labcalc_python_basics

November 12, 2025

1 Nozioni di base in Python

Non ci sono dichiarazioni in python!

[1]:
```
x = 1.2332233
y = -123.231222e2
z = 454
a = "ciao"
```

2 stampa sullo schermo delle variabili

[2]:
```
print("x = ", x)
```

```
x =  1.2332233
```

[3]:
```
print("x + y = ", x+y, "z = ", z)
```

```
x + y =  -12321.8889767 z =  454
```

[4]:
```
print(x,y,z)
```

```
1.2332233 -12323.1222 454
```

3 stampa formattata delle variabili

si usano gli stessi descrittori visti in C con le stesse regole - %f per razionali con la virgola - %d per interi - %g notazione compatta con cifre significative - %e notazione scientifica

In C avreste scritto

```
printf("x = %.3g \t y= %.3e \t z = %d", x,y,z)
```

[5]:
```
print("x = %.3g \t y= %.3e \t z = %d"%(x,y,z))
```

```
x = 1.23         y= -1.232e+04    z = 454
```

[6]:
```
pippo = "stringa lunga a piacere"
gino = 23
dino = pippo + str(gino)
```

stringhe sono come interi e razionali e si possono facilmente manipolare senza usare funzioni come `sprintf()`

[7]:
```
nome = "albert"
cognome = "einstein"
nome_cognome = nome + " " + cognome
print(nome_cognome)
```

```
albert einstein
```

4 if/else if/else if/elif/else

- si usa **elif** al posto di **else if**
- Non si usano le {} come in C
- indentazione + uso di : sostituisce le {} per indicare cicli e condizioni

[8]:
```
x = 0.23
if(x>0):
print("x>0")
elif(x<0):
print("x<0")
else:
print("x=0")
```

```
  Cell In[8], line 3
    print("x>0")
    ^
IndentationError: expected an indented block
```

[9]:
```
x = -0.6
if(x>0):
    print("x>0")
elif(x<0):
    print("x<0")
else:
    print("x=0")
```

```
x<0
```

5 ciclo while

```
while(altezza>0) {


}
```

In python 2^3 si scrive 2**3 (funzione potenza)

[10]:
```
altezza  = 10 # metri
t = 0 # sec
grav = 9.81 # m/s^2
h0 = altezza
while(altezza>0):
    print("t: %.2f s h: %.3f m"%(t, altezza) )
    altezza = h0 - 0.5*grav*t**2
    t = t + 0.05
```

```
t: 0.00 s h: 10.000 m
t: 0.05 s h: 10.000 m
t: 0.10 s h: 9.988 m
t: 0.15 s h: 9.951 m
t: 0.20 s h: 9.890 m
t: 0.25 s h: 9.804 m
t: 0.30 s h: 9.693 m
t: 0.35 s h: 9.559 m
t: 0.40 s h: 9.399 m
t: 0.45 s h: 9.215 m
t: 0.50 s h: 9.007 m
t: 0.55 s h: 8.774 m
t: 0.60 s h: 8.516 m
t: 0.65 s h: 8.234 m
t: 0.70 s h: 7.928 m
t: 0.75 s h: 7.597 m
t: 0.80 s h: 7.241 m
t: 0.85 s h: 6.861 m
t: 0.90 s h: 6.456 m
t: 0.95 s h: 6.027 m
t: 1.00 s h: 5.573 m
t: 1.05 s h: 5.095 m
t: 1.10 s h: 4.592 m
t: 1.15 s h: 4.065 m
t: 1.20 s h: 3.513 m
t: 1.25 s h: 2.937 m
t: 1.30 s h: 2.336 m
t: 1.35 s h: 1.711 m
t: 1.40 s h: 1.061 m
t: 1.45 s h: 0.386 m
```

6 Non esiste il ciclo do/while

si usa `break` con il ciclo while!

[11]:
```
while True:
  x = input("inserisci x>0: ")
  if(float(x)>0):
    break

x = float(x)
print("x = ", x)

if(x>0):
    print("x>0")
elif(x<0):
    print("x<0")
else:
    print("x=0")
```

```
inserisci x>0:  -6
inserisci x>0:  0.3243
```

```
x =  0.3243
x>0
```

7 Ciclo for

la funzione `range(min, max, step)` permette di ottenere l'equivalente del C

```
for(int i = 0; i< 10; i++) {

}
```

[12]:
```
for i in range(0,10,1):
  print("i = ", i)
```

```
i =  0
i =  1
i =  2
i =  3
i =  4
i =  5
i =  6
i =  7
i =  8
i =  9
```

[13]:
```
for i in range(0,20,3):
  print("i = ", i)
```

```
i =  0
i =  3
i =  6
```

```
i =  9
i =  12
i =  15
i =  18
```

8 Libreria matematica

In python su usano **moduli** al posto delle librerie

- I moduli vanno importati prima di essere usati.
- Un modulo si importa in un qualsiasi momento (non solo all'inzio) purchè prima di usarne le funzioni
- dato che i nomi sono lunghi e completi spesso si usa un nome abbreviato

[14]:
```
import math as m
```

[15]:
```
x = m.pi
print(x)
```

```
3.141592653589793
```

[16]:
```
print("x = %.60f"%x)
```

```
x = 3.141592653589793115997963468544185161590576171875000000000000
```

[17]:
```
y = m.sqrt( x)
print(y)
```

```
1.7724538509055159
```

[18]:
```
print("%.90f"%y)
```

```
1.772453850905515881919427556567825376987457275390625000000000000000000000000000000000
000000000000
```

9 Array (lista) in python

basta usare [] per creare un array di N valori

[19]:
```
dati = [ 1.23, 3.3453, -1.232, -4.532, 2, 45, 0 , -7.6]

print(dati)

print("lunghezza dati = ", len(dati))
```

```
[1.23, 3.3453, -1.232, -4.532, 2, 45, 0, -7.6]
lunghezza dati =  8
```

per ottenere l'equivalente di questo in C

```
for(int i=0; i< N; i++) {
  printf("dati[%d] = %.2f",i,dati[i])
}
```

si usa la funzione `len(dati)` per ottenere direttamente la lunghezza dell'array `dati`

[20]:
```
for i in range(0,len(dati)):
  print("dati[%d] = %.2f"%(i,dati[i]))
```

```
dati[0] = 1.23
dati[1] = 3.35
dati[2] = -1.23
dati[3] = -4.53
dati[4] = 2.00
dati[5] = 45.00
dati[6] = 0.00
dati[7] = -7.60
```

Il modo piu' naturale in python è di usare l'iteratore sugli elemento dell'array

[21]:
```
for val in dati:
  print("val = %.2f"%val)
```

```
val = 1.23
val = 3.35
val = -1.23
val = -4.53
val = 2.00
val = 45.00
val = 0.00
val = -7.60
```

possiamo aggiungere nuovi valori all'array estendendone la lunghezza

[22]:
```
print("# elementi dati: ", len(dati))

dati.append(9.8)

print("# elementi dati: ", len(dati))

print(dati)
```

```
# elementi dati:  8
# elementi dati:  9
[1.23, 3.3453, -1.232, -4.532, 2, 45, 0, -7.6, 9.8]
```

si possono anche usare gli indici come in C

[23]:
```
print( dati[0])
```

```
1.23
```

[24]:
```
print(dati[3])
```

```
-4.532
```

[25]:
```
print(dati[len(dati)-1])
```

```
9.8
```

[26]:
```
print(dati[-1])
```

```
9.8
```

[27]:
```
print(dati[-2])
```

```
-7.6
```

con gli indici negativi si accede dall'ultimi elemento indietro!