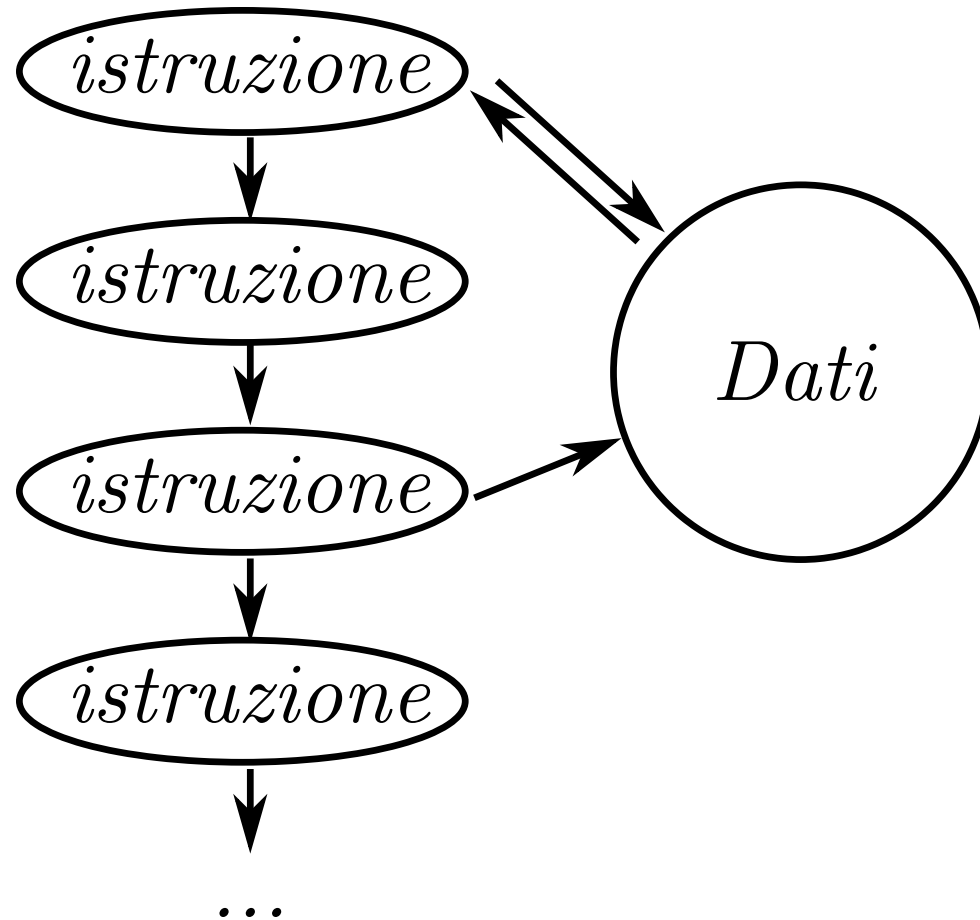


# Programmazione (scientifica) in python

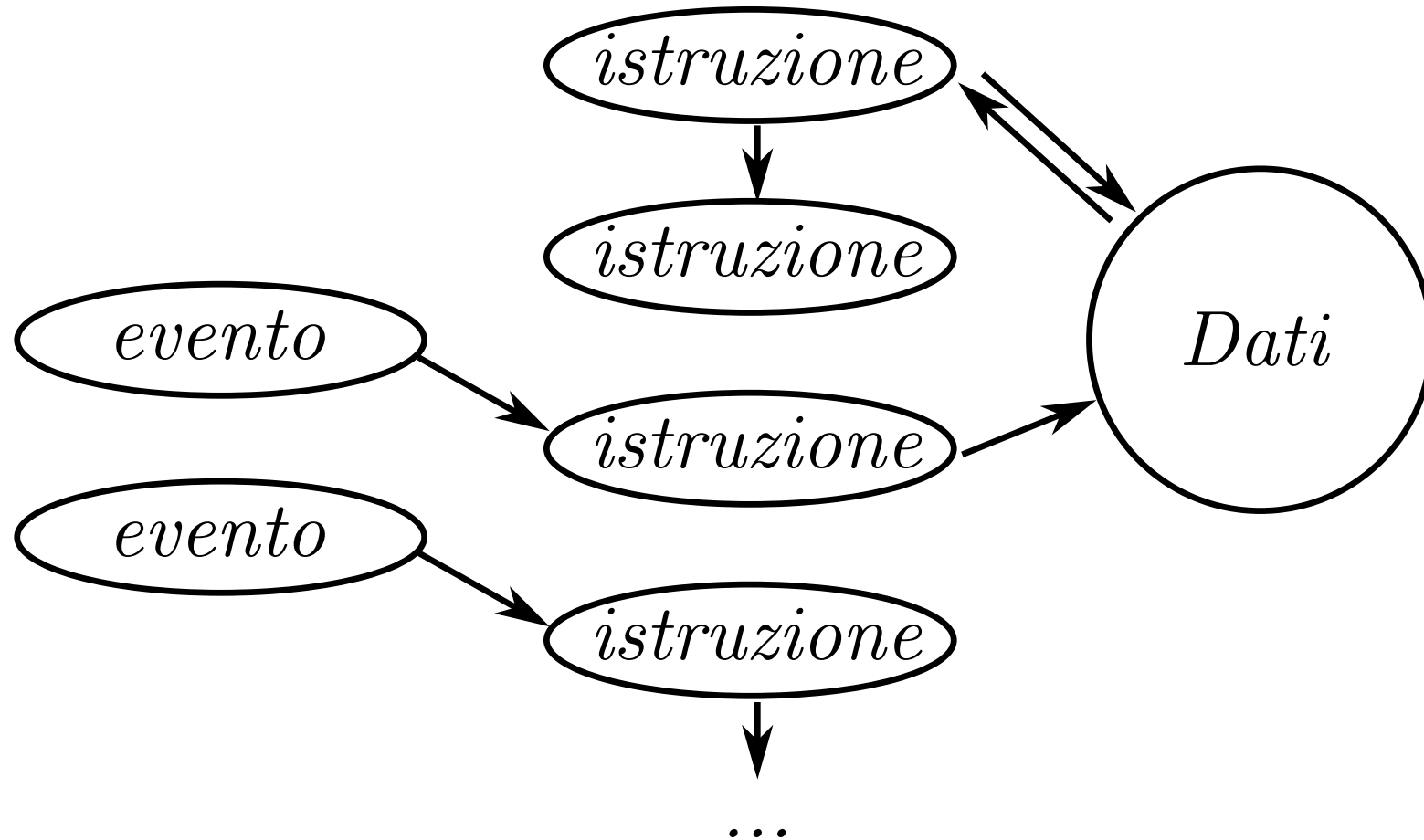
Liceo Scientifico e Sportivo Statale "A. Tassoni"

12/02/2018

