Laboratorio 8

Elementi di Informatica e Programmazione

Esercizio 1: cifratura di Cesare - encrypt

Scrivere il programma `caesarEncrypt.py` che cifri un testo usando la cifratura di Cesare: il testo da cifrare è contenuto in un file il cui nome viene fornito come primo parametro sulla riga dei comandi; il testo cifrato viene visualizzato sul flusso di output standard.

Il parametro PARAM che governa la cifratura (che, come vedremo, è un numero intero positivo) può essere fornito come secondo parametro sulla riga dei comandi (e, in sua assenza, viene assunto uguale a 3). Il programma non interagisce con il flusso di input standard.

La cifratura di Cesare opera sostituendo ciascuna lettera del testo (i caratteri che non sono lettere rimangono identici nel testo cifrato) con un'altra lettera determinata procedendo "in avanti" nell'alfabeto di un numero di posti uguale a PARAM, ripartendo dall'inizio se si arriva all'ultima lettera.

Ad esempio, con `PARAM=3`, la lettera A viene sostituita dalla lettera D, B dalla E, e così via, fino a W sostituita da Z, X sostituita da A, Y sostituita da B e Z sostituita da C. Le lettere maiuscole rimangono maiuscole e le lettere minuscole rimangono minuscole. Ovviamente questo algoritmo di cifratura è facilmente decifrabile.

Nota per il collaudo: se `PARAM=26` o multiplo, il testo cifrato deve coincidere con il testo da cifrare

```
def encrypt(text, param):
    output = ""
    for c in text:
        if "a" <= c <= "z":
            enc_c = chr(ord('a') + ((ord(c) - ord('a') + param) % 26))
        elif "A" <= c <= "Z":
            enc_c = chr(ord('A') + ((ord(c) - ord('A') + param) % 26))
        else:
            enc_c = c
```

```
        output += enc_c
    return output

def main():
    import sys
    if len(sys.argv) == 3:
        path = sys.argv[1]
        param = sys.argv[2]
    elif len(sys.argv) == 2:
        path = sys.argv[1]
        param = 3
    else:
        exit("non ci sono abbastanza argomenti!")

    with open(path, "r", encoding="utf-8") as fp:
        text = fp.read()

    enc = encrypt(text, param)
    print(enc)
```

Esercizio 2: cifratura di Cesare - decrypt

scrivere il programma `caesarDecrypt.py` che decifri un testo che sia stato cifrato usando la cifratura di Cesare: il testo da decifrare è contenuto in un file il cui nome viene fornito come primo parametro sulla riga dei comandi.

Il testo decifrato viene visualizzato sul flusso di output standard. Il parametro PARAM che governa la decifrazione può essere fornito come secondo parametro sulla riga dei comandi (e, in sua assenza, viene assunto uguale a 3) e deve essere identico a quello utilizzato per la cifratura. Il programma non interagisce con il flusso di input standard. Per la soluzione, partire dal programma `caesarEncrypt.py` e fare il minor numero di modifiche possibili.

```
def decrypt(text, param):
    output = ""
    for c in text:
        if "a" <= c <= "z":
            dec_c = chr(ord('a') + ((ord(c) - ord('a') - param) % 26))
        elif "A" <= c <= "Z":
            dec_c = chr(ord('A') + ((ord(c) - ord('A') - param) % 26))
        else:
```

```
            dec_c = c
        output += dec_c
    return output

def main():
    import sys
    if len(sys.argv) == 3:
        path = sys.argv[1]
        param = sys.argv[2]
    elif len(sys.argv) == 2:
        path = sys.argv[1]
        param = 3
    else:
        exit("non ci sono abbastanza argomenti!")

    with open(path, "r", encoding="utf-8") as fp:
        text = fp.read()

    dec = decrypt(text, param)
    print(dec)
```

Esercizio 3: scomposizione in fattori

Scrivere il modulo `factor_generator.py` contenente la definizione della classe `FactorGenerator` che effettui la scomposizione di un numero intero positivo nei suoi fattori primi. La classe deve avere:

- un costruttore che riceve come unico parametro un numero intero e verifica che sia positivo e maggiore di uno (sollevando `ValueError` in caso contrario): sarà il numero da scomporre in fattori primi;
- un metodo di esemplare, `next_factor`, che non riceve parametri espliciti e che, a ogni sua successiva invocazione, restituisce uno dei fattori primi in cui viene scomposto il numero originariamente fornito nel costruttore; invocando un numero sufficiente di volte tale metodo, si ottengono tutti e soli i fattori primi del numero (eventualmente ripetuti), in modo che moltiplicandoli tra loro si ottenga il numero originario (ad esempio, il numero 150 viene scomposto nei suoi fattori primi 2, 3, 5, 5); se il metodo viene invocato dopo che ha già fornito tutti i fattori primi, esso lancia `StopIteration` (attenzione, non `ValueError`, che non avrebbe senso, dal momento che il metodo non ha parametri espliciti…);
- un metodo di esemplare, `has_more_factors`, che non riceve parametri espliciti e che restituisce il valore booleano `True` se e solo se esistono fattori primi non ancora restituiti

da `next_factor` (cioè se `next_factor` è ancora invocabile).

Porre particolare attenzione all'individuazione delle informazioni di stato necessarie al funzionamento della classe, tentando di minimizzarne la dimensione (suggerimento: è sufficiente un'unica variabile di tipo int).

Soluzione

```
class FactorGenerator:
    def __init__(self, number):
        if number < 1:
            raise ValueError
        self._number = number

    def has_more_factors(self):
        return self._number > 1

    def next_factor(self):
        if self._number <= 1:
            raise StopIteration
        i = 2
        while i <= self._number:
            if self._number % i == 0:
                self._number = self._number // i
                return i
            else:
                i += 1

fs = FactorGenerator(123)
while fs.has_more_factors():
    f = fs.next_factor()
    print(f)
```