



ЧТО БУДЕТ НА ЭКРАНЕ?



otus

Начнем в **20:01** [МСК]

# ML Basic

## Функции

otus.ru

```
for k in range(3):  
    for i in range(6+2*k):  
        print(1, end='')  
        for j in range(i):  
            print(0, end='')  
        print()  
print('1\n1\n1')
```

За что?



Меня не видно?  
Не слышно?  
Запись не ведется?



Пишите в чат  
или кричите



ЧТО БУДЕТ НА ЭКРАНЕ?



otus

Начнем в **20:01** [МСК]

# ML Basic

## Функции

otus.ru

```
for k in range(3):  
    for i in range(6+2*k):  
        print(1, end='')  
        for j in range(i):  
            print(0, end='')  
        print()  
print('1\n1\n1')
```

За что?



# Что будет на экране?



```
for k in range(3):  
    for i in range(6+2*k):  
        print(1, end='')  
        for j in range(i):  
            print(0, end='')  
        print()  
print('1\n1\n1')
```





# Что будет на экране?

**Непечатаемые знаки** — условные символы форматирования, которые не отображаются при печати.

```
for k in range(3):  
    for i in range(6+2*k):  
        print(1, end='')  
        for j in range(i):  
            print(0, end='')  
        print()  
print('1\n1\n1')
```



# А так?



`line(n)` выводит строку из n нулей

```
line(3)
000
|
```

```
for k in range(3):
    for i in range(6+2*k):
        print(1, end='')
        line(i)
        print()
print('1\n1\n1')
```



# А так?



`triangle(n)` ВЫВОДИТ  
треугольник высотой `n`

```
triangle(4)
1
10
100
1000
|
```

```
for k in range(3):
```

```
    triangle(6+2k)
```

```
    print('1\n1\n1')
```





OTUS | ОНЛАЙН ОБРАЗОВАНИЕ 



# Built-In функции

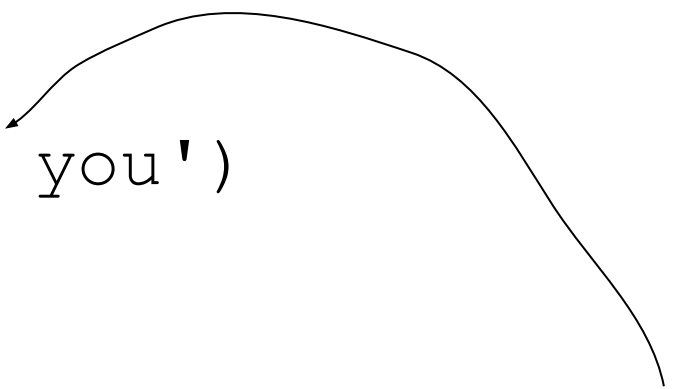
# Built-In

```
print('I like you')
```

<https://python-reference.readthedocs.io/en/latest/docs/functions/print.html>

# Built-In

```
print('I like you')
```

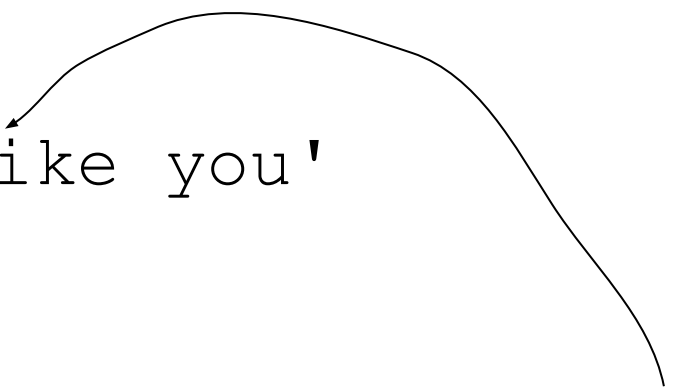


**Аргументы** — это конкретные значения или данные, передаваемые функции при ее вызове.

<https://python-reference.readthedocs.io/en/latest/docs/functions/print.html>

# Built-In

```
phrase = 'I like you'  
print(phrase)
```



**Аргументы** — это конкретные значения или данные, передаваемые функции при ее вызове.

Аргументы могут быть константами, переменными, выражениями или даже результатами других функций.

<https://python-reference.readthedocs.io/en/latest/docs/functions/print.html>

# Built-In

```
print(input())
```



**Аргументы** — это конкретные значения или данные, передаваемые функции при ее вызове.

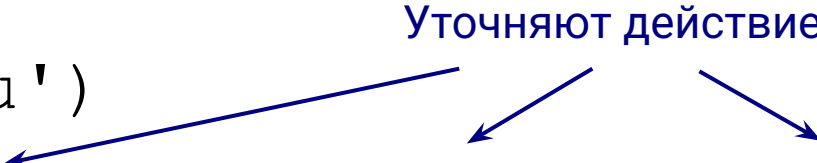
Аргументы могут быть константами, переменными, выражениями или даже результатами других функций.

<https://python-reference.readthedocs.io/en/latest/docs/functions/print.html>

# Параметры

Уточняют действие

```
print('I like you')  
print('I like you', flush=True, end='')  
print('I like you', end='', flush=True)
```



<https://python-reference.readthedocs.io/en/latest/docs/functions/print.html>

# Параметры

```
print('I like you')  
print('I like you', flush=True, end='')  
print('I like you', end='', flush=True)
```

Уточняют действие

Позиционный параметр

Именованный параметр

<https://python-reference.readthedocs.io/en/latest/docs/functions/print.html>

# Пользовательские функции



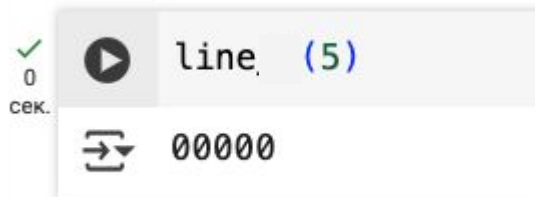
# Своя функция!



`line(n)` выводит строку из `n` нулей

```
line(3)
000
|
```

`line(5)`



# Своя функция!

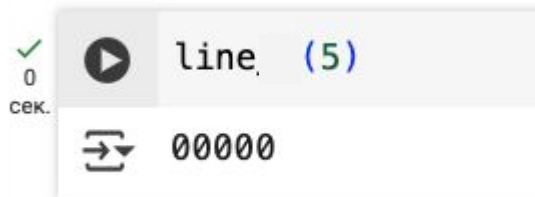


`line(n)` выводит строку из `n` нулей

```
line(3)
000
|
```

```
def line(n):
    values = ["0" for _ in range(n)]
    line = "".join(values)
    print(line)
```

```
line(5)
```



# Своя функция!

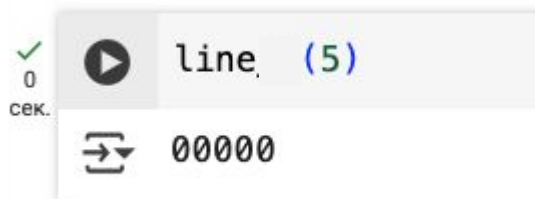


`line(n)` выводит строку из `n` нулей

```
line(3)
000
|
```

```
def line(n):
    for _ in range(n):
        print(0, end="")
```

```
line(5)
```



# Своя функция!

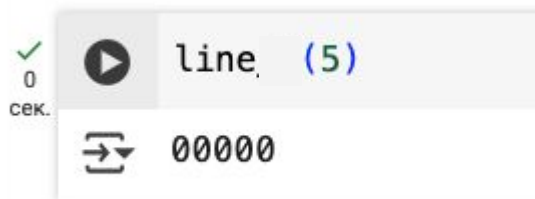


`line(n)` выводит строку из `n` нулей

```
line(3)
000
|
```

```
def line(n):
    for _ in range(n):
        print(0, end="")
```

```
line(5)
```



# Своя функция!

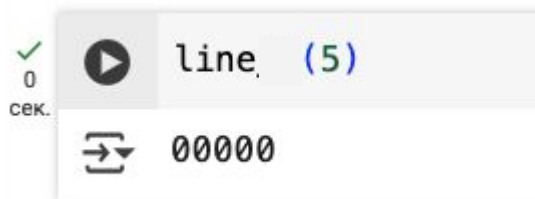


`line(n)` выводит строку из `n` нулей

```
line(3)
000
|
```

```
def line(n, sep):
    for _ in range(n):
        print(0, end=sep)
```

```
line(5, "+")
```



# Своя функция!



`line(n)` выводит строку из `n` нулей

```
line(3)
000
|
```

```
def line(n, sep):
    for _ in range(n):
        print(0, end=sep)
```

```
line(5, "+")
```

✓  
0  
сек.

```
▶ def line(n, sep):
    for _ in range(n):
        print(0, end=sep)
```

```
line(5, "+")
```

⇒ 0+0+0+0+0+

# Своя функция!



Параметры

```
def line(n, sep):
```

```
    for _ in range(n):  
        print(0, end=sep)
```

Тело функции

`line(n)` выводит строку из `n` нулей

```
line(3)  
000  
|
```

```
line(5, "+")
```

Аргументы

```
0 сек. ▶ def line(n, sep):  
          for _ in range(n):  
              print(0, end=sep)  
  
          line(5, "+")  
⇒ 0+0+0+0+0+
```

# Своя функция!



Параметры

```
def line(bober, sep):
```

```
    for _ in range(bober):  
        print(0, end=sep)
```

Тело функции

`line(n)` выводит строку из `n` нулей

```
line(3)  
000  
|
```

```
line(5, "+")
```

Аргументы

```
0 сек. ▶ def line(n, sep):  
    for _ in range(n):  
        print(0, end=sep)  
  
line(5, "+")  
⇒ 0+0+0+0+0+
```



# Своя функция!



Параметры

```
def line(bober, sep):
```

```
    for _ in range(bober):  
        print(0, end=sep)
```

Тело функции

`line(n)` выводит строку из `n` нулей

```
line(3)  
000  
|
```

`x = 5`

`line(x, "+")`

Аргументы

```
0 сек. ▶ def line(n, sep):  
          for _ in range(n):  
              print(0, end=sep)  
  
          line(5, "+")  
⇒ 0+0+0+0+0+
```



# Своя функция!

Параметры

```
def line(bober, sep=""):
```

```
    for _ in range(bober):  
        print(0, end=sep)
```

Тело функции

`line(n)` выводит строку из `n` нулей

```
line(3)  
000  
|
```

`x = 5`

```
line(x, sep="+")
```

Аргументы

0 сек.

```
def line(n, sep):  
    for _ in range(n):  
        print(0, end=sep)
```

```
line(5, "+")
```

```
0+0+0+0+0+
```

# Своя функция!



Параметры

```
def line(bober, sep=""):
```

```
    for _ in range(bober):  
        print(0, end=sep)
```

Тело функции

```
x = 5
```

```
line(x)
```

Аргументы

`line(n)` выводит строку из `n` нулей

```
line(3)  
000  
|
```

0  
сек.

```
def line(n, sep):  
    for _ in range(n):  
        print(0, end=sep)
```

```
line(5, "+")
```

```
0+0+0+0+0+
```

# Своя функция!



`triangle(n)` Выводит  
треугольник высотой `n`

```
triangle(4)
1
10
100
1000
|
```

```
def line(bobber, sep):
    for _ in range(bobber):
        print(0, end=sep)
```

```
x = 5
line(5, "+")
triangle(4)
```

# Своя функция!



`triangle(n)` ВЫВОДИТ  
треугольник высотой `n`

```
triangle(4)
1
10
100
1000
|
```

```
def line(bober, sep):
    for _ in range(bober):
        print(0, end=sep)
```

```
def triangle(rows):
    for i in range(rows):
        print(1, end='')
        line(i, "")
        print()
```

```
triangle(4)
```

# Синтаксис функций





# Синтаксис

---

```
def function_name(parameters):  
    pass
```

- `def` – ключевое слово, обозначающее начало функции.
- `function_name` – имя функции, которое должно быть описательным.
- `parameters` – переменные, передаваемые в функцию. Параметры указываются в круглых скобках, и они могут быть пустыми.
- `pass` – “реализую потом”

# Разминка

---



<https://ahaslides.com/XU603>





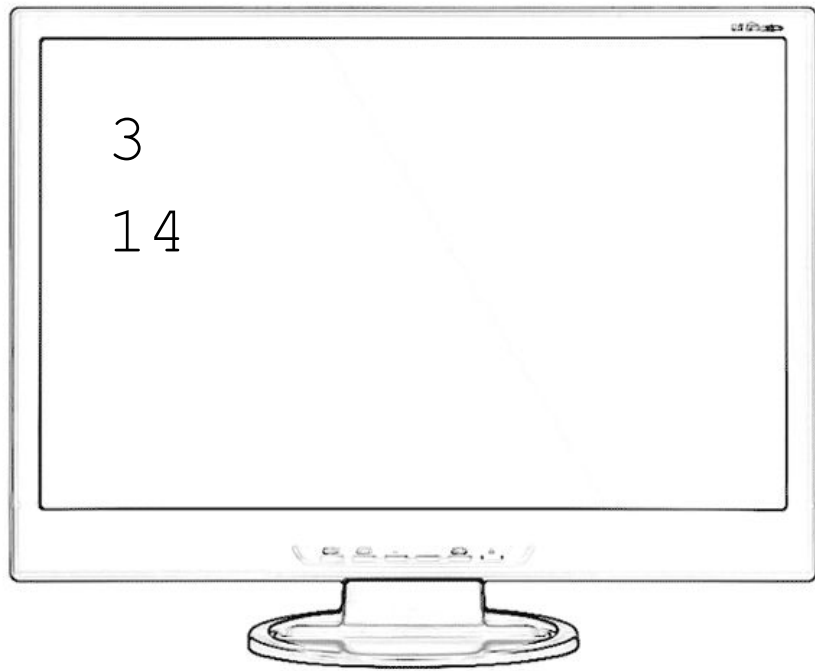
**Вопрос 2**

# Вопрос 1



Что будет храниться в `c` после выполнения кода?

```
a = int(input())  
b = int(input())  
c = a + b
```



