6.1) Definendo le liste come mostrato in basso, cerca di prevedere (senza usare il computer) quale sarà la risposta nei casi **6.1 A**,**B**,**C**,**D** (l'ultimo caso è particolarmente interessante).

```
>>> a=[3,6,2,1]
>>> b=["io","tu","egli"]
>>> c=[1,"Roma"]
>>> d=[a,b,c]
```

6.1A `>>> print(a[2])`

6.1B `>>> print(b[1])`

6.1C `>>> print(c[0])`

6.1D `>>> print(d[0])`

Riprendiamo l'ultimo problema e riflettiamo sulla possibile ortografia adottata da $Python$ nel caso di liste annidate. Come scoprirete, la soluzione ortografica è quella suggerita dal buonsenso:

6.2) Utilizzando soltanto la lista `d` quella definita sopra, per la precisione `d=[[3, 6, 2, 1], ['io', 'tu', 'egli'], [1, 'Roma']]`, e senza mai citare `a`, `b` o `c`, né scrivere stringhe,….

6.2A Stampa sullo schermo la scritta `egli`

6.2C Stampa sullo schermo la scritta `Roma`

Impareremo a scrivere liste di vari tipi per risolvere diversi problemi. Iniziamo però a maneggiare con questo oggetto partendo da una lista già preparata:

6.3) I computer del laboratorio di informatica sono stati aggiornati in modo da contenere il file $Esercizio1.py$. Questo permette di svolgere questa serie di esercizi:

Vai in modalità di programmazione e premetti a tutti codici che dovrai scrivere la riga seguente:

```
from Esercizio1 import p
```

Questa istruzione ha l'effetto di importare la lista `p` contenente i primi 100 numeri primi, necessaria allo svolgimento dei problemi elencati in basso (in seguito* si trova anche il codice scritto per generare `p`)

6.3A Scrivi un programma che stampi tutti i primi 100 numeri primi.

6.3B Scrivi un programma che chieda all'utente quale numero primo si vuole stampare (il primo, il secondo, …non oltre il centesimo) e poi lo faccia.

6.3C Scrivi un programma che stampi i primi 100 numeri primi in ordine decrescente (dal maggiore al minore)

6.3D Scrivi un programma che stampi tutti i primi 100 numeri primi di indice pari (quindi il secondo, il quarto, e così via fino al centesimo)

6.3E Scrivi un programma che, dei primi 100 numeri primi, stampi soltanto i numeri primi di indice primo, quindi il secondo, il terzo, il quinto e così via fino al novantasettesimo. A questo proposito è utile sapere che 97 è il ventiquattresimo numero primo.

*A fianco è mostrato il codice che genera la lista `p`. Esso è basato sul crivello di Eratostene nella sua versione più semplice (senza migliorie o scorciatoie) e permette di trovare velocemente i numeri primi fino a n. Il meccanismo non può prevedere quanti numeri primi troverà, l'assegnazione iniziale $n = 541$ è stata fissata "ad arte" in modo da trovare proprio 100 numeri primi.

```
n=541
a=[1]*n
p=[]
for i in range(2,n+1):
    if a[i-2]==1:
        for j in range(2*i,n+1,i):
            a[j-2]=0
for i in range(2,n+1):
    if a[i-2]==1:
        p=p+[i]
```

6.4) Cerca di capire cosa stamperà a schermo il seguente programma:

```
a="parola" #definisco a
b=[1,2]    #definisco b
c=7        #definisco c
d=[a,b,c]  #definisco d

try:
    print(len(a))   #tenta di stampare a schermo la lunghezza di a
except:
    print("Non posso stampare a") #se non riesce...

try:
    print(len(b))   #tenta di stampare a schermo la lunghezza di b
except:
    print("Non posso stampare b") #se non riesce...

try:
    print(len(c))   #tenta di stampare a schermo la lunghezza di c
except:
    print("Non posso stampare c") #se non riesce...

try:
    print(len(d))   #tenta di stampare a schermo la lunghezza di d
except:
    print("Non posso stampare d") #se non riesce...
```

6.5) Considera il seguente codice (privo di commenti). A e B sono due stringhe che corrispondono alle possibili "risposte" del programma:

```
from Esercizio1 import p
```

```
A="…"
B="…"

a=int(input("Scrivi un numero intero tra 1 e 540))
if a>0 and a<541:
    if a in p: print(A)
    else: print(B)
```

Che frasi devono corrispondere ad A e B?

I seguenti programmi si basano sulla lista p. Metti quindi in testa ai programmi la seguente istruzione:

```
from Esercizio1 import p
```

6.6A Scrivi un algoritmo che verifichi la congettura di *Goldbach* per numeri "piccoli", scrivi cioè un programma che richieda in ingresso un numero pari n compreso fra 4 e 540 (cerca di gestire anche eventuali input errati) e che trovi due numeri **primi** tali che la loro somma sia proprio n.
Esempio. Se $n = 46$ il programma potrebbe stampare $3 + 43$.

6.6B Migliora il programma precedente il modo che vengano stampate **tutte** le coppie diverse formati da due numeri primi tali che la somma sia 46.
Esempio. Se $n = 46$ il programma dovrebbe stampare $3 + 43$, $5 + 41$, $17 + 29$, $23 + 23$

6.7 Nel programma $Esercizio2.py$ (che troverete sui vostri PC) è definita la lista numerica (composta soltanto da numeri interi postivi) denominata con Z. Importala e poi scrivi dei programmi per rispondere alle seguenti domande:

a) Quanti elementi ha Z?

b) Quale è la media aritmetica degli elementi di Z?

c) Quanti numeri compresi tra 10 e 20 (10 e 20 inclusi) vi sono in Z?

d) Quanti elementi di Z sono divisibili per 3?

e) Quale è il più grande numero di Z?

f) Quale è il secondo più grande numero contenuto in Z?

g) Quale numero compare più spesso in Z? Quante volte occorre?