# Modello "computazionale" (pre-1980)



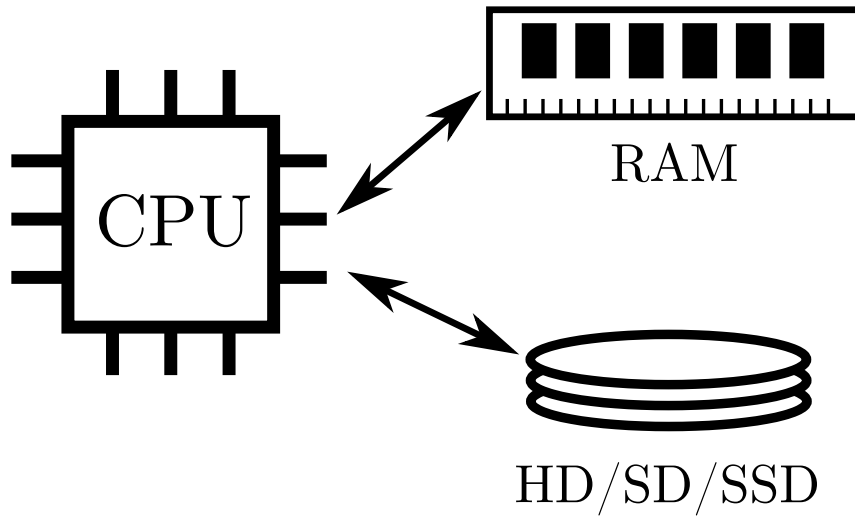
# Modello "computazionale" (post-1980)



... del quale non ci occuperemo...

# Componenti

?