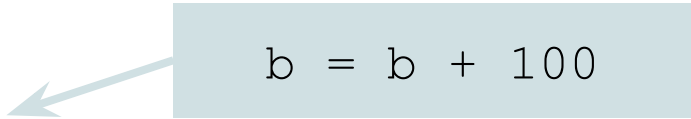
# Приведение типов

---

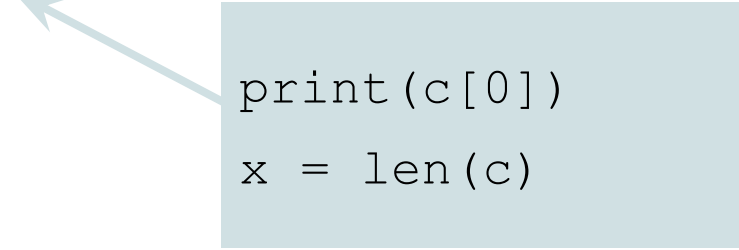
```
a = '123'
```

```
b = int(a)
```

```
c = str(b)
```



```
b = b + 100
```



```
print(c[0])
```

```
x = len(c)
```

# Считывание

---

```
s = input()
```

```
n = int(input())
```

```
print(n%2)
```

# Built-In функции

---

```
s = input()
```

```
n = int(input())
```

```
print(n%2)
```

# Чем отличается `print()` ?

---

```
s = input()
```

```
n = int(input())
```

```
print(n%2)
```

# Своя функция!

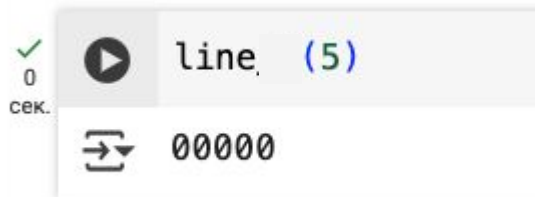


`line(n)` выводит строку из `n` нулей

```
line(3)
000
|
```

```
def line(n):
    values = ["0" for _ in range(n)]
    res = "".join(values)
    print(res)
```

```
line(5)
```



# Своя функция!

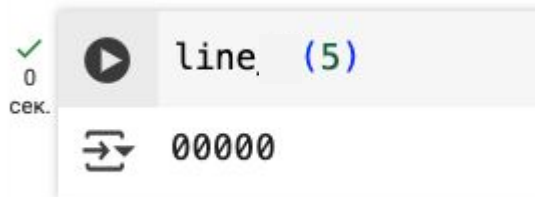


`line(n)` выводит строку из `n` нулей

```
line(3)
000
|
```

```
def line(n):
    values = ["0" for _ in range(n)]
    res = "".join(values)
    return res
```

```
result = line(5)
y = f(x)
```





# Своя функция!

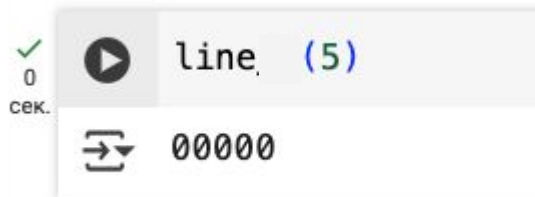


`line(n)` выводит строку из `n` нулей

```
line(3)
000
|
```

```
def line(n):
    values = ["0" for _ in range(n)]
    line = "".join(values)
    return line
```

```
result = line(5)
print(result)
```



# Оператор return

# Своя функция!



```
def line(n):  
    values = ["0" for _ in range(n)]  
    res = "".join(values)  
    print(res)
```

```
line(5)
```

```
def line(n):  
    values = ["0" for _ in range(n)]  
    res = "".join(values)  
    return res
```

```
result = line(5)  
print(result)
```

# Своя функция!

---



```
def nedo_square(n):  
    return n ** 2 - 1  
  
result = nedo_square(5)  
print(result)
```



# Синтаксис

---

```
def function_name(parameters):  
    return value
```

- `def` – ключевое слово, обозначающее начало функции.
- `function_name` – имя функции, которое должно быть описательным.
- `parameters` – переменные, передаваемые в функцию. Параметры указываются в круглых скобках, и они могут быть пустыми.
- `pass` – “реализую потом”
- `return` – оператор, который возвращает значение из функции. Если `return` не указан, функция возвращает `None`.



# Синтаксис

```
[ ] # Код без использования функций
a = 5
b = 10
print(a + b)

c = 7
d = 3
print(c + d)

e = 1
f = 9
print(e + f)
```



15  
10  
10

```
[ ] # Код с использованием функции
def add_numbers(x, y):
    return x + y

print(add_numbers(5, 10))
print(add_numbers(7, 3))
print(add_numbers(1, 9))
```



15  
10  
10

# Про “хороший код”

# Какой код лучше?

---

Критерий	Как оценить?	Как измерить?
<a href="#">Работает ожидаемо?</a>	Формальная верификация	Тесты
<a href="#">Работает долго?</a>	Выч. сложность; Трудоемкость	Running time; Wall-clock time
<a href="#">Занимает много памяти?</a>	Емкостная сложность	Профилирование
<a href="#">Сложно реализовывать?</a>	Цикломатическая сложность	Time tracking
<a href="#">Сложно читать?</a>	Когнитивная сложность	Оценка на ревью



# Зачем читать?

---

1. Корректность
2. Развивать
3. Эффективность
4. Расход памяти (утечки, оптимизация)

*“Код пишут один раз,  
а читают  
многократно”*

# Зачем читать?

---

1. Корректность
2. Развивать
3. Эффективность
4. Расход памяти (утечки, оптимизация)

*“Важно писать не код без ошибок, а такой код, в котором ошибки сложно допустить и легко найти”*

# Вариант 1

---

1. **Code review** — это проверка кода не его разработчиком
2. **Рефакторинг** – процесс улучшения структуры, качества и читаемости кода без изменения его функциональности (чистка и упрощение)

# Варианты

- Code review
- LLM
- Статический анализ



Ekaterina Dmitrieva

Today

Привет!

Оцени, пожалуйста, читаемость этого кода, написанного на Python:

```
python
for i in range(n - 1):
    for j in range(n - 1 - i):
        if values[j] > values[j + 1]:
            values[j], values[j + 1] = values[j + 1], values[j]
```



ChatGPT + Midjourney

Привет!

■ Читаемость кода

Код, который ты предоставил, выглядит достаточно читаемым, но можно улучшить его восприятие.

1. Имена переменных: Лучше использовать более понятные имена вместо `i`, `j`, и `values`, чтобы было понятно, что они означают.
2. Комментарий: Добавь комментарий, объясняющий, что именно делает этот код (например, сортировка).
3. Отступы: Отступы хороши, но можно выравнивать линии, чтобы повысить их наглядность.

Вот улучшенная версия:

```
python
for outer_index in range(n - 1):
    for inner_index in range(n - 1 - outer_index):
        if elements[inner_index] > elements[inner_index + 1]:
            elements[inner_index], elements[inner_index + 1] =
elements[inner_index + 1], elements[inner_index]
```

Такой вариант будет понятнее для других разработчиков.

Ekaterina Dmitrieva

Today

Привет!

Оцени, пожалуйста, читаемость этого кода, написанного на Python:

```
python
for i in range(n - 1):
    for j in range(n - 1 - i):
        if values[j] > values[j + 1]:
            values[j], values[j + 1] = values[j + 1], values[j]
```



ChatGPT + Midjourney

Привет!

### ■ Читаемость кода

Код, который ты предоставил, выглядит достаточно читаемым, но можно улучшить его восприятие.

1. Имена переменных: Лучше использовать более понятные имена вместо `i`, `j`, и `values`, чтобы было понятно, что они означают.
2. Комментарий: Добавь комментарий, объясняющий, что именно делает этот код (например, сортировка).
3. Отступы: Отступы хороши, но можно выровнять линии, чтобы повисить их наглядность.

Вот улучшенная версия:

```
python
for outer_index in range(n - 1):
    for inner_index in range(n - 1 - outer_index):
        if elements[inner_index] > elements[inner_index + 1]:
            elements[inner_index], elements[inner_index + 1] =
elements[inner_index + 1], elements[inner_index]
```

Такой вариант будет понятнее для других разработчиков.

# Варианты

---

- Code review
- LLM
- Статический анализ

# Фредерик Брукс

---

- **Essential complexity** – необходимая сложность, которая определяется самой проблемой и никак не может быть удалена.
- **Accidental complexity** – непреднамеренная сложность, которая добавляется программистами во время проектирования и написания кода



# 2007 год, Sonar Source

- Предложили метрику **КОГНИТИВНОЙ СЛОЖНОСТИ** (Cognitive complexity)

← 4 / 423 issues ↻

Refactor this method to reduce its Cognitive Complexity from 40 to the 15 allowed.

Code Smell +15

1	+1
2	+2 (incl 1 for nesting)
3	+3 (incl 2 for nesting)
4	+4 (incl 3 for nesting)
5	+3 (incl 2 for nesting)
6	+3 (incl 2 for nesting)
7	+1
8	+4 (incl 3 for nesting)
9	+1
10	+5 (incl 4 for nesting)
11	+6 (incl 5 for nesting)
12	+1
13	+4 (incl 3 for nesting)
14	+1
15	+1

alt + ↑ ↓ to navigate issue locations

Refactor this method to reduce its Cognitive Complexity from 40 to the 15 allowed. Why is this an issue? 7 years ago L47

Code Smell Critical Open Not assigned 10min effort Comment No tags

```
48 {
49     string digest = null;
50     StringBuilder typeName = new StringBuilder();
51     IList<int> nestedParameterCounts = genericNameProvider.GetNestedParameterCounts();
52     1 for (int i = 0; i < format.Length; i++)
53     {
54         char ch = format[i];
55         2 if (ch == '{')
56         {
57             i++;
58             int start = i;
59             3 for (; i < format.Length; i++)
60                 4 if (format[i] == '}')
61                     break;
62             5 if (i == format.Length) throw new InvalidDataContractException(String.Format("Generic Name Brace Mismatch",
63             6 if (format[start] == '#' 7 && i == (start + 1))
64             {
65                 8 if (nestedParameterCounts.Count > 1 9 || !genericNameProvider.ParametersFromBuiltInNamespaces)
66                 {
67                     10 if (digest == null)
68                     {
69                         StringBuilder namespaces = new StringBuilder(genericNameProvider.GetNamespaces());
70                         11 foreach (int count in nestedParameterCounts)
71                             namespaces.Insert(0, count).Insert(0, " ");
72                         digest = GetNamespacesDigest(namespaces.ToString());
73                     }
74                 }
75             }
76         }
77     }
```

SonarQube



# 2007 год, Sonar Source

- Предложили метрику **КОГНИТИВНОЙ сложности** (Cognitive complexity)

← 4 / 423 issues ↻

Refactor this method to reduce its Cognitive Complexity from 40 to the 15 allowed.

Code Smell +15

1	+1
2	+2 (incl 1 for nesting)
3	+3 (incl 2 for nesting)
4	+4 (incl 3 for nesting)
5	+3 (incl 2 for nesting)
6	+3 (incl 2 for nesting)
7	+1
8	+4 (incl 3 for nesting)
9	+1
10	+5 (incl 4 for nesting)
11	+6 (incl 5 for nesting)
12	+1
13	+4 (incl 3 for nesting)
14	+1
15	+1

alt + ↑ ↓ to navigate issue locations

Refactor this method to reduce its Cognitive Complexity from 40 to the 15 allowed.

```
48 {
49     string digest = null;
50     StringBuilder typeName = new S
51     IList<int> nestedParameterCou
52     1 for (int i = 0; i < format.Len
53     {
54         char ch = format[i];
55         2 if (ch == '{')
56         {
57             i++;
58             int start = i;
59             3 for (; i < format.Length; i++)
60                 4 if (format[i] == '}')
61                     break;
62             5 if (i == format.Length) throw new InvalidDataContractException(String.Format("Generic Name Brace Mismatch",
63             6 if (format[start] == '#' 7 && i == (start + 1))
64             {
65                 8 if (nestedParameterCounts.Count > 1 9 || !genericNameProvider.ParametersFromBuiltInNamespaces)
66                 {
67                     10 if (digest == null)
68                     {
69                         StringBuilder namespaces = new StringBuilder(genericNameProvider.GetNamespaces());
70                         11 foreach (int count in nestedParameterCounts)
71                             namespaces.Insert(0, count).Insert(0, " ");
72                         digest = GetNamespacesDigest(namespaces.ToString());
73     }
```


По-умолчанию, показателями  
хорошего кода являются:

- 25 (семейство С-языков)
- 15 (все остальные языки)

SonarQube

# Эксперимент

---

A simple line drawing of a stick figure running to the right. The figure has a circular head, a rectangular torso, and two legs. Its right arm is bent with a clenched fist, and its left arm is also bent. A large, light-gray speech bubble with a black outline originates from the figure's mouth. Inside the speech bubble, the word 'Абакан' is written in a black, sans-serif font. The figure is positioned on a thin, wavy horizontal line that represents the ground.

