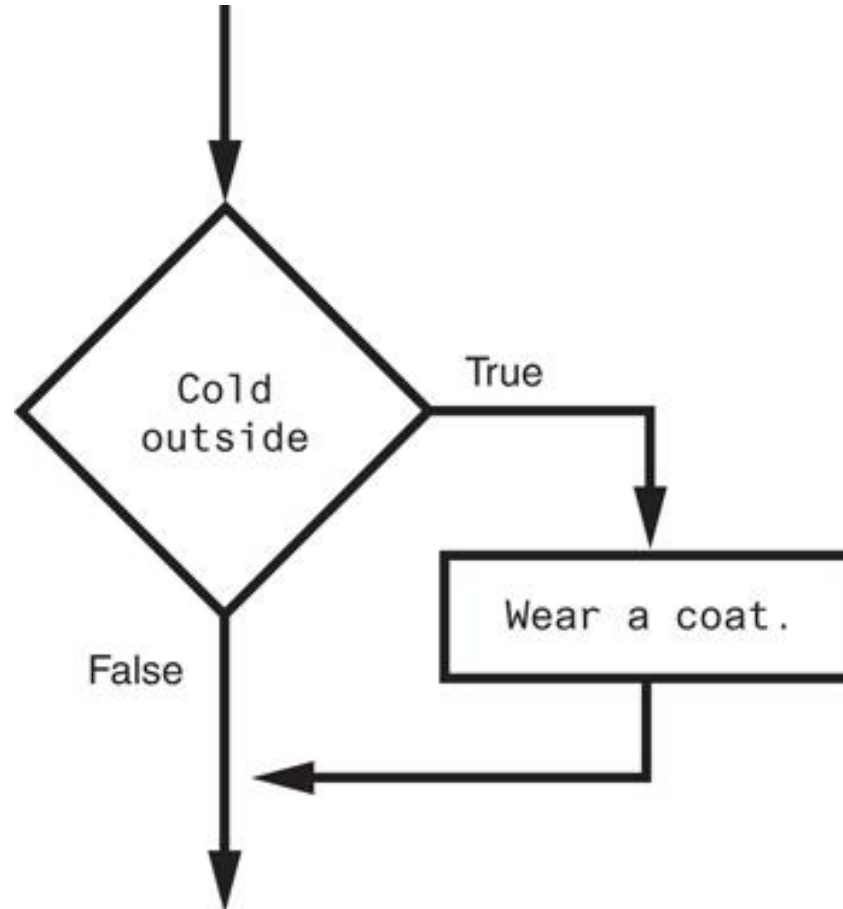
- Control structure: logical design that controls order in which set of statements execute
- Sequence structure: set of statements that execute in the order they appear
- Decision structure: specific action(s) performed only if a condition exists
  - Also known as selection structure



# The `if` Statement (2 of 4)

- In flowchart, diamond represents true/false condition that must be tested
- Actions can be *conditionally executed*
  - Performed only when a condition is true
- Single alternative decision structure: provides only one alternative path of execution
  - If condition is not true, exit the structure

# The `if` Statement (3 of 4)



**Figure 3-1** A simple decision structure

# The `if` Statement (4 of 4)

- Python syntax:

```
if condition:  
    Statement  
    Statement
```

- First line known as the `if` clause
  - Includes the keyword `if` followed by condition
    - The condition can be true or false
    - When the `if` statement executes, the condition is tested, and if it is true the block statements are executed. otherwise, block statements are skipped

# Attenzione ai rientri!

---

- I rientri sono l'unico elemento sintattico che indica quali istruzioni fanno parte di una istruzione condizionale.
- Le istruzioni che seguono una istruzione condizionale (senza farne parte) devono quindi essere scritte senza rientri.

# Esempio rientri

---

Si considerino le due sequenze di istruzioni:

```
if x > 0:  
    print("A")  
    print("B")
```

```
if x > 0:  
    print("A")  
print("B")
```

Nella sequenza a sinistra le due istruzioni `print` sono scritte con un rientro rispetto a `if`: questo significa che fanno entrambe parte dell'istruzione condizionale e quindi verranno eseguite solo se la condizione `x > 0` sarà vera.

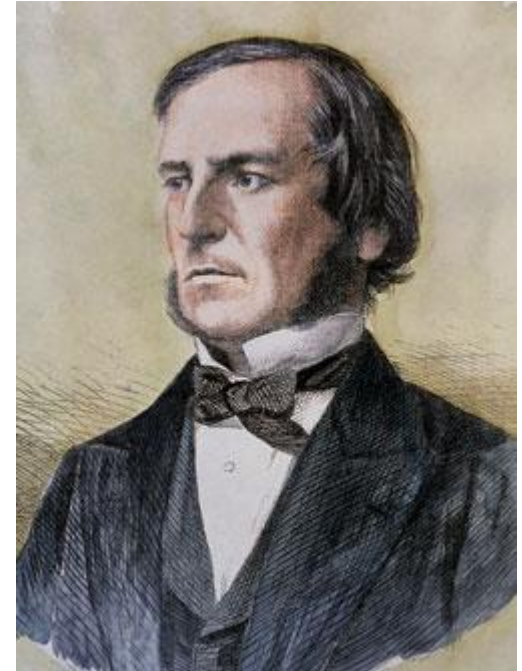
Nella sequenza a destra solo la prima istruzione `print` dopo `if` è scritta con un rientro e quindi solo essa fa parte dell'istruzione condizionale; la seconda istruzione `print` verrà invece eseguita dopo l'istruzione condizionale, indipendentemente dal valore di verità della condizione `x > 0`.

# Algebra di Boole

---

Il matematico britannico George Boole (1815-1864) trovò il modo di legare argomenti logici ad un linguaggio che potesse essere manipolato matematicamente.

Il sistema di Boole era basato su un approccio binario, in grado di differenziare gli argomenti in base alla presenza o all'assenza di una determinata proprietà.



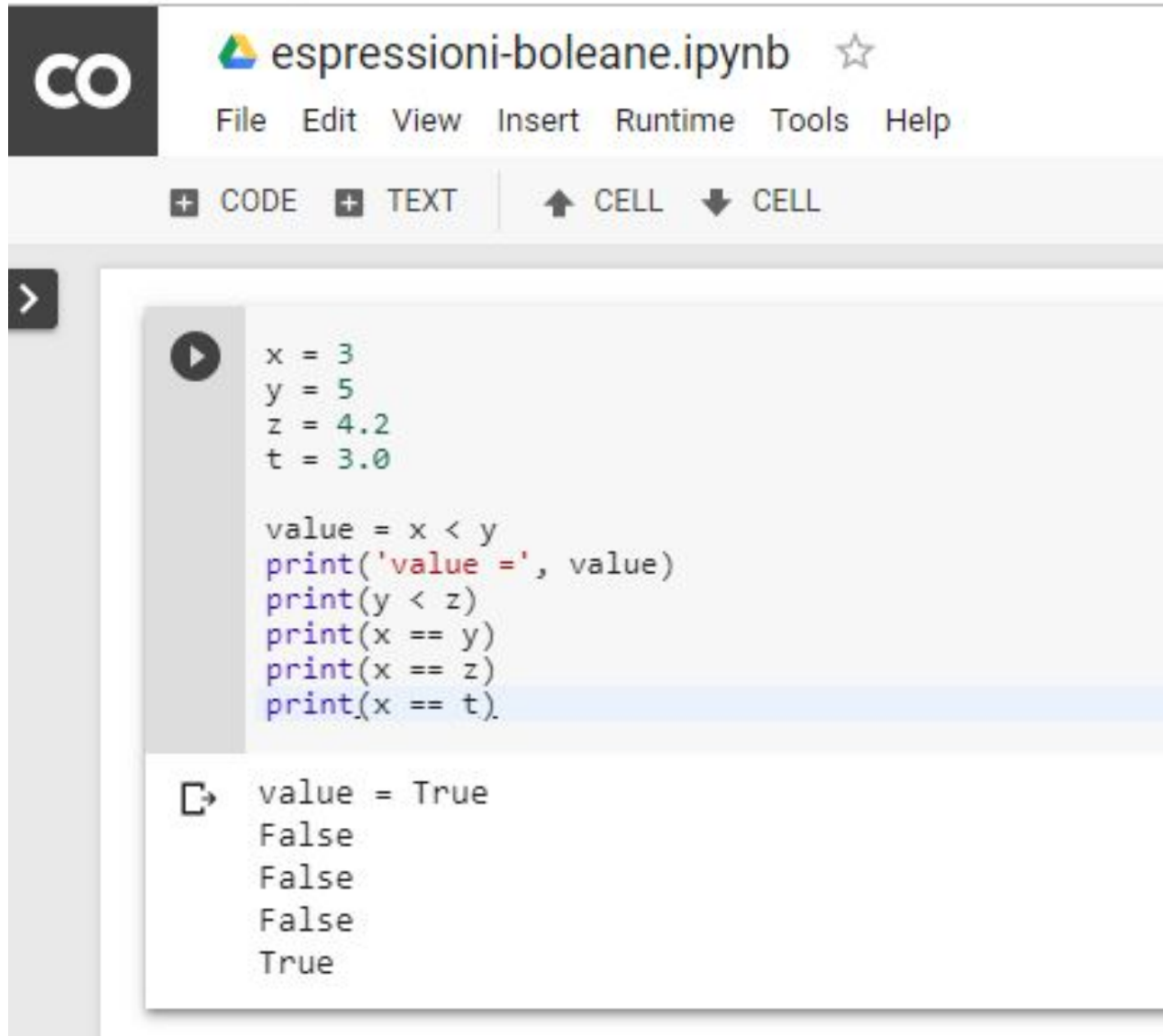
# Espressioni booleane

---

I valori booleani possono essere prodotti da:

- Le costanti `True` o `False`.
- Una variabile cui sia stato assegnato un valore booleano.
- Una relazione fra due espressioni per mezzo degli operatori relazionali come `==`, `!=`, `<`, `>`, `<=`, `>=`

# Espressioni booleane: esempio Colab



The screenshot shows a Google Colab notebook interface. At the top left is the Colab logo (two overlapping circles). To its right is the file name "espressioni-boleane.ipynb" with a star icon. Below the file name is a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". Underneath the menu bar are buttons for "+ CODE", "+ TEXT", "↑ CELL", and "↓ CELL". The main area contains a code cell with a play button icon on the left. The code in the cell is:

```
x = 3
y = 5
z = 4.2
t = 3.0

value = x < y
print('value =', value)
print(y < z)
print(x == y)
print(x == z)
print(x == t).
```

Below the code cell is the output, which consists of five lines of text:

```
value = True
False
False
False
True
```



# Boolean Expressions and Relational Operators (1 of 5)

- Boolean expression: expression tested by if statement to determine if it is true or false
  - Example:  $a > b$ 
    - `true` if `a` is greater than `b`; `false` otherwise
- Relational operator: determines whether a specific relationship exists between two values
  - Example: greater than ( $>$ )

# Boolean Expressions and Relational Operators (2 of 5)

- $>=$  and  $<=$  operators test more than one relationship
  - It is enough for one of the relationships to exist for the expression to be true
- $==$  operator determines whether the two operands are equal to one another
  - Do not confuse with assignment operator ( $=$ )
- $!=$  operator determines whether the two operands are not equal

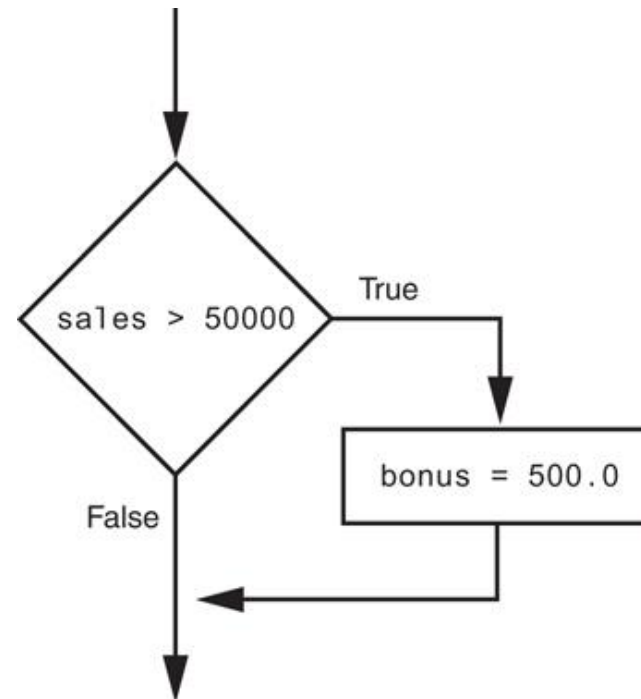
# Boolean Expressions and Relational Operators (3 of 5)

Table 3-2 Boolean expressions using relational operators

Expression	Meaning
$x > y$	Is $x$ greater than $y$ ?
$x < y$	Is $x$ less than $y$ ?
$x \geq y$	Is $x$ greater than or equal to $y$ ?
$x \leq y$	Is $x$ less than or equal to $y$ ?
$x == y$	Is $x$ equal to $y$ ?
$x != y$	Is $x$ not equal to $y$ ?

# Boolean Expressions and Relational Operators (4 of 5)

- Using a Boolean expression with the  $>$  relational operator



**Figure 3-3** Example decision structure

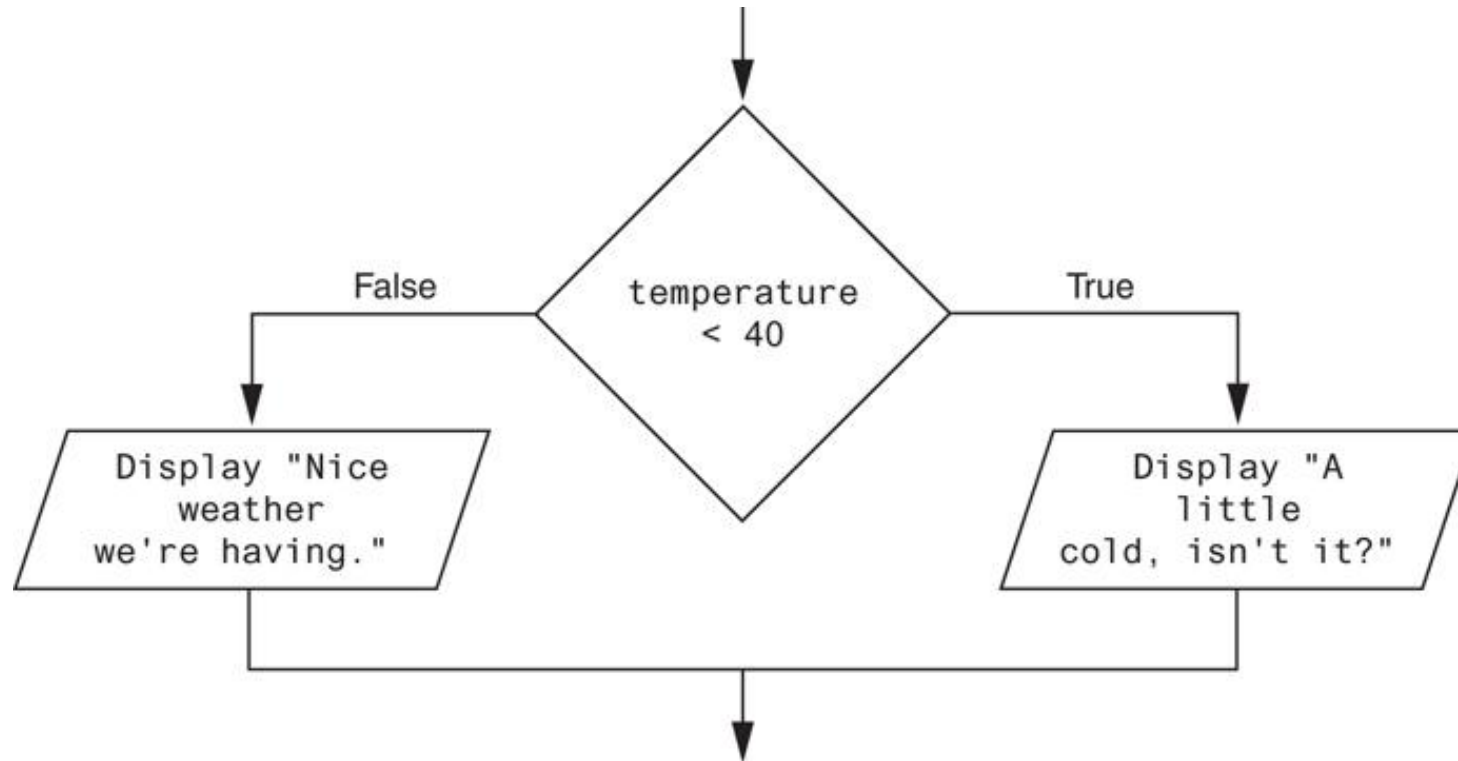
# Boolean Expressions and Relational Operators (5 of 5)

- Any relational operator can be used in a decision block
  - Example: `if balance == 0`
  - Example: `if payment != balance`
- It is possible to have a block inside another block
  - Example: `if` statement inside a function
  - Statements in inner block must be indented with respect to the outer block

