Corso Python

Lezione A03 – Programmazione strutturata

Alberto Montresor

Università di Trento

2021/01/30

Acknowledgments: Stefano Teso

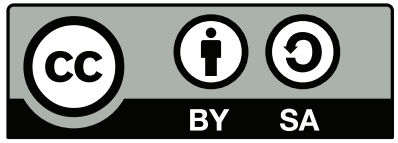
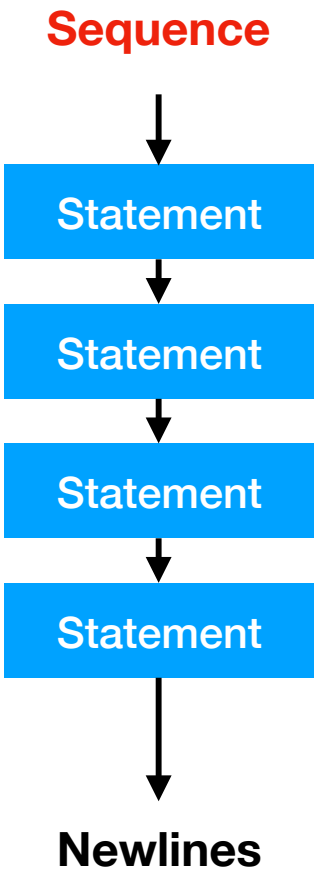
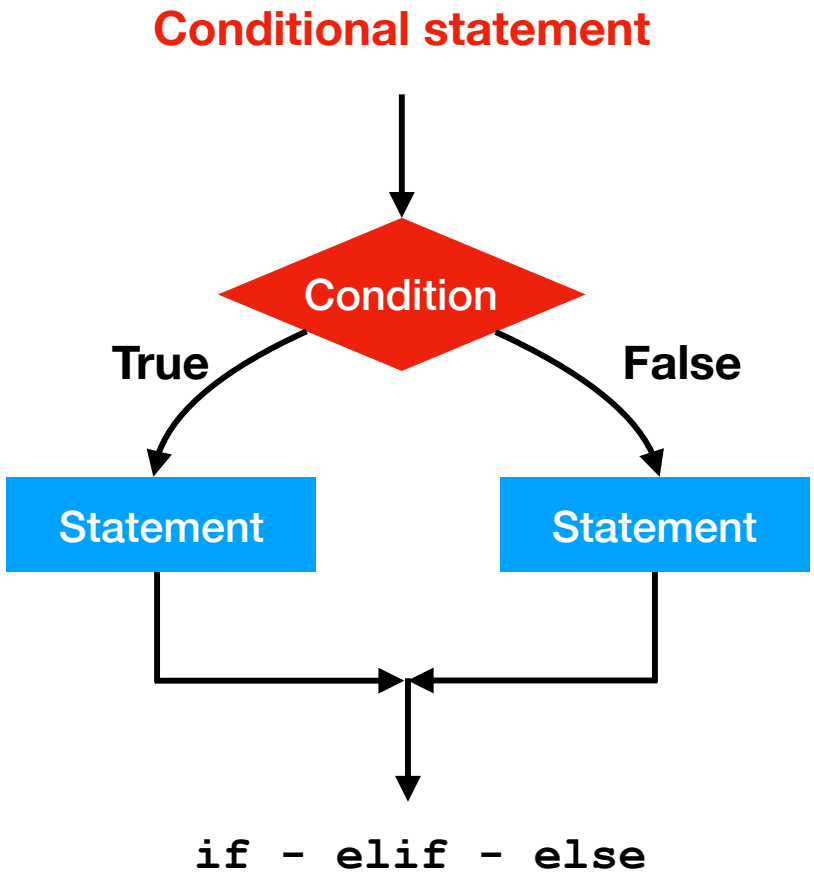


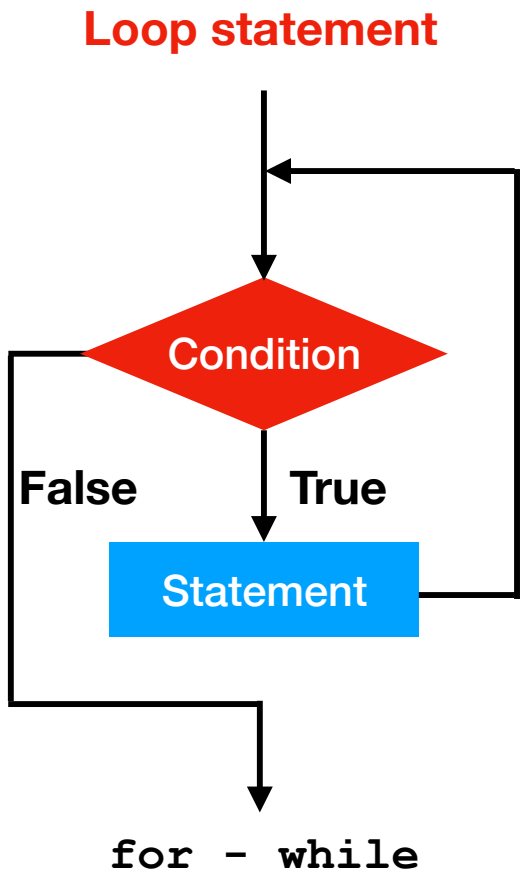
Table of contents

Programmazione strutturata

Dopo aver esaminato gli i tipi primitivi, esaminiamo le strutture di controllo di base:





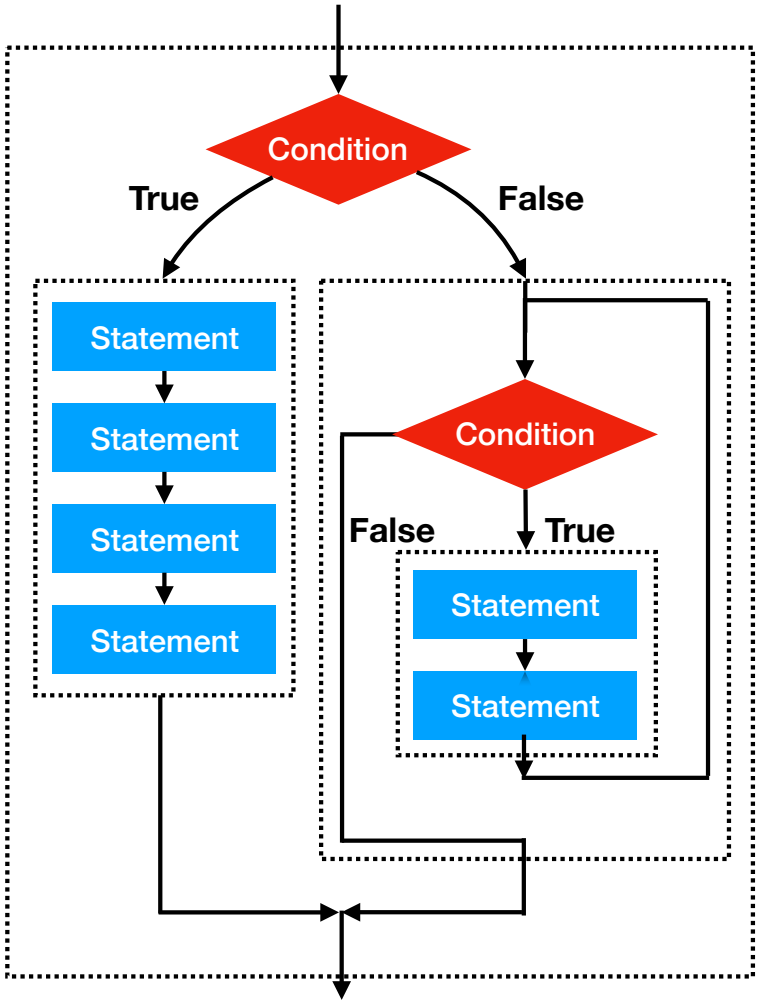


Programmazione strutturata

Blocchi

I blocchi sono utilizzati per raccogliere insieme gruppi di istruzioni.

I linguaggi strutturati hanno una sintassi per identificare i blocchi in modo formale



Alcuni semplici esempi

Ciclo for

```
L = [1,2,3,4]
print("---------------")
for val in L:
    print(">> ", end="")
    print(val, val*val)
print("---------------")
```

Alcuni semplici esempi

Ciclo for

```
L = [1,2,3,4]
print("---------------")
for val in L:
    print(">> ", end="")
    print(val, val*val)
print("---------------")
```

```
---------------
>> 1 1
>> 2 4
>> 3 9
>> 4 16
---------------
```

Istruzioni if - else

```
answer = input()
if (answer.lower() == "yes"):
    print("Good!")
else:
    print("Too bad!")
```

Python vs resto del mondo: Indentazione

Python

I blocchi sono identificati dall'indentazione

```
L = [1,2,3,4]
print("----")
for val in L
    print("** ", end="")
    print(val, val*val)
print("----")
```

Java

I blocchi sono identificati da parentesi graffe

```
System.out.println("--------")
for (int val=1; val <= 4; i++) {
  System.out.print("** ");
  System.out.println(val + " "
      + val*val)
}
System.out.println("--------")
```

Regole

- End-of-line termina un'istruzione
- End-of-indentation termina un blocco

Errori comuni

```
L = [1,2,3,4]
print("---------------")
for val in L:
print("** ", end="")
print(val, val**2)
print("---------------")
```

```
grannysmith:~ montreso$ python errors.py
  File "errors.py", line 4
    print("** ", end="")
        ^
IndentationError:  expected an indented block
```

Errori comuni

```
L = [1,2,3,4]
print("---------------")
for val in L:
      print("** ", end="")
    print(val, val**2)
print("---------------")
```

```
grannysmith:~ montreso$ python errors.py
  File "errors.py", line 5
    print(val, val**2)
                     ^
IndentationError:  unindent does not match any
outer indentation level
```

Errori comuni

```
L = [1,2,3,4]
print("---------------")
for val in L:
    print("** ", end="")
      print(val, val**2)
print("---------------")
```

```
grannysmith:~ montreso$ python errors.py
  File "errors.py", line 5
    print(val, val**2)
    ^
IndentationError:  unexpected indent
```

Altri dettagli sulla formattazione

Il contenuto delle parentesi `()`, `[]`, `{ }` può essere diviso su più linee. In assenza di parentesi, si ottiene un errore.

```
L = [ "A Game of Thrones", "A Clash of Kings",
      "A Storm of Swords", "A Feast for Crows",
      "A Dance with Dragons"
]
S1 = ("abcefghijklmnopqrstuvwxyz" +
       "ABCDEFGHIJKLMNOPQRSTUVWXYZ")
S2 = "abcefghijklmnopqrstuvwxyz" +
       "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

grannysmith:~ montreso$ python prova.py
  File "prova.py", line 12
    S2 = "abcefghijklmnopqrstuvwxyz" +
SyntaxError:  invalid syntax
```

Altri dettagli sulla formattazione

Sequenze in-line

Le sequenze di istruzioni sono normalmente scritte una per riga. E' possibile usare `;` per accorpare più istruzioni in una sola riga.

```
a = 1 ; b = 2 ; c = a + b
```

Comandi strutturati in-line

È possibile evitare l'indentazione in comandi strutturati, scrivendo una singola istruzione nella stessa riga.

```
if val > maximum: maximum = val; pos = i
```

Entrambi gli approcci sono considerati non-pitonici e sconsigliati

Table of contents

Istruzioni `if` – `else`

Istruzioni condizionali

Un'istruzione condizionale permette di scrivere del codice che viene eseguito se e solo se una certa condizione è soddisfatta.

```
print("Guess what number I'm thinking: ")
guess = input()
if guess == 5633839494:
    print("It's correct!")
else:
    print("Sorry, wrong number")
```

BTW, vedete qualche problema in questo codice?

Istruzioni `if` – `else`

Istruzioni condizionali

Un'istruzione condizionale permette di scrivere del codice che viene eseguito se e solo se una certa condizione è soddisfatta.

```
print("Guess what number I'm thinking: ")
guess = int(input())
if guess == 5633839494:
    print("It's correct!")
else:
    print("Sorry, wrong number")
```

Il valore di ritorno di `input()` è una stringa, confrontata con un numero restituisce sempre `False`

Istruzioni `if` – `elif` – `else`

`if`, `elif` e `else` formano una "catena": solo il primo ramo la cui condizione restituisce `True` viene eseguito.

```
print("Insert a number between 1 and 4")
value = int(input())
if value == 1:
    print("Very good choice")
elif value == 2:
    print("The first even number")
elif value == 3:
    print("The first Fermat number")
elif value == 4:
    print("An highly totient number")
else:
    print("I said, between 1 and 4")
```

Istruzioni `if` – `elif` – `else`

`if`, `elif` e `else` formano una "catena": solo il primo ramo la cui condizione restituisce `True` viene eseguito.

Qual è la differenza fra questi due pezzi di codice?

```
value = 2
if (value == 2):
  print("Even and prime")
elif (value % 2 == 0):
  print("Even")
```

```
value = 2
if (value == 2):
  print("Even and prime")
if (value % 2 == 0):
  print("Even")
```

Istruzioni `if` – `elif` – `else`

`if`, `elif` e `else` formano una "catena": solo il primo ramo la cui condizione restituisce `True` viene eseguito.