# I computer sono (più) stupidi

Dal 1946 conoscono UNA SOLA lingua:

```
01001000 10000011 11101100 00011000  
11000111 01000100 00100100 00000100  
00000001 00000000 00000000 00000000  
11000111 00000100 00100100 00000010  
.....
```

ed altre decine di migliaia di istruzioni eseguono la somma

$$a = 1 + 2$$

# I computer sono (più) stupidi

Dal 1946 conoscono UNA SOLA lingua:

```
01001000 10000011 11101100 00011000
11000111 01000100 00100100 00000100
00000001 00000000 00000000 00000000
11000111 00000100 00100100 00000010
.....
```

ed altre decine di migliaia di istruzioni eseguono la somma

$$a = 1 + 2$$

Dati ed istruzioni sono in [codice binario](#).

# I linguaggi di programmazione...

... traducono istruzioni comprensibili per un essere umano,

ad es.  $a = 1 + 2$ ,

# I linguaggi di programmazione...

... traducono istruzioni comprensibili per un essere umano,

ad es.  $a = 1 + 2$ ,

... in istruzioni comprensibili per un computer (smartphone/tablet etc.)

```
01001000 10000011 11101100 00011000
11000111 01000100 00100100 00000100
00000001 00000000 00000000 00000000
11000111 00000100 00100100 00000010
.....
```



# I linguaggi di programmazione...

... traducono istruzioni comprensibili per un essere umano,

ad es.  $a = 1 + 2$ ,

... in istruzioni comprensibili per un computer (smartphone/tablet etc.)

```
01001000 10000011 11101100 00011000
11000111 01000100 00100100 00000100
00000001 00000000 00000000 00000000
11000111 00000100 00100100 00000010
.....
```

Esiste circa **un migliaio** di linguaggi di programmazione diversi.

# Il linguaggio Python™ ([www.python.org](http://www.python.org))

- il nome deriva dal gruppo comico inglese *Monty python*
- *non* è nuovissimo: prima versione rilasciata nel 1991 da Guido van Rossum (BFDL)



# Il linguaggio Python™ ([www.python.org](http://www.python.org))

- il nome deriva dal gruppo comico inglese *Monty python*
- *non* è nuovissimo: prima versione rilasciata nel 1991 da Guido van Rossum (BFDL)



... MA...

- è "relativamente semplice" da usare
- gira su tutti i sistemi operativi (Windows, Linux, Mac, etc...)
- è un linguaggio estremamente versatile (vedi prossima slide)
- è molto utilizzato nella comunità scientifica mondiale (vedi prossima slide)
- è un linguaggio "open source"
- documentazione dettagliata e completa  
[docs.python.org](http://docs.python.org)
- "a oggetti", "funzionale", "dinamico"

# Che ci faccio con python™?

## Programmazione scientifica: scienze pure

- **Matematica:** numpy, scipy, sympy, matplotlib, ...
- **Statistica:** pandas, pystan, pymc3, scikit-learn, ...

## Programmazione scientifica: scienze applicate

- **Astronomia:** astropy, sunpy, yt, gammapy, ...
- **Fisica:** pyroot, heppy, scikit-hep, EMpy, thermopy, CFDPython, ...
- **Chimica:** chempy, chemlab, ...
- **Biologia:** biopython, ...

# Che ci faccio con python™?

In più....

- **Programmazione web:** django, flask, ...
- **Programmazione per dispositivi (iOS, Android, arduino):** beeware, pyserial, ...
- **Elaborazione immagini:** pillow, scikit-image...
- **Grafica 3D:** blender, mayavi, ...
- **Crittografia:** pycrypto, cryptography, ...

