Примеры решения задач

1. Генерация списка чисел.

Для получения списков значений в Python используется концепция **list comprehension** (генератор списков), которая позволяет задать логику, по которой должен быть заполнен список. Например, список из 10 случайных чисел:

```
import random
randomlist = [random.randint(1, 9) for value in range(0, 10)]
print('Первый список:', randomlist)
```

Это же можно сделать при помощи цикла, но нужно учесть, что так

```
randomlist2 = []
for index in range(0, 10):
    randomlist2[index] = random.randint(1, 9)
```

этого сделать **нельзя**. Все из-за того, что список мы изначально назначили пустым, с 0 элементов, а значит, в нем нет элемента с индексом 1, чтобы задать ему значение.

Правильно будет так:

```
randomlist2 = []
for index in range(0, 10):
    randomlist2.append(random.randint(1, 9))
print('Второй список:', randomlist2)
```

Не задавать значение, а добавлять элементы в список. Теперь, когда в списке есть элементы, мы можем их изменять первым способом:

```
for index in range(0, 10):
    randomlist2[index] = random.randint(1, 9)
print('Второй список:', randomlist2)
```

List comprehension позволяет использовать любую функцию для задания элемента, не только **random.randint**.

Например, $f(x) = x^2$, возведение в квадрат:

```
import math
list_of_squares = [math.pow(value, 2) for value in range(0, 10)]
print('Список квадратов:', list_of_squares)
```

или значения логарифма чисел в диапазоне, $f(x) = \lg(x)$:

```
list_of_log = [math.log10(value) for value in range(10, 30)]
print('Логарифмы по основанию 10:', list_of_log)
```

2. Генерация списка чисел.

Кроме того, что можно указывать количество элементов, в **list comprehension** можно также указывать условие, которому значение должно удовлетворять, чтобы попасть в список. Например, список четных чисел в диапазоне [0, 19].

```
odd_list = [value for value in range(1, 20) if value % 2 == 0]
print('Список четных:', odd_list)
```

Это то же самое, что и конструкция:

```
odd_list2 = []
for value in range(1, 20):
    if value % 2 == 0:
        odd_list2.append(value)
print('Список четных 2:', odd_list2)
```

3. Обработка элементов списка.

Дан список из 20 случайных чисел. Переписать его элементы в новый список следующим образом: сначала все четные, затем все нечетные.

```
import random
a = [random.randint(1, 10) for i in range(10)]
print(a)
b = [i for i in a if i%2 == 0] + [i for i in a if I % 2 != 0]
print(b)
```

4. Заполнение и вывод двумерного списка при помощи циклов.

Заполнить список из 7 вложенных списков, в каждом по 7 элементов в шахматном порядке.

```
n = 7
a = []
k = 1
for i in range(n):
    b = []
    for j in range(n):
        b.append(k)
        if k == 0:
            k = 1
        else:
            k = 0
    a.append(b)
for i in range(len(a)):
    for j in range(len(a[i])):
        print(a[i][j], end=' ')
    print()
```

5. Заполнение двумерного списка при помощи **list comprehension**.

Создать матрицу (таблицу) при помощи **list comprehension** также можно. Например, матрица 3x3:

```
matrix = [[elem for elem in range(0, 3)] for lines in range(0, 3)]
print(matrix)
```

Другой вариант вывода:

```
for line in matrix:
    print(line)
```

Также можно ввести значения вручную построчно. Считываем строку и разбиваем ее на элементы (**split**) по пробелам. Поскольку **split** возвращает список, по нему можно пройти при помощи **for**:

```
matrix2 = [[elem for elem in input().split(' ')] for lines in range(0, 3)]
for line in matrix2:
    print(line)
```

6. Особенности хранения вложенных списков.