Абакан

# Алгоритм

---

- **Описание:** <https://redirect.sonarsource.com/doc/cognitive-complexity.html>
- Игнорировать структуры языков программирования, позволяющих сокращать написание кода и делать его более читаемым.
- Увеличивать метрику на единицу для каждого оператора, прерывающего поток исполнения кода
  - Циклические конструкции `for`, `while`, `do while`, ...
  - Условные конструкции: `if`, `#if`, `#ifdef`, тернарные операторы
- Увеличивать метрику на единицу в случае вложенных (nested) конструкций

# Пример 1

---

```
for i in range(n - 1):  
    for j in range(n - 1 - i):  
        if values[j] > values[j + 1]:  
            values[j], values[j + 1] = values[j + 1], values[j]
```

# Пример 1

---

```
for i in range(n - 1): +1
    for j in range(n - 1 - i):
        if values[j] > values[j + 1]:
            values[j], values[j + 1] = values[j + 1], values[j]
```

# Пример 1

---

```
for i in range(n - 1): +1
    for j in range(n - 1 - i): +2
        if values[j] > values[j + 1]:
            values[j], values[j + 1] = values[j + 1], values[j]
```

# Пример 1

---

```
for i in range(n - 1): +1
    for j in range(n - 1 - i): +2
        if values[j] > values[j + 1]: +3
            values[j], values[j + 1] = values[j + 1], values[j]
```

# Пример 1

---

```
for i in range(n - 1): +1
    for j in range(n - 1 - i): +2
        if values[j] > values[j + 1]: +3
            tmp = values[j],
            values[j] = values[j + 1]
            values[j + 1] = tmp
```



# Пример 1

---

```
for i in range(n - 1): +1
    for j in range(n - 1 - i): +2
        if values[j] > values[j + 1]: +3
            tmp = values[j],
            values[j] = values[j + 1]
            values[j + 1] = tmp
```

**Итого: 6**

# Пример 1

---


```
for i in range(n - 1): +1
    for j in range(n - 1 - i): +2
        if values[j] > values[j + 1]: +3
            tmp = values[j],
            values[j] = values[j + 1]
            values[j + 1] = tmp
```

Как уменьшить?

# Пример 1

---

```
for i in range(n - 1): +1
    for j in range(n - 1 - i): +2
        if values[j] > values[j + 1]: +3
            tmp = values[j],
            values[j] = values[j + 1]
            values[j + 1] = tmp
```



Как уменьшить?

# Пример 1

---

```
def one_pass(steps):  
    for j in range(steps): +1  
        if values[j] > values[j + 1]: +2  
            values[j], values[j + 1] = values[j + 1], values[j]  
  
for i in range(n - 1):  
    one_pass(n - 1 - i) +1
```

# Пример 2

---

```
int sumOfPrimes(int max) {  
    int total = 0;  
    OUT: for (int i = 1; i <= max; ++i) { // +1  
        for (int j = 2; j < i; ++j) {      // +2  
            if (i % j == 0) {              // +3  
                continue OUT;             // +1  
            }  
        }  
        total += i;  
    }  
    return total;  
}  
// Cognitive Complexity 7
```

```
String getWords(int number) {  
    switch (number) { // +1  
        case 1:  
            return "one";  
        case 2:  
            return "a couple";  
        case 3:  
            return "a few";  
        default:  
            return "lots";  
    }  
}  
// Cognitive Complexity 1
```

# Статьи

---

Грязный код → Ошибки → Неверные результаты → Неверные выводы

	AG		DSTC		SST-2		RT	
	Dist	Ent	Dist	Ent	Dist	Ent	Dist	Ent
<b>he attack:</b>	0.69	9.81	0.34	7.48	0.72	8.64	0.71	9.30
rdBug	0.75 ↑	9.95 ↑	0.46 ↑	7.90 ↑	0.76 ↑	8.72 ↑	0.75 ↑	9.37 ↑
	0.69	9.82 ↑	0.38 ↑	7.61 ↑	0.72	8.64	0.71	9.30
	0.73 ↑	9.91 ↑	0.40 ↑	7.75 ↑	0.74 ↑	8.68 ↑	0.73 ↑	9.34 ↑
ger	0.74 ↑	9.99 ↑	0.42	7.80 ↑	0.75 ↑	8.71 ↑	0.73 ↑	9.36 ↑
gFool (our)	0.71 ↑	9.98 ↑	0.43 ↑	7.76 ↑	0.76	8.84 ↑	0.74 ↑	9.38 ↑
our)	0.63 ↓	9.53 ↓	0.41 ↑	7.73 ↑	0.73 ↑	8.77 ↑	0.72 ↑	9.34 ↑
with DL (our)	0.69	9.89 ↑	0.41 ↑	7.72 ↑	0.74 ↑	8.78 ↑	0.73 ↑	9.36 ↑

# Тайпхинтинг



# C++

```
#include <iostream>

using namespace std;

void google(int size) {
    cout << 1;
    for (int j = 0; j < size; j++) {
        cout << 0;
    }
    cout << endl;
}

void crown(int width) {
    for (int i = 0; i < width; i++) {
        google(i);
    }
}

int main() {
    int k
    for (k = 0; k < 5; k++) {
        crown(6 + k);
    }
    cout << "1\n1\n1" << endl;
    return 0;
}
```





# C++

```
#include <iostream>

using namespace std;

void google(int size) {
    cout << 1;
    for (int j = 0; j < size; j++) {
        cout << 0;
    }
    cout << endl;
}

void crown(int width) {
    for (int i = 0; i < width; i++) {
        google(i);
    }
}

int main() {
    int k
    for (k = 0; k < 5; k++) {
        crown(6 + k);
    }
    cout << "1\n1\n1" << endl;
    return 0;
}
```

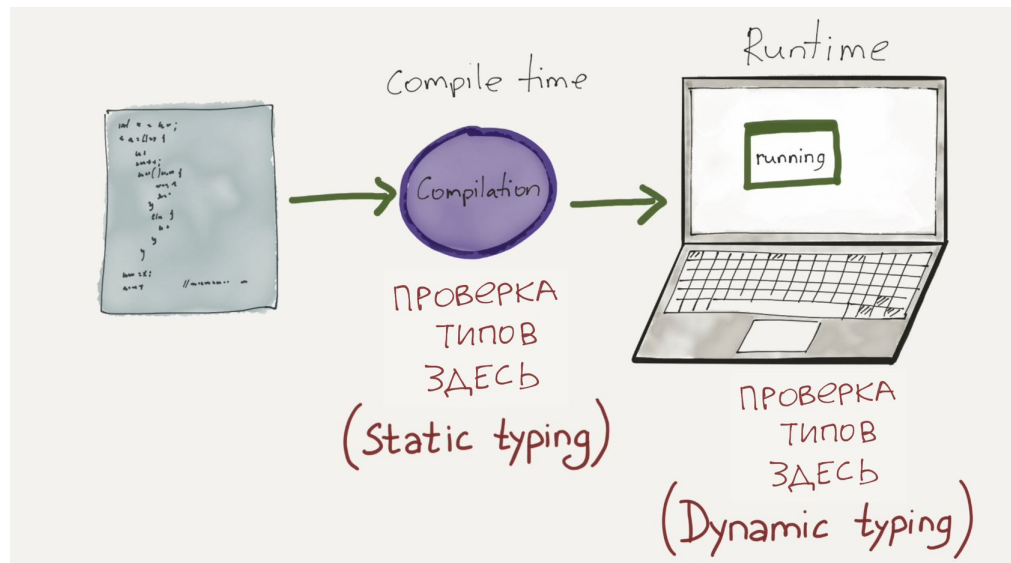
```
def google(size):
    print(1, end='')
    for _ in range(size):
        print(0, end='')
    print()
```

```
def crown(width):
    for i in range(width):
        google(i)
```

```
for k in range(5):
    crown(6 + k)
print('1\n1\n1')
```



# Типизация



Зачем вообще нужно  
определять типы  
данных?

# Утиная типизация

**Утиная типизация** — конкретный тип или класс объекта не важен, а важны лишь свойства и методы, которыми этот объект обладает.

**Duck test:** «Если это выглядит как утка, плавает как утка и крякает как утка, то это, вероятно, и есть утка».



# Аннотирование типов в Python

- Python останется языком с динамической типизацией,
- Начиная с Python 3.6+ есть typehints
- PEP-310
- PEP-0484
- PEP-0526

**Полезные ссылки:** <https://habr.com/ru/companies/lamoda/articles/432656/>

# Аннотирование типов в Python

- Аннотации типов просто считываются интерпретатором Python и никак более не обрабатываются,
- Но доступны для использования из стороннего кода
- ... Для статического анализа
- PyCharm, PyLint, ...

**Полезные ссылки:** <https://habr.com/ru/companies/lamoda/articles/432656/>

# Как пользоваться?

```
def be_my_son(word, n):  
    for _ in range(n):  
        print(word, end=" ")
```

```
value = 100500  
be_my_son("Maaaam!", value)
```

# Как пользоваться?

```
from typing import *
```

```
def be_my_son(word, n):  
    for _ in range(n):  
        print(word, end=" ")
```

До Python 3.6



```
value = 100500  
be_my_son("Maaaam!", value)
```



# Как пользоваться?

```
from typing import *  
  
def be_my_son(word, n):  
    for _ in range(n):  
        print(word, end=" ")  
  
value: int = 100500  
be_my_son("Maaaam!", value)
```



# Как пользоваться?

```
from typing import *
```

```
def be_my_son(word, n):  
    for _ in range(n):  
        print(word, end=" ")
```

```
value: int = 100500  
value = "Сто пятсот"
```

# PyCharm

```
7 value: int = 100500
8 value = "Сто пятьсот"
9 my_son( word: "Мааааааа!" value)
10
```

Run google x

```
my_son("Мааааааа!", value)
File "/Users/dmi3eva-k8/PycharmProjects/katya_zone/linter/google.py", line 3, in my_
for _ in range(n):
TypeError: 'str' object cannot be interpreted as an integer

Process finished with exit code 1
```

# Как пользоваться?

```
from typing import *  
  
def be_my_son(word, n):  
    for _ in range(n):  
        print(word, end=" ")  
  
value: int = 100500  
be_my_son("Maaaam!", value)
```

# Как пользоваться?

```
from typing import *  
  
def be_my_son(word: str, n: int):  
    for _ in range(n):  
        print(word, end=" ")  
  
value: int = 100500  
be_my_son("Maaaam!", value)
```

# Как пользоваться?

```
from typing import *  
  
def be_my_son(word: str, n: int):  
    for _ in range(n):  
        print(word, end=" ")  
  
value: int = 100500  
be_my_son("Maaaam!", value)
```

# Ошибка

```
from typing import *

def be_my_son(word: str, n: int):
    for _ in range(n):
        print(word, end=" ")

value: int = 100500
be_my_son(100500, "Maaaaam!")
```

# Пока не работает!

```
from typing import *
```

```
def get_phrase(word: str, n: int):  
    ...
```

```
value: int = 100500
```

```
replica = get_phrase("Maaaam!", value)
```



# Пока не работает!

```
from typing import *
```

```
def get_phrase(word: str, n: int):  
    pass
```

```
value: int = 100500
```

```
replica = get_phrase("Maaaam!", value)
```





# Пока не работает!

```
from typing import *
```

```
def get_phrase(word: str, n: int):  
    for _ in range(n):  
        phrase = phrase + "Maaaam! "
```

```
value: int = 100500
```

```
replica = get_phrase("Maaaam!", value)
```



# Пока не работает!

```
from typing import *
```

```
def get_phrase(word: str, n: int):  
    phrase = " Maaaam!"  
    for _ in range(n):  
        phrase = phrase + "Maaaam! "
```

```
value: int = 100500
```

```
replica = get_phrase("Maaaam!", value)
```



# Пока не работает!

```
from typing import *
```

```
def get_phrase(word: str, n: int):  
    phrase = " Maaaam!"  
    for _ in range(n):  
        phrase = phrase + "Maaaam! "  
    return phrase
```

```
value: int = 100500
```

```
replica = get_phrase("Maaaam!", value)
```



# Работает!

```
from typing import *

def get_phrase(word: str, n: int):
    phrase = " Maaaam!"
    for _ in range(n):
        phrase = phrase + "Maaaam! "
    return phrase.strip()

value: int = 100500

replica = get_phrase("Maaaam!", value)
```



# Работает!

```
from typing import *

def get_phrase(word: str, n: int) -> str:
    phrase = " Maaaam!"
    for _ in range(n):
        phrase = phrase + "Maaaam! "
    return phrase.strip()

value: int = 100500

replica = get_phrase("Maaaam!", value)
```

Дорогой ценой я выучил урок: для небольших программ динамическая типизация – благо. Но для более крупных программ необходим более дисциплинированный подход.