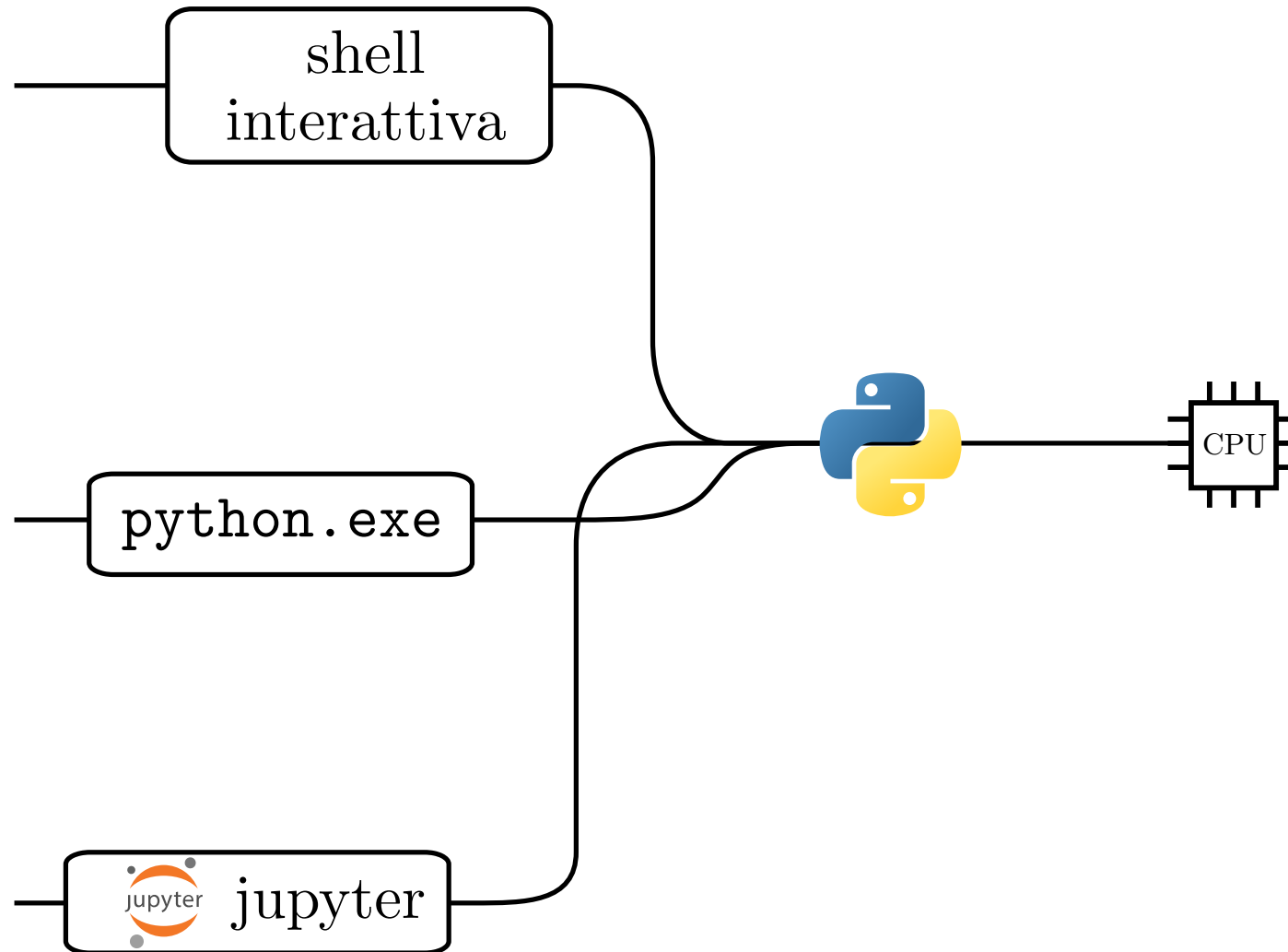
... ed altre 128100+ librerie sul PyPI...

