Laboratorio 7

Elementi di Informatica e Programmazione

Esercizio 1: Palindromi ricorsivi

Scrivere il programma `recursiveIsPalindrome.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 programma non deve utilizzare costrutti iterativi (cioè non deve avere nessun tipo di ciclo). 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"…)

Soluzione

```
def recursive_is_palindrome(s):
    if len(s) == 0 or len(s) == 1:
        # Caso base: una stringa di lunghezza 0 o 1
        # è palindroma per definizione
        return True
    if s[0] != s[-1]:
        # Se il primo e l'ultimo carattere sono diversi
        # allora la stringa non è un palindromo
        return False
    else:
```

```
        # Altrimenti controllo che la stringa ottenuta
        # rimuovendo il primo e l'ultimo carattere sia un palindromo
        return recursive_is_palindrome(s[1:-1])

# Stringhe di test che sono palindrome
tests_positive = [
    "",
    "a",
    "anna",
    "radar",
]

for s in tests_positive:
    assert recursive_is_palindrome(s)

# Stringhe di test che non sono palindrome
tests_negative = [
    "ab",
    "aac"
]

for s in tests_negative:
    assert not recursive_is_palindrome(s)
```

Esercizio 2: conteggio della frequenza delle lettere

Scrivere il programma letterfreq.py che calcoli la frequenza delle singole lettere dell'alfabeto all'interno di un file di testo (un'informazione utile nell'ambito della crittografia e della compressione):

- chiede all'utente il nome del file da elaborare
- se l'utente scrive la stringa vuota, si usi il testo American Fairy Tales by L. Frank Baum, reperibile all'indirizzo http://www.gutenberg.org/cache/epub/4357/pg4357.txt, che potete salvare in un file locale dal nome che preferite
- altrimenti, il file si trova nel file system locale (e la stringa può essere il solo nome, oppure un percorso relativo o assoluto)
- nel testo da elaborare, dopo averlo convertito in maiuscolo, conta le occorrenze delle singole lettere (ignorando tutti i caratteri che non sono lettere)
- visualizza una tabella di due colonne: nella prima colonna vanno riportate le lettere dell'alfabeto inglese, in ordine lessicografico (cioè, in questo caso, alfabetico), nella sec-

onda colonna la percentuale di lettere del testo che sono uguali a quella indicata in tale riga, riportando i dati con due cifre decimali nella parte frazionaria, allineati a destra

Esempio:

```
A 1.32%
B 12.20%
C 0.01%
D 3.00%
...
```

Osservate come solitamente nei testi in lingua inglese la lettera che ricorre più frequentemente sia la e, mentre nei testi in lingua italiana la "competizione" sia accesa tra le lettere e e a.

Soluzione

```python
def print_letter_freq(path):
    with open(path, "r", encoding="utf-8") as fp:
        text = fp.read().upper()

    character_counts = dict()
    total_characters = len(text)
    for character in text:
        if character in character_counts:
            # se il carattere ha già un conteggio inizializzato,
            # allora lo incremento
            character_counts[character] += 1
        else:
            # altrimenti lo inizializzo a 1
            character_counts[character] = 1

    # calcoliamo le frazioni e le stampiamo
    for character, count in character_counts.items():
        perc = count / total_characters * 100.0
        # usiamo `repr` per visualizzare anche i caratteri "invisibili"
        # come gli spazi bianchi, e formattiamo la percentuale
        # con un numero limitato (ad esempio 3) di cifre decimali
        print("%s %.3f" % (repr(character), perc))

path = input("inserire il nome di un file: ").strip()
if path == "":
    path = "american_fairy_tales.txt"
```

```
print_letter_freq(path)
```

Esercizio 3: triangoli

Progettare il modulo `triangle.py` che contenga la definizione della classe `Triangle`, i cui esemplari rappresentino triangoli geometrici nel piano cartesiano, oltre a codice di collaudo.

Per prima cosa definire la classe `Point` che descriva un punto nel piano cartesiano: il costruttore riceve le coordinate `x` e `y` del punto; il metodo di esemplare `x()` restituisce la coordinata `x` e il metodo di esemplare `y()` restituisce la coordinata `y`, mentre il metodo di esemplare `xy` restituisce una tupla contenente, nell'ordine, la coordinata `x` e la coordinata `y`; il metodo di classe `delta(p1, p2)` calcola e restituisce la distanza tra il punto `p1` e il punto `p2`.