Qual è la differenza fra questi due pezzi di codice?

```
value = 2
if (value == 2):
  print("Even and prime")
elif (value % 2 == 0):
  print("Even")
```

```
Even and prime
```

```
value = 2
if (value == 2):
  print("Even and prime")
if (value % 2 == 0):
  print("Even")
```

```
Even and prime
Even
```

Istruzioni `if` – `elif` – `else`

Cosa viene stampato da questo pezzo di codice?

```
if condition1:
    print("A")
elif condition2:
    print("B")
else:
    print("C")
```

	condition2==True	condition2==False
condition1==True		
condition1==False		

Istruzioni `if` – `elif` – `else`

Cosa viene stampato da questo pezzo di codice?

```
if condition1:
    print("A")
elif condition2:
    print("B")
else:
    print("C")
```

	`condition2==True`	`condition2==False`
`condition1==True`	A	A
`condition1==False`	B	C

Istruzioni `if` – `elif` – `else`

Come possiamo riottenere il comportamento descritto dalla tabella, senza usare `elif` / `else`, ma usando gli operatori `and`/`or`/`not`?

	`condition2==True`	`condition2==False`
`condition1==True`	A	A
`condition1==False`	B	C

Istruzioni `if` – `elif` – `else`

Come possiamo riottenere il comportamento descritto dalla tabella, senza usare `elif` / `else`, ma usando gli operatori `and`/`or`/`not`?

	`condition2==True`	`condition2==False`
`condition1==True`	A	A
`condition1==False`	B	C

```
if condition1:
    print("A")
if not condition1 and condition2:
    print("B")
if not condition1 and not condition2:
    print("C")
```

Istruzioni `if` – `elif` – `else`

Come possiamo riottenere il comportamento descritto dalla tabella, senza usare `elif` / `else`, ma usando gli operatori `and`/`or`/`not`?

	`condition2==True`	`condition2==False`
`condition1==True`	A	B
`condition2==False`	B	C

Istruzioni `if` – `elif` – `else`

Come possiamo riottenere il comportamento descritto dalla tabella, senza usare `elif` / `else`, ma usando gli operatori `and`/`or`/`not`?

	`condition2==True`	`condition2==False`
`condition1==True`	A	B
`condition2==False`	B	C

```
if condition1 and condition2:
    print("A")
if (  (condition1 and not condition2) or
      (not condition1 and condition2) ):
    print("B")
if not condition1 and not condition2:
    print("C")
```

Table of contents

Ciclo for

Ciclo for

Il ciclo `for` è un iteratore generico di Python: può iterare attraverso gli item di qualunque sequenza ordinata o altri oggetti iterabili.

- Funziona su stringhe, liste, tuple, dizionari e altri oggetti iterabili definiti nelle librerie
- È possibile definire i propri tipi iterabili

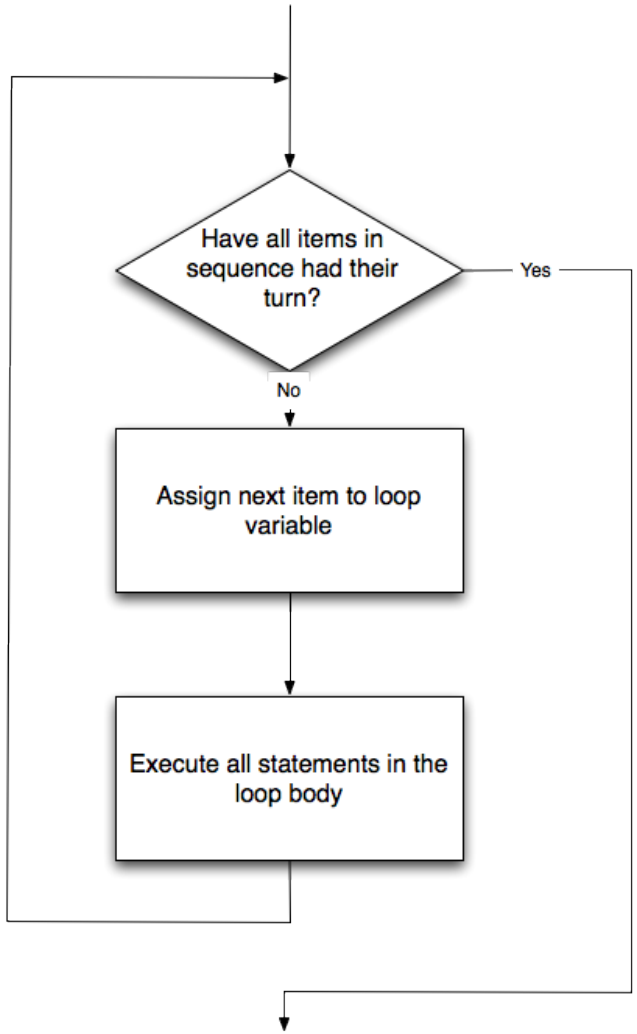
```
L = ["Venezia", "Verona", "Padova", "Vicenza",
     "Treviso", "Belluno", "Rovigo"]
for provincia in L:
    print(provincia, "si trova in Veneto")
```

Ciclo for

```
for var in sequence:
    statement1
    statement2
    ...
```

- `var` è la variabile di ciclo; ad ogni iterazione, viene assegnata ad uno dei valori della lista, fino a quando non vengono tutti usati
- `sequence` è la sequenza di ciclo;
- le istruzioni da seguire sono chiamate il corpo del ciclo

Codelens



Iterazione

`str`	`for` itera sui caratteri
`list`	`for` itera sugli elementi
`tuple`	`for` itera sugli elementi
`dict`	`for` itera sulle chiavi

```
for k in [1,2,3,4]:
    print(k, end=' ')
print("")
for k in ('a','b','c'):
    print(k, end=' ')
print("")
for k in "IBM":
    print(k, end='.')
print("")
```

Iterazione

`str`	`for` itera sui caratteri
`list`	`for` itera sugli elementi
`tuple`	`for` itera sugli elementi
`dict`	`for` itera sulle chiavi

```
for k in [1,2,3,4]:
    print(k, end=' ')
print("")
for k in ('a','b','c'):
    print(k, end=' ')
print("")
for k in "IBM":
    print(k, end='.')
print("")
```

```
1 2 3 4
a b c
I.B.M.
```

Range

Range

La funzione built-in `range()` restituisce un oggetto iterabile che può essere utilizzato per scorrere un insieme di interi.

`range(stop)`

Restituisce una sequenza di interi da `0` a `stop-1`

```
for k in range(4):
    print(k, end = ' ')
print("")
```

Range

Range

La funzione built-in `range()` restituisce un oggetto iterabile che può essere utilizzato per scorrere un insieme di interi.

`range(stop)`

Restituisce una sequenza di interi da `0` a `stop-1`

```
for k in range(4):
    print(k, end = ' ')
print("")
```

```
0 1 2 3
```

Range

Range

La funzione built-in `range()` restituisce un oggetto iterabile che può essere utilizzato per scorrere un insieme di interi.

