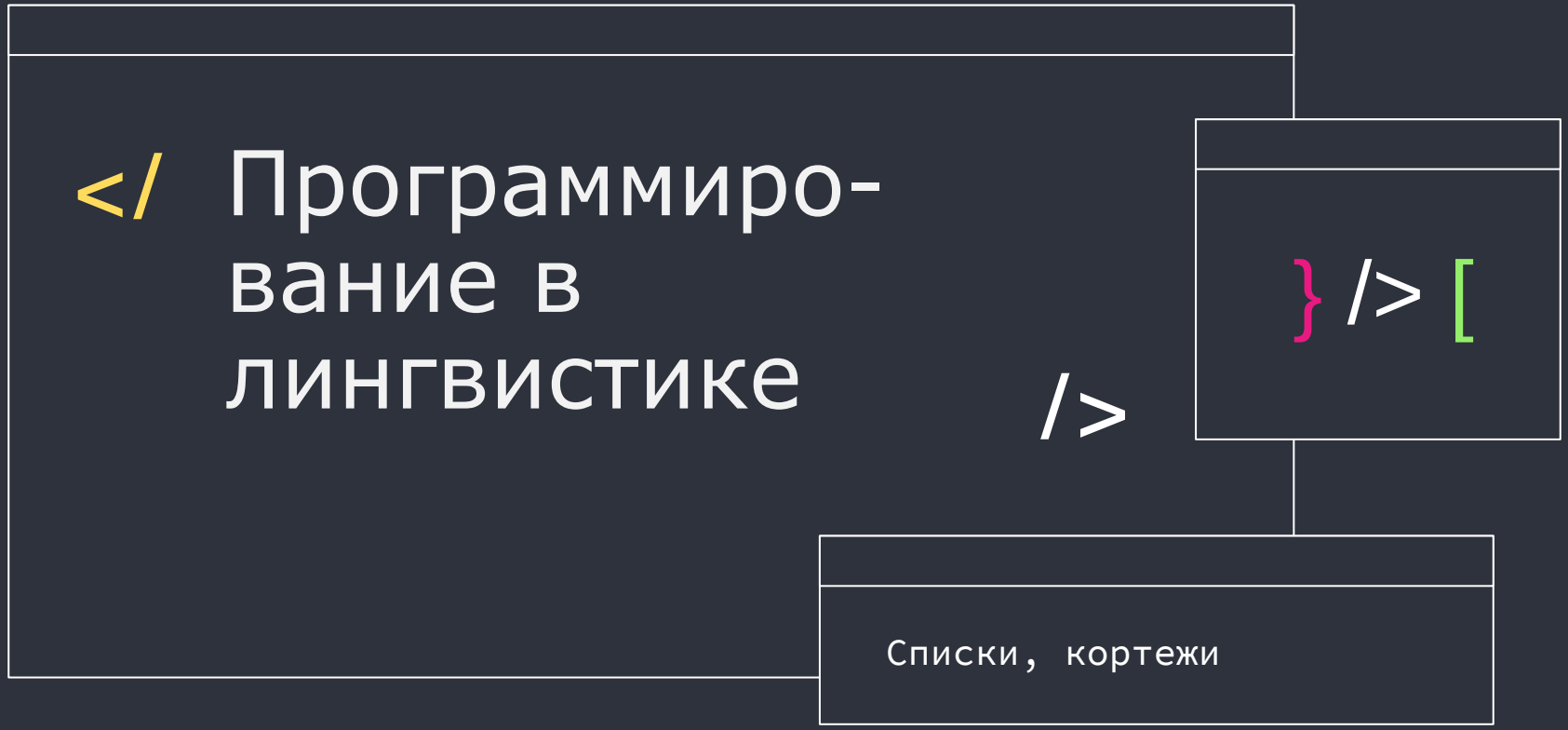


1 0 1 1 0 1 1 0 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1

</ Типы данных Python

Итерируемые	Неитерируемые
Iterable	Non-iterable
строки, списки, словари и т.д.	числа, bool, None

Вызываемые	Невызываемые
Callable	Non-callable
Функции, методы, объекты классов	Все прочее

Изменяемые	Неизменяемые
Mutable	Immutable
списки, словари, множества и т.д.	строки, числа, кортежи, bool, None

Хэшируемые	Нехэшируемые
Hashable	Non-hashable
Все неизменяемое + классы с __hash__	Все прочее

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Списки в Python

```
ints = [1, 2, 3]          # Список int
strs = ['1', '2', '3']    # Список str
lists = [[1], [2], [3]]   # Список списков
mixed = [1, "2", [3]]     # Смешанный список
```

Список (list) – итерируемый изменяемый объект.

