Laboratorio 5

Elementi di Informatica e Programmazione

Esercizio 1: modulo `myinput`

Scrivere il modulo `myinput.py` che contenga le seguenti funzioni di utilità per l'acquisizione di dati dall'utente:

- `inputYesNo(message, yes, no)` restituisce `True` se l'utente ha scritto la stringa fornita come secondo argomento, restituisce `False` se l'utente ha scritto la stringa fornita come terzo argomento; ignora la differenza tra maiuscole e minuscole. Esempio di utilizzo:

```
if inputYesNo("Vuoi continuare? (S/N) ", "S", "N"):
    # vuole continuare
```

- `inputStringStartingWith(message, startingString)` restituisce una stringa che inizia con la stringa `startingString`; se `startingString` è la stringa vuota, viene restituita la prima stringa digitata dall'utente (anche se è la stringa vuota)
- `inputStringEndingWith(message, endingString)` restituisce una stringa che termina con la stringa `endingString`; se `endingString` è la stringa vuota, viene restituita la prima stringa digitata dall'utente (anche se è la stringa vuota)
- `inputStringContaining(message, substring)` restituisce una stringa che contiene la stringa substring; se substring è la stringa vuota, viene restituita la prima stringa digitata dall'utente (anche se è la stringa vuota)
- `isDecimalInteger(s)` restituisce True se e solo se la stringa s contiene un numero intero decimale, che ha questo formato: zero o più spazi iniziali, un eventuale segno meno, una o più cifre decimali (da 0 a 9), zero o più spazi finali (quindi, ad esempio, non ci può essere uno spazio tra il segno meno e la prima cifra del numero)
- `inputDecimalInteger(message)` restituisce un numero intero; la funzione deve utilizzare in modo opportuno la funzione isDecimalInteger

- `inputPositiveDecimalInteger(message)` restituisce un numero intero positivo; la funzione deve utilizzare in modo opportuno la funzione `inputDecimalInteger`
- `inputNegativeDecimalInteger(message)` restituisce un numero intero negativo; la funzione deve utilizzare in modo opportuno la funzione inputDecimalInteger
- `inputNonPositiveDecimalInteger(message)` restituisce un numero intero non positivo; la funzione deve utilizzare in modo opportuno la funzione inputDecimalInteger
- `inputNonNegativeDecimalInteger(message)` restituisce un numero intero non negativo; la funzione deve utilizzare in modo opportuno la funzione inputDecimalInteger
- `isFloating(s)` restituisce True se e solo se la stringa s contiene un numero decimale in virgola mobile, che ha questo formato: zero o più spazi iniziali, un eventuale segno meno, una o più cifre decimali (da 0 a 9), un eventuale separatore decimale (il carattere "punto") seguito da una o più cifre decimali, un'eventuale lettera "e" (maiuscola o minuscola) seguita da un eventuale segno meno e da una o più cifre decimali, zero o più spazi finali
- `inputFloating(message)` restituisce un numero in virgola mobile; la funzione deve utilizzare in modo opportuno la funzione isFloating

Tutte le funzioni di tipo `input...` devono chiedere ripetutamente il dato all'utente finché questo non rispetta le specifiche della funzione, riproponendo il messaggio message (senza visualizzare messaggi d'errore). Tutte le funzioni di tipo `is...`, invece, non devono avere alcuna interazione con l'utente (né in input né in output).