`range(start, stop)`

Restituisce una sequenza di interi da `start` a `stop-1`

```
for k in range(1,5):
    print(k, end = ' ')
print("")
```

Range

Range

La funzione built-in `range()` restituisce un oggetto iterabile che può essere utilizzato per scorrere un insieme di interi.

`range(start, stop)`

Restituisce una sequenza di interi da `start` a `stop-1`

```
for k in range(1,5):
    print(k, end = ' ')
print("")
```

```
1 2 3 4
```

Range

Range

La funzione built-in `range()` restituisce un oggetto iterabile che può essere utilizzato per scorrere un insieme di interi.

`range(start, stop, increment)`

Restituisce una sequenza di interi da `start` a `stop-1`, con incremento `increment`

```
for k in range(2,10,2):
    print(k, end = ' ')
print("")
```

Range

Range

La funzione built-in `range()` restituisce un oggetto iterabile che può essere utilizzato per scorrere un insieme di interi.

`range(start, stop, increment)`

Restituisce una sequenza di interi da `start` a `stop-1`, con incremento `increment`

```
for k in range(2,10,2):
    print(k, end = ' ')
print("")
```

```
2 4 6 8
```

Range - Differenze fra 2.x e 3.x

```
Python 2.7.13 (default, Apr 23 2017, 16:50:35)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.42.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> L = range(10)
>>> print(L)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
Python 3.5.3 |Anaconda 4.4.0 (x86_64)| (default, Mar  6 2017, 12:15:08)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> L = range(10)
>>> print(L)
range(0, 10)
```

Range - Differenze fra 2.x e 3.x

Python 2.x

`range()` restituisce una lista che contiene gli interi desiderati. La lista è contenuta nella memoria, rendendo `range()` inefficiente quando il numero di iterazioni è molto grande

Se volete utilizzare l'approccio 3.x in 2.x, utilizzate `xrange()` al suo posto.

Python 3.x

In Python 3.x, `range()` genera i numeri mano a mano che l'iteratore glieli chiede, senza memorizzarli in una lista.

Se volete creare una lista, utilizzate la funzione built-in `list()`:

```
L = list(range(10))
print(L)
[0,1,2,3,4,5,6,7,8,9]
```

Esempi for – Summa

Sommiamo i numeri contenuti in L

```
L = [1, 25, 6, 27, 57, 12]
total = 0
for number in L:
    total = total + number
print("The sum is", total)
```

Codice alternativo (usando funzioni built-in)

```
L = [1, 25, 6, 27, 57, 12]
print(sum(L))
```

Esempi for – Massimo

Il codice seguente restituisce il valore massimo contenuto in L

```
L = [1, 25, 6, 27, 57, 12]
max_so_far = L[0]
for number in L:
    if number > max_so_far:
        max_so_far = number
print("The maximum is", max_so_far)
```

Codice alternativo (usando funzioni built-in)

```
L = [1, 25, 6, 27, 57, 12]
print(max(L))
```

Esempi for – Fibonacci

Il codice seguente calcola i primi 20 numeri di Fibonacci

$$F[n] = \begin{cases} F[n-1] + F[n-2] & n > 2 \\ 1 & n \leq 2 \end{cases}$$

```
F = [1,1]
for k in range(18):
    F.append(F[-1]+F[-2])
print(F)
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377,
610, 987, 1597, 2584, 4181, 6765]
```

Cicli `for` annidati

Due o più cicli possono essere "annidati", nel senso che uno è contenuto nell'altro
Considerate il codice seguente, che lista tutti i possibili "quarti d'ora"

```
L = []
for h in range(24):
    for m in range(0, 60, 15):
        L.append(str(h)+":"+str(m))
print(L)
```

```
['0:0', '0:15', '0:30', '0:45', '1:0', '1:15', '1:30',
 '1:45','2:0', '2:15', '2:30', '2:45', '3:0', '3:15',
 '3:30', '3:45', ...
```

Cicli `for` annidati

Il codice seguente controlla se esistono oggetti ripetuti nella lista

```
L = [1,3,5,6]
for x in L:
    for y in L:
        if x == y:
            print(x, "is repeated")
```

Vedete qualche problema?

Cicli for annidati

Il codice seguente controlla se esistono oggetti ripetuti nella lista

```
L = [1,3,5,6]
for x in L:
    for y in L:
        if x == y:
            print(x, "is repeated")
```

Vedete qualche problema?

```
1 is repeated
3 is repeated
5 is repeated
6 is repeated
```

Cicli `for` annidati

Il codice seguente controlla se esistono oggetti ripetuti nella lista

```
L = [1,3,5,6,1,8,3]
for i in range(len(L)):
    for j in range(i+1,len(L)):
        if L[i] == L[j]:
            print(L[i], "is repeated at", i, j)
```

Cicli for annidati

Il codice seguente controlla se esistono oggetti ripetuti nella lista

```
L = [1,3,5,6,1,8,3]
for i in range(len(L)):
    for j in range(i+1,len(L)):
        if L[i] == L[j]:
            print(L[i], "is repeated at", i, j)
```

```
1 is repeated at 0 4
3 is repeated at 1 6
```

Cicli `for` annidati

Il codice seguente controlla se esistono oggetti ripetuti nella lista

```
L = [1,3,5,6,1,8,3,4,3]
for i in range(len(L)):
    for j in range(i+1,len(L)):
        if L[i] == L[j]:
            print(L[i], "is repeated at", i, j)
```

Cosa viene stampato?

Cicli for annidati

Il codice seguente controlla se esistono oggetti ripetuti nella lista

```
L = [1,3,5,6,1,8,3,4,3]
for i in range(len(L)):
    for j in range(i+1,len(L)):
        if L[i] == L[j]:
            print(L[i], "is repeated at", i, j)
```

Cosa viene stampato?

```
1 is repeated at 0 4
3 is repeated at 1 6
3 is repeated at 1 8
3 is repeated at 6 8
```

Esercizio

Data una lista `L` di interi, stampare `True` se `L` contiene due valori distinti la cui somma è 18

Esempio: Se `L=[3,7,12,11,8,32,7,5]`, stampa `True` in quanto `7+11=18`.

Esercizio

Data una lista `L` di interi, stampare `True` se `L` contiene due valori distinti la cui somma è 18

Esempio: Se `L=[3,7,12,11,8,32,7,5]`, stampa `True` in quanto `7+11=18`.

```
found = False
for i in range(len(L)):
    for j in range(i+1,len(L)):
        if L[i]+L[j] == 18:
            found = True
print(found)
```

Esercizio

Data una lista `L` di interi, stampare una coppia di indici le cui celle sommano a 18 (se esiste)

Esempio: Se `L=[3,7,12,11,8,32,7,6]`, stampa `1,3` in quanto `L[1]+L[3]=18` e l'algoritmo si ferma alla prima coppia incontrata.