# The `if-else` Statement (1 of 3)

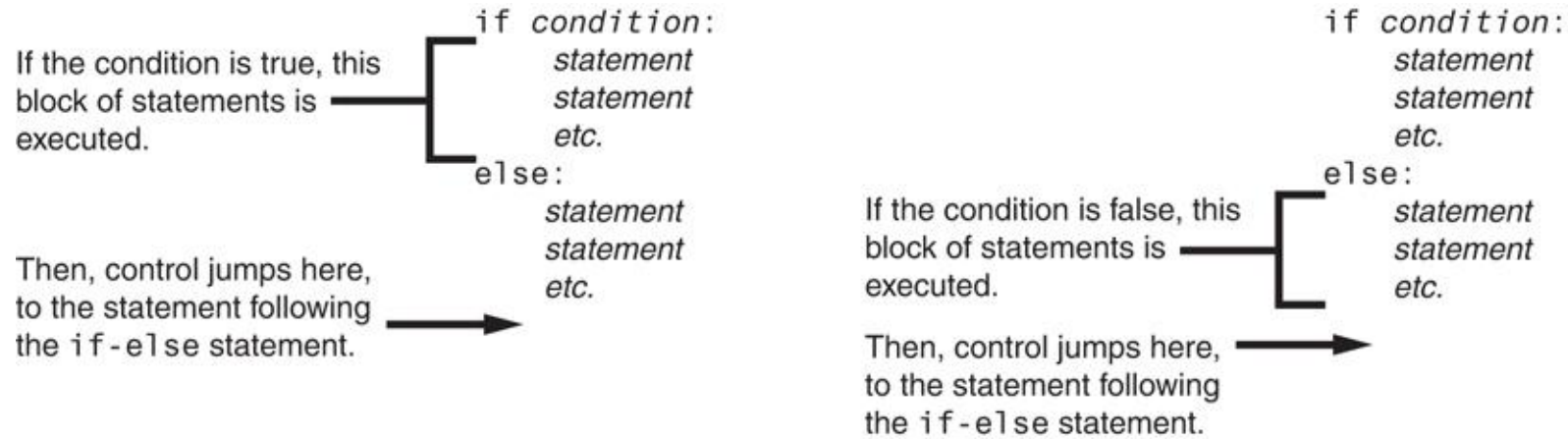
- Dual alternative decision structure: two possible paths of execution
  - One is taken if the condition is true, and the other if the condition is false
  - Syntax: `if condition:`  
    `statements`  
    `else:`  
    `other statements`
  - `if` clause and `else` clause must be aligned
  - Statements must be consistently indented

# The `if-else` Statement (2 of 3)



**Figure 3-5** A dual alternative decision structure

# The `if-else` Statement (3 of 3)



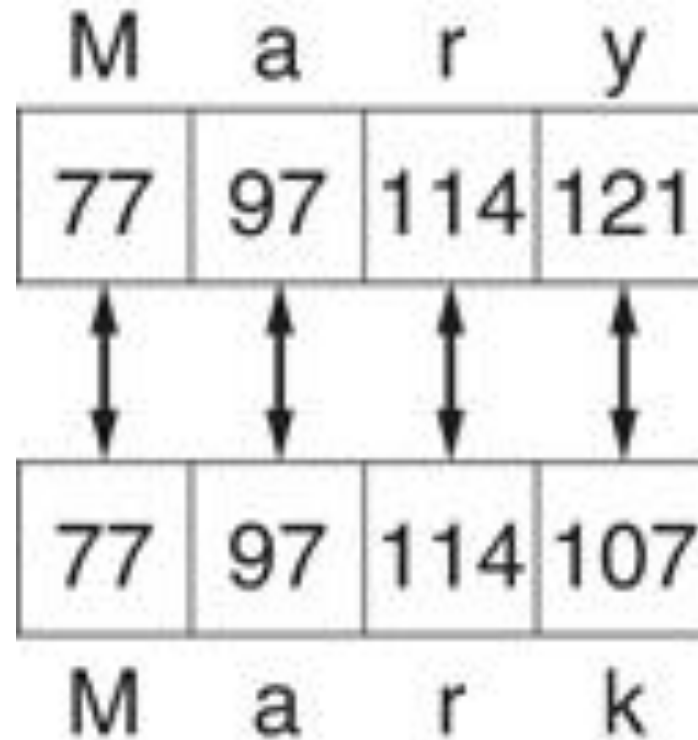
**Figure 3-6** Conditional execution in an `if-else` statement



# Comparing Strings (1 of 2)

- Strings can be compared using the == and != operators
- String comparisons are case sensitive
- Strings can be compared using >, <, >=, and <=
  - Compared character by character based on the ASCII values for each character
  - If shorter word is substring of longer word, longer word is greater than shorter word

# Comparing Strings (2 of 2)



**Figure 3-9** Comparing each character in a string



✓  
0 s

```
[4] name1 = 'Mary'  
     name2 = 'Mark'  
     if name1 == name2:  
         print('I nomi sono identici.')     else:  
         print('I nomi sono diversi.')
```

I nomi sono diversi.

✓  
0 s

```
▶ print(ord("y"))
```

121

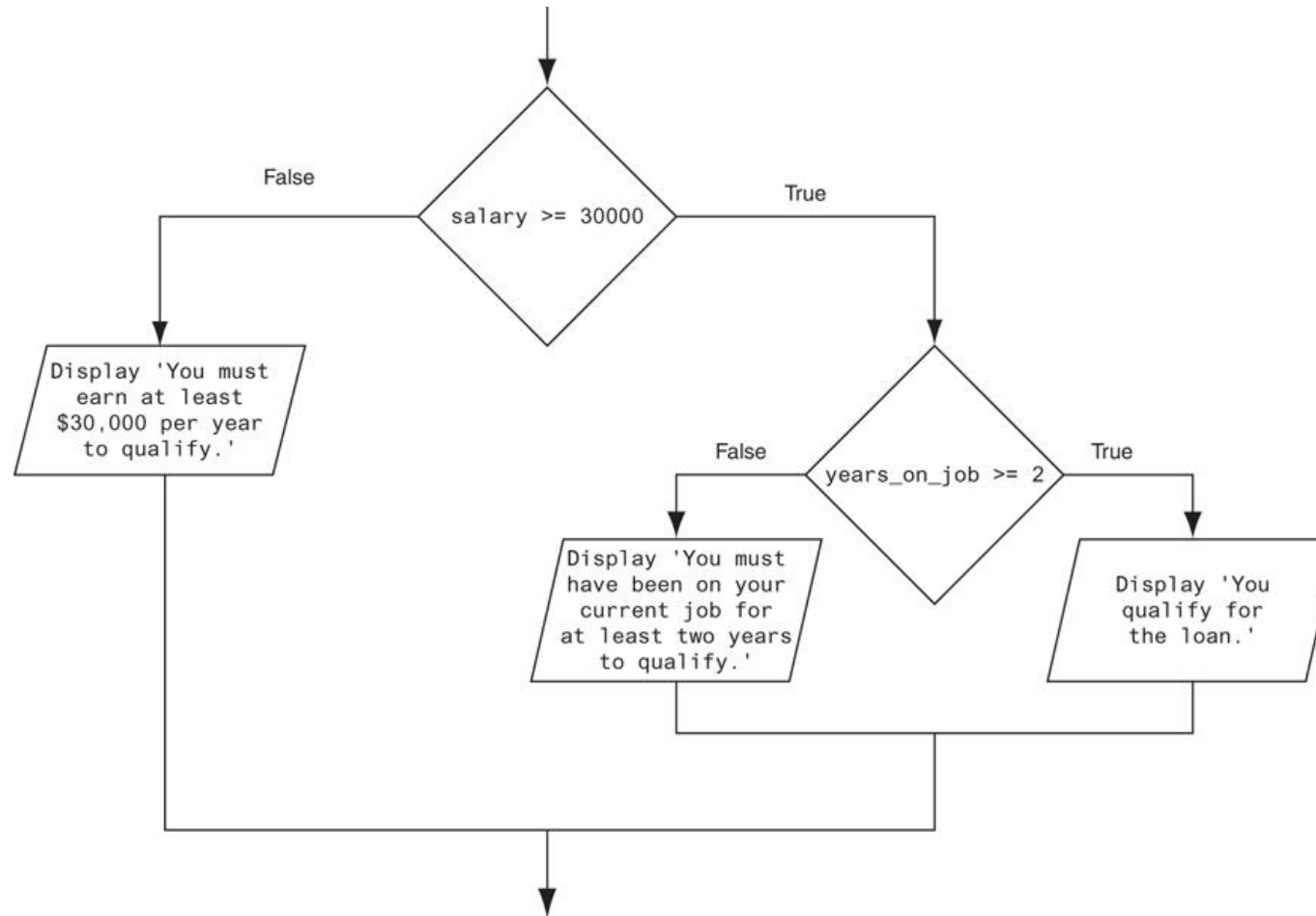
## **ord(c, /)**

Given a string representing one Unicode character, return an integer representing the Unicode code point of that character. For example, `ord('a')` returns the integer `97` and `ord('€')` (Euro sign) returns `8364`. This is the inverse of `chr()`.