Возьмем список из 2-х элементов.

```
list = ['a', 'b']
print(list)
```

```
['a', 'b']
```

Сделаем список из этих элементов, повторяющихся 4 раза.

```
list_multiplied = list * 4
print(list_multiplied)
```

```
['a', 'b', 'a', 'b', 'a', 'b', 'a', 'b']
```

Изменим 1-й элемент в полученном списке.

```
list_multiplied[0] = 'c'
print(list_multiplied)
```

```
['c', 'b', 'a', 'b', 'a', 'b', 'a', 'b']
```

Теперь попробуем сделать список списков из нашего **list**.

```
list_of_lists = [list] * 4
print(list_of_lists)
```

```
[['a', 'b'], ['a', 'b'], ['a', 'b'], ['a', 'b']]
```

В полученном списке изменим значение 1-го элемента одного из вложенных списков.

```
list_of_lists[0][0] = 'p'
print(list_of_lists)
```

```
[['p', 'b'], ['p', 'b'], ['p', 'b'], ['p', 'b']]
```

Результат объясняется особенностями хранения данных в памяти. Переменные с одинаковыми данными хранятся в одной ячейке памяти.

```
print(list_of_lists[0] is list_of_lists[1])
```

```
True
```

С точки зрения списка **list_of_lists** ничего не менялось, его элементы по-прежнему ссылаются на одну и ту же ячейку, а значения в этой ячейке изменились:

```
print(list)
```

```
['p', 'b']
```

7. Где использовать кортежи.

Технически разница между кортежами и списками в том, что кортежи после создания нельзя изменять. Засчет этого все операции с кортежами выполняются быстрее, чем со списками, ведь они хранятся как единый фиксированный объект в памяти.

Однако смысловая разница в том, что списки часто используют для хранения однородных элементов, а кортежи – для разнородных.

Например, хранение событий и дат:

```
python_birth_date = ('Python', 20, 'февраля', 1991)
print('Дата рождения:', python_birth_date)
```

Если мы решим записать даты рождения других языков, то у нас получатся такие же структуры:

```
basic_birth_date = ('Basic', 1, 'мая', 1964)
print('Дата рождения:', basic_birth_date)
```

Можно менять тип данных отдельных элементов, сохраняя смысл всей структуры:

```
perl_birth_date = ('Perl', 18, 12, 1987)
java_birth_date = ('Java', 23, 'мая', 1995)
php_birth_date = ('PHP', 8, 'июня', 1995)
javascript_birth_date = ('JavaScript', 4, 12, 1995)
go_birth_date = ('Go', 10, 11, 2009)
print(perl_birth_date)
print(java_birth_date)
print(php_birth_date)
print(javascript_birth_date)
print(go_birth_date)
```

Даже для тех случаев, когда точная дата неизвестна, можно сохранять структуру:

```
pascal_birth_date = ('Pascal', '', '', 1970)
c_birth_date = ('C', None, None, 1972)
print(pascal_birth_date)
print(c_birth_date)
```

Все это свершившиеся факты, нет смысла их изменять, тем более, что-то удалять. Но новые языки продолжают появляться, а значит, из этих событий можно составить список:

```
language_birth_dates = [python_birth_date, basic_birth_date,
                        perl_birth_date, java_birth_date,
                        php_birth_date, javascript_birth_date,
                        go_birth_date, pascal_birth_date,
                        c_birth_date]
print(language_birth_dates)
```

Заметьте, что сами по себе события – это однородные элементы.

Список можно пополнять:

```
language_birth_dates.append( ('C++', None, None, 1983) )
print(language_birth_dates)
```

Другой пример – полки, со стоящими на них в случайном порядке банками с вареньями и соленьями. Для их описания очень подходят кортежи, особенно если полки поделены на ячейки, а на банках наклейки:

```
first_shelf = ('Клубника', 'Малина', 'Огурцы', 'Огурцы', 'Помидоры')
```

Полки бывают разной длины. Под рукой оказались такие доски:

```
second_shelf = ('Малина', 'Вишня', 'Вишня с косточками', 'Яблоки')
third_shelf = ('Яблочный компот', 'Кабачковая икра', 'Помидоры', 'Разносол')
```

Такие полки заполняются банками в сезон, а в остальное время опустошаются только сами банки, значит из этих полок тоже можно сделать кортеж кортежей:

```
shelves = (first_shelf, second_shelf, third_shelf)
print(shelves)
```

И часто можно услышать что-то подобное (нумерация с нуля):

```
print('На первой полке вторая слева:', shelves[0][1])
print('На последней полке третья справа:', shelves[-1][-3])
```