# Un avvertimento prima di cominciare...

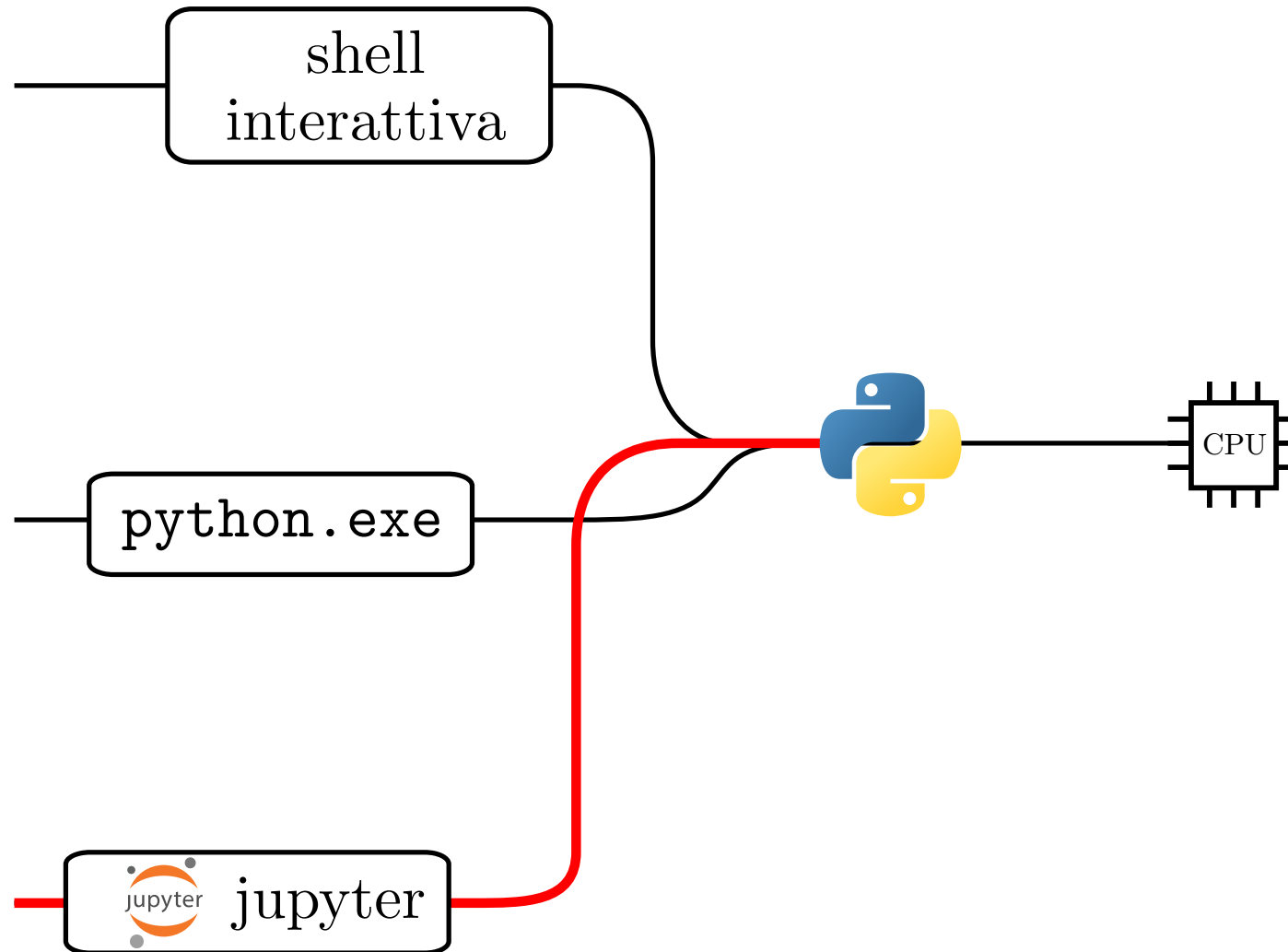
*A chi discende nello stesso fiume sopraggiungono acque sempre nuove.*

(Eraclito, DS 12)

# La triplice via di python



# La triplice via di python





# Output

```
In [ ]: 3
```

0

```
In [ ]: print(3)  
        print(3, 5, 28)
```

# Output

```
In [ ]: 3
```

0

```
In [ ]: print(3)
        print(3, 5, 28)
```

## Commenti

```
In [ ]: print(3) # questo è un commento
        # print(3) tutta questa riga non viene eseguita
```

# Matematica

## Addizione

```
In [ ]: 3 + 4
```

## Moltiplicazione

```
In [ ]: 13 * 24
```

## Elevamento a potenza

```
In [ ]: 16**3
```

# Matematica

## Divisione

```
In [ ]: 4/3
```

## Divisione intera

```
In [ ]: 4 // 3
```

## Resto della divisione

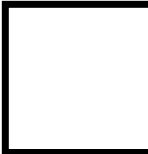
```
In [ ]: 1189 % 32
```

## Espressioni

```
In [ ]: (5+2)*((4 // 3)*3)**(3/5)
```

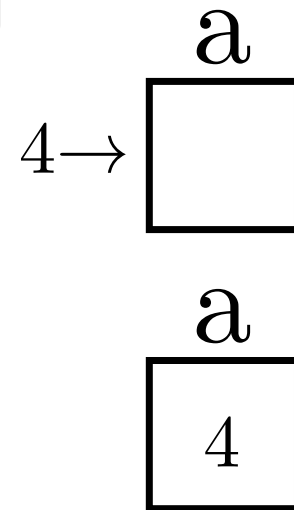
Variabili

a

A square box with a black border, positioned below the letter 'a'.