Гвидо Ван Россум, фанат Монти Пайтон

# Ошибка

```
from typing import *

def be_my_son(word: str, n: int):
    for _ in range(n):
        print(word, end=" ")

value: int = 100500
be_my_son(100500, "Maaaaam!")
```

# Ошибка

```
from typing import *

def be_my_son(word: str, n: int) -> NoReturn:
    for _ in range(n):
        print(word, end=" ")

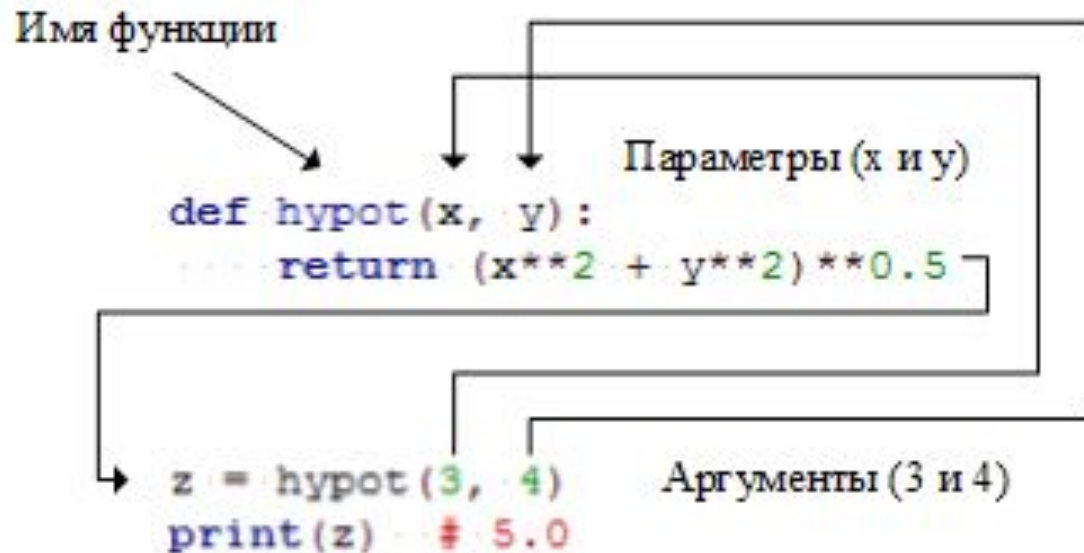
value: int = 100500
be_my_son(100500, "Maaaaam!")
```



# Параметры и аргументы

# Параметры vs. Аргументы

---



# Параметры vs. Аргументы

---

**Параметры** — это переменные, которые определяются в заголовке функции и используются внутри неё.

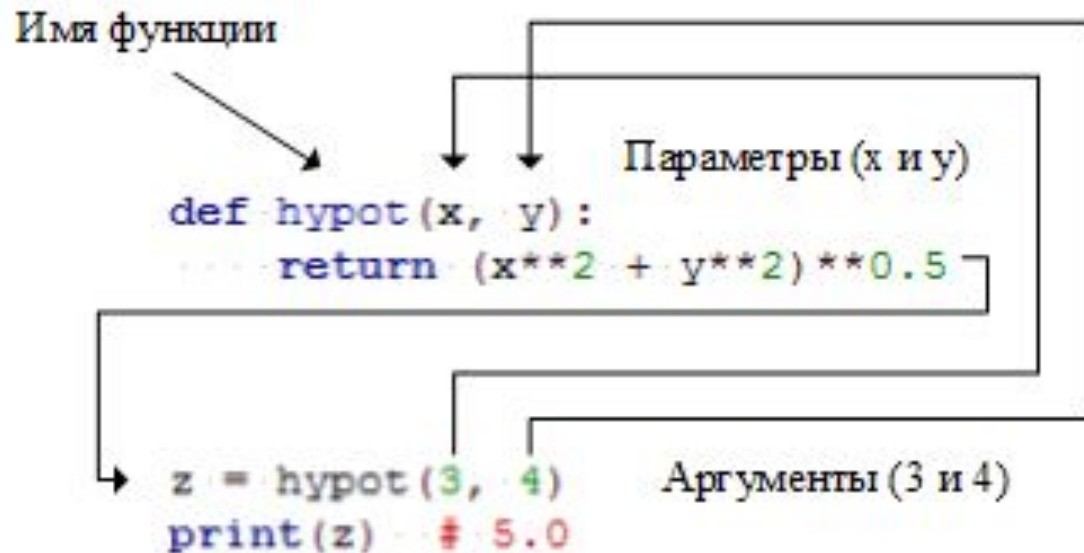
- Они служат как «заполнители» для значений, которые будут использованы в теле функции.

**Аргументы** — это конкретные значения или данные, передаваемые функции при ее вызове.

- Аргументы подставляются в места параметров при выполнении функции.
- Аргументы могут быть константами, переменными, выражениями или даже результатами других функций.

# Параметры vs. Аргументы

---



# Область видимости

---

```
from typing import *

def get_phrase(word: str, n: int) -> str:
    phrase = " Maaaam!"
    for _ in range(n):
        phrase = phrase + "Maaaam! "
    return phrase.strip()

replica = get_phrase("Maaaam!", value)
print(phrase)
```

# Область видимости

---

```
from typing import *

def get_phrase(word: str, n: int) -> str:
    phrase = " Maaaam!"
    for _ in range(n):
        phrase = phrase + "Maaaam! "
    return phrase.strip()

replica = get_phrase("Maaaam!", value)
print(phrase)
```

# Виды параметров

# Позиционные параметры

```
from typing import *  
  
def be_my_son(word: str, n: int) -> NoReturn:  
    for _ in range(n):  
        print(word, end=" ")  
  
be_my_son(100500, "Maaaam!")
```



# Позиционные параметры

```
from typing import *
```

```
def be_my_son(word: str, n: int) -> NoReturn:  
    for _ in range(n):  
        print(word, end=" ")
```

```
be_my_son(100500, "Maaaam!")
```

# Позиционные параметры

```
from typing import *

def be_my_son(word: str, n: int) -> NoReturn:
    for _ in range(n):
        print(word, end=" ")

be_my_son("Maaaam!", 100500)
```



# Именованные параметры

```
from typing import *  
  
def be_my_son(word: str, n: int) -> NoReturn:  
    for _ in range(n):  
        print(word, end=" ")  
  
be_my_son(n=100500, word="Maaaam!")
```

# Параметры по умолчанию

```
def be_my_son(word: str, n: int) -> NoReturn:  
    for _ in range(n):  
        print(word, end=" ")  
  
be_my_son("Maaaam!", 100500)
```

# Строго после позиционных

```
def be_my_son(word: str, n: int = 100500) -> NoReturn:
    for _ in range(n):
        print(word, end=" ")

be_my_son("Maaaam!", 100500)
```

# Можно не указывать

```
def be_my_son(word: str, n: int = 100500) -> NoReturn:  
    for _ in range(n):  
        print(word, end=" ")
```

```
be_my_son("Maaaam!")
```

# Параметры

```
print('I like you')  
print('I like you', flush=True, end='')  
print('I like you', end='', flush=True)
```

Уточняют действие

Позиционный аргумент

Именованный аргумент по умолчанию

<https://python-reference.readthedocs.io/en/latest/docs/functions/print.html>

# Произвольное количество аргументов



# Параметры

```
print('I like you')
```

<https://python-reference.readthedocs.io/en/latest/docs/functions/print.html>

# Параметры

```
print('I like you')
```

```
print('I like you', 'And you!')
```

<https://python-reference.readthedocs.io/en/latest/docs/functions/print.html>

# Параметры

```
print('I like you')
```

```
print('I like you', 'And you!', 3)
```

<https://python-reference.readthedocs.io/en/latest/docs/functions/print.html>

# Параметры

```
print('I like you')
```

```
print('I like you', 'And you!', 3)
```

<https://python-reference.readthedocs.io/en/latest/docs/functions/print.html>

# Параметры

```
print('I like you')
```

```
print('I like you', 'And you!', 3)
```

<https://python-reference.readthedocs.io/en/latest/docs/functions/print.html>

# Произвольное количество аргументов

```
def get_value(*args):  
    return args[0]
```

```
result = get_value(1, 2, 3, 4)  
print(result)    # Вывод: 1
```

# Произвольное количество аргументов

```
def get_value(*args):  
    return args[-1]
```

```
result = get_value(1, 2, 3, 4)  
print(result)
```

# Произвольное количество аргументов

```
def get_value(*args):  
    return args[-1]
```

```
result = get_value(1, 2, 3, 4)  
print(result)
```

```
a = [1,2,3]  
b = [*a,4,5,6]  
print(b)  # [1,2,3,4,5,6]
```



# Произвольное количество именованных

```
def print_info(**kwargs):  
    for _k, _v in kwargs.items():  
        print(_k, end="")  
        print(_v  
  
print_info(name="Alice", age=30, city="New York")
```

# И то, и то

```
def print_info(*args, **kwargs):  
    print(args[0])  
    for _k, _v in kwargs.items():  
        print(_k, end=" ")  
        print(_v)  
  
print_info(1, 2, 3, 4, name="Alice", age=30, city="New York")
```

# Рекурсия

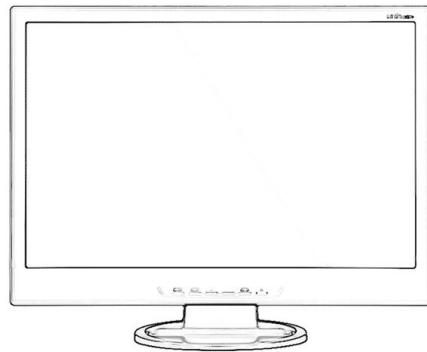
**Вопрос 5**

# Вариант 3

```
def do(n):  
    if n <= 0:  
        print("0", end="")  
    else:  
        print("1", end="")  
        do(n-1)
```

do(3)

do(3)

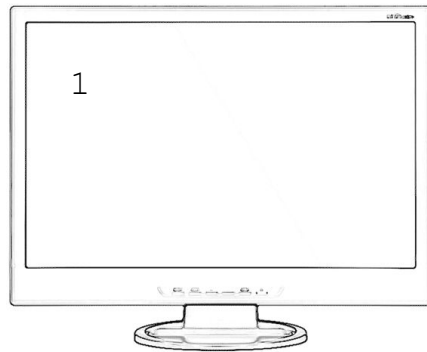


## Вариант 3

```
def do(n):  
    if n <= 0:  
        print("0", end="")  
    else:  
        print("1", end="")  
        do(n-1)
```

do(3)

do(3)

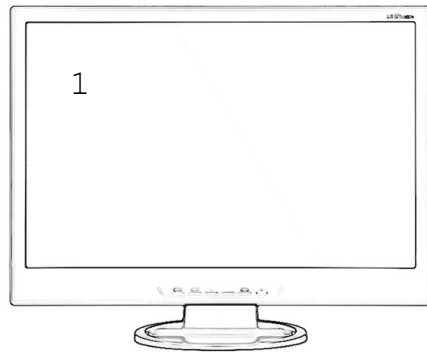


# Вариант 3

```
def do(n):  
    if n <= 0:  
        print("0", end="")  
    else:  
        print("1", end="")  
        do(n-1)
```

do(3)

```
do(3)  
do(2)
```

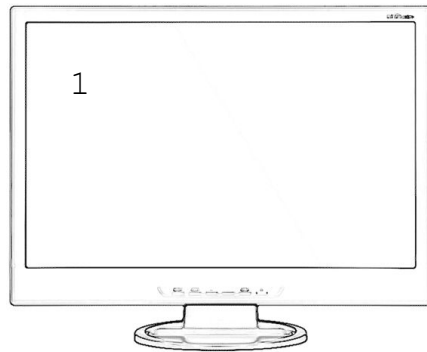


## Вариант 3

```
def do(n):  
    if n <= 0:  
        print("0", end="")  
    else:  
        print("1", end="")  
        do(n-1)
```

do(3)

```
do(3)  
    do(2)
```



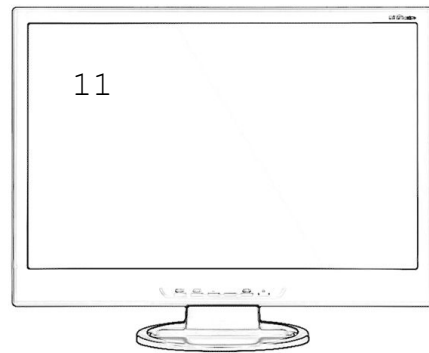


## Вариант 3

```
def do(n):  
    if n <= 0:  
        print("0", end="")  
    else:  
        print("1", end="")  
        do(n-1)
```

do(3)

```
do(3)  
    do(2)
```

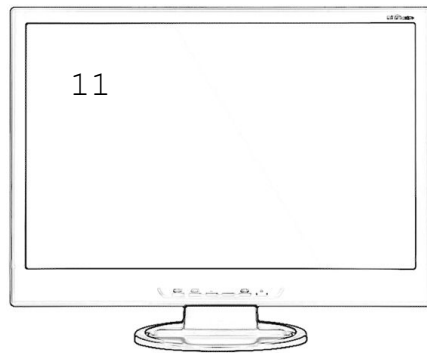


## Вариант 3

```
def do(n):  
    if n <= 0:  
        print("0", end="")  
    else:  
        print("1", end="")  
        do(n-1)
```