Soluzione

```
def inputYesNo(message, yes, no) :
    while True :
        s = input(message)
        if s == yes :
            return True
        if s == no :
            return False

def inputStringStartingWith(message, startingString) :
    while True :
        s = input(message)
        if s.startswith(startingString) :
            return s

def inputStringEndingWith(message, endingString) :
    while True :
```

```
        s = input(message)
        if s.endswith(endingString) :
            return s

def inputStringContaining(message, substring) :
    while True :
        s = input(message)
        if substring in s :
            return s

def isDecimalInteger(s) :
    atLeastOneDigit = False
    i = 0
    while i < len(s) : # spazi iniziali ?
        if s[i] != ' ' :
            break
        i += 1 # il valore finale di i corrisponde al primo carattere
               # della stringa che NON è uno spazio (eventualmente zero)
    if i < len(s) and s[i] == '-' : # segno meno ?
        i += 1
    while i < len(s) : # cifre decimali ?
        if not s[i].isdigit() :
            break
        atLeastOneDigit = True
        i += 1
    while i < len(s) : # spazi finali ?
        if s[i] != ' ' :
            break
        i += 1
    # se i == len(s) vuol dire che sono riuscito a scandire tutta
    # la stringa rispettando il formato, devo solo controllare che
    # ci sia almeno una cifra numerica (potrebbero essere tutti spazi)
    return i == len(s) and atLeastOneDigit
    # osservate bene l'istruzione qui sopra... quando si deve eseguire
    # una cosa come questa
    #
    #     if condizione :
    #         return True
    #     else:
    #         return False
    #
```

```
    # si può semplicemente scrivere
    #
    #     return condizione
    #
    # perché? capire...
    # analogamente, al posto di
    #
    #     if condizione :
    #         variabile = True
    #     else :
    #         variabile = False
    #
    # si può scrivere
    #
    #     variabile = condizione

def inputDecimalInteger(message) :
    while True :
        s = input(message)
        if isDecimalInteger(s) : # sicuramente int(s) non fallirà
            return int(s)

def inputPositiveDecimalInteger(message) :
    while True :
        n = inputDecimalInteger(message)
        if n > 0 :
            return n

def inputNegativeDecimalInteger(message) :
    while True :
        n = inputDecimalInteger(message)
        if n < 0 :
            return n

def inputNonPositiveDecimalInteger(message) :
    while True :
        n = inputDecimalInteger(message)
        if n <= 0 :
            return n

def inputNonNegativeDecimalInteger(message) :
    while True :
```

```
        n = inputDecimalInteger(message)
        if n >= 0 :
            return n

def isFloating(s) :
    atLeastOneDigit = False
    i = 0
    while i < len(s) : # spazi iniziali ?
        if s[i] != ' ' :
            break
        i += 1 # il valore finale di i corrisponde al primo carattere
               # della stringa che NON è uno spazio (eventualmente zero)
    if i < len(s) and s[i] == '-' : # segno meno ?
        i += 1
    while i < len(s) : # cifre decimali della parte intera ?
        if not s[i].isdigit() :
            break
        atLeastOneDigit = True
        i += 1
    if i < len(s) and s[i] == '.' : # separatore decimale ?
        i += 1
    while i < len(s) : # cifre decimali della parte frazionaria ?
        if not s[i].isdigit() :
            break
        atLeastOneDigit = True
        i += 1
    # parte esponenziale facoltativa...
    eLetterIsPresent = False
    atLeastOneExponentDigit = False
    if i < len(s) and s[i].upper() == 'E' : # lettera "e" maiuscola o minuscola ?
        eLetterIsPresent = True
        i += 1
        if i < len(s) and s[i] == '-' : # segno meno dell'esponente ?
            i += 1
        while i < len(s) : # cifre decimali dell'esponente ?
            if not s[i].isdigit() :
                break
            atLeastOneExponentDigit = True
            i += 1
    while i < len(s) : # spazi finali ?
        if s[i] != ' ' :
```

```
            break
        i += 1
    return i == len(s) and atLeastOneDigit \
             and (eLetterIsPresent == atLeastOneExponentDigit)

def inputFloating(message) :
    while True :
        s = input(message)
        if isFloating(s) : # sicuramente float(s) non fallirà
            return float(s)
```

Tutte le funzioni che cominciano con `is` possono anche essere implementate con le espressioni regolari. Ad esempio la funzione `isFloating` può essere implementata con le funzioni regolari come segue:

```
def isFloating(s):
    import re
    match = re.search(
      r"^\s*-?[0-9]+(\.[0-9]+)?([eE]-?[0-9]+)?\s*$",
      s
    )
    return match is not None

assert isFloating("0.234")
assert not isFloating(".234")
assert not isFloating("abcd")
assert not isFloating("0.2 abcd")
assert isFloating("12.234")
assert isFloating("-12.234")
assert isFloating("-12.234e12")
assert isFloating("-12.234e-12")
assert isFloating("-12e-12")
assert isFloating("  -12e-12  ")
```

Per chiarezza, scomponiamo l'espressione regolare:

- `\s*` all'inizio e alla fine corrisponde a "zero o più spazi"
- i caratteri `^` e `$` indicano che l'espressione regolare deve coprire l'intera stringa
- procedendo in ordine con il resto degli elementi dell'espressione regolare:
  - `-?` indica un segno `-` opzionale
  - `[0-9]+` indica una o più cifre decimale

