

# Kurs języka Python

## Lista 3.

Poniżej są zadania polegające na implementacji funkcji zwracających listy liczb naturalnych. Każde z zadań należy wykonać w dwóch wersjach: w wersji z listą składaną i wersję funkcyjną.

Wersja z *listą składaną* powinna być w postaci jednej listy składanej, zawierającej być może inną listę składaną. W przypadku bardzo długich wyrażeń akceptowane będzie wydzielenie podlisty składanej

```
def zadana_funkcja(n):
    lista_tymcz = [ lista skladana ]
    return [ lista_skladana_zawierajaca lista_tymcz ]
```

*Implementacja funkcyjna* powinna korzystać z funkcji dedykowanych do operacji na listach: `filter`, `map` czy `reduce`.

Zbadaj, która wersja jest szybsza.

Do zaprogramowania powyższych zadań wystarczą standardowe funkcje i operatory, nie ma potrzeby korzystania z dodatkowych modułów.

**Zadanie 1.** Zaprogramuj jednoargumentowe funkcje `pierwsze_skladana(n)` i `pierwsze_funkcyjna(n)`, które zwracają listę liczb pierwszych nie większych niż  $n$ , na przykład

```
>>> pierwsze(20)
[2, 3, 5, 7, 11, 13, 17, 19]
```

**Zadanie 2.** Zaprogramuj jednoargumentowe funkcje `doskonale_skladana(n)` i `doskonale_funkcyjna(n)`, które zwracają listę liczb doskonałych nie większych niż  $n$ , na przykład

```
>>> doskonale(1000)
[6, 28, 496, 8128]
```

**Zadanie 3.** Zaprogramuj jednoargumentowe funkcję `rozklad_skladana(n)` i `rozklad_funkcyjna(n)` które obliczają rozkład liczby  $n$  na czynniki pierwsze i zwracają jako wynik listę par  $[(p_1, w_1), (p_2, w_2), \dots, (p_k, w_k)]$  taką, że  $n = p_1^{w_1} * p_2^{w_2} * \dots * p_k^{w_k}$  oraz  $p_1, \dots, p_k$  są różnymi liczbami pierwszymi. Na przykład

```
>>> rozklad(756)
[(2, 2), (3, 3), (7, 1)]
```

Ponieważ w tym zadaniu może być potrzebna lista liczb pierwszych, można zaimplementować pomocniczą funkcję sprawdzającą pierwszość liczby bądź zwracającą listę liczb pierwszych. W przypadku tej funkcji pomocniczej implementacja może być dowolna.

**Zadanie 4.** Zaprogramuj jednoargumentowe funkcje `zaprzyjaznione_skladana(n)` i `zaprzyjaznione_funkcyjna(n)`, które zwracają listę par liczb zaprzyjaznionych nie większych niż  $n$ , na przykład

```
>>> zaprzyjaznione(1300)
[(220, 284), (1184, 1210)]
```

Odpowiednie definicje można znaleźć np. w polskiej Wikipedii. Wybierz dwa z podanych zadań. Każde zadanie jest warte 2 pkt.