Esercizio

Data una lista `L` di interi, stampare una coppia di indici le cui celle sommano a 18 (se esiste)

Esempio: Se `L=[3,7,12,11,8,32,7,6]`, stampa `1,3` in quanto `L[1]+L[3]=18` e l'algoritmo si ferma alla prima coppia incontrata.

```
found = False
for i in range(len(L)):
    for j in range(i+1,len(L)):
        if L[i]+L[j] == 18 and not found:
            found = True
            pos_i = i
            pos_j = j
if found:
    print(pos_i, pos_j)
```

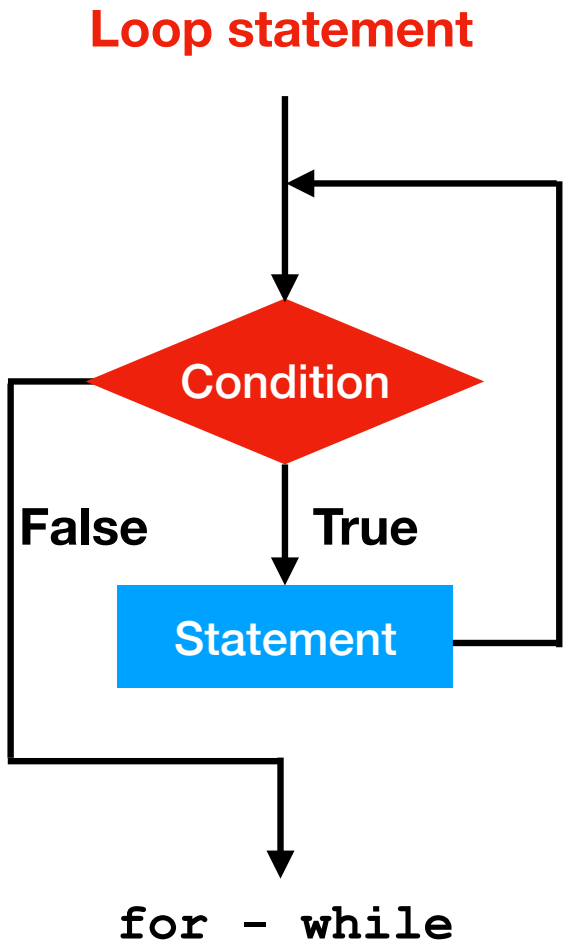
Table of contents

Cicli while

While

L'istruzione `while` permette di scrivere codice che si ripete fintanto che una certa condizione è vera. Si ferma non appena la condizione è falsa.

```
while condition:
    # statements that may change
    # the condition from
    # True to False
```



Cicli `while`

Differenze fra `while` e `for`

- `for` element `in` collection: esegue n volte, dove n è la dimensione della collezione
- `while` condition: esegue un numero indefinito di volte, fintanto che la condizione è vera.

Utilizzo di while

L'istruzione **while** è utile quando il numero di iterazioni non può essere conosciuto a priori, per esempio quando si interagisce con un utente.

```
while input("Do you want me to stop? ").lower() != "yes":
    print("Then I'll keep going!")
```

Da meno pitonico a più pitonico

```
tot = 0
i=0
while i < len(L):
  tot = tot + L[i]
  i = i +1
```

```
tot = 0
for i in range(len(L)):
  tot = tot + L[i]
```

```
tot = 0
for v in L:
  tot = tot + v
```

```
tot = sum(L)
```

Esercizio

Scrivere un programma che pone una domanda e la ripete fino a quando la risposta è corretta.

Esercizio

Scrivere un programma che pone una domanda e la ripete fino a quando la risposta è corretta.

```
answer = ""
while (answer != "rome"):
    print("What is the capital of Italy?")
    answer = input().lower()
    if answer != "rome":
        print("Sorry, wrong answer; retry")
```

Esercizio

Scrivere un programma che pone una domanda e la ripete fino a quando la risposta è corretta oppure fino a quando si sono superate tre risposte sbagliate.

Esercizio

Scrivere un programma che pone una domanda e la ripete fino a quando la risposta è corretta oppure fino a quando si sono superate tre risposte sbagliate.

```
answer = ""
attempts = 0
while (answer != "rome" and attempts < 3):
    print("What is the capital of Italy?")
    answer = input().lower()
    if answer != "rome":
        print("Sorry, wrong answer; retry")
        attempts = attempts+1
if (attempts == 3):
    print("The capital is Rome! Goat!")
else:
    print("Very good")
```

Esercizio

Sia L una lista di interi e sia $L_{sum}(k)$ la somma dei primi k valori di L:

$$L_{sum}(k) = \sum_{i=0}^{k-1} L[i].$$

Scrivere un programma che prende L ed un valore intero *threshold* come input e stampa il numero k di elementitali che $L_{sum}(k) \geq threshold$. Stampa -1 se la somma di tutti gli elementi è più piccola di *threshold*.

Per esempio,

- Se $L = [1, 4, 3, 12, 7]$ e $threshold = 7$, l'output è 3, ovvero la somma dei primi tre elementi (8) è maggiore di *threshold*.
- Se $L = [1, 2, 3, 4]$ e $threshold = 11$, l'output è -1, ovvero la somma di tutti gli elementi (10) è inferiore a *threshold*.

Esercizio

```
L = list(range(10))
threshold = 20
tot = 0
i = 0
while (tot <= threshold and i < len(L)):
    tot = tot + L[i]
    i = i+1
print(i)
if (tot >= threshold):
    print(i-1)
else:
    print(-1)
```

Esercizio

Scrivere un programma che crea una lista contenente tutti i numeri di Fibonacci più piccoli di 1,000,000.

Esercizio

Scrivere un programma che crea una lista contenente tutti i numeri di Fibonacci più piccoli di 1,000,000.

```
F = [1,1]
over = False

while not over:
    next = F[-1]+F[-2]
    if next > 1000000:
        over = True
    else:
        F.append(next)
```

```
F = [1,1]
while F[-1]<1000000:
    F.append(F[-1]+F[-2])
F.pop(-1)
```

Esercizio

Esercizio

La sequenza $3n+1$ è definita come segue: dato un numero n, se n è pari dividetelo per 2; se n è dispari, moltiplicatelo per 3 e aggiungete 1. Fermatevi quando raggiungete il valore 1.

Esempio: per $n = 3$, la sequenza è $[3, 10, 5, 16, 8, 4, 2, 1]$.

Scrivete un programma che crea una lista D, tale che per ogni valore n compreso fra 1 e 50, $D[n]$ contiene la lunghezza della sequenza così generata. Nel caso di $n = 3$, la lunghezza è 8. Nel caso di $n = 27$, la lunghezza è 111.