* Все элементы в списке индексируются (как в строке).
* К спискам применимы срезы.
* Список может включать в себя объекты любых типов.
* Список может состоять из объектов разных типов.
* Порядок элементов в списке строго определен.
* В списке могут быть повторяющиеся элементы.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Как создать список?

1. Прямо записать его.

```
lst1 = [1, 2, 3]
lst2 = ["a", "b", "c"]
lst3 = [1, "a", 3]
```

2. Добавлять в пустой список элементы с помощью append().

```
lst = []
for i in range(1, 4):
  lst.append(i)
lst
```

```
[1, 2, 3]
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Как создать список?

3. Преобразовать в список другой итерируемый объект.

```
lst = list(range(1, 4))
lst
```

```
[1, 2, 3]
```

Range() -> list of ints

```
lst = list("123")
lst
```

```
['1', '2', '3']
```

str -> list of strings

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Как создать список?

4. Заранее проинициализировать список нулями (дефолтными значениями) и заполнить его в цикле for.

```
lst = [0] * 5
for i in range(5):
  lst[i] = i * 2
lst
```

```
[0, 2, 4, 6, 8]
```

<- Заменяет элементы в списке из пяти нолей на удвоенные числа в диапазоне от 0 до 4 (эти числа в цикле также используются в качестве индексов списка)

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Как создать список?

5. С помощью list comprehension (генератора списков).

```
a = "1234"
lst = [int(i) for i in a if i != "2"]
lst
```

```
[1, 3, 4]
```

<- Создает из строки список чисел, пропуская двойки

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ List comprehension

```
lst = [func(i) for i in x if i]
```

Генератор списков – собранный в одну строку упрощенный цикл for, результатом выполнения которого будет являться список.

Цикл for:

```
a = "1234"
lst = []
for i in a:
  if i != "2":
    lst.append(int(i))
```

Генератор списков:

```
a = "1234"
lst = [int(i) for i in a if i != "2"]
```

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ List comprehension

```
lst = [int(sym) for sym in a if sym != "2"]
```

> lst – название будущего списка.

> int(sym) – функция, применяемая к каждому элементу итерируемого объекта. Можно обойтись без применения функций.

> for sym in a – «для каждого элемента (sym) в объекте a».

> If sym != “2” – условие, позволяющее пропускать символы, которые равны символу “2”. Проверка может отсутствовать.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Пример кода

```
lst = list(range(1, 11))
squares = [x ** 2 for x in lst]
print('Squares are:', *squares)
odds = [x for x in lst if x % 2]
print('Odds are:', *odds)
```

```
Squares are: 1 4 9 16 25 36 49 64 81 100
Odds are: 1 3 5 7 9
```

1. Мы создаем список из чисел от 1 до 10 с помощью range().
2. Мы получаем квадраты этих чисел, применяя операцию к каждому элементу.
3. Мы получаем только нечетные числа, проверяя остаток от деления каждого элемента списка.

Оператор * означает распаковку. *odds будет передавать каждый элемент по отдельности, то есть:
print(*odds) = print(odds[0], odds[1], odds[2], odds[3], odds[4], odds[5])

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Операции со списками

```
lst = [1, 2, 3] + [4, 5, 6]
lst
```

```
[1, 2, 3, 4, 5, 6]
```

```
lst = ["a", "h"] * 3
lst
```

```
['a', 'h', 'a', 'h', 'a', 'h']
```

Списки можно складывать и умножать на число.

Результат такой же, как и при сложении и умножении строк.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Операции со списками

```
lst = [2, 3, "cat", ["a"]]
lst[0] *= 2
lst[1] = lst[0] - lst[1]
lst[2] = lst[2].replace("c", "r")
lst[3].append("b")
lst
```

```
[4, 1, 'rat', ['a', 'b']]
```

К элементу списка можно применять операции, функции и методы, которые применимы к типу данных этого элемента.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Добавление элемента в список

```
lst = [1, 2, 3]
lst.append(4)
lst.append("5")
lst.append([6, 7])
lst
```

```
[1, 2, 3, 4, '5', [6, 7]]
```

list.append() позволяет добавить один элемент в конец списка.

Если мы передаем туда список, то он не расширяет текущий, а добавляется в качестве элемента.

Получается список внутри списка.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Удаление элемента из списка

```
lst = [1, 2, 3, 4, 5]
del lst[0] # с помощью del
lst[0:1] = [] # с помощью срезов
elem = lst.pop(0) # с помощью pop
print(elem) # pop вернул удаленный элемент
lst.remove(4) # с помощью remove (по значению)
lst
```

```
3
[5]
```

Можно удалять элементы по индексам: del, срезы и pop.

Метод .pop() не просто удаляет элемент, но и возвращает его (можно положить в переменную).

Для удаления по значению есть метод .remove(). Он удаляет первое вхождение элемента. Если не находит – ValueError.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Вставка элемента в список

```
lst = ["a", "c"]
lst.insert(1, "b")
lst
```

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

list.insert(index, elem) – метод списков для вставки элемента по индексу.

