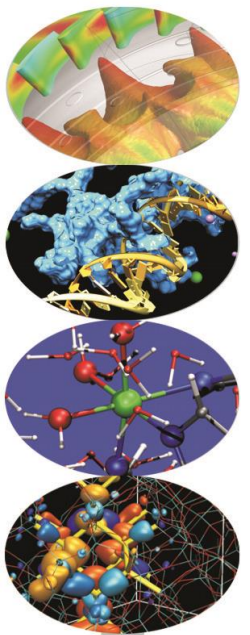
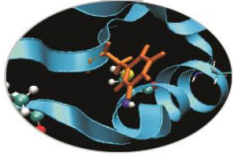


# Costrutti





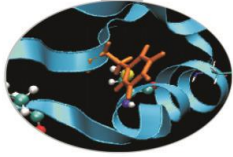
# Control Flow

Con *control flow* (o *strutture di controllo*) si intende l'insieme dei costrutti che servono a gestire il flusso di esecuzione di un programma, cioè a specificare quando e come eseguire gli statements del codice.

Nei più diffusi linguaggi di programmazione i costrutti variano ma possono essere classificati macroscopicamente in:

- *Costrutti condizionali*, l'esecuzione dipende dal raggiungimento di certe condizioni.
- *Costrutti iterativi*, gli statements vengono eseguiti zero o più volte.
- *Costrutti non locali*, gestione delle eccezioni.
- *Costrutti fondamentali*, funzioni, metodi.

Python in quanto linguaggio *imperativo* dispone dei più comuni costrutti di controllo. L'indentazione del codice fornisce l'accorpamento dei comandi in *blocchi* di statements, al posto delle parentesi usate nei più tradizionali linguaggi di programmazione, p.e. C/C++.

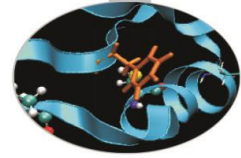


# If Statement

Il costrutto condizionale *if* ha la seguente sintassi:

```
if condition1:  
    block 1  
elif condition2:  
    block 2  
    ...  
elif conditionN:  
    block N  
else:  
    block statements
```

*If* è un costrutto condizionale: se la condizione è soddisfatta viene eseguito il blocco di istruzioni corrispondente altrimenti si analizzano le condizioni di verità dei blocchi *elif*.



# If Statement

Il blocco *if* può avere più blocchi *elif*.

*elif* è la versione abbreviata di *else if*. Il costrutto *elif* è opzionale.

*else* corrisponde al caso di default, come *elif* è opzionale.

In un *if statement* al più viene eseguito un unico blocco (*suite*) di istruzioni.

## Esempio

```
>>> x=2000
```

```
>>> if x<10:
```

```
    print 'Piccolo'
```

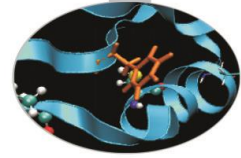
```
elif x<100:
```

```
    print 'Medio'
```

```
else:
```

```
    print 'Grande'
```

```
'Grande'
```



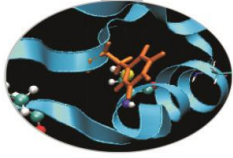
# If Statement

L'espressione condizionale che deve essere verificata utilizza i comuni operatori logici che il linguaggio mette a disposizione:

- Comparazione:  $<$ ,  $<=$ ,  $>$ ,  $>=$ ,  $==$ ,  $!=$
- Identità: *is*, *is not*
- Membership: *in*, *not in*
- Logical: *not*, *and*, *or*

## Esempio

```
if answer=='copy':  
    copyfile='file.tmp'  
elif answer=='run' or answer=='execute':  
    run=True  
elif answer=='quit':  
    quit=True  
else:  
    print "No Valid Answer "
```



# Switch Case in Python

Il costrutto *switch-case* permette di controllare il flusso del programma in base al valore di una variabile o espressione.

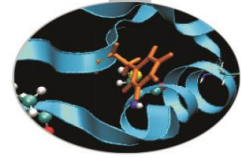
Python non dispone del costrutto condizionale switch, la gestione di espressioni condizionali con *N* sottocasi viene gestita con il costrutto *if-elif-else*.

Python ha un'alternativa più semplice ed espressiva per il costrutto *switch-case* presente in altri linguaggi di programmazione.

In molti linguaggi come in C/C++ il costrutto *switch -case* può gestire solo dei valori interi. La soluzione implementata in Python permette di gestire il costrutto *case* in modo generale.

Il funzionamento del costrutto *switch-case* può essere emulato utilizzando opportunamente gli array associativi – dictionary.

```
{key1: f1, key2: f2,...keyN: fN}
```



# For Statement

Il costrutto *for* è un costrutto iterativo che viene utilizzato per ripetere una suite di istruzioni un numero definito di volte.