Utilizzo di while – Esercizio

```
MAX = 50
L = [0]*(MAX+1)
for n in range(1,MAX+1):
    count = 0
    start = n
    while n > 1:
        if (n % 2 == 0):
            n = n // 2
        else:
            n = 3*n+1
        count = count + 1
    L[start] = count
print(L[1:])
```

Table of contents

Break/continue – amore e odio

break

All'interno di un ciclo (sia `for` che `while`), un'istruzione `break` interrompe l'esecuzione del ciclo.

```
sums = []
tot = 0
for x in range(1,100):
    tot = tot+x
    if tot>300:
        break
    sums.append(tot)
print(sums)
```

```
[1, 3, 6, 10, 15, 21, 28,
 36, 45, 55, 66, 78, 91,
 105, 120, 136, 153, 171,
 190, 210, 231, 253, 276,
 300]
```

Break/continue – amore e odio

break

Questa versione stampa lo stesso output; è molto meglio senza `break`!

```
sums = []
tot = 0
x = 1
while (tot+x <= 300):
    tot = tot + x
    sums.append(tot)
    x = x + 1
print(sums)
```

```
[1, 3, 6, 10, 15, 21, 28,
 36, 45, 55, 66, 78, 91,
 105, 120, 136, 153, 171,
 190, 210, 231, 253, 276,
 300]
```

Break/continue – amore e odio

continue

All'interno di un ciclo (sia `for` che `while`), un'istruzione `continue` interrompe l'esecuzione dell'iterazione corrente del ciclo e passa alla successiva.

```
LS = []
for x in range(1,31):
    if x%2 != 0 and x%3 != 0:
        continue
    LS.append(x)
print(LS)
```

Break/continue – amore e odio

continue

All'interno di un ciclo (sia `for` che `while`), un'istruzione `continue` interrompe l'esecuzione dell'iterazione corrente del ciclo e passa alla successiva.

```
LS = []
for x in range(1,31):
    if x%2 != 0 and x%3 != 0:
        continue
    LS.append(x)
print(LS)
```

```
[2, 3, 4, 6, 8, 9, 10, 12,
 14, 15, 16, 18, 20, 21,
 22, 24, 26, 27, 28, 30]
```

Break/continue – amore e odio

continue

Questa versione stampa lo stesso output; è molto meglio senza `break`!

```
LS = []
for x in range(1,31):
    if x%2 == 0 or x%3 == 0:
        LS.append(x)
print(LS)
```

```
[2, 3, 4, 6, 8, 9, 10, 12,
 14, 15, 16, 18, 20, 21,
 22, 24, 26, 27, 28, 30]
```

Comportamento break e continue

```
for number in range(10):
    print(number, end=" ")
print("")

for number in range(10):
    print(number, end=" ")
    break
print("")

for number in range(10):
    print(number, end=" ")
    continue
print("")
```

Comportamento break e continue

```
for number in range(10):
    print(number, end=" ")
print("")
```

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

```
for number in range(10):
    print(number, end=" ")
    break
print("")
```

```
for number in range(10):
    print(number, end=" ")
    continue
print("")
```

Comportamento break e continue

```
for number in range(10):
    print(number, end=" ")
print("")
```

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

```
for number in range(10):
    print(number, end=" ")
    break
print("")
```

```
0
```

```
for number in range(10):
    print(number, end=" ")
    continue
print("")
```

Comportamento break e continue

```
for number in range(10):
    print(number, end=" ")
print("")
```

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

```
for number in range(10):
    print(number, end=" ")
    break
print("")
```

```
0
```

```
for number in range(10):
    print(number, end=" ")
    continue
print("")
```

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

Comportamento break e continue

```
for number in range(1,11):
    print(number, end=" ")
    if number % 4 == 0:
        break
print("")

for number in range(1,11):
    if number % 4 == 0:
        break
    print(number, end=" ")
print("")
```

Comportamento break e continue

```
for number in range(1,11):
    print(number, end=" ")
    if number % 4 == 0:
        break
print("")
```

```
1 2 3 4
```

```
for number in range(1,11):
    if number % 4 == 0:
        break
    print(number, end=" ")
print("")
```

Comportamento break e continue

```
for number in range(1,11):
    print(number, end=" ")
    if number % 4 == 0:
        break
print("")
```

1 2 3 4

```
for number in range(1,11):
    if number % 4 == 0:
        break
    print(number, end=" ")
print("")
```

1 2 3

Evitate `break` e `continue`!

Using break and continue frequently makes code hard to follow. But if replacing them makes the code even harder to follow, then that's a bad change.

E.W. Dijkstra. *Go To Statement Considered Harmful.* Communications of the ACM, Vol. 11 (1968) 147-148

Table of contents

List Comprehension

List comprehension

L'operatore di list comprehension permette di filtrare o trasformare una lista. La lista originale non viene modificata; ne viene creata una nuova.

La sintassi astratta è:

```
filtered = [expression(element)
            for element in original
            if condition(element)]
```

List Comprehension

List comprehension per filtrare

Dato un oggetto iterabile arbitrario `original`, possiamo creare una nuova lista che contiene solo queli elementi di `original` che soddisfano una certa condizione.

La sintassi astratta è:

```
filtered = [element
            for element in original
            if condition(element)]
```

List comprehension - Esempio

Example

Generate la lista dei numeri naturali minori o uguali a 30 che sono divisibile per 2 o 3.

$$L = \{n : 1 \leq n \leq 30 \textbf{ and } (n \bmod 2 = 0 \textbf{ or } n \bmod 3 = 0)\}$$

```
L = [n for n in range(1,31) if n%2 == 0 or n%3 == 0]
```

```
[2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16,
 18, 20, 21, 22, 24, 26, 27, 28, 30]
```

List comprehension - Esercizio

Esercizio

Data una lista di sequenze DNA rappresentate come stringhe, restituite solo quelle sequenze che contengono un'adenosina ("A")

```
sequences = ["ACTGG", "CCTGT", "ATTTA", "TATAGC"]
```

List comprehension - Esercizio

Esercizio

Data una lista di sequenze DNA rappresentate come stringhe, restituite solo quelle sequenze che contengono un'adenosina ("A")

```
sequences = ["ACTGG", "CCTGT", "ATTTA", "TATAGC"]
L = [seq for seq in sequences if "A" in seq]
```

```
['ACTGG', 'ATTTA', 'TATAGC']
```

List comprehension - Esercizio

Esercizio

Data una lista di numeri di telefono rappresentati come stringhe, ritornate solo quelli di Verona (prefisso "045")

```
numbers = ["04599904523", "0461304534",
           "0288662244", "0458346157"]
```

List comprehension - Esercizio

Esercizio

Data una lista di numeri di telefono rappresentati come stringhe, ritornate solo quelli di Verona (prefisso "045")

```
numbers = ["04599904523", "0461304534",
           "0288662244", "0458346157"]
L = [number for number in numbers if number.startswith("045")]
```

```
['04599904523', '0458346157']
```

Note

Il nome della variabile "temporanea" che contiene l'elemento corrente (negli esempi precedenti, `n` e `seq`, respectively) è arbitraria.