do(3)

```
do(3)  
    do(2)  
        do(1)
```

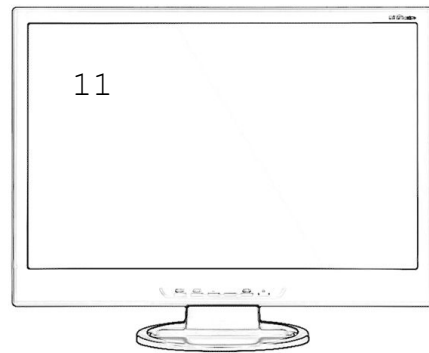


## Вариант 3

```
def do(n):  
    if n <= 0:  
        print("0", end="")  
    else:  
        print("1", end="")  
        do(n-1)
```

do(3)

```
do(3)  
    do(2)  
        do(1)
```

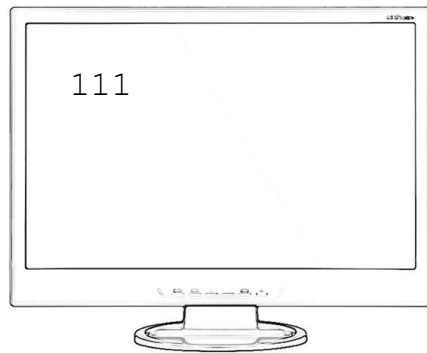


## Вариант 3

```
def do(n):  
    if n <= 0:  
        print("0", end="")  
    else:  
        print("1", end="")  
        do(n-1)
```

do(3)

```
do(3)  
    do(2)  
        do(1)
```

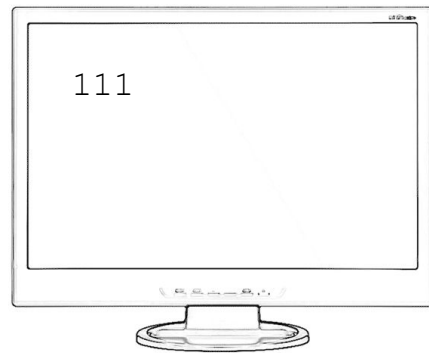


## Вариант 3

```
def do(n):  
    if n <= 0:  
        print("0", end="")  
    else:  
        print("1", end="")  
        do(n-1)
```

do(3)

```
do(3)  
  do(2)  
    do(1)  
      do(0)
```

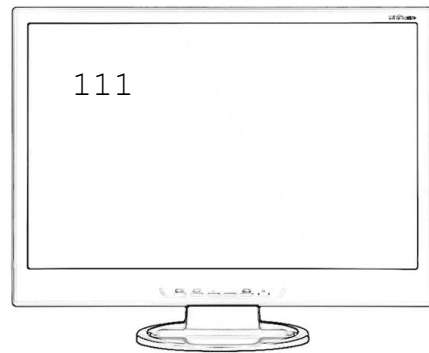


## Вариант 3

```
def do(n):  
    if n <= 0:  
        print("0", end="")  
    else:  
        print("1", end="")  
        do(n-1)
```

do(3)

```
do(3)  
  do(2)  
    do(1)  
      do(0)
```

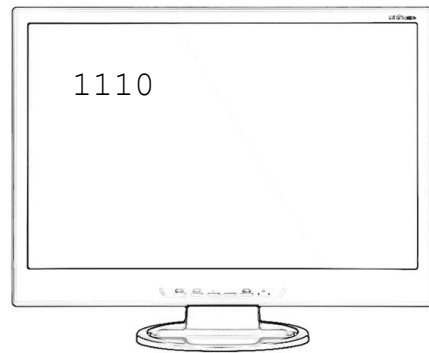


## Вариант 3

```
def do(n):  
    if n <= 0:  
        print("0", end="")  
    else:  
        print("1", end="")  
        do(n-1)
```

do(3)

```
do(3)  
  do(2)  
    do(1)  
      do(0)
```

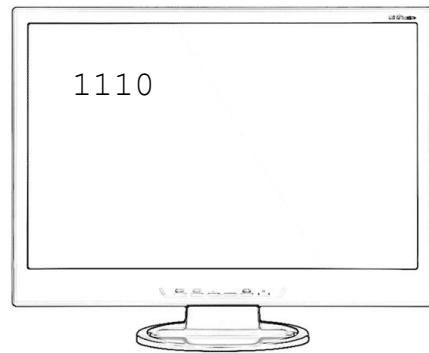


## Вариант 3

```
def do(n):  
    if n <= 0:  
        print("0", end="")  
    else:  
        print("1", end="")  
        do(n-1)
```

do(3)

```
do(3)  
    do(2)  
        do(1)
```



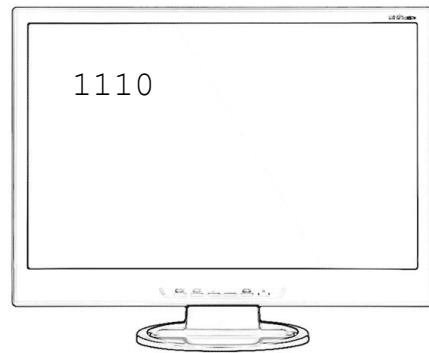


## Вариант 3

```
def do(n):  
    if n <= 0:  
        print("0", end="")  
    else:  
        print("1", end="")  
        do(n-1)
```

do(3)

```
do(3)  
  do(2)
```

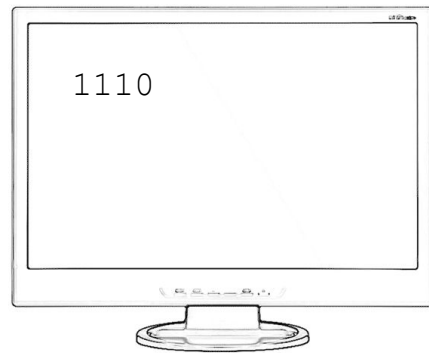


## Вариант 3

```
def do(n):  
    if n <= 0:  
        print("0", end="")  
    else:  
        print("1", end="")  
        do(n-1)
```

do(3)

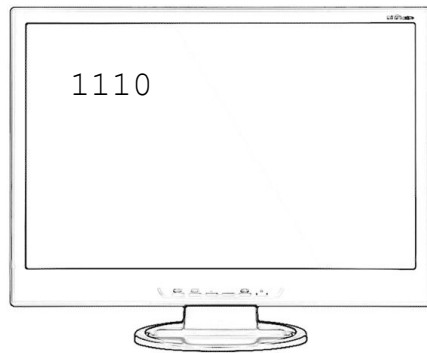
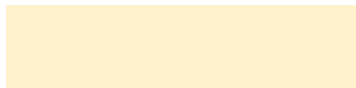
do(3)



## Вариант 3

```
def do(n):  
    if n <= 0:  
        print("0", end="")  
    else:  
        print("1", end="")  
        do(n-1)
```

do(3)

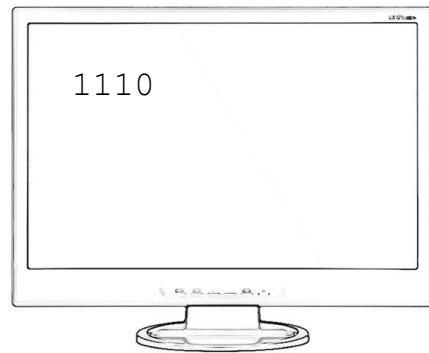


# Вариант 3

```
import sys
sys.setrecursionlimit(limit)
```

```
def do(n):
    if n <= 0:
        print("0", end="")
    else:
        print("1", end="")
        do(n-1)
```

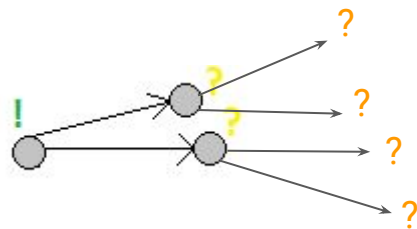
```
do(3)
```



**c return**

## Вариант 3

```
def solve(n):  
    if n < 2:  
        return 1  
  
    uplift_n = solve(n - 1) + solve(n - 2)  
  
    return uplift_n
```

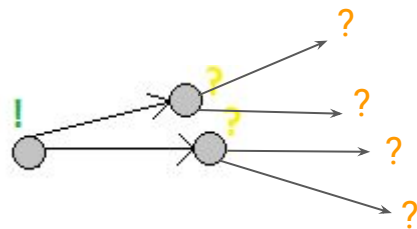


solve(4) =

```
n = int(input())  
result = solve(n)  
print(result)
```

# Вариант 3

```
def solve(n):  
    if n < 2:  
        return 1  
  
    uplift_n = solve(n - 1) + solve(n - 2)  
  
    return uplift_n
```



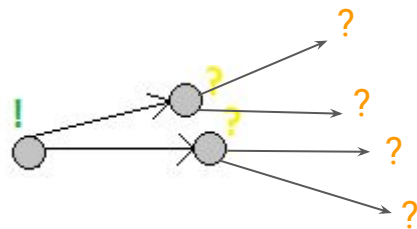
$\text{solve}(4) = \text{solve}(3) + \text{solve}(2)$

```
n = int(input())  
result = solve(n)  
print(result)
```

# Вариант 3

```
def solve(n):  
    if n < 2:  
        return 1  
  
    uplift_n = solve(n - 1) + solve(n - 2)  
  
    return uplift_n
```

```
n = int(input())  
result = solve(n)  
print(result)
```



$\text{solve}(4) = \text{solve}(3) + \text{solve}(2)$

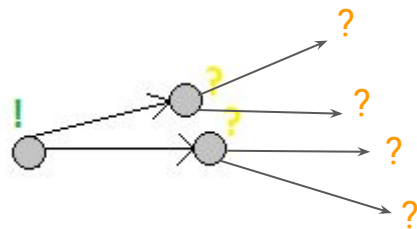
$\text{solve}(3) =$

$\text{solve}(2) =$



# Вариант 3

```
def solve(n):  
    if n < 2:  
        return 1  
  
    uplift_n = solve(n - 1) + solve(n - 2)  
  
    return uplift_n  
  
n = int(input())  
result = solve(n)  
print(result)
```



$\text{solve}(4) = \text{solve}(3) + \text{solve}(2)$

$\text{solve}(3) = \text{solve}(2) + \text{solve}(1)$

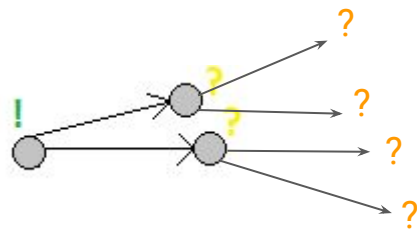
$\text{solve}(2) =$

$\text{solve}(1) =$

# Вариант 3

```
def solve(n):  
    if n < 2:  
        return 1  
  
    uplift_n = solve(n - 1) + solve(n - 2)  
  
    return uplift_n
```

```
n = int(input())  
result = solve(n)  
print(result)
```



$\text{solve}(4) = \text{solve}(3) + \text{solve}(2)$

$\text{solve}(3) = \text{solve}(2) + \text{solve}(1)$

$\text{solve}(2) = \text{solve}(1) + \text{solve}(0)$

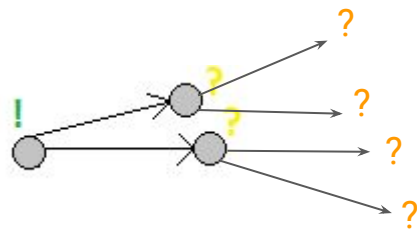
$\text{solve}(1) =$

$\text{solve}(0) =$

# Вариант 3

```
def solve(n):  
    if n < 2:  
        return 1  
  
    uplift_n = solve(n - 1) + solve(n - 2)  
  
    return uplift_n
```

```
n = int(input())  
result = solve(n)  
print(result)
```

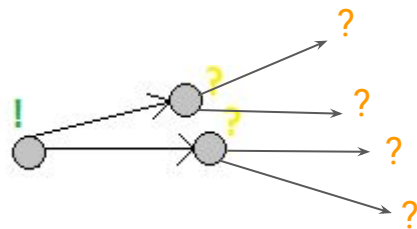


```
solve(4) = solve(3) + solve(2)  
solve(3) = solve(2) + solve(1)  
solve(2) = solve(1) + solve(0)  
solve(1) = 1  
solve(0) = 1
```

# Вариант 3

```
def solve(n):  
    if n < 2:  
        return 1  
  
    uplift_n = solve(n - 1) + solve(n - 2)  
  
    return uplift_n
```

```
n = int(input())  
result = solve(n)  
print(result)
```

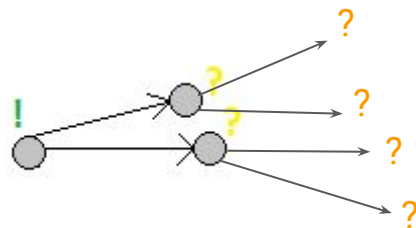


```
solve(4) = solve(3) + solve(2)  
solve(3) = solve(2) + solve(1)  
solve(2) = solve(1) + solve(0) = 1 + 1 = 2  
solve(1) = 1  
solve(0) = 1
```

# Вариант 3

```
def solve(n):  
    if n < 2:  
        return 1  
  
    uplift_n = solve(n - 1) + solve(n - 2)  
  
    return uplift_n
```

```
n = int(input())  
result = solve(n)  
print(result)
```



