LABORATORIO DI PYTHON

ESERCIZI VARI, PRATICHE DI DEBUG E DI TRACING

29 Marzo 2019

Debug

ESERCIZIO A – COSA NON VA IN QUESTO CODICE?

Scrivere una funzione che prende come parametro una tupla `t` e restituisce `True` se tutti i valori sono in ordine **strettamente crescente**, `False` altrimenti.

```
1  def crescente(t):
2      for i in range(len(t)):
3          if t[i+1]>t[i]:
4              return True
5      return False
6
7  print(crescente((1,2,3,4,5,6))) # Atteso: True
8  print(crescente((1,2,2,4))) # Atteso: False
9  print(crescente((7,1,4,5,3))) # Atteso: False
10 print(crescente((1,))) # Atteso: True
11 print(crescente(())) # Atteso: True
```

ESERCIZIO A – SOLUZIONE

```
1  def crescente(t):
2      # risolvo index out of range con len(t)-1
3      for i in range(len(t)-1):
4          # per una maggiore efficièNza
5          # cerco almeno uno minore
6          if t[i+1]<=t[i]:
7              return False
8      # nessuno è minore
9      return True
```

ESERCIZIO B

Scrivere una funzione che restituisce una tupla contenente tutti i divisori propri (`n` escluso) di un numero naturale `n` preso come parametro.

N.B. Un numero è perfetto se il numero è uguale alla somma dei divisori propri.

Scrivere una funzione che prende come parametro un numero naturale `n` e restituisce `True` se il numero è un numero perfetto, `False` altrimenti. Usare la funzione creata precedente.

Esempi di numeri perfetti sono il 6, 28, 496, 8128, 33550336.

ESERCIZIO B – COSA NON VA IN QUESTO CODICE?

```
1  def divisori_propri(n):
2      divisori = ()
3      for i in range(n):
4          if n%i = 0:
5              divisori += i
6
7  def numero_perfetto(n):
8      divisori = divisori_propri(n)
9      somma_divisori = 0
10     for divisore in divisori:
11         somma_divisori += divisore
12         if somma_divisori == n:
13             return True
14         else:
15             return False
```

ESERCIZIO B – SOLUZIONE I

```
1  def divisori_propri(n):
2      divisori = ()
3      for i in range(n):
4          ## = non è un operatore di confronto valido
5          if n%i == 0:
6              divisori += i
7
8  def numero_perfetto(n):
9      divisori = divisori_propri(n)
10     somma_divisori = 0
11     for divisore in divisori:
12         somma_divisori += divisore
13     # attenzione! il ciclo deve finire per poter confrontare n
14     if somma_divisori == n:
15         return True
16     else:
```

ESERCIZIO B – SOLUZIONE II

```
17          return False
18
19  print(divisoripropri(10)) # Atteso: (1,2,5)
20  print(divisoripropri(1)) # Atteso: ()
21  print(divisoripropri(3)) # Atteso (1,)
22  print(divisoripropri(6)) # Atteso: (1,2,3)
23  print(divisoripropri(28)) # Atteso: (1,2,4,7,14)
24  #False
25  print(numperfetto(1))
26  print(numperfetto(3))
27  print(numperfetto(10**4))
28  print()
29  #True
30  print(numperfetto(6))
31  print(numperfetto(28))
32  print(numperfetto(496))
33  print(numperfetto(8128))
```

ESERCIZIO C – COSA NON VA IN QUESTO CODICE?

Un plateau è una sottosequenza di almeno due elementi contigui con lo stesso valore. Scrivere una funzione `plateau(t)` che, data una tupla `t`, restituisce la tupla degli elementi distinti di `t` che sono valori di un plateau.

Esempio `plateau((3,3,0,2,2,2,0,3,3,4,4))` restituisce `(3,2,4)`.

```
1  def plateau(t):
2      for i in range(len(t-1)):
3          if t[i] == t[i+1] and risultato[-1] != t[i]:
4              risultato += t[i]
5          return risultato
```

ESERCIZIO C – SOLUZIONE

```
1  def plateau(t):
2      # inizializzo una tupla vuota
3      risultato = ()
4      # il range del ciclo for termina con len(t) - 1
5      # in questo modo posso accedere a i + 1 nel corpo del for
6      for i in range(len(t)-1):
7          # risultato non deve contenere già l'elemento i-esimo
8          if t[i] == t[i+1] and t[i] not in risultato
9              # non è possibile sommare tuple ed interi
10             risultato += (t[i],)
11     return risultato # fuori dal cilo for
12
13 print(plat((3,3,0,2,2,2,0,3,3,4,4))) #atteso (3,2,4)
14 print(plat(())) #atteso ()
15 print(plat((1,3,5,2))) #atteso ()
16 print(plat((1,1,1,1,1,1))) #atteso (1,)
```

ALTRI ESERCIZI, EVENTUALMENTE DA FINIRE A CASA.

Scrivere una funzione `confronta(T1, T2)` che prese due stringhe `T1` e `T2` restituisce una tupla contenente gli `i` tali che `T1[i]` è uguale ad uno dei caratteri `T2[i-1]`, `T2[i]`, `T2[i+1]`.