- `(\.[0-9]+)?` indica un gruppo opzionale, formato da un punto e da una o più cifre decimali
- `([eE]-?[0-9]+)?` indica un altro gruppo opzionale (si noti il `?` alla fine) composto da:
  - `[eE]` una lettera `e`, maiuscola o minuscola
  - `-?` un segno meno opzionale
  - `[0-9]+` una o più cifre decimali

Esercizio 2: nomi dei numeri

Scrivere il programma `number_name.py` che acquisisca un numero intero positivo minore di un milione e ne visualizzi la descrizione in italiano, seguendo le regole riportate qui: https://it.wiktionary.org/wiki/Appendice:Tutti_i_numeri _in_lettere (attenzione ai molti casi particolari…). Definire funzioni opportune in modo da evitare, per quanto possibile, la duplicazione di codice.

Potete usare le funzioni del modulo `myinput` sviluppato nell'esercizio precedente per gestire l'interazione con l'utente.

Soluzione

```
from myinput import *
from sys import exit

def main() :
    n = inputPositiveDecimalInteger("Numero intero positivo minore di un milione: ")
    if n >= 1_000_000 : # notare l'uso di _ per separare le migliaia: è ammesso
        exit("Numero troppo grande!")
    s = ""
    # 1 <= n <= 999999
    thousands = n // 1000
    # 1 <= thousands <= 999
    if thousands > 1 :
        s += upTo999(thousands)
        s += "mila"
    elif thousands == 1 :
        s += "mille"
    x = n % 1000
    # 1 <= x <= 999
    s += upTo999(x)
```

```
    if n % 10 == 3 and n != 3 :
        s = s[:-1] + "é"
    print(s)

def upTo999(n) :
    s = ""
    hundreds = n // 100
    # 1 <= hundreds <= 9
    if hundreds > 1 :
        s += upTo9(hundreds)
    if hundreds >= 1 :
        s += "cento"
    n = n % 100
    # 1 <= n <= 99
    s += upTo99(n)
    return s

def upTo99(n) :
    if n <= 9 : return upTo9(n)
    if n == 10 : return "dieci"
    if n == 11 : return "undici"
    if n == 12 : return "dodici"
    if n == 13 : return "tredici"
    if n == 14 : return "quattordici"
    if n == 15 : return "quindici"
    if n == 16 : return "sedici"
    if n == 17 : return "diciassette"
    if n == 18 : return "diciotto"
    if n == 19 : return "diciannove"
    tens = n // 10
    # 2 <= tens <= 9
    if tens == 2 : s = "venti"
    elif tens == 3 : s = "trenta"
    elif tens == 4 : s = "quaranta"
    elif tens == 5 : s = "cinquanta"
    elif tens == 6 : s = "sessanta"
    elif tens == 7 : s = "settanta"
    elif tens == 8 : s = "ottanta"
    elif tens == 9 : s = "novanta"
    units = upTo9(n % 10)
    if units == "uno" or units == "otto" :
```

```
        s = s[:-1] # tolgo la vocale finale delle decine
    return s + units


def upTo9(n) :
    if n == 1 : return "uno"
    if n == 2 : return "due"
    if n == 3 : return "tre"
    if n == 4 : return "quattro"
    if n == 5 : return "cinque"
    if n == 6 : return "sei"
    if n == 7 : return "sette"
    if n == 8 : return "otto"
    if n == 9 : return "nove"
    return ""


main()
```

Esercizio 3: gioco fantasy - inventario

In un videogioco a tema fantasy ciascun protagonista ha a disposizione un inventario di oggetti. Il videogioco rappresenta l'inventario come un dizionario dove le chiavi sono stringhe che descrivono l'oggetto e il valore è un intero rappresentante il numero di oggetti di quel tipo. Ad esempio, il dizionario

```
{'corda': 1, 'torcia': 6, "monete d'oro": 42, 'spada': 1, 'freccia': 12}
```

rappresenta un inventario con una corda, 6 torce, 42 monete, una spada e 12 frecce.

Sviluppate una funzione `display_inventory()` che dato un inventario lo stampi come segue:

```
Inventario:
12 freccia
42 monete d'oro
1 corda
6 torcia
1 spada
Numero totale di elementi: 62
```

L'ordine degli elementi non è importante.