# Nested Decision Structures and the `if-elif-else` Statement (1 of 3)

- A decision structure can be nested inside another decision structure
  - Commonly needed in programs
  - Example:
    - Determine if someone qualifies for a loan, they must meet two conditions:
      - Must earn at least \$30,000/year
      - Must have been employed for at least two years
    - Check first condition, and if it is true, check second condition

# Nested Decision Structures and the `if-elif-else` Statement (2 of 3)



**Figure 3-12** A nested decision structure

# Nested Decision Structures and the `if-elif-else` Statement (3 of 3)

- Important to use proper indentation in a nested decision structure
  - Important for Python interpreter
  - Makes code more readable for programmer
  - Rules for writing nested if statements:
    - `else` clause should align with matching `if` clause
    - Statements in each block must be consistently indented

# The `if-elif-else` Statement (1 of 3)

- `if-elif-else` statement: special version of a decision structure
  - Makes logic of nested decision structures simpler to write
    - Can include multiple `elif` statements
  - Syntax:

```
if condition_1:  
    statement(s)  
elif condition_2:  
    statement(s)  
elif condition_3:  
    statement(s)  
else  
    statement(s)
```



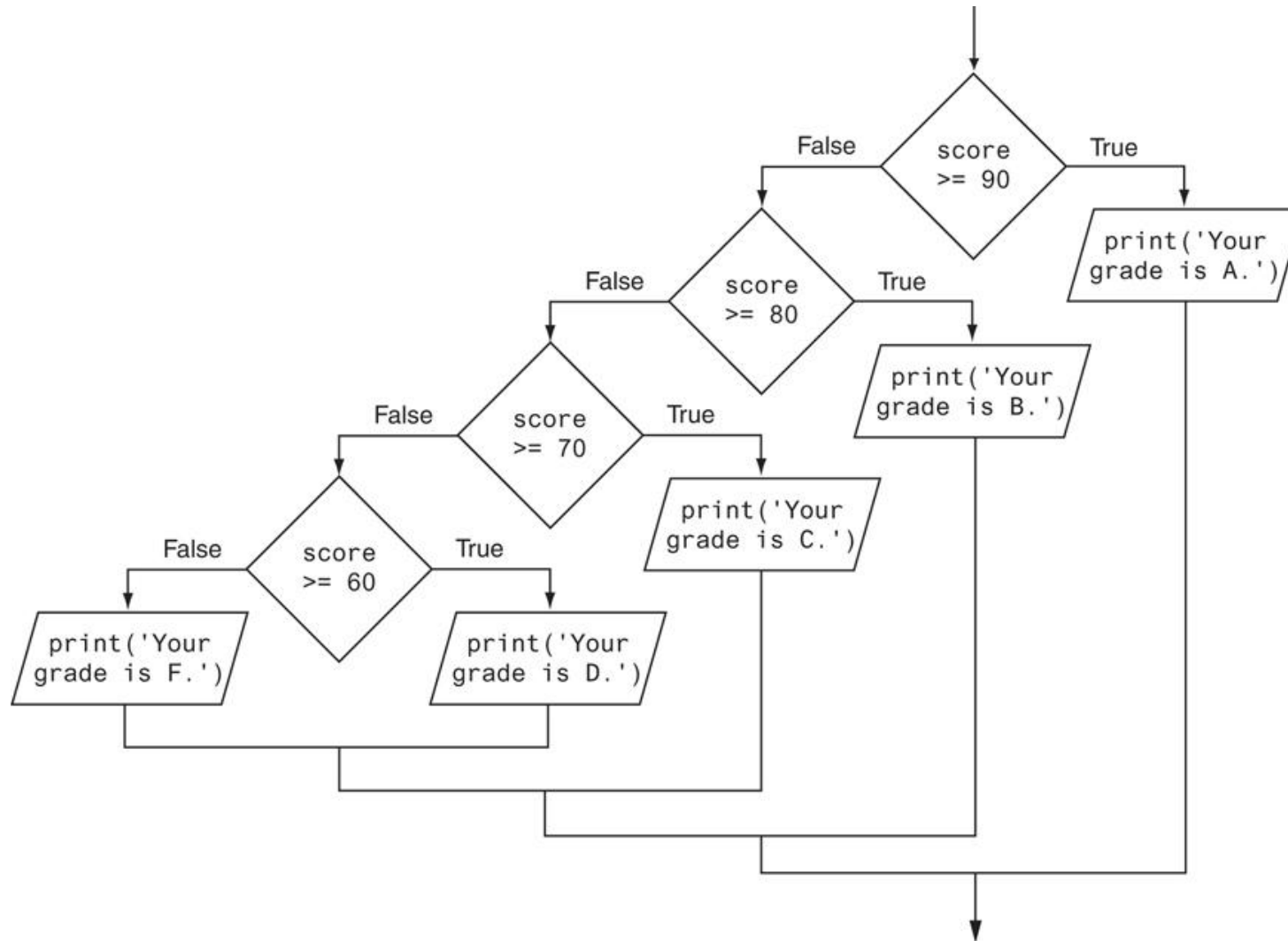
Insert as many `elif` clauses as necessary.

# The `if-elif-else` Statement (2 of 3)

- Alignment used with `if-elif-else` statement:
  - `if`, `elif`, and `else` clauses are all aligned
  - Conditionally executed blocks are consistently indented
- `if-elif-else` statement is never required, but logic easier to follow
  - Can be accomplished by nested `if-else`
    - Code can become complex, and indentation can cause problematic long lines



# The `if-elif-else` Statement (3 of 3)



**Figure 3-15** Nested decision structure to determine a grade

# Logical Operators

- Logical operators: operators that can be used to create complex Boolean expressions
  - `and` operator and `or` operator: binary operators, connect two Boolean expressions into a compound Boolean expression
  - `not` operator: unary operator, reverses the truth of its Boolean operand

# Operatore booleano `and`

---

Se `x` e `y` sono booleani, possiamo completamente determinare i valori possibili di `x and y` usando la seguente «tavola di verità»:

<code>x</code>	<code>y</code>	<code>x and y</code>
False	False	False
False	True	False
True	False	False
True	True	True

**`x and y` è False a meno che le variabili `x` e `y` non siano entrambe True**

# Operatore booleano `or`

---

Se  $x$  e  $y$  sono booleani, possiamo completamente determinare i valori possibili di  $x \text{ or } y$  usando la seguente «tavola di verità»:

<b>x</b>	<b>y</b>	<b>x or y</b>
False	False	False
False	True	True
True	False	True
True	True	True

**$x \text{ or } y$  è True a meno che le variabili  $x$  e  $y$  non siano entrambe False**

# Operatore booleano `not`

---

Se `x` è un booleano, possiamo completamente determinare i valori possibili di `not x` usando la seguente «tavola di verità»:

<code>x</code>	<code>not x</code>
False	True
True	False

**`not x` è True se `x` è False ed è False se `x` è True**

# Short-Circuit Evaluation

- Short circuit evaluation: deciding the value of a compound Boolean expression after evaluating only one sub expression
  - Performed by the `or` and `and` operators
    - For `or` operator: If left operand is true, compound expression is true. Otherwise, evaluate right operand
    - For `and` operator: If left operand is false, compound expression is false. Otherwise, evaluate right operand

# Checking Numeric Ranges with Logical Operators

- To determine whether a numeric value is within a specific range of values, use `and`
  - Example: `x >= 10 and x <= 20`
- To determine whether a numeric value is outside of a specific range of values, use `or`
  - Example: `x < 10 or x > 20`

# Boolean Variables

- Boolean variable: references one of two values, `True` or `False`
  - Represented by `bool` data type
- Commonly used as flags
  - Flag: variable that signals when some condition exists in a program
    - Flag set to `False` means the condition does not exist
    - Flag set to `True` means the condition exists

