

1. Написать функцию на Python

Задание: Напиши функцию, которая удалит дубликаты в списке. Список не отсортирован. Необходимо сохранить порядок сортировки оригинального списка. Какая асимптотическая сложность у этой функций?

Решение:

In [1]:

```
def remove_dupes(data):  
    """  
    Функция для удаления дубликатов в списке.  
    Принимает на вход произвольный список (или другой итерируемый объект).  
    Возвращает новый список, содержащий только уникальные элементы из исходного  
    и в том же порядке, в каком они впервые встречались в нём.  
  
    >>> remove_dupes([1, 2, 3, 1])  
    [1, 2, 3]  
  
    >>> remove_dupes([1, 3, 2, 1, 5, 3, 5, 1, 4])  
    [1, 3, 2, 5, 4]  
  
    >>> remove_dupes(['a', 'b', 'c', 'a'])  
    ['a', 'b', 'c']  
  
    >>> remove_dupes('abca')  
    ['a', 'b', 'c']  
  
    >>> remove_dupes([(1, 0), (1, 2), (0, 1), (2, 0), (1, 0), (0, 1)])  
    [(1, 0), (1, 2), (0, 1), (2, 0)]  
  
    >>> remove_dupes(1231)  
    TypeError: Данные не итерируемы  
    """  
  
    uniques = set()  
    res = list()  
    try:  
        for item in data:  
            if item not in uniques:  
                uniques.add(item)  
                res.append(item)  
  
        return res  
    except TypeError:  
        print('TypeError: Данные не итерируемы')
```

При вызове данной функции создаются множество `uniques` и список `res`. Множество предназначено для обеспечения *уникальности* элементов в результирующем списке, а список — для сохранения *порядка* элементов исходного списка. Использование множества необязательно, т.к. уникальность элементов можно было бы проверять и по списку `res`, однако операция проверки на вхождение элемента во множество `in` имеет асимптотическую сложность $O(1)$, в то время как для списков она имеет сложность $O(n)$.

Для вычисления асимптотической сложности функции необходимо определить сложность каждой операции, связанной со входными данными:

```
for item in data: # O(n) - проход по всем элементам
    if item not in uniques: # O(1) - проверка на вхождение во множество оптимизи
        рована
        uniques.add(item) # O(1) - добавление элемента во множество
        res.append(item) # O(1) - добавление элемента в конец списка
```

Асимптотическая сложность функции: $O(n) * (O(1) + O(1) + O(1)) = O(n)$. В том случае, если бы не использовалось множество, сложность стала бы квадратичной: $O(n^2)$.

In [2]:

```
import doctest
doctest.testmod()
```

Out[2]:

```
TestResults(failed=0, attempted=6)
```

In [3]:

```
a = [1, 2, 3, 1]
b = [1, 3, 2, 1, 5, 3, 5, 1, 4]
c = ['a', 'b', 'c', 'a']
d = 'abca'
e = [(1, 0), (1, 2), (0, 1), (2, 0), (1, 0), (0, 1)]

for i in [a, b, c, d, e]:
    print(i, '-->', remove_dupes(i))
```

```
[1, 2, 3, 1] --> [1, 2, 3]
[1, 3, 2, 1, 5, 3, 5, 1, 4] --> [1, 3, 2, 5, 4]
['a', 'b', 'c', 'a'] --> ['a', 'b', 'c']
abca --> ['a', 'b', 'c']
[(1, 0), (1, 2), (0, 1), (2, 0), (1, 0), (0, 1)] --> [(1, 0), (1, 2), (0, 1), (2, 0)]
```

In [4]:

```
remove_dupes(1231)
```

TypeError: Данные не итерируемы

2. Написать SQL запрос

Задание: Дана таблица employees всех сотрудников компании. Поля:

- full_name TEXT (PK),
- position TEXT,
- department TEXT.

Напиши запрос, выводящий все отделы, в которых меньше 5 разработчиков (position = 'Software Developer').

Решение:

```
SELECT department
FROM employees
WHERE position = 'Software Developer'
GROUP BY department
HAVING count(*) < 5
```

При выполнении этого запроса происходит следующее:

1. Считывается таблица-источник данных employees (оператор FROM);
2. Из этой таблицы выбираются те строки, которые удовлетворяют указанному условию (оператор WHERE);
3. Выбранные строки группируются по значениям поля department (оператор GROUP BY);
4. Из полученных групп выбираются удовлетворяющие агрегирующему условию (оператор HAVING);
5. Выполняется проекция для выбранных групп — отображаются значения их поля department (оператор SELECT).

3. Решить задачу

Задание: Подкинули монету N раз. Кол-во случаев, когда выпал орёл, на 10% больше, чем кол-во случаев, когда выпала решка. При каком N мы можем сказать, что монета «нечестная» (орёл и решка выпадают с разной вероятностью)?

