Tutoriat 6 – Recursivitate. Lambda funcții. Sortări liste
- rezolvări -

1. Ipoteza Collatz spune că plecând de la orice număr natural, dacă aplicăm repetat următoarea operație

$$n \longmapsto \begin{cases} \frac{n}{2} & , daca\ n\ par \\ 3n+1 & , daca\ n\ impar \end{cases}$$

șirul ce rezultă va atinge valoarea `1`. Implementați o funcție recursivă, care returnează numărul de operații efectuate până la atingerea valorii `1`.

```python
def collatz( n, cnt = 0 ):
    if n == 1:
        return cnt
    if n % 2 == 0:
        return collatz( n / 2, cnt + 1 )
    return collatz( 3 * n + 1, cnt + 1 )

def collatz1( n ):
    if n == 1:
        return 0
    if n % 2 == 0:
        return 1 + collatz( n / 2 )
    return 1 + collatz( 3 * n + 1 )

print( collatz( int( input('n = ') ) ) )
```

2. În teoria informației, distanța Levenshtein măsoară diferența dintre două cuvinte, ca fiind numărul minim de operații elementare (inserare, ștergere, modificare pe câte un caracter) ce pot fi făcute pe primul cuvânt, pentru a fi obținut cel de-al doilea cuvânt. Distanța Levenshtein dintre două cuvinte `a,b` (de lungimi `|a|`, respectiv `|b|`) este dată de `lev(a, b)`, unde

$$lev(a,b) = \begin{cases} |a| & daca\ |b| = 0 \\ |b| & daca\ |a| = 0 \\ lev\big(\ tail(a), tail(b)\big) & daca\ a[0] = b[0] \\ 1 + min \begin{cases} lev(\ tail(a), b\ ) \\ lev(\ a, tail(b)\ ) \\ lev(\ tail(a), tail(b)\ ) \end{cases} & altfel \end{cases}$$

unde `tail(x)=` șirul format din `x` prin eliminarea primului caracter. Scrieți o funcție recursivă care calculează distanța Levenshtein dintre două cuvinte.

```
def lev( a, b ):
    if len( a ) == 0:
        return len( b )
    if len( b ) == 0:
        return len( a )
    if a[0] == b[0]:
        return lev( a[1:], b[1:] )
    return 1 + min( lev( a[1:], b ), lev( a, b[1:] ), lev( a[1:], b[1:] ) )

print( lev( input('a = '), input('b = ') ) )
```

3. Scrieți o funcție recursivă care să calculeze numărul de cifre prime ale unui număr natural primit ca parametru.

```
def cifreprime( n ):
    if n < 10 and n in [ 2, 3, 5, 7 ]:
        return 1
    if n < 10:
        return 0
    if n % 10 in [ 2, 3, 5, 7 ]:
        return 1 + cifreprime( n // 10 )
    return cifreprime( n // 10 )

print( cifreprime( int( input('n = ') ) ) )
```

4. Scrieți o lambda funcție care primește ca parametru un număr natural și întoarce răsturnatul său. Apelați funcția pentru un număr citit de la tastatură și afișați rezultatul. Toată rezolvarea problemei trebuie să aibă o singură linie.

```
print( ( lambda x: int( str(x)[::-1] ) )( int( input('n = ') ) ) )
```

5. Se citește o listă de numere naturale. Sortați lista descrescător după numărul de divizori.

```
lista = [ int( x ) for x in input().split() ]
lista.sort( key=lambda x: len( [ d for d in range( 1, x + 1 ) if x % d == 0 ] ),
reverse=True )
print( lista )
```

6. Se citește o listă de numere naturale. Sortați lista crescător după numărul de cifre, iar, în caz de egalitate, descrescător după media aritmetică a cifrelor.

```
lista = [ int( x ) for x in input().split() ]
lista.sort( key=lambda x: ( len( str(x) ), -sum( [ int(x) for x in str(x) ] ) /
len( [ int(x) for x in str(x) ] ) ) )
print( lista )
```

7. Se citește o listă de numere naturale. Sortați lista crescător după numărul de cifre **2**, iar, în caz de egalitate, crescător după valoarea efectivă.

```
lista = [ int(x) for x in input().split() ]
lista.sort( key=lambda x: ( str(x).count( '2' ), x ) )
print( lista )
```