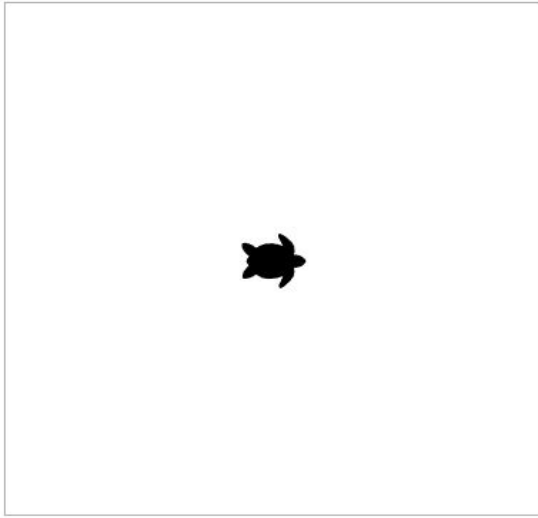


# Turtle Graphics: Determining the State of the Turtle

```
[9] !pip3 install ColabTurtlePlus
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting ColabTurtlePlus
  Downloading ColabTurtlePlus-2.0.1-py3-none-any.whl (31 kB)
Installing collected packages: ColabTurtlePlus
Successfully installed ColabTurtlePlus-2.0.1

import ColabTurtlePlus.Turtle as ta

ta.clearscreen()
ta.setup(300,300)
ta.showborder()
ta.shape("turtle")
```

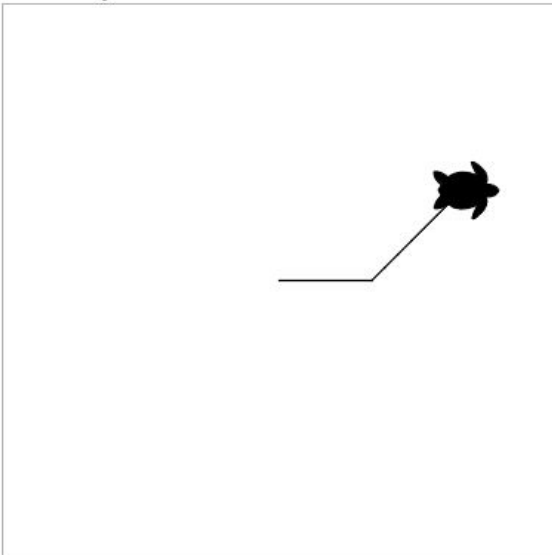


# Turtle Graphics: Determining the State of the Turtle

0 s

```
print("posizione iniziale:",ta.pos())
print("x:",ta.xcor(),"y:",ta.ycor())
ta.forward(50)
print("posizione dopo forward:",ta.pos())
ta.goto(100,50)
print("posizione dopo goto:",ta.pos())
```

posizione iniziale: (0.0, 0.0)  
x: 0.0 y: 0.0




posizione dopo forward: (50.0, 0.0)  
posizione dopo goto: (100.0, 50.0)

# Turtle Graphics: Determining the State of the Turtle

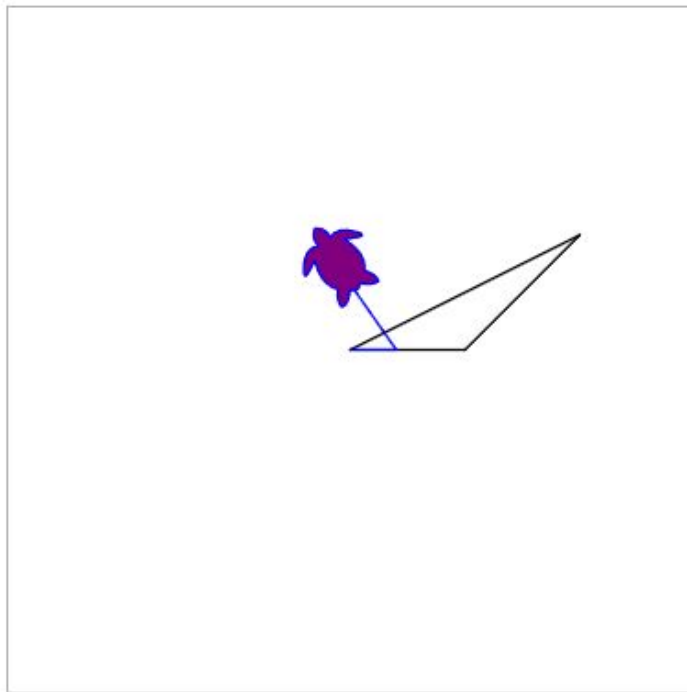
0s

```
if ta.ycor() > 0:  
    ta.goto(0, 0)
```



# Turtle Graphics: Determining the State of the Turtle

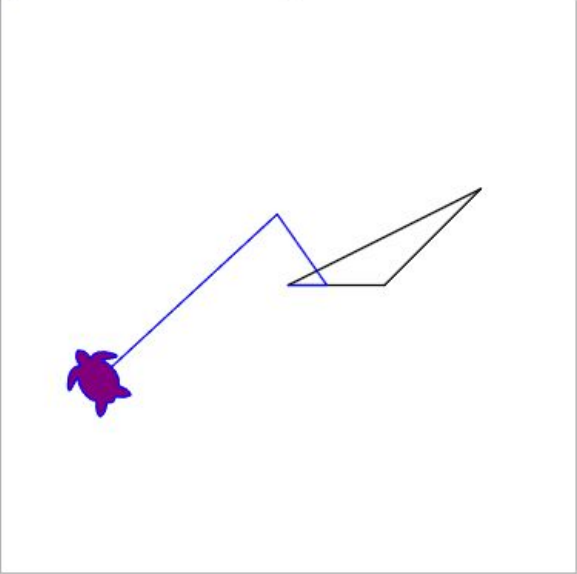
```
ta.color("blue","purple")  
ta.forward(20)  
ta.left(125)  
ta.forward(45)
```



# Turtle Graphics: Determining the State of the Turtle

```
0s ✓ ▶ print(ta.pos())  
if ta.xcor() > 100 and ta.ycor() < 200:  
    ta.goto(0, 0)  
else:  
    ta.goto(-100, -50)  
print(ta.pos())
```

↗ (-5.811000000000007, 36.861999999999995)



(-100.0, -50.0)

# La funzione `input()`

---

La funzione built-in `input` può essere utilizzata in due forme.

```
valore = input()
```

l'interprete resta in attesa che l'utente inserisca nella shell, attraverso la tastiera, una sequenza di caratteri che dovrà essere conclusa dal tasto <INVIO>. Questa sequenza di caratteri è poi inserita in una stringa che viene restituita come valore.

```
dato = input(messaggio)
```

l'interprete stampa nella shell `messaggio`, che deve essere una stringa, poi procede come indicato sopra.

La stringa `messaggio` viene di norma usata per indicare all'utente che il programma è in attesa di ricevere un particolare dato in ingresso.

# La funzione `input()`

---



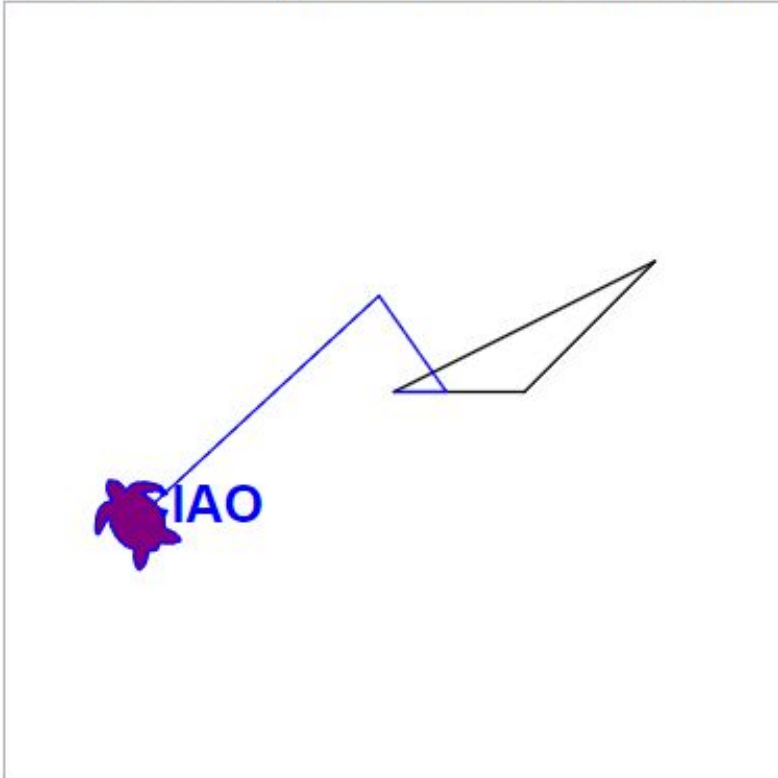
```
▶ testo = input("Cosa vuoi stampare vicino la tartaruga? ")  
ta.write(testo,font=("Arial", 20, "bold"))
```

... Cosa vuoi stampare vicino la tartaruga?

