Programmazione in Python per la bioinformatica

University of Verona

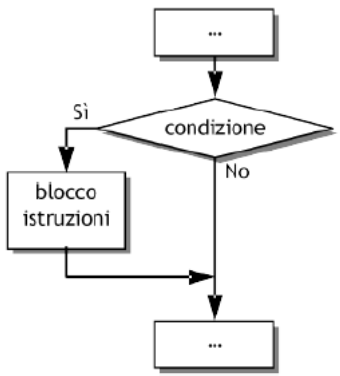
Sommario

- Selezione (if)
- Cicli (while – for)

Cambiare il flusso di esecuzione di un programma

Selezione Semplice

Selezione Semplice



Pseudocodice

```
....
SE condizione
    ALLORA
        blocco istruzioni
FINESE
...
```

```
>>> my_str = "acgtgtaggtatgagggtac"
>>> if 'atg' in my_str :
...     print("my_str contiene 'atg'")
...
my_str contiene 'atg'
>>> if 'ccc' in my_str :
...     print("my_str contiene 'ccc'")
...
>>>
```

in Python l'indentazione ha un significato sintattico!

```
>>> if 'ccc' not in my_str :
... print("ok")
  File "<stdin>", line 2
    print("ok")
        ^
IndentationError: expected an indented block
```

Condizioni

Boolean Expression

- la condizione nell'if e' una *Boolean Expression*
- Boolean expression: espressione che vale *true* o *false*
- Boolean expression: costruite con gli operatori di:
  - Comparazione (Comparison)
  - Appartenenza (Membership)
  - Identità (Identity)

Comparazione

Operatori di comparazione

```
>>> 'a' == 'A' #Equal
False
>>> 'a' != 'A' #Not equal
True
>>> 'a' < 'A' #less
False
>>> 'a' > 'A' #greater
True
>>> 10 <= 5*2 #less or equal
True
>>> 10 >= 5*2 #greater or equal
True
```

Appartenenza

Operatori di appartenenza

```
>>> 'atg' not in 'atgatgatgataga' #caratteri in una stringa
False
>>> 'atg' in ['atg','cta','cga'] #elemento in una lista
True
>>> 'atg' not in {'atg','cta','cga'} #elemento in un insieme
False
>>> 'atg' in ('atg','cta','cga') #elemento in una tupla
True
```

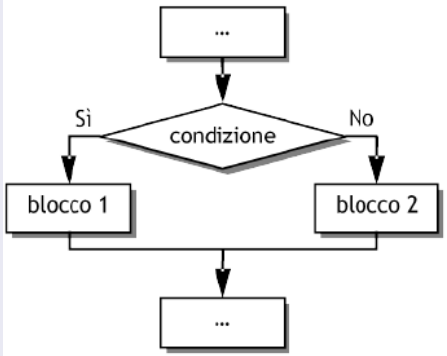
Identità

Operatori di identità

```
>>> my_list = ['atg','cga','tcg']
>>> my_list2 = my_list #assegno my_list a my_list2
>>> my_list2 == my_list #controlla uguaglianza
True
>>> my_list2 is my_list #controlla a quale oggetto puntano
True
>>> my_new_list = my_list[:] #creo una nuova lista (nuova memoria)
>>> my_new_list == my_list #controlla uguaglianza [true]
True
>>> my_new_list is my_list #controlla dove puntano [false]
False
```

Selezione a due vie

Selezione a due vie



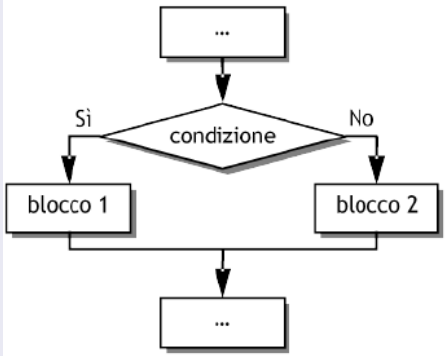
Pseudocodice

```
....
SE condizione
    ALLORA
        bbcco 1
    ALTRIMENTI
        bbcco 2
FINESE
...
```

```
>>> if 'cga' in 'cgacagttc' :
...     print("cga e' contenuto")
... else:
...     print("cga NON e' contenuto")
...
cga e' contenuto
```