# Variabili

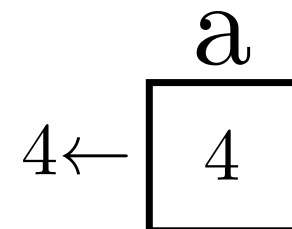
```
In [ ]: a = 4
```



# Variabili

```
In [ ]: a = 4
```

```
In [ ]: print(a)
```



# Variabili

```
In [ ]: a = 4
```

```
In [ ]: print(a)
```

```
In [ ]: a = 4  
        b = 28  
        c, d = -3, 10  
        print(a + b + c + d)
```



# Nomi di variabili

E' bene evitare i nomi di particolari "oggetti" che già hanno un altro significato in python, ad es

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

# Nomi di variabili

E' bene evitare i nomi di particolari "oggetti" che già hanno un altro significato in python, ad es

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Se si vuole dare un "significato" ad una variabile, si possono usare due sistemi:

```
variabile_molto_importante = 42  
variabileMoltoImportante = 38
```

# Nomi di variabili

E' bene evitare i nomi di particolari "oggetti" che già hanno un altro significato in python, ad es

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Se si vuole dare un "significato" ad una variabile, si possono usare due sistemi:

```
variabile_molto_importante = 42  
variabileMoltoImportante = 38
```

Python è *case-sensitive*:

```
a = 3  
A = 6  
print(a)
```

# Tipi di dati: numeri

## Interi (`int`)

```
n = 13  
m = -9930
```

compresi tra  $-2^{63}$  e  $2^{63} - 1$

## Reali (`float`)

```
x = -2.31  
y = 42.9e-23
```

dove  $y = 42,9 \cdot 10^{-23}$ . Sono compresi tra  $1,7 \cdot 10^{308}$  e  $2,2 \cdot 10^{-308}$

```
print(1e330, 1e-330)
```

# Tipi di dati: numeri

python sceglie automaticamente quale tipologia di numero da utilizzare...

```
a = 5      # <- int  
b = 5.0    # <- float
```

... e converte, quando si fanno le operazioni, nel tipo di numero necessario.

```
c = a / 2   # <- float  
print(c)
```

# Tipi di dati: stringhe

rappresentano una sequenza di caratteri (`str`)

```
s = "Del bel Panaro il pian, sotto due scorte,"  
t = 'A predar vanno i Bolognesi armati;'  
print(s)  
print(t)
```

# Tipi di dati: stringhe

rappresentano una sequenza di caratteri (`str`)

```
s = "Del bel Panaro il pian, sotto due scorte,"  
t = 'A predar vanno i Bolognesi armati;'  
print(s)  
print(t)
```

se si vuole dividere una stringa su più righe

```
s = """Del bel Panaro il pian, sotto due scorte,  
A predar vanno i Bolognesi armati;  
E da Gherardo altri condotti a morte,  
Altri dal Potta son rotti e fugati. """
```

# Operazioni con le stringhe

## Concatenazione

```
s, t = "Ciao ", "mondo!"  
print(s + t)
```

## Ripetizione

```
s, t = "Ciao ", "mondo!"  
print(s*3)  
print( (s+t)*3 + s )
```



# Operazioni con le stringhe

## Affettamento ("slicing")

```
s = "Del bel Panaro il pian, sotto due scorte"  
print(s[5])  
print(s[3:9])  
print(s[0:14:3])  
# ---  
print(s[39])
```

# Operazioni con le stringhe

## Affettamento ("slicing")

```
s = "Del bel Panaro il pian, sotto due scorte"  
print(s[5])  
print(s[3:9])  
print(s[0:14:3])  
# ---  
print(s[39])
```

```
print(s[-3])  
print(s[-10:-3])  
print(s[-3:-10])    # <- !!  
print(s[-3:-10:-1])
```

# Operazioni con le stringhe

## Affettamento ("slicing")

```
s = "Del bel Panaro il pian, sotto due scorte"  
print(s[5])  
print(s[3:9])  
print(s[0:14:3])  
# ---  
print(s[39])
```

```
print(s[-3])  
print(s[-10:-3])  
print(s[-3:-10])      # <- !!  
print(s[-3:-10:-1])
```