# La funzione `input()`

```
✓ 1m ▶ testo = input("Cosa vuoi stampare vicino la tartaruga? ")  
ta.write(testo,font=({"Arial", 20, "bold"}))
```

↳ Cosa vuoi stampare vicino la tartaruga? CIAO





# La funzione `input()` con dati numerici

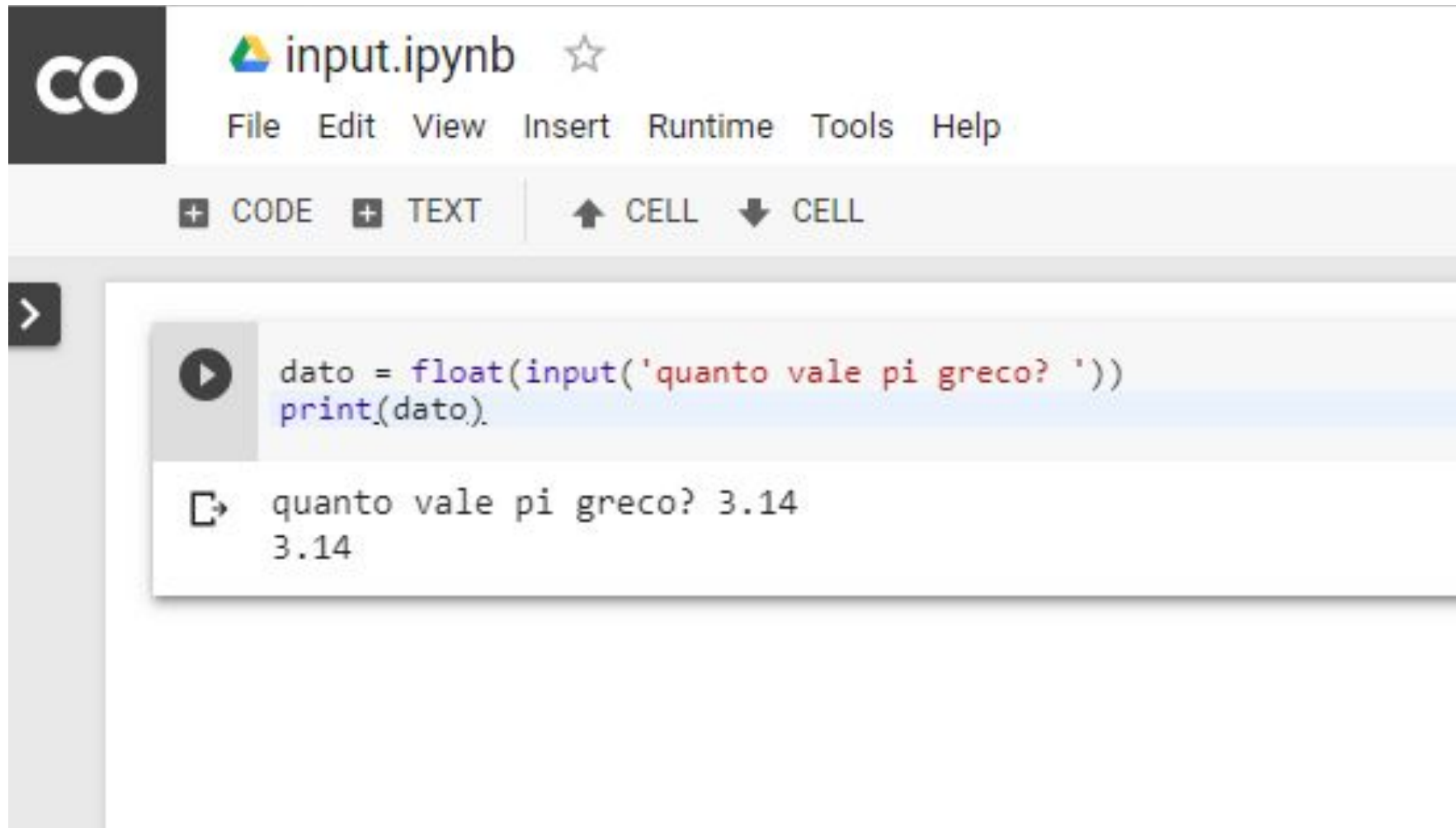
---

Nel caso di dati numerici, occorrerà convertire con `int` o `float` in intero o frazionario il valore `str` restituito da `input`

Esempio

```
dato = float(input('quanto vale pi greco? '))
```

# La funzione `input()` : Esempio Colab



The screenshot shows a Google Colab notebook interface. At the top left is the Colab logo. The notebook title is "input.ipynb" with a star icon. Below the title is a menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". Underneath the menu bar are buttons for "+ CODE", "+ TEXT", "↑ CELL", and "↓ CELL". The main area of the notebook contains a code cell with a play button icon on the left. The code in the cell is:

```
dato = float(input('quanto vale pi greco? '))  
print(dato)
```

Below the code cell, the output is displayed, showing the prompt and the user's input:

```
quanto vale pi greco? 3.14  
3.14
```

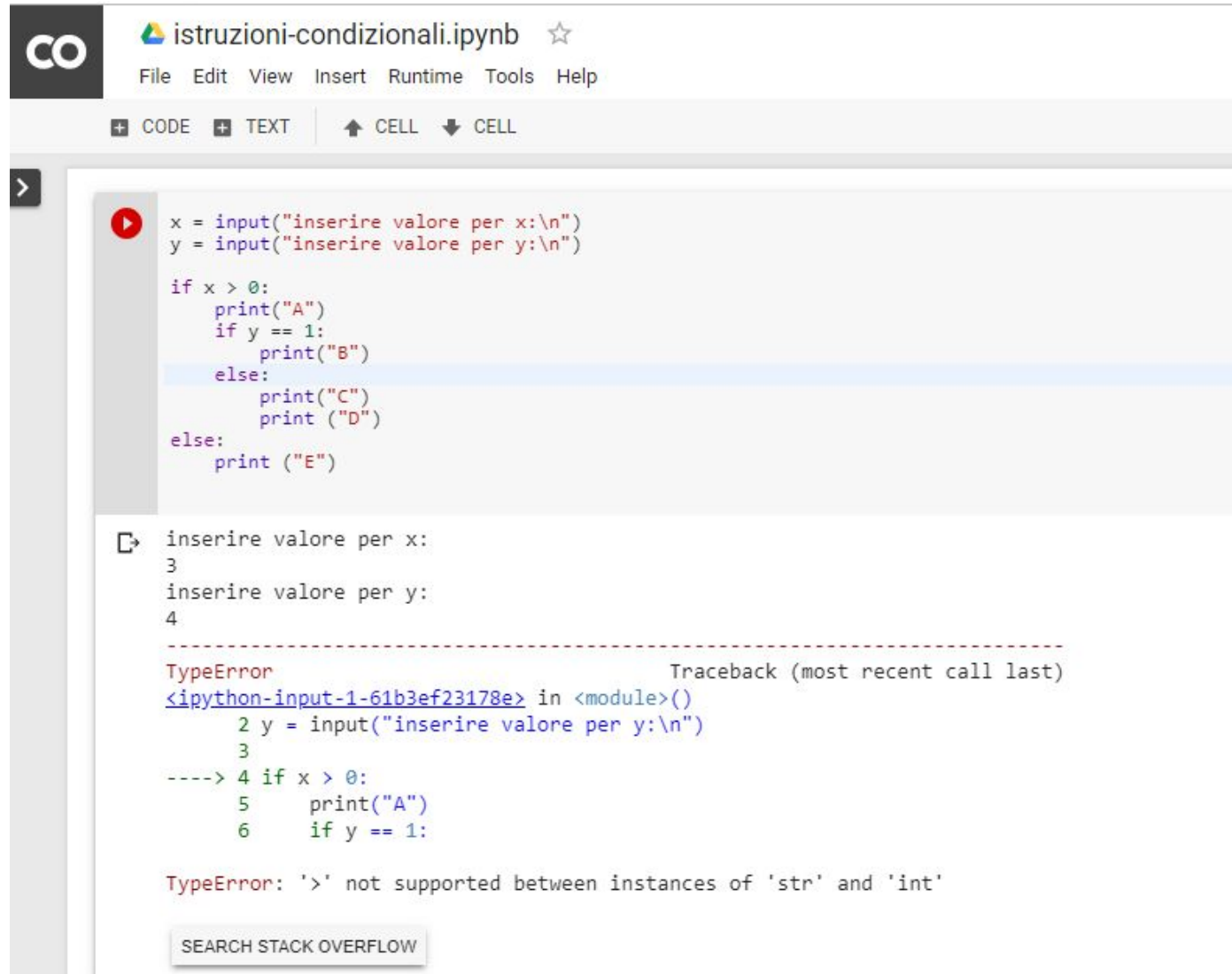
# Esercizio

---

Recuperare da tastiera i valori di x e y e verificare quale sia il risultato del codice seguente inserendo  $x = 3$  e  $y = 4$

```
if x > 0:
    print("A")
    if y == 1:
        print("B")
    else:
        print("C")
        print("D")
else:
    print("E")
```