La classe `Triangle` ha un costruttore che richiede tre punti (cioè tre esemplari di `Point`) e solleva l'eccezione `ValueError` se tali punti non costituiscono un triangolo (ricordando che tre punti definiscono un triangolo se e solo se ciascuno dei tre segmenti da essi definiti è minore della somma degli altri due). Per decidere quali siano le migliori informazioni di stato per un tale oggetto, è bene analizzare la sua interfaccia pubblica, così definita:

- il metodo `area()` restituisce l'area del triangolo (può essere utile ricordare la formula di Erone)
- il metodo `height()` restituisce l'altezza relativa al lato maggiore
- il metodo `isScalene()` restituisce True se e solo se il triangolo è scaleno
- il metodo `isIsosceles()` restituisce True se e solo se il triangolo è isoscele
- il metodo `isEquilateral()` restituisce True se e solo se il triangolo è equilatero (ovviamente un triangolo equilatero è anche isoscele)
- il metodo `isRight()` restituisce True se e solo se il triangolo è rettangolo (ovviamente un triangolo rettangolo può anche essere isoscele o scaleno)

Soluzione

```
# Importiamo la funzione `isclose` che consente di verificare
# se due numeri floating point sono "sufficientemente vicini".
# Come abbiamo visto infatti confrontare per uguaglianza i numeri floating
# point ha spesso poco senso.
from math import isclose

class Point :

    def __init__(self, x, y) :
```

```
        self._x = x
        self._y = y

    def x(self) :
        return self._x

    def y(self) :
        return self._y

    def xy(self) :
        return (self._x, self._y)

    def delta(p1, p2) :
        return ((p1._x - p2._x)**2 + (p1._y - p2._y)**2)**(1/2)

class Triangle :
    # dato che i singoli punti che definiscono il triangolo non vengono
    # utilizzati dai metodi, che invece usano le lunghezze dei lati, è
    # più comodo usare quelle come variabili di esemplare; siccome, poi,
    # alcuni metodi hanno bisogno di sapere quale sia il lato più lungo,
    # memorizziamo le lunghezze dei lati in ordine decrescente in una
    # tupla (non serve una lista, perché nessun metodo li modifica)
    def __init__(self, p1, p2, p3) :
        sides = (Point.delta(p1,p2), Point.delta(p2,p3), Point.delta(p1,p3))
        maxSide = max(sides)
        minSide = min(sides)
        middleSide = sum(sides) - minSide - maxSide
        self._sides = (maxSide, middleSide, minSide)
        if maxSide >= middleSide + minSide :
            raise ValueError("I punti non definiscono un triangolo")

    def area(self) : # uso la formula di Erone
        (d1, d2, d3) = self._sides
        p = (d1 + d2 + d3) / 2
        return (p * (p - d1) * (p - d2) * (p - d3))**(1/2)

    def height(self) : # l'altezza relativa al lato maggiore
        return 2 * self.area() / self._sides[0]

    def isScalene(self) :
        (d1, d2, d3) = self._sides
```

```
        return not isclose(d1, d2) and not isclose(d2, d3) and not isclose(d1, d3)

    def isIsosceles(self) :
        (d1, d2, d3) = self._sides
        return isclose(d1, d2) or isclose(d2, d3) or isclose(d1, d3)

    def isEquilateral(self) :
        (d1, d2, d3) = self._sides
        # osservare che, ovviamente, la relazione indotta da isclose NON è
        # transitiva... quindi devo controllare le tre coppie di lati
        # infatti, ad esempio, d1 potrebbe essere prossimo a d2 e d2
        # potrebbe essere prossimo a d3, ma la differenza tra d1 e d3
        # potrebbe essere eccessiva
        return isclose(d1, d2) and isclose(d2, d3) and isclose(d1, d3)

    def isRight(self) :
        (d1, d2, d3) = self._sides
        # verifico se la formula pitagorica è rispettata, l'ipotenusa
        # è ovviamente il lato più lungo, cioè d1
        return isclose(d1*d1, d2*d2 + d3*d3)

# codice di collaudo
t = Triangle(Point(0, 0), Point(1, 1), Point(0, 1))
print(t.area())
print(t.isRight()) # osservare che senza l'uso di isclose sarebbe False
print(t.isScalene())
print(t.isEquilateral())
print(t.isIsosceles())
# aggiungere molte altre prove...
```