Решение: В первую очередь следует определить, что именно нам известно из постановки задачи.

Исходя из эмпирического определения вероятности, предложение "Кол-во случаев, когда выпал орёл, на 10% больше, чем кол-во случаев, когда выпала решка" можно интерпретировать следующим образом: "Вероятность выпадения орла на 10% больше, чем вероятность выпадения решки."

Обозначим вероятность выпадения решки за q , тогда вероятность выпадения орла равна $p = (q + 0.1)$. Сумма вероятностей выпадения орла и решки равна 1:

$$\begin{aligned}(q + 0.1) + q &= 1 \\ q &= 0.45\end{aligned}$$

Таким образом, вероятность выпадения орла p равна 0.55. Если принять выпадение орла за 1, а выпадение решки за 0, то это число является *выборочным матожиданием* для указанной в условии задачи последовательности из N бросков монеты. Отношение количества раз, когда выпал орёл, к количеству раз, когда выпала решка, равно отношению вероятностей этих событий:

$$\frac{0.55}{0.45} = \frac{11n}{9n},$$

где n - некоторый целочисленный множитель.

Из этого можно сделать вывод, что для обеспечения указанного в условии задачи соотношения количества выпадений орла к количеству выпадений решки общее количество бросков монеты N должно быть кратно 20, т.к.:

$$N = 11n + 9n = (11 + 9)n = 20n$$

После анализа исходных данных можно приступить к выбору пути решения задачи. Черета бросков монеты описывается биномиальным распределением: $Y \sim \text{Bin}(N, p)$. Один из возможных вариантов решения задачи — использовать центральную предельную теорему для аппроксимации этого распределения нормальным распределением:

$$\text{Bin}(N, p) \approx N(Np, Npq),$$

где $Np = 0.55N$ — матожидание, а $Npq = (0.55 * 0, 45)N$ — дисперсия этого распределения. В случае "честной" монеты, у которой вероятность выпадения орла и решки r равны 0.5, матожидание аппроксимирующего распределения имело бы значение $Nr = 0.5N$. Чтобы доказать, что монета "нечестная", число бросков N должно быть таким, чтобы $0.5N$ не попадало в доверительный интервал в окрестности $0.55N$:

$$(0.55N - Z\sqrt{(0.55 * 0, 45)N}; 0.55N + Z\sqrt{(0.55 * 0, 45)N}),$$

где Z — стандартизованная оценка — множитель, определяющий ширину доверительного интервала. В зависимости от необходимого уровня "уверенности" в том, что монета "нечестная", можно выбрать следующие значения Z :

- $Z = 1.6449$ — уровень доверия 90%;
- $Z = 1.9599$ — уровень доверия 95%;
- $Z = 2$ — уровень доверия 95.45% (правило "двух сигм");
- $Z = 3$ — уровень доверия 99.73% (правило "трёх сигм").

Т.к. $0.5N < 0.55N$, то определение необходимого значения N сводится к решению неравенства:

$$\begin{aligned}0.5N &< 0.55N - Z\sqrt{(0.55 * 0, 45)N} \\ 0.05N &> Z\sqrt{(0.55 * 0, 45)N} \\ 0.05^2 N^2 &> Z^2(0.55 * 0, 45)N \\ N &> Z^2 \frac{0.55 * 0, 45}{0.05^2}\end{aligned}$$

С учётом того, что $N = 20n, n \in \mathbb{N}$:

$$N = 20 * \left\lceil Z^2 \frac{0.55 * 0.45}{0.05^2 * 20} \right\rceil$$

In [5]:

```
from math import ceil

for Z, loc in [(1.6449, 0.9), (1.9599, 0.95), (2, 0.9545), (3, 0.9973)]:
    N = 20 * ceil((Z**2.0)*(0.55*0.45)/(0.05**2.0*20))
    print('Z: %-6.6s --> N: %s (уровень доверия: %s)' % (Z, N, loc))
```

```
Z: 1.6449 --> N: 280 (уровень доверия: 0.9)
Z: 1.9599 --> N: 400 (уровень доверия: 0.95)
Z: 2      --> N: 400 (уровень доверия: 0.9545)
Z: 3      --> N: 900 (уровень доверия: 0.9973)
```

В принципе, уровня доверия 90% уже достаточно для того, чтобы отвергнуть гипотезу о том, что монета "честная", поскольку слева от соответствующего доверительного интервала значения распределения возникают только в 5% случаев. Значение 5% является одним из популярных уровней статистической значимости, поэтому можно утверждать, что ответом на поставленный в задаче вопрос является число $N = 280$.