Selezione a due vie

Selezione a due vie



Pseudocodice

```
....
SE condizione
    ALLORA
        bbcco 1
    ALTRIMENTI
        bbcco 2
FINESE
...
```

```
>>> if 'cga' in 'cgacagttc' :
...     print("cga e' contenuto")
... else:
...     print("cga NON e' contenuto")
...
cga e' contenuto
```

Selezione ad *n* vie

Selezione ad *n* vie

Controlliamo varie condizioni in una sola struttura di selezione

elif: parola chiave per rappresentare *else if*

```
>>> my_str = 'atcatcgaAAatcga'
>>> if 'a' in my_str :
...     print("%s contiene il carattere 'a'" % my_str)
... elif 'A' in my_str :
...     print("%s contiene il carattere 'A'" % my_str)
... else:
...     print("%s non contiene 'a' o 'A'" % my_str)
...
atcatcgaAAatcga contiene il carattere 'a'
```

Operatori Logici

Operatori Logici

- *and*: vero se entrambe le cond. sono vere
- *or*: vero se almeno una delle due cond. è vera
- *not*: vero se la condizione è falsa

```
>>> my_str = 'atcatcgaAAatcga'
>>> if 'a' in my_str or 'A' in my_str :
...     print("%s contiene 'a' oppure 'A'" % my_str)
... else:
...     print("%s non contiene 'a' o 'A'" % my_str)
...
atcatcgaAAatcga contiene 'a' oppure 'A'
```

Esercizi su Selezione

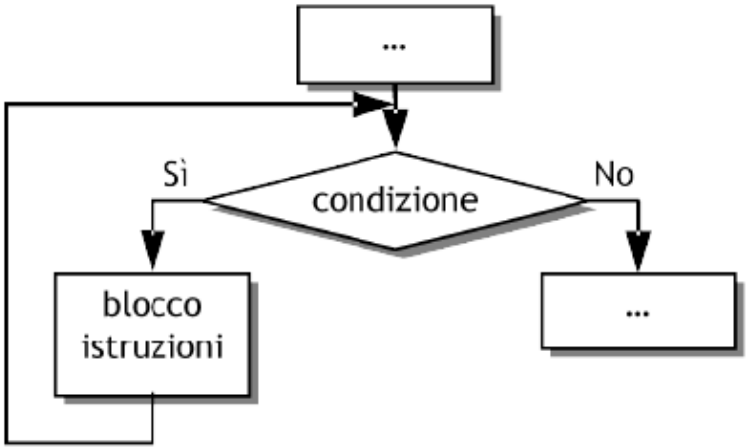
Eserc if

Q1 scrivere uno script che chieda all'utente una stringa di dna. La stringa viene considerata valida se contiene solo caratteri 'atcg'. Controllare se la stringa e' valida e stampare un messaggio corrispondente. Scaricare e modificare il file eserc-if-Q1.py [Sol: eserc-if-Q1.sol]

Q2 scrivere uno script che prende in input un nucleotide e stampa il nucleotide complementare (i.e., a:t,c:g,g:c,t:a). Se il carattere non e' un nucleotide stampa un messaggio di errore. Scaricare e modificare il file eserc-if-Q2.py [Sol: eserc-if-Q2.sol]

Il concetto di ciclo

Ripetere le istruzioni



Pseudocodice

```
....
QUANDO condizione ESEGUI
    blocco istruzioni
RIPETI
....
```

Ciclo while

Trovare gli indici di tutte le occorrenze di 'atg' in una stringa di dna

loop-example.py

```
dna = input("inserisci una sequenza di dna:\n")
pos = dna.find('atg');
while pos>-1 :
    print("'atg' found at %d" % pos)
    pos = dna.find('atg',pos+1)
```

```
inserisci una sequenza di dna:
atgttaatatgatgaaggatg
'atg' found at 0
'atg' found at 8
'atg' found at 11
'atg' found at 18
```

Ciclo for

Data una lista di stringhe ritornare il numero di occorrenze della sottostringa 'atg' in ciascuna stringa

loop-for-example.py

```
my_list = ['atgatg','cattagc','acct']
for dna in my_list :
    print("%s %d" % (dna,dna.count('atg')))
```

```
atgatg 2
cattagc 0
acct 0
```

