Settimana 4

Elementi di Informatica e Programmazione

Esercizio 1

Progettare il programma **printSelectedMonth.py** che chieda all'utente un numero intero compreso tra 1 e 12 e visualizzi il nome del mese corrispondente, come in questo esempio di funzionamento:

> Inserisci il numero del mese (1-12): 5

«Maggio»

NON SI PUò USARE L'ENUNCIATO if !!

MOLTO IMPORTANTE: (come sempre) individuare un algoritmo che risolva il problema e verificarlo con cura PRIMA di iniziare a scrivere il programma.

Esercizio 2 (slide 1/2)

Scrivere il programma **isPalindrome.py** che verifichi se una stringa, fornita dall'utente, è un palindromo oppure no.

Si ricorda che una stringa è un palindromo se è composta da una sequenza di caratteri (anche non alfabetici) che possa essere letta allo stesso identico modo anche al contrario (es. "radar", "anna", "inni", "xyz%u%zyx").

Il problema può essere risolto in modo "banale" generando la stringa inversa e poi confrontando le due stringhe: la stringa originaria è un palindromo se e solo se le due stringhe sono uguali (e, in tal caso, sono ovviamente entrambe un palindromo).

Tale soluzione richiede, però, molta memoria aggiuntiva, impegnata per la stringa inversa (si pensi a una stringa contenente un milione di caratteri...), anche quando sarebbe sufficiente confrontare pochi caratteri per decidere che la stringa NON è un palindromo (ad esempio, se il suo primo carattere è diverso dal suo ultimo carattere).

Esercizio 2 (slide 2/2)

Cercare, quindi, di individuare un algoritmo che effettui la verifica richiesta SENZA costruire la stringa inversa.

Verificare il corretto funzionamento del programma con:

- una stringa di lunghezza pari che sia un palindromo
- una stringa di lunghezza dispari che sia un palindromo
- una stringa di lunghezza pari che non sia un palindromo
- una stringa di lunghezza dispari che non sia un palindromo
- una stringa di lunghezza unitaria (che è ovviamente un palindromo)
- una stringa di lunghezza zero (che è ragionevole ritenere sia un palindromo, dato che niente la rende "non un palindromo"...)

Esercizio 3

Progettare il programma **printPrimes.py** che calcoli e visualizzi in ordine crescente, un numero per riga, tutti i numeri primi fino a un valore massimo introdotto inizialmente dall'utente; chiedere ripetutamente tale valore all'utente finché non viene introdotto un numero intero maggiore di uno.

Ricordare che un numero intero maggiore di uno è primo se è divisibile con resto nullo soltanto per il numero uno e per se stesso.

MOLTO IMPORTANTE: (come sempre) individuare un algoritmo che risolva il problema e verificarlo con cura PRIMA di iniziare a scrivere il programma.

Esercizio 4

Scrivere il programma **isSubstring.py** che

- chieda all'utente di introdurre due stringhe (una per riga), **s1** e **s2**; ciascuna stringa è costituita da tutti i caratteri presenti sulla riga, compresi eventuali spazi iniziali, finali e/o intermedi
- verifichi (visualizzando al termine un messaggio opportuno) se la seconda stringa, **s2**, è una sottostringa di **s1**, cioè se esiste una coppia di numeri interi non negativi, **x** e **y**, per cui **s1[x:y]** restituisce una stringa uguale a **s2**

Il programma non deve utilizzare metodi che operano su stringhe.

Verificare che il programma gestisca correttamente la situazione in cui **s2** è la stringa vuota, che deve risultare sottostringa di qualsiasi stringa.

Come è invece ragionevole che si comporti il programma se **s1** è la stringa vuota?

Esercizio 5

ATTENZIONE! ***Risolvere questo esercizio soltanto DOPO aver risolto l'esercizio precedente***.

Scrivere il programma **isSubstring2.py** che svolga esattamente la stessa elaborazione del programma **isSubstring.py** descritto nell'esercizio precedente, utilizzando, però, l'operatore di estrazione di sottostringhe SENZA il carattere "due punti", cioè estraendo soltanto sottostringhe di lunghezza unitaria (rimane, ovviamente, possibile invocare la funzione **len**).

Il programma non deve utilizzare metodi che operano su stringhe.

Esercizio 6

ATTENZIONE! ***E' consigliato risolvere gli esercizi isSubstring.py e isSubstring2.py prima di risolvere questo esercizio***.