$\text{solve}(4) = \text{solve}(3) + \text{solve}(2)$

$\text{solve}(3) = \text{solve}(2) + \text{solve}(1) = 2 + 1 = 3$

$\text{solve}(2) = \text{solve}(1) + \text{solve}(0) = 1 + 1 = 2$

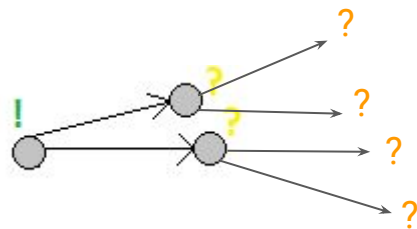
$\text{solve}(1) = 1$

$\text{solve}(0) = 1$

# Вариант 3

```
def solve(n):  
    if n < 2:  
        return 1  
  
    uplift_n = solve(n - 1) + solve(n - 2)  
  
    return uplift_n
```

```
n = int(input())  
result = solve(n)  
print(result)
```



$\text{solve}(4) = \text{solve}(3) + \text{solve}(2) = 3 + 2 = 5$

$\text{solve}(3) = \text{solve}(2) + \text{solve}(1) = 2 + 1 = 3$

$\text{solve}(2) = \text{solve}(1) + \text{solve}(0) = 1 + 1 = 2$

$\text{solve}(1) = 1$

$\text{solve}(0) = 1$

# Вариант 3

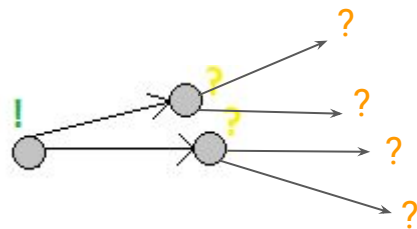
```
import sys

sys.setrecursionlimit(limit)

def solve(n):
    if n < 2:
        return 1

    uplift_n = solve(n - 1) + solve(n - 2)
    return uplift_n

n = int(input())
result = solve(n)
```



$$\text{solve}(4) = \text{solve}(3) + \text{solve}(2) = 3 + 2 = 5$$

$$\text{solve}(3) = \text{solve}(2) + \text{solve}(1) = 2 + 1 = 3$$

$$\text{solve}(2) = \text{solve}(1) + \text{solve}(0) = 1 + 1 = 2$$

$$\text{solve}(1) = 1$$

$$\text{solve}(0) = 1$$

# Задание

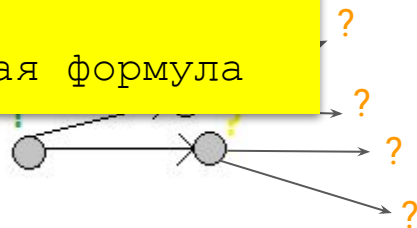
```
def f(n):  
    if n % 2 == 0 or n == 1:  
        return 2  
  
    uplift_n = f(n - 1) * f(n - 2)  
  
    return uplift_n  
  
n = int(input())  
result = f(n)  
print(result)
```

**A:**  $f(4) = ?$

**B:**  $f(5) = ?$

**C:**  $f(11) = ?$

**D\*:** Аналитическая формула



$\text{solve}(4) = \text{solve}(3) + \text{solve}(2) = 3 + 2 = 5$

$\text{solve}(3) = \text{solve}(2) + \text{solve}(1) = 2 + 1 = 3$

$\text{solve}(2) = \text{solve}(1) + \text{solve}(0) = 1 + 1 = 2$

$\text{solve}(1) = 1$

$\text{solve}(0) = 1$



# Задание

**A:**  $f(4) = 2$

**B:**  $f(5) = 8$

**C:**  $f(11) = 64$

**D\*:** Для нечетных  $2^{n//2 + 1}$

```
def f(n):
```

```
    if n % 2 == 0:
```

```
        return 2
```

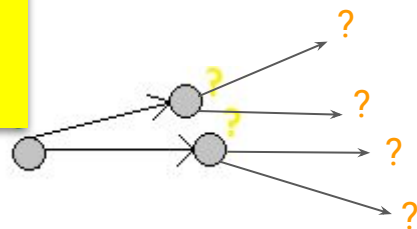
```
    uplift_n = f(n - 1) * f(n - 2)
```

```
    return uplift_n
```

```
n = int(input())
```

```
result = f(n)
```

```
print(result)
```



$\text{solve}(4) = \text{solve}(3) + \text{solve}(2) = 3 + 2 = 5$

$\text{solve}(3) = \text{solve}(2) + \text{solve}(1) = 2 + 1 = 3$

$\text{solve}(2) = \text{solve}(1) + \text{solve}(0) = 1 + 1 = 2$

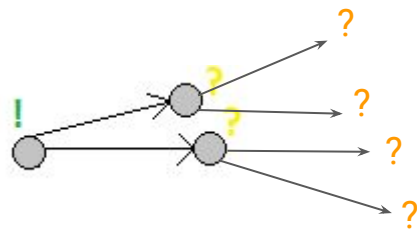
$\text{solve}(1) = 1$

$\text{solve}(0) = 1$

# Вариант 3

```
def solve(n):  
    uplift_n = solve(n - 1) + solve(n - 2)  
    return uplift_n
```

```
n = int(input())  
result = solve(n)  
print(result)
```

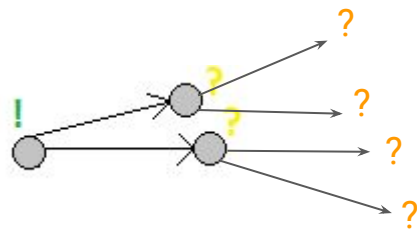


# Вариант 3

```
def solve(n):  
    if n < 2:  
        return 1  
    uplift_n = solve(n - 1) + solve(n - 2)  
    return uplift_n
```

Выход из рекурсии

```
n = int(input())  
result = solve(n)  
print(result)
```



# lambda-функции



# Боярский диалект языка С



- <https://habr.com/ru/articles/41561/>

использовати площадь каковычно амин

наместе двояко провѣряти чего глаголют молчаливо  
кагбе

ѣжѣли получалка. сломалася молчаливо тогдауж  
кагбе

молвити "Не лепо молвишь, барин!" амин

возвѣрнути нуль спасихоспади1

ага

возвѣрнути один амин

ага

# Боярский диалект языка Python

---



```
print("Hello, world!")
```

# Боярский диалект языка Python

---



```
print("Благо, земляне!")
```

```
молви("Благо, земляне!")
```



# Боярский диалект языка Python

---



```
def молви(txt):  
    print(txt)
```

```
молви("Благо, земляне!")
```



# Боярский диалект языка Python

---



```
def молви(txt):  
    print(txt)
```



```
молви("Благо, земляне!", end="\n\n")
```

# Боярский диалект языка Python

---



```
молви = print
```

```
молви("Благо, земляне!", end="\n\n")
```

# Принцип



```
import this
```

Всё в Python является объектами:

```
>>> isinstance(1, object)
True
>>> isinstance(list(), object)
True
>>> isinstance(True, object)
True
>>> def foo():
...     pass
...
>>> isinstance(foo, object)
True
```

# Принцип



```
import this
```

Всё в Python является объектами:

Каждый объект содержит как минимум три вида данных:

- Счётчик ссылок
- Тип
- Значение

# Лямбда-функции



```
def defined_cube(y):  
    return y * y * y
```

```
lambda_cube = lambda y: y*y*y  
print(defined_cube(2))  
print(lambda_cube(2))
```

ключевое слово	параметры функции	исполняемое выражение
lambda	a, b	: a * b

lambda表达式

```
lambda x, y: x + y
```

def func(x, y):  
 return x + y



# Лямбда-функции

---



```
def defined_nesumma(x, y):  
    return x + y - 1
```

```
lambda_nesumma = lambda x, y: x + y  
print(defined_nesumma(2, 4))  
print(lambda_nesumma(2, 4))
```

# Функция высшего порядка

---



```
numbers = [-100, 12, 2, 97, -4]
numbers.sort()
print(numbers)
```

# Функция высшего порядка

---



```
numbers = [-100, 12, 2, 97, -4]  
numbers.sort()  
print(numbers)
```

A black silhouette of a computer monitor with a stand, positioned on the right side of the slide. The screen of the monitor displays the sorted list.

```
[-100, -4, 2, 12, 97]
```



# Функция высшего порядка



```
numbers = [-100, 12, 2, 97, -4]
numbers.sort(???)
print(numbers)
```

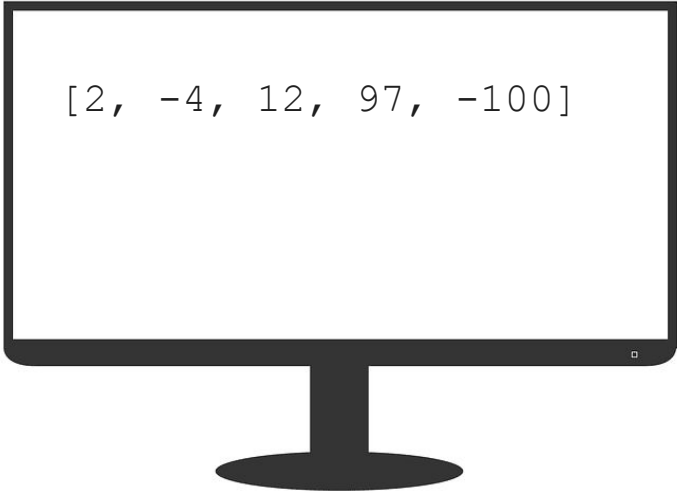
A black silhouette of a computer monitor with a stand, positioned on the right side of the slide. The screen of the monitor displays the sorted list.

```
[2, -4, 12, 97, -100]
```

# Функция высшего порядка



```
numbers = [-100, 12, 2, 97, -4]
numbers.sort(key=comparator)
print(numbers)
```

A black silhouette of a computer monitor with a stand, positioned on the right side of the slide. The screen of the monitor displays the sorted list.

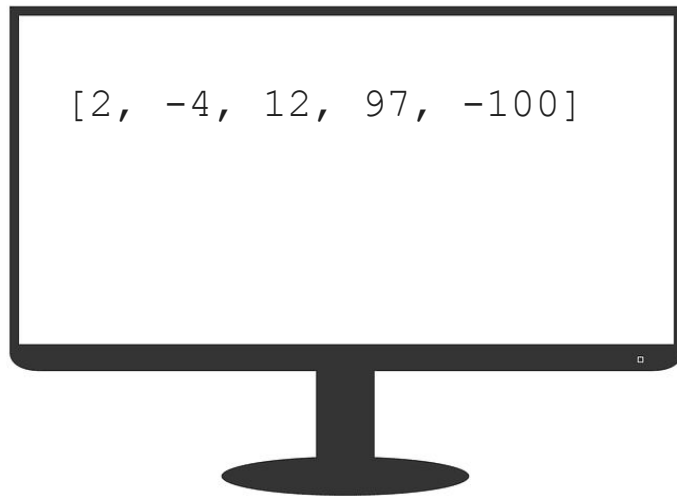
```
[2, -4, 12, 97, -100]
```

# Функция высшего порядка



```
def comparator(x):  
    return abs(x)
```

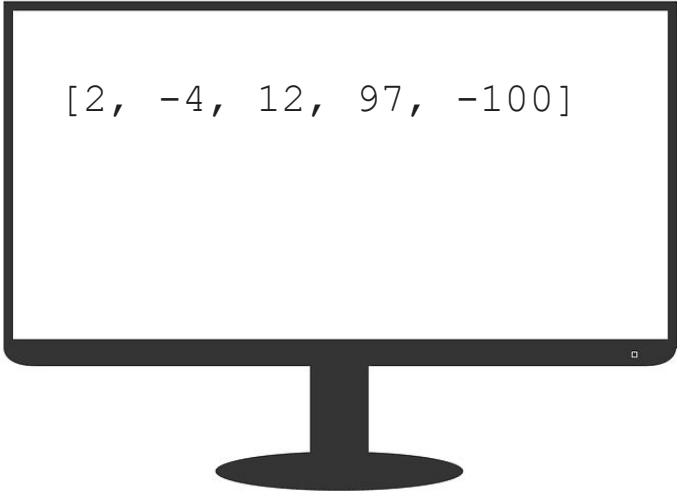
```
numbers = [-100, 12, 2, 97, -4]  
numbers.sort(key=comparator)  
print(numbers)
```



# Функция высшего порядка



```
numbers = [-100, 12, 2, 97, -4]
numbers.sort(key=lambda x: abs(x))
print(numbers)
```

A black silhouette of a computer monitor with a stand, positioned on the right side of the slide. The screen of the monitor displays the output of the Python code.

```
[2, -4, 12, 97, -100]
```

# Функция высшего порядка



```
numbers = [-100, 12, 2, 97, -4]
numbers.sort(key=lambda x: x**2)
print(numbers)
```



# Функция высшего порядка



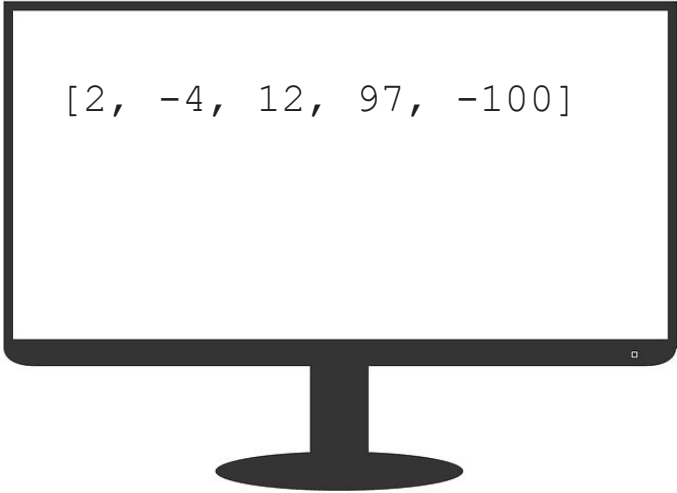
```
numbers = [-100, 12, 2, 97, -4]
numbers.sort(key=lambda x: -x)
print(numbers)
```