			N		e		l				m		e		z		z		o	
		+	---	+	---	+	---	+	---	+	---	+	---	+	---	+	---	+	---	+
[n]	->		0		1		2		3		4		5		6		7		8	
[n:m]	->	0		1		2		3		4		5		6		7		8		9
[-n]	->		-9		-8		-7		-6		-5		-4		-3		-2		-1	
[-n:-m]	->	-9		-8		-7		-6		-5		-4		-3		-2		-1		

# Operazioni con le stringhe

## Affettamento ("slicing")

```
print(s[:3])  
print(s[8:-2])  
print(s[7:2])
```

# Operazioni con le stringhe

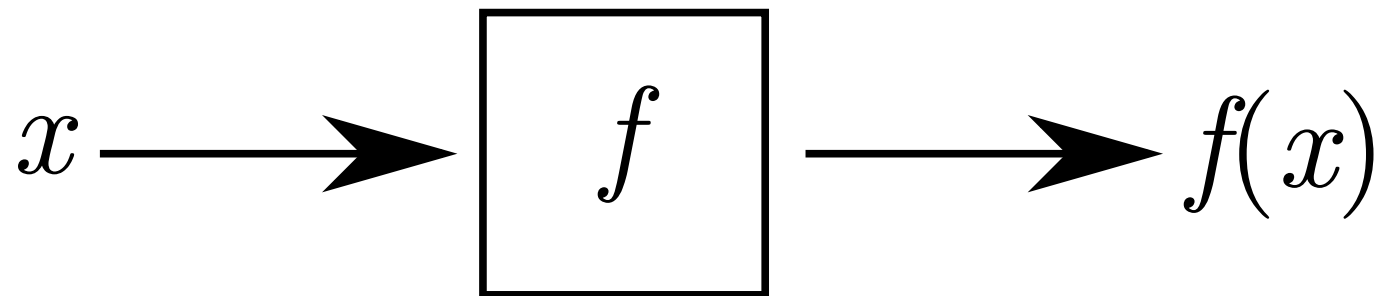
## Affettamento ("slicing")

```
print(s[::3])  
print(s[8::-2])  
print(s[:7:2])
```

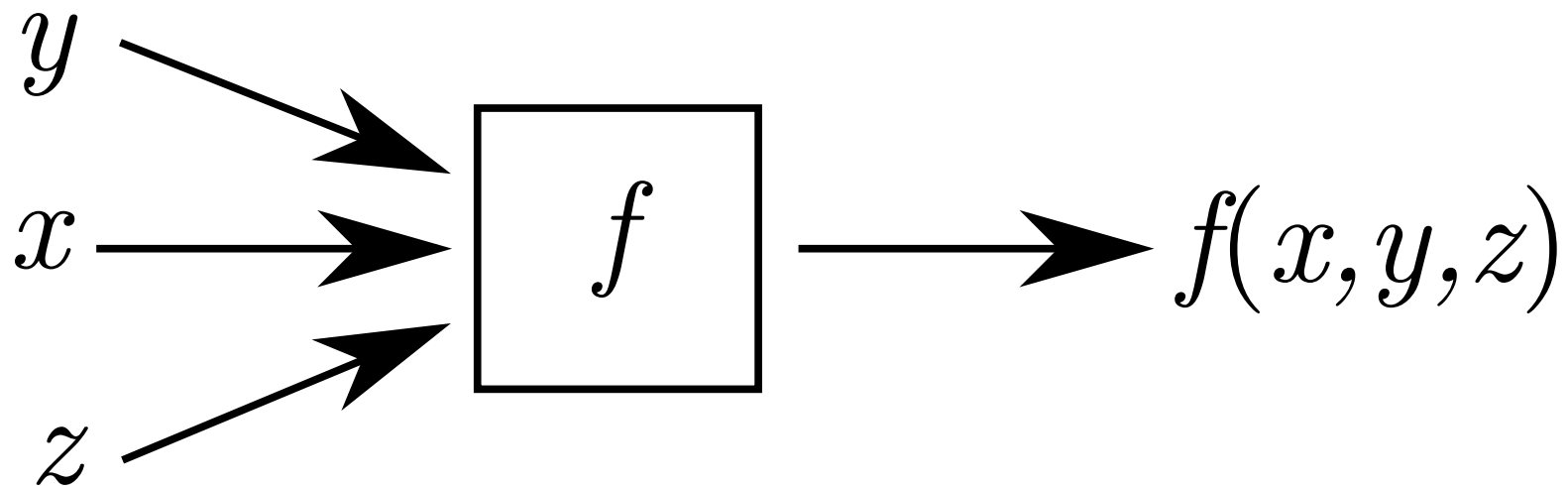
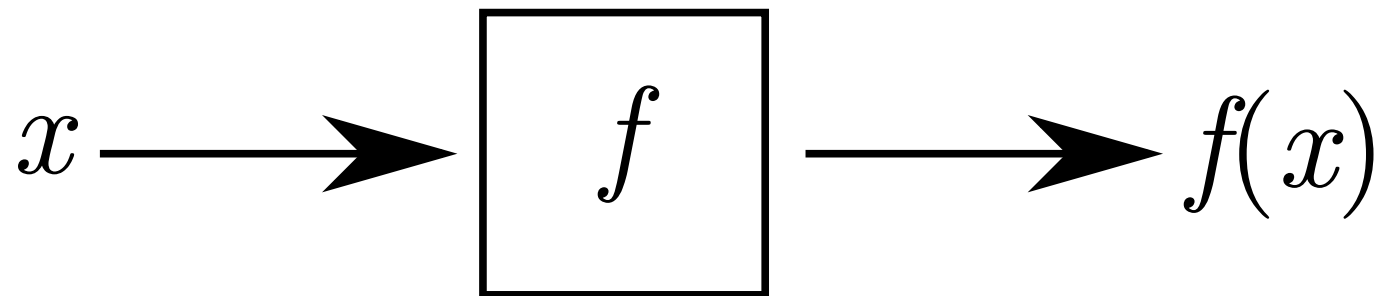
Le stringhe sono *immutabili*

```
s[0] = "n" # <- !!
```