Questi pezzi di codice sono identici:

```
L = [n for n in range(1,31) if n%2 == 0 or n%3 == 0]
```

```
L = [pippo for pippo in range(1,31)
           if pippo%2 == 0 or pippo%3 == 0]
```

List Comprehension

List comprehension come trasformazione

Dato un oggetto iterabile arbitrario `original`, è possibile utilizzare una list comprehension per trasformare i valori in qualche modo.

La sintassi astratta è:

```
transformed = [expression(element)
               for element in original]
```

`expression` deve essere basata su `element`, ma a volte è possibile che `element` non sia menzionato.

List comprehension - Esercizi

Esercizio

Generate una lista di quadrati dei numeri naturali fra `1` e `30`.

$$L = \{n^2 : 1 \leq n \leq 30\}$$

List comprehension - Esercizi

Esercizio

Generate una lista di quadrati dei numeri naturali fra 1 e 30.

$$L = \{n^2 : 1 \leq n \leq 30\}$$

```
L = [n*n for n in range(1,31)]
print(L)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196,
 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625,
 676, 729, 784, 841, 900]
```

List comprehension – Esercizi

Esercizio

Data una lista di stringhe che rappresentano una parte della struttura 3D di una catena di proteine, calcolate una lista che contenga, per ogni riga, una lista contenente le tre ultime coordinate.

```
atoms = [
    "SER A 96 77.253 20.522 75.007",
    "VAL A 97 76.066 22.304 71.921",
    "PRO A 98 77.731 23.371 68.681",
    "SER A 99 80.136 26.246 68.973",
    "GLN A 100 79.039 29.534 67.364",
    "LYS A 101 81.787 32.022 68.157",
]
```

List comprehension – Esercizi

Esercizio

Data una lista di stringhe che rappresentano una parte della struttura 3D di una catena di proteine, calcolate una lista che contenga, per ogni riga, una lista contenente le tre ultime coordinate.

```
atoms = [
    "SER A 96 77.253 20.522 75.007",
    "VAL A 97 76.066 22.304 71.921",
    "PRO A 98 77.731 23.371 68.681",
    "SER A 99 80.136 26.246 68.973",
    "GLN A 100 79.039 29.534 67.364",
    "LYS A 101 81.787 32.022 68.157",
]
coords = [row.split()[-3:] for row in atoms]
print(coords)
```

```
[['77.253', '20.522', '75.007'], ['76.066', '22.304', '71.921']
```

List comprehension – esempio

Ma! Il risultato è una lista di liste di stringhe, io volevo le tre coordinate come float! Cosa possiamo fare?

List comprehension – esempio

Ma! Il risultato è una lista di liste di stringhe, io volevo le tre coordinate come float! Cosa possiamo fare?

```
coords = [
    [float(coord) for coord in row.split()[-3:]]
    for row in atoms]
print(coords)
```

```
[[77.253, 20.522, 75.007], [76.066, 22.304, 71.921], ...
```

List comprehension – esempio

Ma! Il risultato è una lista di liste, io volevo una lista di tuple da utilizzare in un dizionario.

List comprehension – esempio

Ma! Il risultato è una lista di liste, io volevo una lista di tuple da utilizzare in un dizionario.

```
coords = [
    tuple([float(coord) for coord in row.split()[-3:]])
    for row in atoms]
print(coords)
```

```
[(77.253, 20.522, 75.007), (76.066, 22.304, 71.921), ...
```

List Comprehension

Tutto insieme, ora!

Data una lista arbitraria `original`, possiamo utilizzare una list comprehension per trasformare e filtrare gli elementi nella lista

La sintassi astratta è:

```
transformed = [expression(element)
               for element in original
               if condition(element)]
```

List Comprehension

Usando gli esempi sviluppati finora, vogliamo solo le coordinate delle serine:

List Comprehension

Usando gli esempi sviluppati finora, vogliamo solo le coordinate delle serine:

```
coords = [
    tuple([float(coord) for coord in row.split()[-3:]])
    for row in atoms
    if row.startswith("SER")]
print(coords)
```

```
[(77.253, 20.522, 75.007), (80.136, 26.246, 68.973)]
```

List comprehension e liste di liste

Diciamo che vogliamo creare una matrice bi-dimensionale $n \times n$, inizializzata a zero. Scriviamo così:

```
n = 4
L = [[0]*n]*n
print(L)
```

```
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Cosa succede se faccio `L[1][1] = 5`?

List comprehension e liste di liste

Diciamo che vogliamo creare una matrice bi-dimensionale $n \times n$, inizializzata a zero. Scriviamo così:

```
n = 4
L = [[0]*n]*n
print(L)
```

```
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Cosa succede se faccio `L[1][1] = 5`?

```
[[0, 5, 0, 0], [0, 5, 0, 0], [0, 5, 0, 0], [0, 5, 0, 0]]
```

Perchè?

List comprehension e liste di liste

Il modo migliore per inizializzare una lista di liste è di usare una list comprehension o un ciclo.

```
n = 4
L = [[0]*n for i in range(n)]
L[1][1] = 5
print(L)
```

```
[[0, 0, 0, 0], [0, 5, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Table of contents

Esercizio

Problema

Scrivete un algoritmo che prenda in input interi positivi e li inserisca in una lista. Il programma termina quando si inserisce 0, al che viene stampata la lista.

Esercizio

Problema

Scrivete un algoritmo che prenda in input interi positivi e li inserisca in una lista. Il programma termina quando si inserisce 0, al che viene stampata la lista.

```
L = []
n = int(input())
while n != 0:
    L.append(n)
    n = input()
print(L)
```

Esercizio

Problema

Scrivere un programma che data una lista di numeri, stampi un istogramma fatto di asterischi. Ad esempio, [2,3] verrà stampato come

```
**
***
```

Esercizio

Problema

Scrivere un programma che data una lista di numeri, stampi un istogramma fatto di asterischi. Ad esempio, [2,3] verrà stampato come

```
**
***
```

```
L = [1,3,2,5]
for n in L:
    print("*" * n)
```

Esercizio più complicato: provate a fare un istogramma verticale, invece che orizzontale

Esercizio

Problema

Scrivere un programma che prenda in input un testo `original` e rimuova tutti i caratteri del testo che sono "punteggiatura"

Esercizio

Problema

Scrivere un programma che prenda in input un testo `original` e rimuova tutti i caratteri del testo che sono "punteggiatura"

```
original="""Quel ramo del lago di Como, che volge a mezzogiorno,"""

punc = ",;.:’"

text = "".join([char for char in original if char not in punc])
```

Esercizio

Problema

Scrivere un programma che prenda in input un testo `text` senza "punteggiatura" e restituisca una lista di tutte le lunghezze delle parole presenti.

Esercizio

Problema

Scrivere un programma che prenda in input un testo `text` senza "punteggiatura" e restituisca una lista di tutte le lunghezze delle parole presenti.