# Esempio in Colab



```
x = input("inserire valore per x:\n")
y = input("inserire valore per y:\n")

if x > 0:
    print("A")
    if y == 1:
        print("B")
    else:
        print("C")
        print("D")
else:
    print("E")
```

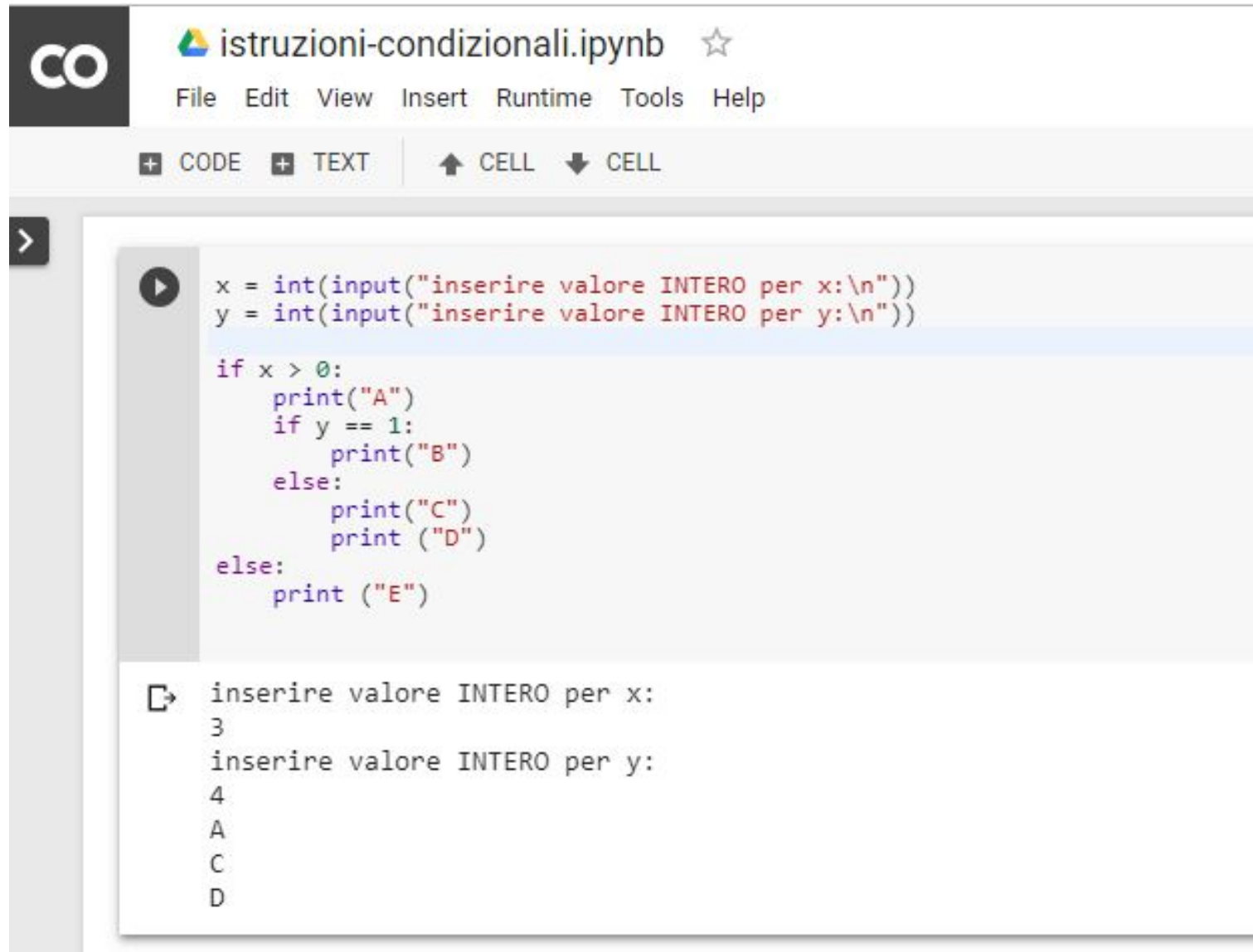
inserire valore per x:  
3  
inserire valore per y:  
4

-----  
TypeError Traceback (most recent call last)  
<ipython-input-1-61b3ef23178e> in <module>()  
 2 y = input("inserire valore per y:\n")  
 3  
----> 4 if x > 0:  
 5 print("A")  
 6 if y == 1:

TypeError: '>' not supported between instances of 'str' and 'int'

SEARCH STACK OVERFLOW

# Cast dei dati



The screenshot shows a Jupyter Notebook interface. At the top, there is a dark header with the 'CO' logo on the left, the file name 'istruzioni-condizionali.ipynb' in the center, and a star icon on the right. Below the header is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Underneath the menu bar is a toolbar with icons for '+ CODE', '+ TEXT', '↑ CELL', and '↓ CELL'. The main area contains a code cell with a play button icon on the left. The code in the cell is as follows:

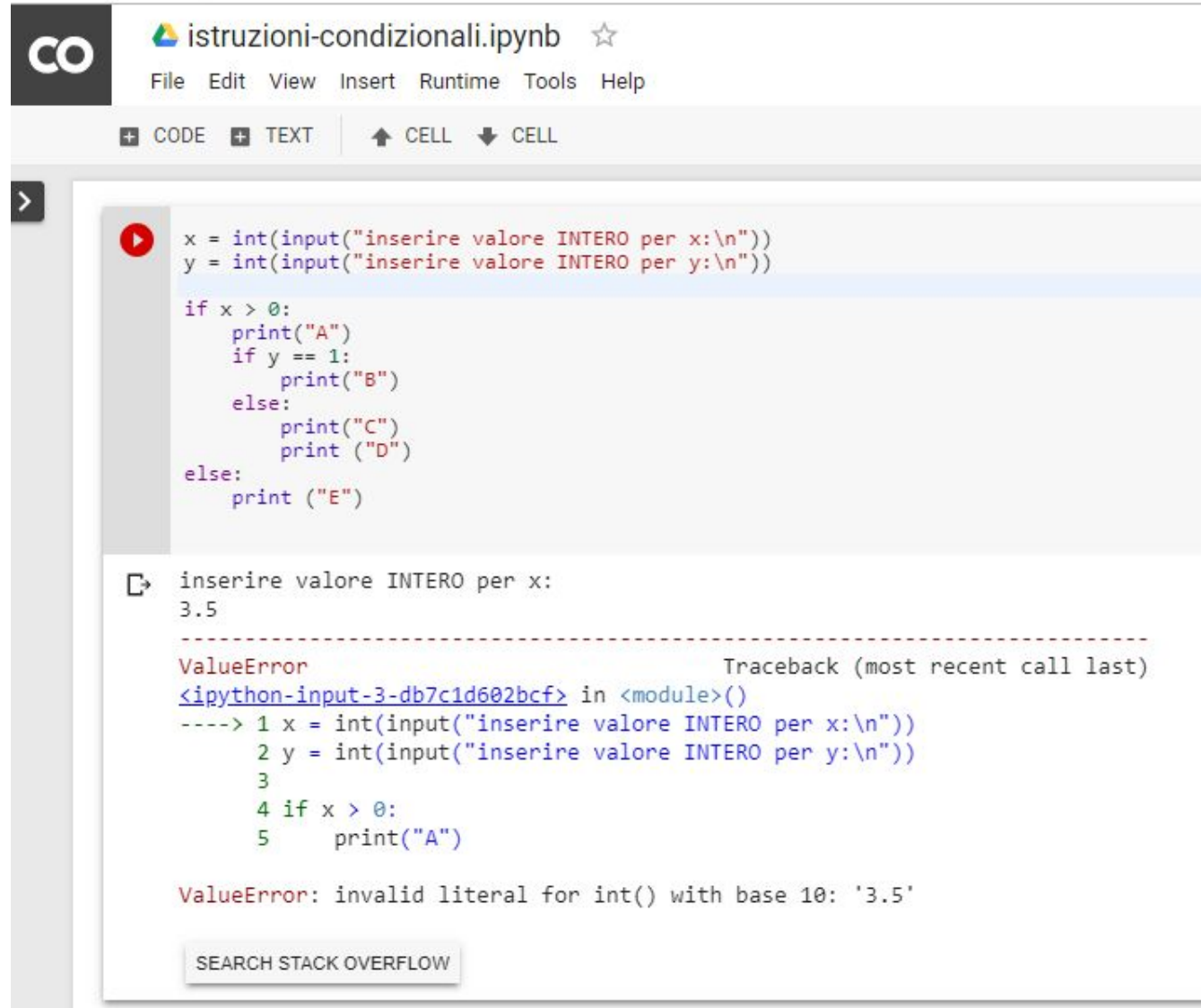
```
x = int(input("inserire valore INTERO per x:\n"))
y = int(input("inserire valore INTERO per y:\n"))

if x > 0:
    print("A")
    if y == 1:
        print("B")
    else:
        print("C")
        print ("D")
else:
    print ("E")
```