Una stringa **s2** è una **sottosequenza** di un'altra stringa **s1** se e solo se tutti i caratteri di **s2** sono presenti in **s1** nello stesso ordine (anche se in posizioni diverse e non consecutive). Ad esempio, **gatto** è una sottosequenza di **g**r**atto** e **xyz** è una sottosequenza di 2**x**ppp**y**q**z** ma non di yxz. Se, in base alla definizione data nell'esercizio isSubstring.py, **s2** è una sottostringa di **s1**, è anche (ovviamente...) una sua sottosequenza; viceversa, essere una sottosequenza non implica essere una sottostringa.

Scrivere il programma **isSubsequence.py** che

- chieda all'utente di introdurre due stringhe (una per riga), **s1** e **s2**; ciascuna stringa è costituita da tutti i caratteri presenti sulla riga, compresi eventuali spazi iniziali, finali e/o intermedi
- verifichi (visualizzando al termine un messaggio opportuno) se la seconda stringa, **s2**, è una sottosequenza di **s1**.

Il programma non deve utilizzare metodi che operano su stringhe.

Esercizio 7

Scrivere il programma **tableOfPowers.py** che visualizzi una tavola pitagorica dopo aver chiesto all'utente i valori massimi della base e dell'esponente, che devono essere numeri interi positivi.

Ogni riga della tabella deve contenere le potenze consecutive di una stessa base, con base crescente dall'alto in basso (come in una normale tavola pitagorica).

Porre particolare cura nell'impaginazione della tabella, che deve rispettare queste regole:

In ciascuna riga, due valori consecutivi devono essere separati da almeno uno spazio.

In ciascuna colonna, tutti i valori devono essere "allineati a destra", cioè la cifra che rappresenta le unità di un valore deve trovarsi nella stessa colonna della cifra che rappresenta le unità degli altri valori.

La lunghezza delle righe deve essere quella minima che risulta compatibile con le regole precedenti.

Esercizio 8 (slide 1/2)

Scrivere il programma **twoComplementAdd.py** che

- legga dall'input standard due numeri interi relativi rappresentati in complemento a due e aventi la stessa lunghezza; i numeri devono essere acquisiti sotto forma di stringhe (uno per riga) e il programma deve terminare con una segnalazione d'errore se una delle due stringhe NON è composta da sole cifre binarie e se le due stringhe NON hanno la stessa lunghezza
- calcoli e visualizzi sull'output standard il risultato dell'addizione dei due numeri, sempre rappresentato in complemento a due con lo stesso numero di bit
- nel caso in cui l'addizione abbia dato luogo a overflow, visualizzi nella riga seguente il messaggio **Overflow**

come illustrato nei seguenti esempi di esecuzione del programma

```
Primo numero in complemento a due: 010
Secondo numero (con la stessa lunghezza): 101
010 + 101 = 111

Primo numero in complemento a due: 101
Secondo numero (con la stessa lunghezza): 101
101 + 101 = 010
Overflow

Primo numero in complemento a due: 01000
Secondo numero (con la stessa lunghezza): 101
I due numeri non hanno la stessa lunghezza
```

Esercizio 8 (slide 2/2)

Si noti come, per effetto della rappresentazione in complemento a due, non sia mai necessario esaminare esplicitamente il segno dei due numeri e/o del risultato.

Collaudare il programma eseguendo l'addizione tra 001100110011 e 110011001100 (dopo aver determinato a mano il risultato corretto); ripetere il collaudo eseguendo l'addizione tra 011011011011 e 011011011011. Fare anche qualche collaudo in casi limite... numeri con una sola cifra, uno dei due numeri è lo zero (o entrambi lo sono), uno dei due numeri è il massimo numero positivo con quella determinata lunghezza, uno dei due numeri è il numero negativo con il massimo valore assoluto con quella determinata lunghezza...

IN OGNI CASO, determinare il risultato a mano PRIMA di eseguire il programma.

Esercizio 9

Scrivere il programma **stripDuplicates.py** che legga un'unica riga di testo e, poi, la visualizzi priva di eventuali caratteri **consecutivi** duplicati

Esempio:

legge **Pippo** e scrive **Pipo**

legge **pseudo---casuale** e scrive **pseudo-casuale**

legge **mammmma** e scrive **mama**