Esempio.

```
1 print(confronta('asca', 'lasca')) #(0,1,2,3)
2 print(confronta('la mamma in', 'la nonna ti')) #(0,1,2,7,8,9)
3 print(confronta('acca', 'zonzo')) #()
```

SOLUZIONE

```
1  def confronta (T1,T2):
2      R = ()
3      if T1[0] in T2[:2]:
4          R += (0,)
5      for i in range(1, len(T1)):
6          if T1[i] in T2[i-1:i+2]:
7              R += (i,)
8      return R
9
```

ALTRI ESERCIZI, EVENTUALMENTE DA FINIRE A CASA.

Giulio Cesare era solito criptare i suoi messaggi sostituendo a ogni lettera quella corrispondente dell'alfabeto "spostato in avanti".

Scrivere due funzioni:

- Una funzione che ritorna l'indice del carattere `c` nella stringa `s`, o l'indice della prima volta che compare `c`, se compare più volte, o None se `c` non è presente
- Una funzione che "`modifica`" la stringa `s` presa come parametro sostituendo a ogni lettera la lettera che si trova 13 posizioni piu' avanti nell'alfabeto. Esempio `'b'` diventa `'o'`, `'m'` diventa `'z'`, `'n'` diventa `'a'`. Supponiamo di lavorare solo con alfabeto minuscolo, convertendo eventualmente `s` in tuttominuscolo. Tutti gli altri caratteri (cifre, punteggiatura, spazi) non vengono modificati.

N.B. Esiste la costante `string.ascii_lowercase` nel modulo string che contiene tutti e soli i caratteri dell'alfabeto (senza numeri e punteggiatura).

SOLUZIONE I

```
1  # uso del modulo string
2  import string
3
4  def posizione(c,s):
5      if c not in s:
6          return None
7      for i in range(len(s)):
8          if c == s[i]:
9              return i
10
11 def cesare(s):
12     s = s.lower() #il metodo si chiama lower()
13     #e la costante string.ascii_lowercase
14     cifrata = ''
15     for c in s:
```

SOLUZIONE II

```
16          if c not in string.ascii_lowercase: #se NON e' un
        alfabetico, mantengo invariato
17              cifrata += c
18          else: #c e' un carattere, va sostituito con il suo
        cifrato
19              p = posizione(c, string.ascii_lowercase)
20              #le lettere dell'alfabeto inglese (e len(
        ascii_lowercase)) sono 26
21              pc = (p + 13) % 26 #l'operatore % e' prioritario
        rispetto a +, quindi qui servono parentesi per fare la
        somma prima del resto
22              cc = string.ascii_lowercase[pc] #pc e' l'indice di c
        , cifrato, nell'alfabeto
23              cifrata += cc #devo concatenare il cifrato
24      return cifrata
25
26  print(cesare(cesare("giulio"))) #giulio
```

SOLUZIONE III

```
27 print(cesare("Anche tu, Bruto, figlio mio!")) #napur gh, oehgb,
       svtyvb zvb!
28 print(cesare("napur gh, oehgb, svtyvb zvb!")) #anche tu, bruto,
       figlio mio!
```

ALTRI ESERCIZI, EVENTUALMENTE DA FINIRE A CASA.

Date due tuple di naturali tutti distinti A e B, diciamo che costituiscono un involucro se una delle due compare come sottosequenza contigua dentro l'altra, con almeno un elemento a sinistra e almeno un elemento a destra che non le appartengono.

Esempio. (0,1,2,3,11,16) e (1,2,3) costituiscono un involucro; (1,2,3) e (1,2,3,11) non lo sono.

Scrivere una funzione `involucro(A,B)` che, presi come parametri due tuple di naturali restituisce `True` se e solo se esse costituiscono un involucro.

Test. Usare il seguente codice per testare la funzione con un input.

```
1  A = eval(input("Inserisci una tupla: "))
2  B = eval(input("Inserisci una tupla: "))
3  print(A,B) #verifico se l'input e' corretto
4  print(involucro(A,B))
```

ES. 4 PER CASA - CORREZIONE I

```
1  def involucro(A,B):
2      if len(A)==0 or len(B)==0:
3          if (len(A)==0 and len(B)>=2) or (len(A)>=2 and len(B)
       ==0):
4              return True
5          return False
6
7      if len(A)>len(B):
8          return inv_aux(B,A)
9      else:
10         return inv_aux(A,B)
11
12 def inv_aux(C,L): # L e' la tupla lunga e C quella corta; non
       sono vuote
13     i = 1 #secondo elemento e' 1
14
```

ES. 4 PER CASA - CORREZIONE II

```
15      # i < per lasciare un elemento
16      # Devo continuare se non ho ancora trovato C[0], quindi !=
17      while i < len(L)-len(C) and L[i] != C[0]:
18          i += 1
19      # se C[0] non e' in L
20      if i == len(L)-len(C):
21          return False
22
23      j = 1
24      while j < len(C) and L[i+j] == C[j]:
25          j += 1
26
27      if j == len(C): #e' j che scorre C!
28          return True
29      else:
30          return False
```