Zestaw 2

1. Mamy zagnieżdżoną listę, na przykład: list1 = [1, 2, [3, 4, [5, 6], 5], 3, [4, 5]]. Dodaj element o kolejnej wartości w najbardziej zagnieżdżonej liście. W tym przypadku 7 w miejscu [1, 2, [3, 4, [5, 6, 7], 5], 3, [4, 5]]. Napisz program, który zrobi to uniwersalnie, dla dowolnego zagnieżdżenia, np. dla [1 [2, 3] 4] chodzi o [1 [2, 3, 4] 4], dla [3, 4, [2, [1, 2, [7, 8], 3, 4], 3, 4], 5, 6, 7] powinno być [3, 4, [2, [1, 2, [7, 8, 9], 3, 4], 3, 4], 5, 6, 7]. Jeżeli największe zagnieżdżenie na danym poziomie się powtórzy, należy dodać w obu zagnieżdżeniach, czyli dla [1, [3], [2]] należy uzyskać [1, [3, 4], [2, 3]].

2. Napisać program, który będzie wyświetlał bieżący czas (tak ma to wyglądać: ► 14:48:31 ◄), aktualizowany dynamicznie. Czas można odczytać na wiele sposobów, użyjmy moduł datetime, wtedy, bieżący punkt w czasie dostaniemy: now = datetime.now() i za pomocą składowych now.hour, now.minute, now.second mamy potrzebne wartości. Przy czym dla sekund należy sprytnie podmienić sekundy w zakresie 0..9 tak, żeby przed nimi wyświetlało się zero (np. nie 5, tylko 05). Znaczki na początku i końcu mają kod chr(16) i chr(17). Zegar musi być wyświetlany w nieskończonej pętli funkcją print(), argument end='\r' zapewni nadpisywanie. Potrzebne jest jeszcze (z modułu time) wołanie czegoś typu time.sleep(0.5) w pętli, żeby niepotrzebnie nie odświeżać zbyt często bieżącego odczytu czasu.

3. Dla dowolnego podanego łańcucha znakowego wypisać: ile jest w nim *słów* (poprzez słowo rozumiemy ciąg co najmniej jednego znaku innego niż znak przestankowy, dla uproszczenia przyjmijmy, że liczymy a-z, A-Z i 0-9 jako coś, co składa się na słowa), ile *liter*, ile *cyfr*, oraz wypisać statystykę częstości występowania poszczególnych liter oraz cyfr.

4. Mamy trzy liczby całkowite, x, y, z reprezentujące wymiary prostopadłościanu, oraz pewną liczbę naturalną n. Wypisz listę wszystkich możliwych współrzędnych (i, j, k) na trójwymiarowej siatce, gdzie i+j+k nie jest równe n. Warunki: $0 \leq i \leq x$, $0 \leq j \leq y$, $0 \leq k \leq z$. Rozwiązanie zapisz w postaci list składanych (list comprehesion), ale można zacząć od zagnieżdżonych pętli. Przykład. Niech x = 1, y = 1, z = 2, n = 3. Lista wszystkich permutacji trójek [i, j, k] w tym przykładzie: [[0,0,0], [0,0,1], [0,0,2], [0,1,0], [0,1,1], [0,1,2], [1,0,0], [1,0,1], [1,0,2], [1,1,0], [1,1,1], [1,1,2]]. Elementy, które nie sumują się do 3 to: [[0,0,0], [0,0,1], [0,0,2], [0,1,0], [0,1,1], [1,0,0], [1,0,1], [1,1,0], [1,1,2]]. Parametry x, y, z, n wczytać na początku za pomocą funkcji input().

5. Mamy liczbę naturalną N w zapisie binarnym (czyli składa się tylko z 0 i 1). Binarna przerwa to sekwencja zer otoczonych z lewej i prawej strony 1. Na przykład liczba 9 (decymalnie) binarnie wynosi 1001 i posiada jedną binarną przerwę długości 2. Liczba 529 ma binarną reprezentację 1000010001, zatem ma dwie binarne przerwy, o długości 4 i 3. Liczba 20 ma reprezentację 10100 zawiera zatem jedną binarną przerwę o długości 1. Liczba 15 ma reprezentację 1111, a zatem żadnej binarnej przerwy. Napisz funkcję:

def fun(N)

która dla podanej liczby naturalnej N (uwaga: liczby w systemie dziesiętnym) zwraca długość jej najdłuższej binarnej przerwy, albo 0, jeśli nie ma ani jednej przerwy. Na przykład, dla N = 1041, które binarnie jest 10000010001, ma najdłuższą przerwę binarną 5. Należy przyjąć, że argument N może być z przedziału [1..2147483647]. Wskazówka: warto skorzystać z operatora przesunięcia bitowego >>. Można podejrzeć jak wygląda liczba w zapisie binarnym poprzez rzutowanie bin(N).