Soluzione

```
def display_inventory(inventory):
    print("Inventario")
    cnt = 0
    for (name, count) in inventory.items():
        print(count, name)
        cnt += count
    print("Numero totale di elementi:", cnt)


stuff = {'corda': 1, 'torcia': 6, "monete d'oro": 42, 'spada': 1, 'freccia': 12}
display_inventory(stuff)
```

```
Inventario
1 corda
6 torcia
42 monete d'oro
1 spada
12 freccia
Numero totale di elementi: 62
```

Esercizio 4: anagrammi

Un anagramma è una permutazione delle lettere di una parola compiuta in modo tale da ottenere un'altra parola di senso compiuto. Quindi data una coppia di parole possiamo chiederci se una è l'anagramma dell'altra. Scrivere una funzione `anagrams` che riceve in input una lista di parole e ritorna una lista dei gruppi di parole che sono l'una l'anagramma dell'altra.

Ad esempio, se la lista di parole è:

```
[
  'canori', 'carino',
  'cranio', 'naso',
  'rancio', 'gelo',
  'gole', 'lego'
]
```

la funzione deve ritornare

```
[
  ['gelo', 'gole', 'lego'],
  ['canori', 'carino', 'cranio', 'rancio']
]
```

Attenzione: la parola `naso` non fa parte di nessuna lista di output, perchè non è l'anagramma di nessun'altra parola della lista.

Nella progettazione della funzione considerate i seguenti fatti:

- la relazione di anagramma è simmetrica, ovvero "lego" è anagramma di "gole" e "gole" è anagramma di "lego"
- se ordiniamo le lettere di una parola in ordine alfabetico (ovvero "lego" diventa "eglo"), abbiamo che tutte le parole che sono anagramma l'una dell'altra hanno la stessa sequenza di lettere ordinate.
- se una parola è l'anagramma dell'altra, hanno lo stesso numero di lettere

Per implementare la funzione ricordate che (tra le altre cose):

- per ordinare i caratteri di una stringa `s` potete usare `sorted(s)`. Qual è il risultato? Qual è il tipo di dato?
- potete trasformare una lista di caratteri `l` in una stringa usando `"".join(l)` (controllate la documentazione del metodo `join` delle stringhe)
- potete usare un dizionario per memorizzare i gruppi di parole che sono una l'anagramma dell'altra. Quali sono le chiavi? Quali sono i valori?
- dato un dizionario `d`, con `d.values()` potete iterare sui *valori* contenuti nel dizionario

Per testare la funzione potete usare la lista di parole contenuta nel file `parole.txt` scaricabile da moodle.

Soluzione

```
def anagrams(words):
    # Manteniamo i gruppi di anagrammi in un dizionario.
    # Le chiavi del dizionario sono le sequenze di
    # lettere in ordine alfabetico,
    # i valori sono liste di parole le cui lettere
    # ordinate corrispondono alla chiave.
    anagrams_dictionary = dict()

    for word in words:
        # trasformiamo ogni parola in minuscolo, e
        # rimuoviamo gli spazi iniziali e finali
```

```
        word = word.lower().strip()
        # ordiniamo i caratteri e trasformiamo la lista
        # risultante in una stringa
        sorted_letters = "".join(sorted(word))

        # se la chiave (ovvero la lista di stringhe) non è
        # nel dizionario allora la aggiungiamo, associandola
        # a una lista vuota.
        if sorted_letters not in anagrams_dictionary:
            anagrams_dictionary[sorted_letters] = []
        # aggiungiamo alla lista corrispondente alla chiave
        # la prola corrente
        anagrams_dictionary[sorted_letters].append(word)

    # inizializziamo una lista di output
    output = []
    for group in anagrams_dictionary.values():
        # per ogni gruppo di parole nei valori del dizionario,
        # se il gruppo contiene più di un elemento lo aggiungiamo
        # alla lista di output
        if len(group) > 1:
            output.append(group)

    # (opzionale) ordiniamo l'output per lunghezza dei gruppi
    output.sort(key=len)
    return output

# qui leggiamo tutte le parole dal file e testiamo la funzione
with open("parole.txt") as fh:
    words = fh.readlines()
    for group in anagrams(words):
        print(group)
```