```
lst = ["a", "c"]
lst[1:1] = ["b"]
lst
```

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

list[index:index] = [elem] – вставка с помощью срезов. Можно добавить сразу несколько элементов.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Подсчет элемента в списке

```
lst = [1, 1, 2, 3, 1]
print(lst.count(1))
lst = ["a", "a", "b"]
print(lst.count("a"))
```

```
3
2
```

list.count(elem)

Метод count позволяет подсчитать количество повторов элемента в списке.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Поиск элемента в списке

```
lst = [1, 2, 3, 4]
print(lst.index(3))
lst.index(8)
```

```
2
---------------------------
ValueError
```

list.index(elem)

Метод .index() позволяет получить индекс элемента в списке. Если не найдено, то возвращается ValueError.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Расширение списка

```
lst = [1, 2, 3]
lst += [4, 5]
lst
```

```
[1, 2, 3, 4, 5]
```

```
lst = [1, 2, 3]
lst.extend([4, 5])
lst
```

```
[1, 2, 3, 4, 5]
```

list1.extend(list2) – позволяет вставить элементы из второго списка в конец первого. Работает так же, как и оператор +, только делает это in-place (сразу, без необходимости класть результат в переменную).

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Методы строк .split() и join()

```
lst = "cat dog ball".split()
lst
```

```
['cat', 'dog', 'ball']
```

```
lst = "cat, dog, ball".split(", ")
lst
```

```
['cat', 'dog', 'ball']
```

Метод строки .split() позволяет создать список из строки по указанному разделителю. По умолчанию это пробельный символ.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Методы строк .split() и join()

```
lst = ["a", "b", "c"]
" ".join(lst)
```

```
'a b c'
```

```
lst = ["Коля", "Витя", "Саша"]
" и ".join(lst)
```

```
'Коля и Витя и Саша'
```

Метод строки .join() позволяет соединить список в строку, используя указанный разделитель.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Функции, применимые к спискам

Функция	Результат	Описание
len([1, 2, 3])	3	Возвращает длину списка (сколько в нем элементов)
min([2, 1, 3])	1	Возвращает самый маленький элемент
max([1, 3, 2])	3	Возвращает самый большой элемент
sum([1, 2, 3])	6	Возвращает сумму всех элементов (если все элементы - числа)

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Сравнение списков

[1, 2, 3] < [3, 4, 5]

[1, 2, 3] < [1, 3, 4]

[1, 1, 3] < [1, 1, 4]

[1, 1, 1] == [1, 1, 1]

Списки сравниваются поэлементно, как строки. Неравенство появляется при первом же расхождении между значениями элементов списков.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Копирование списков

```
x = 5
y = x
x += 1
print(x, y)
```

```
6 5
```

Числа (неизменяемый объект)

```
lst1 = [1, 2]
lst2 = lst1
lst2.append(3)
print(lst1, lst2)
```

```
[1, 2, 3] [1, 2, 3]
```

Списки (изменяемый объект)

Если объект **изменяемый**, то Python при присвоении этого объекта другой переменной создаст **ссылку** на первоначальный объект. Изменив мнимую копию, изменится и **оригинальный объект**.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Копирование списков

```
lst1 = [1, 2]
lst2 = lst1[:]
lst2.append(3)
print(lst1, lst2)
```

```
[1, 2] [1, 2, 3]
```

Копию списка можно сделать с помощью пустого среза:
list1 = list2[:]

```
lst1 = [1, 2]
lst2 = lst1.copy()
lst2.append(3)
print(lst1, lst2)
```

```
[1, 2] [1, 2, 3]
```

Либо с помощью метода .copy():
list1 = list2.copy()

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Сортировка списков

```
lst = [1, 4, 3, 2]
lst.sort()
lst
```

```
[1, 2, 3, 4]
```

С помощью in-place метода .sort(). In-place методы возвращают None, если их результат положить в переменную.