Below the code cell is an output cell with a copy icon on the left. The output is as follows:

```
inserire valore INTERO per x:
3
inserire valore INTERO per y:
4
A
C
D
```

# Verifica del tipo di dato



The screenshot shows a Jupyter Notebook interface with the following elements:

- Header:** "co" logo, "istruzioni-condizionali.ipynb" with a star icon, and a menu bar (File, Edit, View, Insert, Runtime, Tools, Help).
- Toolbar:** "+ CODE", "+ TEXT", "↑ CELL", "↓ CELL".
- Code Cell:** Contains Python code for input and conditional logic:

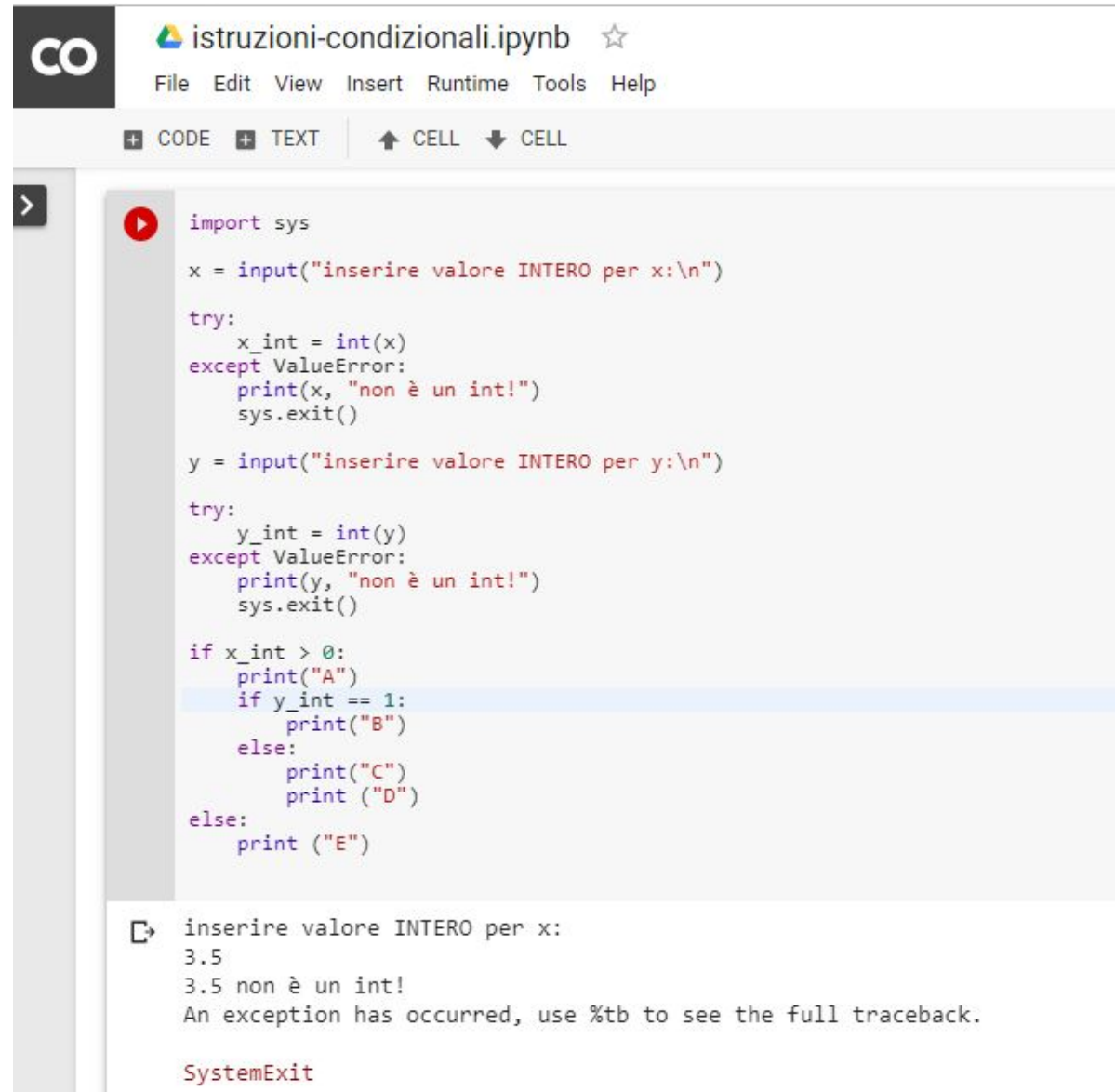
```
x = int(input("inserire valore INTERO per x:\n"))
y = int(input("inserire valore INTERO per y:\n"))

if x > 0:
    print("A")
    if y == 1:
        print("B")
    else:
        print("C")
        print("D")
else:
    print("E")
```
- Input:** A prompt "inserire valore INTERO per x:" followed by the user input "3.5".
- Output:** A `ValueError` exception with a traceback:

```
ValueError                                Traceback (most recent call last)
<ipython-input-3-db7c1d602bcf> in <module>()
----> 1 x = int(input("inserire valore INTERO per x:\n"))
      2 y = int(input("inserire valore INTERO per y:\n"))
      3
      4 if x > 0:
      5     print("A")

ValueError: invalid literal for int() with base 10: '3.5'
```
- Footer:** A button labeled "SEARCH STACK OVERFLOW".

# Gestione delle eccezioni



The screenshot shows a Jupyter Notebook titled "istruzioni-condizionali.ipynb". The code in the cell is as follows:

```
import sys

x = input("inserire valore INTERO per x:\n")

try:
    x_int = int(x)
except ValueError:
    print(x, "non è un int!")
    sys.exit()

y = input("inserire valore INTERO per y:\n")

try:
    y_int = int(y)
except ValueError:
    print(y, "non è un int!")
    sys.exit()

if x_int > 0:
    print("A")
    if y_int == 1:
        print("B")
    else:
        print("C")
        print("D")
else:
    print("E")
```

The output of the code execution is:

```
inserire valore INTERO per x:
3.5
3.5 non è un int!
An exception has occurred, use %tb to see the full traceback.

SystemExit
```

# Esercizio 1

---

Utilizzare il linguaggio di programmazione Python per stampare a video la scritta "prima esercitazione"



# Esercizio 2

---

Modificare il programma precedente per ottenere la stampa di

```
prima esercitazione  
e ne seguiranno altre  
ancora
```

# Esercizio 3

---

Creare un programma contenente istruzioni in Python per la stampa del proprio nome, cognome, e classe nel formato mostrato sotto

```
nome:           Domenico  
cognome:       Bloisi  
anno di nascita: 1982
```

# Esercizio 4

---

Scrivere del codice Python per richiedere all'utente di inserire da tastiera il proprio nome.

Una volta recuperato il nome, esso dovrà essere stampato a video.

Si veda l'esempio seguente:

```
inserisci il tuo nome: Domenico  
nome inserito: Domenico
```

# Esercizio 5

---

Scrivere un programma Python in grado di prendere in ingresso da tastiera un intero  $x$  e stampare a video il valore  $-x$

**Esempio 1:**

```
inserisci valore: 7
```

```
valore con segno invertito:
```

```
-7
```

**Esempio 2:**

```
inserisci valore: -8
```

```
valore con segno invertito:
```

```
8
```

# Esercizio 6

---

Modificare il programma precedente per ottenere la stampa di

`inserisci valore: -8`

`valore con segno invertito: 8`

# Esercizio 7

---

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

# Summary

- This chapter covered:
  - Decision structures, including:
    - Single alternative decision structures
    - Dual alternative decision structures
    - Nested decision structures
  - Relational operators and logical operators as used in creating Boolean expressions
  - String comparison as used in creating Boolean expressions
  - Boolean variables
  - Determining the state of the turtle in Turtle Graphics

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

Modulo di Sistemi di Elaborazione delle Informazioni

# Strutture Decisionali e Logica Booleana



**UNIVERSITÀ DEGLI STUDI DELLA BASILICATA**



Docente:  
**Domenico Daniele Bloisi**