La sintassi:

```
for <variable> in <sequence>:  
    block
```

Python può eseguire dei loop su qualsiasi tipo di sequenza di dati: tuple, liste, stringhe.

## Esempio

```
>>> s = 'Ciao'
```

```
>>> for i in s:
```

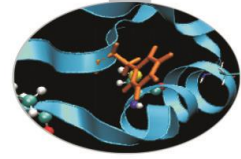
```
    print s
```

```
C
```

```
i
```

```
a
```

```
o
```



# For Statement

Per iterare su una sequenza di interi, Python dispone della funzione built-in `range([start,] stop[, step])` che restituisce una lista di interi .

Per ottimizzare l'efficienza del codice nei loop è preferibile utilizzare la funzione `xrange([start,] stop[, step])`.

Il vantaggio di `xrange` risiede nell'utilizzo della memoria: `xrange` richiede sempre la stessa quantità di memoria indipendentemente dalla dimensione del range di valori da rappresentare.

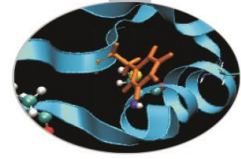
## Esempio

```
x=1000000000
t1=time.clock()
for el in range(x):
    pass
t2=time.clock()
print "Time ", t2-t1, "s"
Time 6.66 s
```

**VS**

```
x=1000000000
t1=time.clock()
for el in xrange(x):
    pass
t2=time.clock()
print "Time ",t2-t1, "s"
Time 1.9 s
```





# While Statement

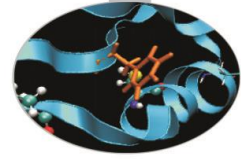
Il costrutto *while* permette di eseguire un blocco di istruzioni un numero indefinito di volte finché una particolare espressione risulta *True*.

La sintassi generale:

```
while (condition):  
    block
```

L'espressione *condition* viene valutata ad ogni ciclo e il ciclo viene ripetuto finché *condition==True*

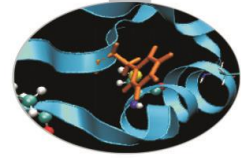
In Python non esiste il costrutto *do-while*. Il ciclo *while* viene eseguito almeno una volta se la condizione di partenza risulta *True*.



# While Statement

## Esempio

```
import random
number=random.randint(0,10)
guess=False
while(not guess):
    s=input('Inserisci un intero: ')
    if s==number:
        print 'Complimenti! Numero segreto indovinato'
        guess=True
    elif s<number:
        print 'Numero deve essere più grande'
        guess=False
    elif s>number:
        print 'Numero deve essere più piccolo'
        guess=True
```



# Continue - Break

L'esecuzione di un loop può essere interrotta qualora siano verificate condizioni eccezionali.

L'istruzione *break* interrompe l'esecuzione del loop. Per i cicli annidati, *break* interrompe esclusivamente l'esecuzione del loop più interno.

L'istruzione *continue* salta all'iterazione successiva di un ciclo. Nel caso di cicli annidati, *continue* agisce sul ciclo più interno.

**for** <variable> **in** >iterable:

statements1

**for** <variable2> **in** <iterable>:

**if** (condition2):

**continue**

**elif** (condition):

**break**

statements2

**while**(condition):

statements1

**if**(condition):

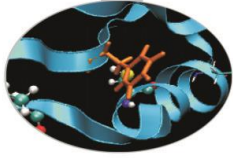
**continue**

**elif**(condition2):

**break**

statements2

# Continue - Break



**Esempio- break:** calcolo dei numeri primi fino ad n

```
n=raw_input('Inserisci un numero intero ')
```

```
n=int(n)
```

```
for k in range(3,n+1):
```

```
    flag=1
```

```
    print 'k', k
```

```
    for i in range(2,k):
```

```
        print 'i',i
```

```
        if k%i == 0:
```

```
            flag=1
```

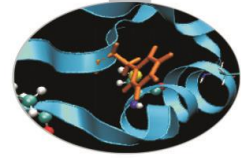
```
            break
```

```
    flag=0
```

```
if flag==0:
```

```
    print 'Numero primo:', k
```

# Continue - Break



## Output

Inserisci un numero intero 6

k 3

i 2

Numero primo: 3

k 4

i 2

k 6

i 2

k 5

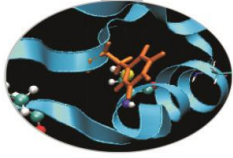
i 2

i 3

i 4

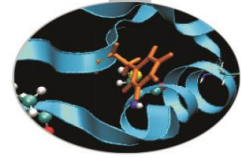
Numero primo: 5

# Continue - Break



**Esempio:** calcolo della distanza minima tra un punto  $p_1$  e un insieme di punti nel primo quadrante

```
from math import *
from pylab import*
lista=[(2,3),(4,5),(1,7),(-1,6),(-2,-4),(2,-8),(1,1)]
p1=input('Inserisci la coordinata lungo x: ')
p2=input('Inserisci la coordinata lungo y: ')
point=(p1,p2)
c_point=()
dist_min=0
cur_dist=-1
dist=10000000
```