# Fuzioni



# Fuzioni



# Funzioni

in `python` le funzioni si indicano

```
w = f(x, y, z)
```

[alcune funzioni](#) particolari sono già definite per l'ambiente `python`



# Funzioni

in python le funzioni si indicano

```
w = f(x, y, z)
```

[alcune funzioni](#) particolari sono già definite per l'ambiente python

## Lunghezza di una stringa (len)

```
s = "Del bel Panaro il pian, sotto due scorte"  
print("s è lunga", len(s))
```

# Tipi di dati: booleani

Corrispondono a due soli valori: vero (True) o falso (False) Sono il risultato delle operazioni di confronto:

```
a, b, c, d, e, f = 1, 2, 3, 4, 5, 6
print(a == b)    # <- uguale a
print(a < b)     # <- minore di
print(a > c)     # <- maggiore di
print(e != c)    # <- diverso da
print(e >= b)    # <- maggiore o uguale a
print(f <= c)    # <- maggiore o uguale a
```

+ altre due che vedremo più avanti (forse).

# Tipi di dati: booleani

Corrispondono a due soli valori: vero (True) o falso (False) Sono il risultato delle operazioni di confronto:

```
a, b, c, d, e, f = 1, 2, 3, 4, 5, 6
print(a == b)    # <- uguale a
print(a < b)     # <- minore di
print(a > c)     # <- maggiore di
print(e != c)    # <- diverso da
print(e >= b)    # <- maggiore o uguale a
print(f <= c)    # <- maggiore o uguale a
```

+ altre due che vedremo più avanti (forse).

Hanno priorità inferiore rispetto alle operazioni matematiche

```
print( f <= c*2 )
print( e-a > b )
```

# Tipi di dati: booleani

Obbediscono alla cosiddetta [algebra di Boole](#):

```
print( True and True )  
print( True and False )  
print( False or True )  
print( False or True )  
print( not False )  
print( (1+2 > 0) and (3*9 == 27))
```

# Tipi di dati: booleani

Obbediscono alla cosiddetta [algebra di Boole](#):

```
print( True and True )
print( True and False )
print( False or True )
print( False or True )
print( not False )
print( (1+2 > 0) and (3*9 == 27))
```

## Attenzione!

Alcuni particolari dati hanno un particolare "valore di verità", nonostante il dato sia di tipo diverso:

```
print( 0 == False )
print( 3 == True )
print( 3 == False )
print( 3 and True )   # <- !!
print( True and 3 )   # <- !!
print( "ciao" and True ) # <- !!
print( True and "ciao" ) # <- !!
print( not 3 )
print( not 0)
```

gli "operatori booleani" non sono commutativi se i dati sono di tipo diverso