# Функция высшего порядка



```
numbers = [-100, 12, 2, 97, -4]
numbers.sort(key=lambda x: -x)
print(numbers)
```

A black computer monitor with a wide base, displaying the sorted list of numbers.

```
[2, -4, 12, 97, -100]
```



# Функция высшего порядка

---

```
numbers = [[1, 7], [2, 3], [4, 0]]  
numbers.sort(key=lambda x: x[0]+x[1])  
print(numbers)
```





# Функция высшего порядка

---



```
numbers = [(1, 7), (2, 3), (4, 0)]  
numbers.sort(key=lambda x: x[0]+x[1])  
print(numbers)
```

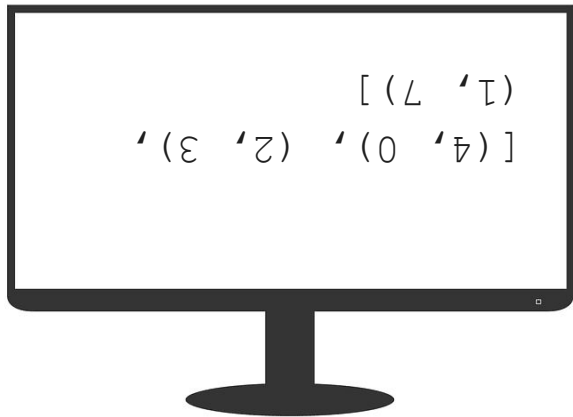




# Функция высшего порядка

---

```
numbers = [(1, 7), (2, 3), (4, 0)]  
numbers.sort(key=lambda x: x[0]+x[1])  
print(numbers)
```



**Вопрос 3**

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

A) `print(x+y)`

Б) `print("x+y")`

В) `print("{x+y}")`

Г) `print("{x}"+"{y}")`

Д) `print("{x}+{y}")`

Е) `print(f"{x}+{y}")`

Ё) Никакой

## Вывод

```
n = 7  
print(n)  
print("There are 7 days in a week")
```

## Вывод

```
n = 7  
print(n)  
print("There are n days in a week")
```

## Вывод

```
n = 7  
print(n)  
print(f"There are {n} days in a week")
```

## Вывод

```
n = 7  
print(n)  
print(f"There are {n} days in a week")
```



The image shows a stylized computer monitor with a grey frame and a white screen. The screen displays the text "There are 7 days in a week" in a monospaced font. The monitor is sitting on a grey base.

```
There are 7 days in a week
```

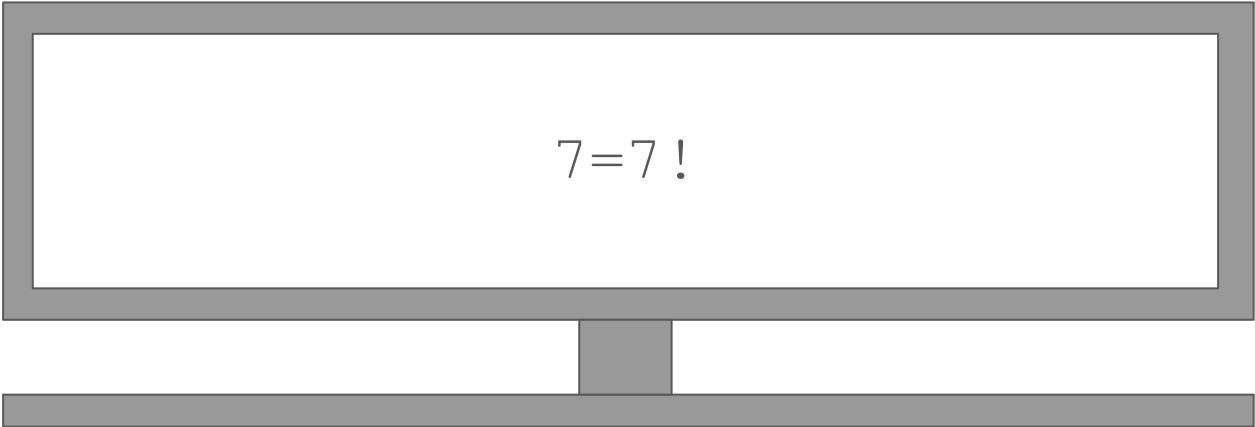


## Вывод

```
n = 7  
print(n)  
print(f"{n}={n}!")
```

# Вывод

```
n = 7  
print(n)  
print(f"{n}={n}!")
```



7=7!

# Вывод

```
n = 7  
w = "seven"  
print(n)  
print(f"{w}={n}!")
```

# Вывод

```
n = 7  
w = "seven"  
print(n)  
print(f"{w}={n}!")
```



seven=7!

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

A) `print(x+y)`

Б) `print("x+y")`

В) `print("{x+y}")`

Г) `print("{x}"+"{y}")`

Д) `print("{x}+{y}")`

Е) `print(f"{x}+{y}")`

Ё) Никакой

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

**A) print(x+y)**

Б) print("x+y")

В) print("{x+y}")

Г) print("{x}"+"{y}")

Д) print("{x}+{y}")

Е) print(f"{x}+{y}")

Ё) Никакой

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

600 **A) print(x+y)**

Б) print("x+y")

В) print("{x+y}")

Г) print("{x}"+"{y}")

Д) print("{x}+{y}")

Е) print(f"{x}+{y}")

Ё) Никакой

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

600 A) `print(x+y)`

**Б) `print("x+y")`**

В) `print("{x+y}")`

Г) `print("{x}"+"{y}")`

Д) `print("{x}+{y}")`

Е) `print(f"{x}+{y}")`

Ё) Никакой



# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

600 A) `print(x+y)`

x+y Б) **`print("x+y")`**

В) `print("{x+y}")`

Г) `print("{x}"+"{y}")`

Д) `print("{x}+{y}")`

Е) `print(f"{x}+{y}")`

Ё) Никакой

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

600 А) `print(x+y)`

x+y Б) `print("x+y")`

**В) `print("{x+y}")`**

Г) `print("{x}"+"{y}")`

Д) `print("{x}+{y}")`

Е) `print(f"{x}+{y}")`

Ё) Никакой

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

600 А) `print(x+y)`

x+y Б) `print("x+y")`

{x+y} В) **`print("{x+y}")`**

Г) `print("{x}"+"{y}")`

Д) `print("{x}+{y}")`

Е) `print(f"{x}+{y}")`

Ё) Никакой

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

600 А) `print(x+y)`

x+y Б) `print("x+y")`

{x+y} В) `print("{x+y}")`

Г) **`print("{x}"+"{y}")`**

Д) `print("{x}+{y}")`

Е) `print(f"{x}+{y}")`

Ё) Никакой

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

600 А) `print(x+y)`

x+y Б) `print("x+y")`

{x+y} В) `print("{x+y}")`

{x}{y} Г) **`print("{x}"+"{y}")`**

Д) `print("{x}+{y}")`

Е) `print(f"{x}+{y}")`

Ё) Никакой

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

600 А) `print(x+y)`

x+y Б) `print("x+y")`

{x+y} В) `print("{x+y}")`

{x}{y} Г) `print("{x}"+"{y}")`

Д) **`print("{x}+{y}")`**

Е) `print(f"{x}+{y}")`

Ё) Никакой

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

600 А) `print(x+y)`

x+y Б) `print("x+y")`

{x+y} В) `print("{x+y}")`

{x}{y} Г) `print("{x}"+"{y}")`

{x}+{y} Д) **`print("{x}+{y}")`**

Е) `print(f"{x}+{y}")`

Ё) Никакой

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

600 А) `print(x+y)`

x+y Б) `print("x+y")`

{x+y} В) `print("{x+y}")`

{x}{y} Г) `print("{x}"+"{y}")`

{x}+{y} Д) `print("{x}+{y}")`

Е) **`print(f"{x}+{y}")`**

Ё) Никакой



# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

600 А) `print(x+y)`

x+y Б) `print("x+y")`

{x+y} В) `print("{x+y}")`

{x}{y} Г) `print("{x}"+"{y}")`

{x}+{y} Д) `print("{x}+{y}")`

100+500 Е) **`print(f"{x}+{y}")`**

Ё) Никакой

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

600 А) `print(x+y)`

x+y Б) `print("x+y")`

{x+y} В) `print("{x+y}")`

{x}{y} Г) `print("{x}"+"{y}")`

{x}+{y} Д) `print("{x}+{y}")`

100+500 Е) `print(f"{x}+{y}")`

Ё) **Никакой**

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

600 А) `print(x+y)`

x+y Б) `print("x+y")`

{x+y} В) `print("{x+y}")`

{x}{y} Г) **`print("{x}"+"{y}")`**

{x}+{y} Д) `print("{x}+{y}")`

100+500 Е) `print(f"{x}+{y}")`

Ё) **Никакой**

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

```
x = 100
```

```
y = 500
```

```
// Одна строка
```

600 А) `print(x+y)`

x+y Б) `print("x+y")`

{x+y} В) `print("{x+y}")`

100500 Г) **`print(f"{x}" + f"{y}")`**

{x}+{y} Д) `print("{x}+{y}")`

100+500 Е) `print(f"{x}+{y}")`

Ё) **Никакой**

# Вопрос 3



С помощью какой строчки кода можно вывести "100500"?

x = 100

y = 500

// Одна строка

600 А) print(x+y)

x+y Б) print("x+y")

{x+y} В) print("{x+y}")

100500 Г) **print(f"{x}" + f"{y}")**

{x}+{y} Д) print("{x}+{y}")

100+500 Е) print(f"{x}+{y}")

Ё) **print(str(x)+str(y))**

# Полезно



```
x = 100
```

```
y = 500
```

```
print(f"{x}")
```

✓  
0  
сек.

```
[12] print(f"{x}")
```



100

# Полезно



```
x = 100
```

```
y = 500
```

```
print(f"{x=} ")
```



0  
сек.

```
[12] print(f"{x}")
```



100



0  
сек.



```
print(f"{x=}")
```



x=100

**Вопрос 4**



## Вопрос 3


---



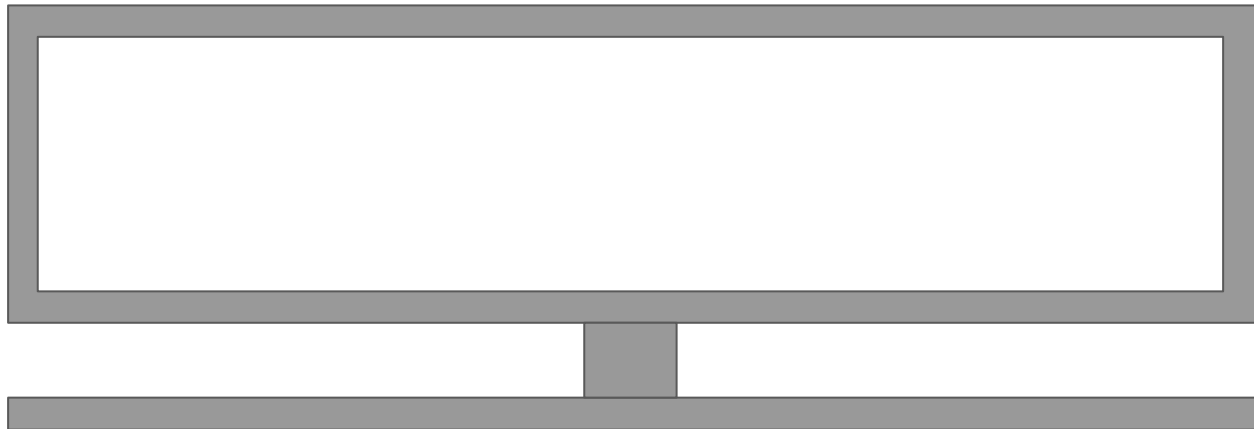
```
word = "seven"  
print(word[-3:])
```

Что будет на экране?