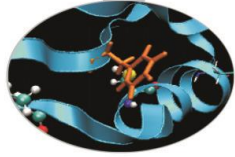
# Continue - Break

```
x=[]  
y=[]  
for k in range(len(lista)):  
    x.append(lista[k][0])  
    y.append(lista[k][1])  
    if lista[k][0]<0 or lista[k][1]< 0:  
        continue  
    cur_dist=sqrt((lista[k][0]-point[0])**2+(lista[k][1]-point[1])**2)  
    if cur_dist==0:  
        print 'Punti coincidenti'  
        break  
    if cur_dist<dist:  
        dist=cur_dist  
        c_point=lista[k]
```

**Jump**

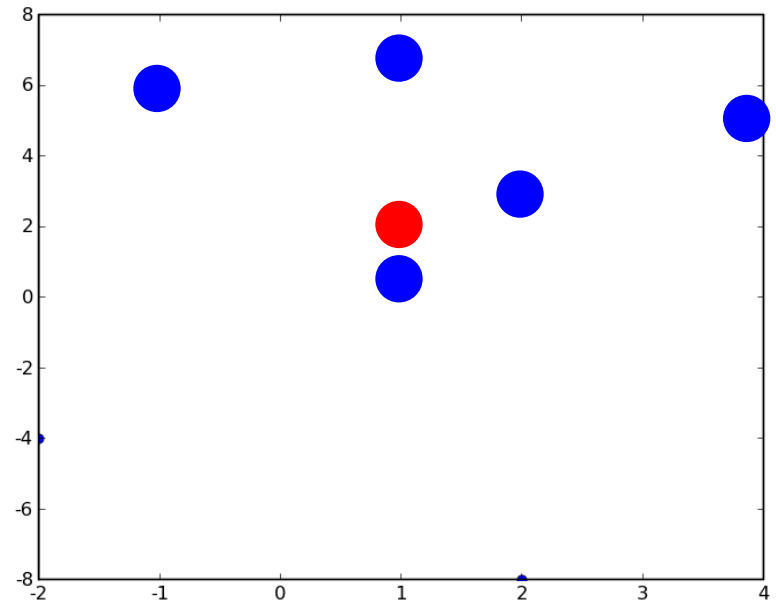
**Break**

# Continue - Break



```
print 'Distanza minima : ', dist
print 'Punto di minimo: ', c_point

plot(x,y,'o',[point[0]],[point[1]],'ro')
```



## #OUTPUT

Inserisci la coordinata lungo x: 1

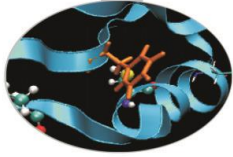
Inserisci la coordinata lungo y: 2

Distanza minima : 1.0

Punto di minimo: (1, 1)



# Pass



- Pass è un'istruzione generica che 'non fa niente'. Può essere inserita in qualsiasi parte del codice.
- E' utile in situazioni in cui è necessario uno statement per ragioni sintattiche.
- Esempio il corpo di una funzione non può essere omissso.

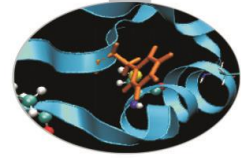
```
>>> def f(a,b):
```

```
    pass
```

```
>>>a=3
```

```
>>>b=4
```

```
>>>f(a,b)
```



# For While & Else Clause

Ai cicli condizionali *for* e *while* può essere associato un *else* statement, con la seguente sintassi:

```
for <variable> in <sequence>:
    block1
else:
    block2

while(condition1):
    block1
else:
    block2
```

Il blocco *else* viene eseguito al termine dell'esecuzione dei rispettivi cicli a patto che questi non siano terminati con l'istruzione *break*.

## Esempio

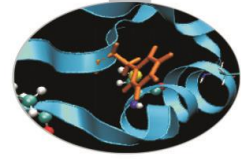
```
n2=1000
```

```
step=5
```

```
print "Number n1 is in range ( %d , %d) with step %d?"%(0,n2,step)
```

```
number=int(raw_input("Insert number n1: "))
```

# For While & Else Clause



```
for i in range(0,n2,step):  
    if i==n1  
        print 'n1 IS in range'  
        break  
else:  
    print 'n1 IS NOT in range'
```

## **#OUTPUT:**

Number n1 is in range ( 0 , 1000) with step 5

Insert n1: 77

n1 IS NOT in range

Insert n1: 55

n1 IS in range