La funzione range

Utilizzo della funzione range

loop-range-example.py

```
print("range(10):")
for i in range(10) : #0,1,2,...,9
    print("%d" % i,end = ',')
print()

print("range(1,11):")
for i in range(1,11) : #1,..,10
    print("%d" % i,end = ',')
print()

print("range(2,10,2):")
for i in range(2,10,2) : #2,..,8
    print("%d" % i,end = ',')
print()
```

La funzione range esecuzione

Esecuzione

```
range(10):
0,1,2,3,4,5,6,7,8,9,
range(1,11):
1,2,3,4,5,6,7,8,9,10,
range(2,10,2):
2,4,6,8,
```

La funzione range esempio

Stampare tutti gli amminoacidi non validi all'interno di una data proteina

protein-example.py

```
protein='SDVIHRYKUUPAKSHGWYVCJRSRFTWMVWWRFRSCRA'
valid='ABCDEFGHIKLMNPQRSTVWXYZ'
for i in range(len(protein)):
    if protein[i] not in valid:
        print("amm. invalido %s in posizione %d"%(protein[i],i))
```

```
amm. invalido U in posizione 8
amm. invalido U in posizione 9
amm. invalido J in posizione 20
```

Uscire dai cicli: *break* e *else* (per cicli!)

Decidere se una proteina è valida

protein-break-example.py

```
protein='SDVIHRYKUUPAKSHGWYVCJRSRFTWMVWWRFRSCRA'
valid='ABCDEFGHIKLMNPQRSTVWXYZ'
for i in range(len(protein)):
    if protein[i] not in valid:
        print("proteina non valida")
        break;
else:
    print("proteina valida")
```

- break: esce dal ciclo che lo contiene
- else: eseguito al termine del ciclo (e.g., quando la condizione del while diventa falsa)

Evitare una iterazione del ciclo: *continue*

Correggere una proteina togliendo i caratteri non validi

protein-continue-example.py

```
protein='SDVIHRYKUUPAKSHGWYVCJRSRFTWMVWWRFRSCRA'
valid='ABCDEFGHIKLMNPQRSTVWXYZ'
correct_protein = ''
for i in range(len(protein)):
    if protein[i] not in valid:
        continue
    correct_protein = correct_protein + protein[i]

print("la sequenza corretta e' :%s" % correct_protein)
```

- continue: passa alla prossima iterazione del ciclo che lo contiene

L'istruzione *pass*

pass: placeholder

protein-pass-example.py

```
protein='SDVIHRYKUUPAKSHGWYVCJRSRFTWMVWWRFRSCRA'
valid='ABCDEFGHIKLMNPQRSTVWXYZ'
for i in range(len(protein)):
    if protein[i] not in valid:
        pass #parte di codice da implementare
else:
    print("proteina valida")
```

- *pass*: non esegue nessuna funzione, utile per rispettare la sintassi del linguaggio
- spesso usato in fase prototipale del codice

Cicli annidati

Posso annidare i cicli

nested-loops.py

```
"""
per ogni carattere stampo gli indici in cui
compare nel resto della stringa
"""
my_str = "actcacggatag"
for i in range(len(my_str)):
    print("%s:" % my_str[i], end=" ")
    for j in range(i+1,len(my_str)):
        if (my_str[i] == my_str[j]):
            print("%d" % j, end = " ")
    print()
```

Esercizi su Cicli

Eserc loops

Q1 scrivere uno script che calcola un numero randomico (X) e chiede all'utente di indovinarlo. Lo script prosegue fino a che l'utente non ha indovinato il numero o ha eseguito un numero massimo di tentativi (e.g., 5). Ad ogni step lo script chiede un numero all'utente (N), se N e' uguale ad X termina e stampa un messaggio di successo. Altrimenti comunica all'utente se N e' maggiore o minore di X. eserc-loops-Q1.py [Sol: eserc-loops-Q1.sol]

Esercizi su Cicli annidati

Eserc nloops

Q1 data una stringa stampare tutte le sottostringhe della stringa data. Usare il ciclo for. Scaricare e modificare il file eserc-nloops-Q1.py [Sol: eserc-nloops-Q1.sol]

Q2 risolvere l'esercizio Q1 usando il ciclo while. Scaricare e modificare il file [Sol: eserc-nloops-Q2.sol]