## Вывод



```
word = "seven"  
print(word[1])
```



# Вывод

```
word = "seven"  
print(word[1])
```

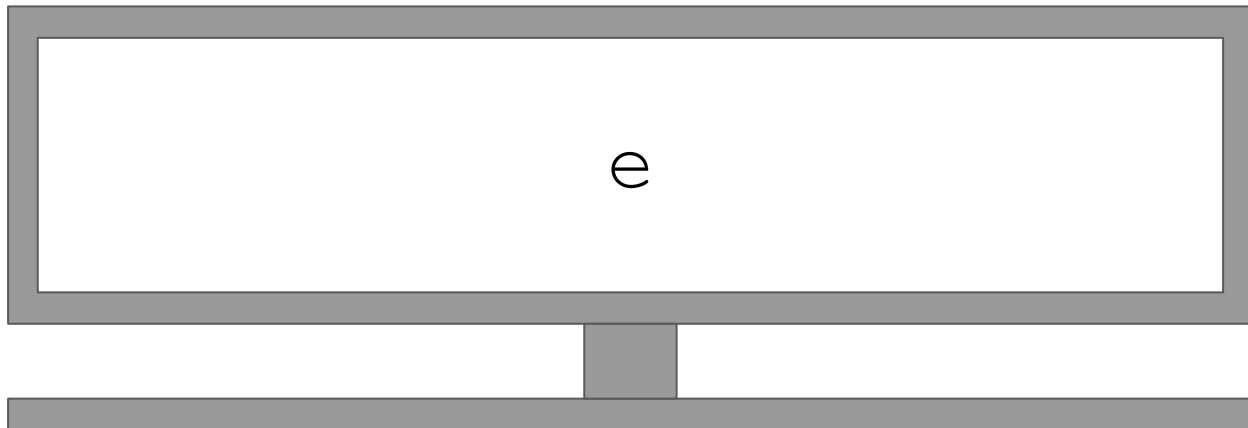
0	1	2	3	4
s	e	v	e	n



# Вывод

```
word = "seven"  
print(word[1])
```

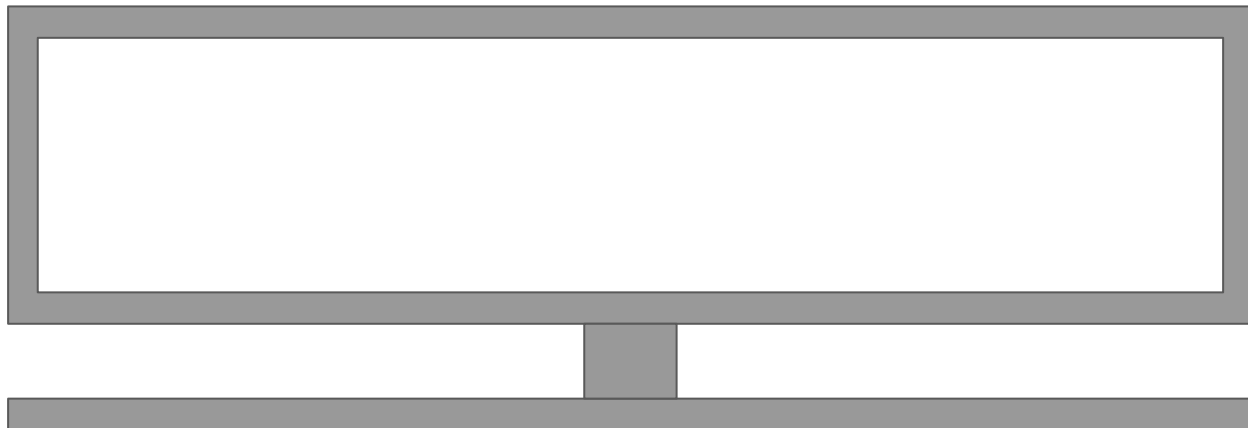
0	1	2	3	4
s	e	v	e	n



# Вывод

```
word = "seven"  
print(word[1:3])
```

0	1	2	3	4
s	e	v	e	n



# Вывод

```
word = "seven"  
print(word[1:3])
```

0	1	2	3	4
s	e	v	e	n

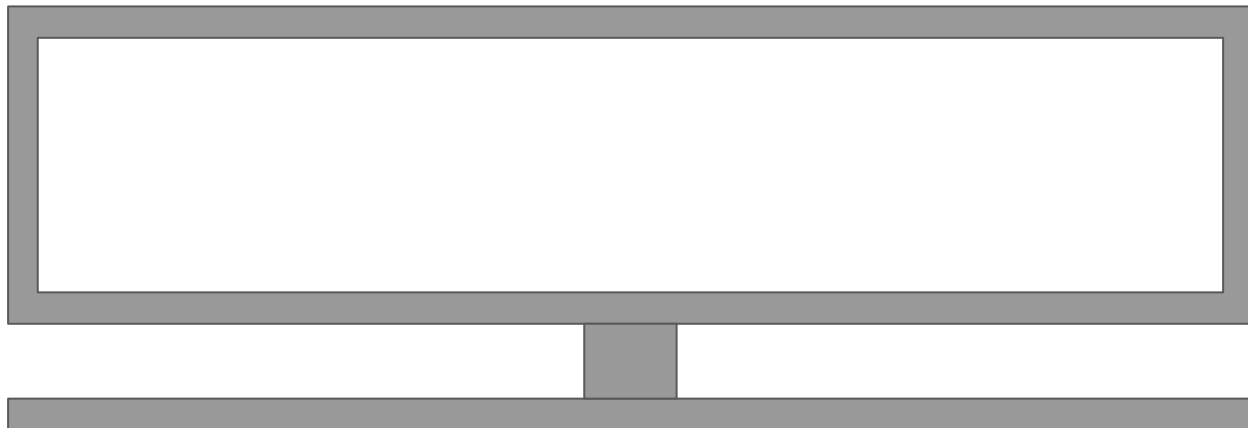


ev

# Вывод

```
word = "seven"  
print(word[:3])
```

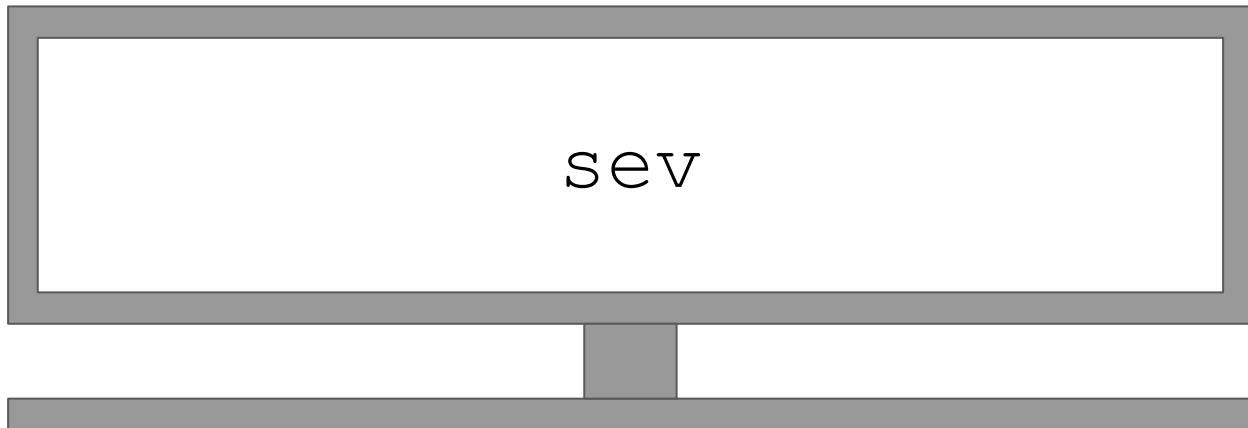
0	1	2	3	4
s	e	v	e	n



# Вывод

```
word = "seven"  
print(word[:3])
```

0	1	2	3	4
s	e	v	e	n





# Вывод

```
word = "seven"  
print(word[1:])
```

0	1	2	3	4
s	e	v	e	n



# Вывод

```
word = "seven"  
print(word[1:])
```

0	1	2	3	4
s	e	v	e	n



even

# Вывод

```
word = "seven"  
print(word[:])
```

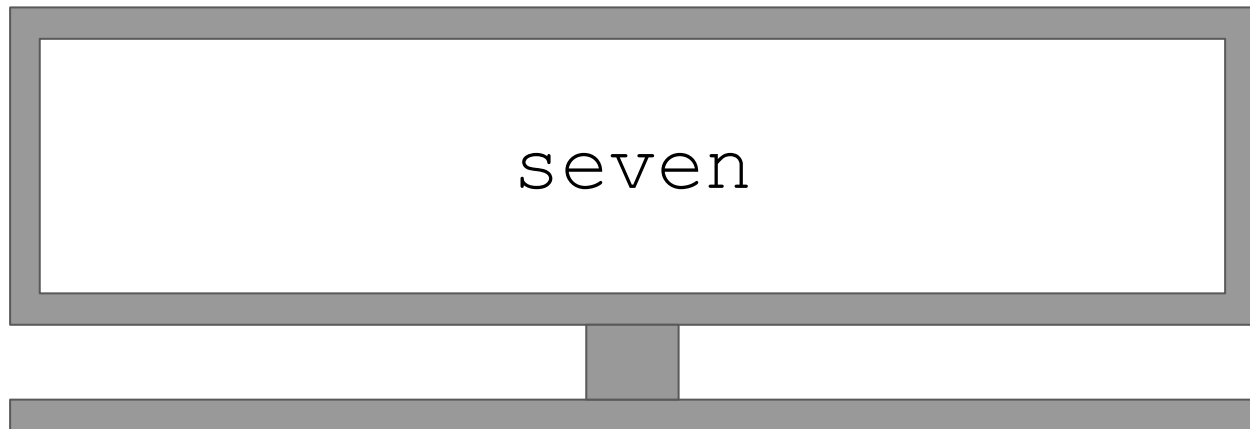
0	1	2	3	4
s	e	v	e	n



# Вывод

```
word = "seven"  
print(word[:])
```

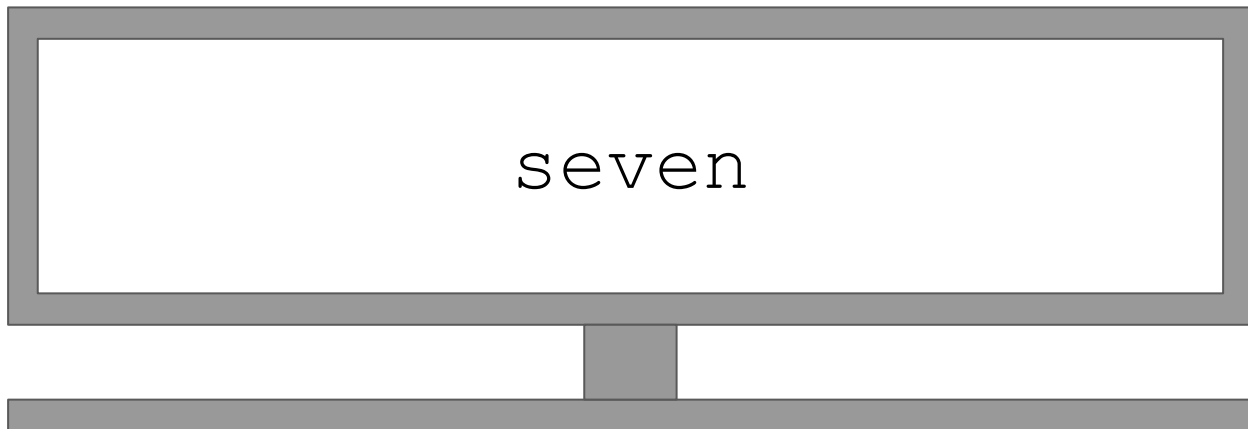
0	1	2	3	4
s	e	v	e	n



## Вывод

```
word = "seven"  
print(word[-1])
```

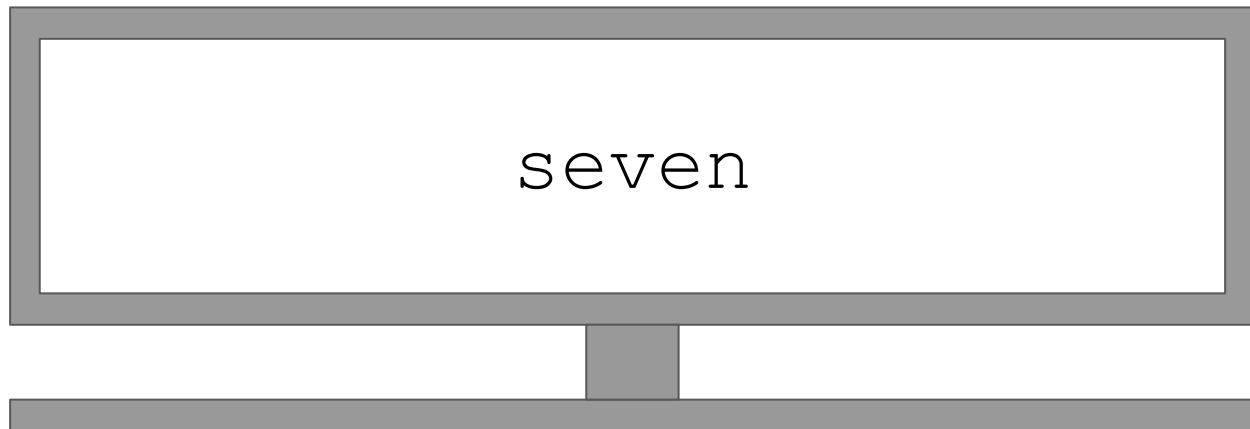
0	1	2	3	4
s	e	v	e	n



# Вывод

```
word = "seven"  
print(word[-1])
```

0	1	2	3	4
s	e	v	e	n



# Вопрос 3



```
word = "seven"  
print(word[-3:])
```

Что будет на экране?

0	1	2	3	4
s	e	v	e	n

# Вопрос 3



```
word = "seven"  
print(word[-3:])
```



```
word = "seven"  
print(word[-3:])
```



ven

Что будет на экране?

0	1	2	3	4
s	e	v	e	n