```
wordlist = text.split()
lenlist = [len(word) for word in wordlist]
```

Esercizio

Problema

Scrivere un programma che prenda in input una lista di numeri positivi (rappresentanti lugnhezze di parole) e costruisca un dizionario che associa ad ogni lunghezza la frequenza (numero di occorrenze) di quella lunghezza

Esercizio

Problema

Scrivere un programma che prenda in input una lista di numeri positivi (rappresentanti lugnhezze di parole) e costruisca un dizionario che associa ad ogni lunghezza la frequenza (numero di occorrenze) di quella lunghezza

```
nmax=max(lenlist)
nmin=min(lenlist)
histogram = {}
for k in range(nmin, nmax+1):
    histogram[k] = lenlist.count(k)
print(histogram)
```

Esercizio

Problema

Dato un testo lungo rappresentato come stringa, calcolate la frequenza di ogni parola che appare in esso e memorizzatelo come dizionario che associa le parole alla loro frequenza.

```
text = """Nel pozzo di San Patrizio c'e' una pazza che lava
una pezza. Arriva un pazzo, con un pezzo di pizza e chiede
alla pazza se ne vuole un pezzo. La pazza rifiuta. Allora
il pazzo prende la pazza, la pezza e la pizza e li butta
nel pozzo di San Patrizio, protettore dei pazzi.
"""
```

Suggerimento: potete utilizzare la funzione `get(k)` dei dizionari, che restituisce `None` se una certa chiave non è presente.

Esercizio

Problema

Dato un numero `n`, create e stampate una lista contenente tutti i modi possibili per ottenere `n` come prodotto di due interi. Fate attenzione alle ripetizioni commutative. Ad esempio, $n = 36$ deve stampare `[(1, 36), (2, 18), (3, 12), (4, 9), (6, 6)]`

Esercizio

Problema

Dato un numero `n`, create e stampate una lista contenente tutti i modi possibili per ottenere `n` come prodotto di due interi. Fate attenzione alle ripetizioni commutative. Ad esempio, $n = 36$ deve stampare `[(1, 36), (2, 18), (3, 12), (4, 9), (6, 6)]`

```
n = 36
L = []
for i in range(n+1):
    for j in range(i,n+1):
        if (i*j==n):
            L.append((i,j))
print(L)

# Through list comprehension
L = [(i,j) for i in range(n+1) for j in range(i,n+1) if i*j==n]
```

Esercizio

Problema

Data una lista di valori, generate la sottolista di valori che appaiono una volta sola nella lista. Per esempio, `L = [1,3,2,3,5,4,5,3,3]` genera `[1, 2, 4]`

Esercizio

Problema

Data una lista di valori, generate la sottolista di valori che appaiono una volta sola nella lista. Per esempio, `L = [1,3,2,3,5,4,5,3,3]` genera `[1, 2, 4]`

```
L = [1,3,2,3,5,4,5,3,3]
SL = [n for n in L if L.count(n)==1]
print(SL)
```

Esercizio

Problema

Data una lista di valori, potenzialmente senza valori ripetuti, generate la sottolista ordinata di valori in cui i valori ripetuti sono stati rimossi. Per esempio, `L = [1,3,2,3,5,4,5,3,3]` dovrebbe generare `L = [1,2,3,4,5]`

Esercizio

Problema

Data una lista di valori, potenzialmente senza valori ripetuti, generate la sottolista ordinata di valori in cui i valori ripetuti sono stati rimossi. Per esempio, `L = [1,3,2,3,5,4,5,3,3]` dovrebbe generare `L = [1,2,3,4,5]`

```
L = [1,3,2,3,5,4,5,3,3]
SL = []
for n in L:
    if (n not in SL):
        SL.append(n)
print(SL.sort())
```

Esercizio

Problema

Date due liste senza ripetizioni, generate la sottolista di valori che appaiono in entrambe.

```
L1 = [1,7,9,3,23,11]
L2 = [7,23,2,4,8,16]
```

Esercizio

Problema

Date due liste senza ripetizioni, generate la sottolista di valori che appaiono in entrambe.

```
L1 = [1,7,9,3,23,11]
L2 = [7,23,2,4,8,16]

SL = []
for v in L1:
  if v in L2:
     SL.append(v)
print(SL)
```

Esercizio

Problema

Date due liste di valori ordinati, generate la sottolista di valori che appartengono ad entrambi. Sfruttate il fatto che le liste di input sono ordinate.

Esercizio

Problema

Date due liste di valori ordinati, generate la sottolista di valori che appartengono ad entrambi. Sfruttate il fatto che le liste di input sono ordinate.

```
A = [1,2,4,8,16,32]
B = [4,8,12,16,20]
LS = []
a = b = 0
while (a < len(A) and b < len(B)):
    if A[a] == B[b]:
        LS.append(A[a])
        a = a+1
        b = b+1
    elif A[a] < B[b]:
        a = a+1
    else:
        b = b+1
print(LS)
```

Esercizio

Esercizio - Sudoku

Data una matrice 9×9, stampate `True` se la matrice è una soluzione accettabile per il Sudoku.

```
L = [[1, 2, 3, 4, 5, 6, 7, 8, 9],
  [4, 5, 6, 7, 8, 9, 1, 2, 3],
  [7, 8, 9, 1, 2, 3, 4, 5, 6],
  [2, 3, 4, 5, 6, 7, 8, 9, 1],
  [5, 6, 7, 8, 9, 1, 2, 3, 4],
  [8, 9, 1, 2, 3, 4, 5, 6, 7],
  [3, 4, 5, 6, 7, 8, 9, 1, 2],
  [6, 7, 8, 9, 1, 2, 3, 4, 5],
  [9, 1, 2, 3, 4, 5, 6, 7, 8] ]
```

Esercizio

Data la lista di stringhe:

```
table = [
    "protein domain start end",
    "YNL275W PF00955 236 498",
    "YHR065C SM00490 335 416",
    "YKL053C-A PF05254 5 72",
    "YOR349W PANTHER 353 414",
]
```

- scrivete un programma che utilizzi la prima riga come nomi di colonna
- per ogni riga, crei un dizionario come questo

```
dictionary = {
    "protein": "YNL275W",
    "domain": "PF00955",
    "start": "236",
    "end":, "498"
}
```

- aggiunga ognuno di questi dizionari ad una lista

Esercizio

Dato:

```
translation_of = {"a": "ade", "c": "cyt",
    "g": "gua", "t": "tym"}
```

traducete la lista:

```
L = ["A", "T", "T", "A", "G", "T", "C"]
```

nella stringa:

```
"ade tym tym ade gua tym cyt"
```

Attenzione ai caratteri maiuscoli/minuscoli. Provate a scrivere il codice che fa l'operazione opposta.