```
lst = [1, 4, 3, 2]
lst = sorted(lst)
lst
```

```
[1, 2, 3, 4]
```

С помощью функции sorted(). Это не in-place функция, так что ее результат надо куда-то класть. Применима к любым итерируемым объектам, но возвращает список.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Сортировка списков: key

```
lst1 = 'apple bin cell'.split()
print(lst1)
lst2 = sorted(lst1, key=len)
print('B:', *lst2)
lst1.sort(key=len, reverse=True)
print('A:', *lst1)
```

```
['apple', 'bin', 'cell']
B: bin cell apple
A: apple cell bin
```

И у функции, и у метода сортировки есть параметр key=. Он может принимать любую функцию, которая принимает элемент списка и возвращает число (можно и символ, тогда по умолчанию будет ord()). В зависимости от return'а сортируются и элементы в списке.

Здесь ключом для сортировки является длина слова (len).

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Обращение списков

```
lst = [1, 2, 3]
lst.reverse()
lst
```

```
[3, 2, 1]
```

```
lst = [1, 2, 3]
lst = reversed(lst)
print(lst)
print(list(lst))
```

```
<list_reverseiterator object at 0x7f2301b5b4c0>
[3, 2, 1]
```

Перевернуть список можно с помощью in-place метода .reverse().

Либо с помощью функции reversed(), которая возвращает reverseiterator. Его можно легко превратить в список с помощью list().

Не забываем про срезы!
lst[::-1]

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Кортежи (tuple)

```
tpl1 = (1, 2, 3)
tpl2 = 1, 2, 3
print(type(tpl1), type(tpl2))
```

```
<class 'tuple'> <class 'tuple'>
```

Кортежи (tuple) похожи на списки, но, в отличие от последних, неизменяемы.

```
tpl1[0] = 5
```

```
---------------------------
TypeError
```

Кортежи являются hashable объектом.

Они хороши, когда нам нужен неизменяемый набор констант.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Множественное присваивание

```
x, y = 4, 5 # присвоит 4 в x, а 5 в y
x = 4, 5 # сделает в x кортеж (4, 5)
x, y = [1, 2] # присвоит 1 в x, а 2 в y
x, y = 4 # вызовет ошибку!
```

Кортежи и списки могут использоваться для множественного присваивания: мы можем одной строкой задать значения переменных через запятую. Первый элемент кладется в первую переменную, второй – во вторую и т.д.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Множественное присваивание

```
lst = []
*lst, x = 1, 2, 3, 4
print('lst:', lst, 'x:', x)
```

```
lst: [1, 2, 3] x: 4
```

С помощью множественного присваивания можно создавать списки.

Здесь приведен загадочный пример, где 1, 2 и 3 кладутся в список lst, а 4 – в x.

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Множественное присваивание

```
x = 4
y = 5
x, y = y, x
print(x, y)
```

```
5 4
```

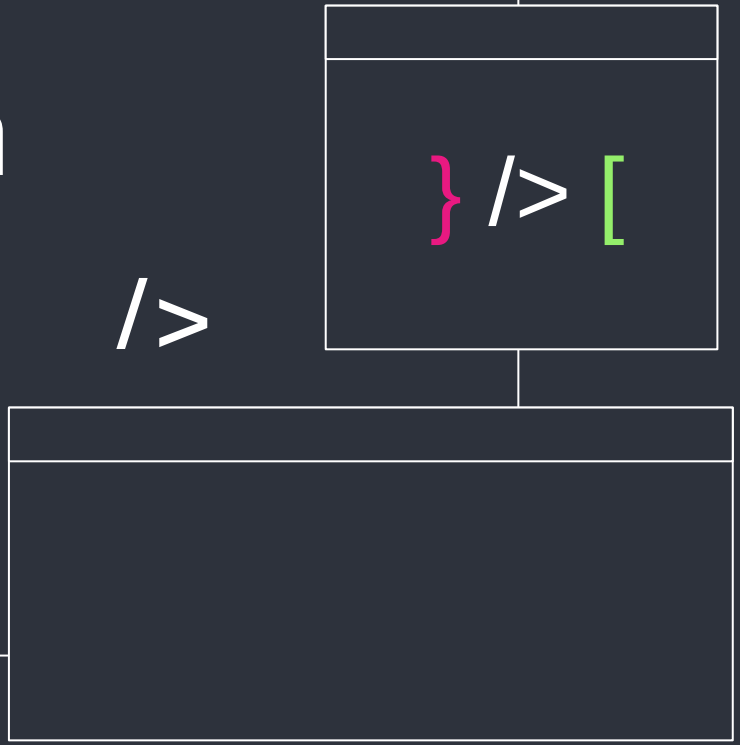
С помощью множественного присваивания можно менять переменные местами.

Очень удобно!

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Спасибо за внимание!

admvereshchagina@gmail.com



1 0 1 1 0 1 1 0 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 1