6.8) Per svolgere i seguenti esercizi passa in modalità di programmazione.

a) Crea la lista "Q" che contenga i primi 100 quadrati perfetti positivi (da 1 a 10.000)

b) Crea la lista "D" che contenga i primi 100 quadrati numeri dispari.

c) Crea la lista che contenga i primi 20 fattoriali "F" (sarà necessario importare la funzione fattoriale scritta tempo fa).

6.9) Considera la seguente funzione denominata $elabora$ presente in $Esercizio2.py$.

```
def elabora(cosa): #la funzione prende un argomento, chiamato "cosa"

    controllo=1    #controllo è variabile di controllo posta a 1 per default
    valore=cosa    #se possibile, in "valore" verrà registrata la conversione …
                   #.. numerica  di "cosa"

    try:           #il blocco try tenta di convertire "cosa" in un numero...
        valore=float(cosa)

    except:
        controllo=0    #se fallisce, la variabile di controllo va a 0

    return([valore,controllo])
```

Che valore restituisce la funzione $elabora$ nei seguenti casi (in modalità $shell$)?

a) `>>> elabora(-1.1)`

b) `>>> elabora("-1.1")`

c) `>>> elabora("A")`

d) `>>> elabora(9)`

e) `>>> elabora([1,2])`

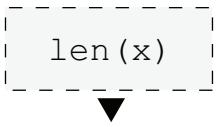
f) `>>> elabora("2")`

6.10) Utilizzando la funzione elabora per gestire le eccezioni, scrivi un programma che chieda all'utente di inserire dei numeri, crei una lista che contenga questi numeri e ne calcoli infine alcuni dati caratteristici (la media aritmetica, la moda, la mediana…) in modo che l'input sia gestito così:

1) L'utente può inserire una quantità di numeri a piacere
2) Se l'utente immette un stringa non numerica, il computer prosegue senza generare errori: rimane quindi possibile inserire altri valori
3) L'immissione dei numeri si conclude quando l'utente preme invio senza aver scritto niente

RIASSUNTO DELLE ISTRUZIONI VISTE

descrizione della funzione **len()**



se x è una lista, len(x) restituisce il numero di elementi di x
se x è una stringa, len(x) resituisce il numero di caratteri di x
in generale, se x è una *sequenza di Python*, len(x) resituisce la sua lunghezza

Esempi (in modalità *shell*)

```
>>> a=[1,2,3]
>>> len(a)
3
```

```
>>> b="ciao!"
>>> len(b)
5
```

```
>>> c=[1,"A,",[0,2]]
>>> len(c)
3
```

descrizione della condizione **in**

Fornisce `True` se l'elemento A fa parte di B e `False` altrimenti (B è una lista, una stringa o una struttura dati composta). Trattandosi di una condizione, `in` può essere usato nelle clausole *if*, *while*, e così via.

Esempi (in modalità *shell*)

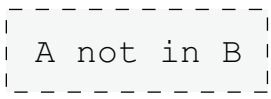
```
>>> a=[1,2,"C"]
>>> b="Ciao"
>>> 1 in a
True
>>> 2 in a
True
>>> "C" in a
True
>>> [1,2] in a
False
>>> "C" in b
True
>>> "Ci" in b
True
```

I casi a fianco mostrano che la condizione "*in*" viene interpretata in modo leggermente diverso nel caso di liste e di stringhe.
L'esempio seguente mette tale differenza in risalto:

```
>>> a=[1]
>>> b="x"

>>> a in a
False
>>> b in b
True
```

La condizione `in` mutua il concetto insiemistico di appartenza (di un elemento ad un insieme) alle strutture dati. Analogamente fa la condizione `not in`, che traduce la relazione di "non appartenenza".



descrizione del metodo **append()**

le liste

lista.**append**(*elemento*)

Se *lista* è il nome di una lista e *elemento* è un oggetto (un numero, una stringa, una lista,...), *lista*.**append**(*elemento*) aggiunge in coda alla lista *lista* l'elemento *elemento*.
Attenzione: la stessa invocazione del metodo *append* modifica la lista *lista*, sarebbe sbagliato scrivere `lista=lista.append(elemento)`.

Esempio 1

```
>>> a=[1,2,3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]
```

Esempio 2

```
>>> a=[1,2,3]
>>> b=[4,5,6]
>>> a.append(b)
>>> a
[1, 2, 3, [4, 5, 6]]
```

descrizione del metodo **extend()**

le liste

lista.**extend**(*aggiunta*)

Se *lista* e *aggiunta* sono i nomi di due liste, *lista*.**extend**(*aggiunta*) attacca a *lista* la lista *aggiunta*.
Attenzione: la stessa invocazione del metodo *extend* modifica la lista *lista*, sarebbe sbagliato scrivere `lista=lista.extend(aggiunta)`.

Esempio 1

```
>>> a=[1,2,3]
>>> b=[4,5,6]
>>> a.extend(b)
>>> a
[1, 2, 3, 4, 5, 6]
```

Esempio 2

```
>>> a=[1,2,3]
>>> b=[4,5,6]
>>> b.append(a)
>>> b
[4, 5, 6, 1, 2, 3]
```