Искусство однострочников

Как решить любую задачу в одно выражение

Александр Харламов, CodeScoring
Telegram: @wignorbo

О себе

- Пишу на Python более 6 лет
- Backend-разработчик в CodeScoring
- Магистрант ИТМО
- 476 дней на литкоде

Зачем слушать этот доклад?

- Начинающим:
  - Основные концепции функционального программирования
  - Идиоматические подходы Python к решению задач
- Продолжающим:
  - Ленивые вычисления
  - Работа с бесконечными последовательностями
- Познавшим жизнь:
  - Мир грязного функционального программирования
  - Поломать мозги над решением задач в одну строку

Вдохновение

- Изучение Elixir

```
defp line_to_point(line) do
   line
   |> String.trim()
   |> String.split(" ")
   |> Enum.map(&Float.parse/1)
   |> Enum.map(&elem(&1, 0))
end
```

Вдохновение

- Изучение Elixir
- Простые задачи на LeetCode

```
class Solution:
    def destCity(self, paths: List[List[str]]) -> str:
        dp = {a: b for a, b in paths}
        node = paths[0][0]
        while node in dp:
            node = dp[node]
        return node
```

Вдохновение

- Изучение Elixir
- Простые задачи на LeetCode

```
class Solution:
    def destCity(self, paths: List[List[str]]) -> str:
        return reduce(lambda x, y: y - x, map(set, zip(*paths))).pop()
```

Вдохновение

- Изучение Elixir
- Простые задачи на LeetCode

```python
class Solution:
    def minimumDeletions(self, s: str) -> int:
        ans = inf
        a_right, b_left = s.count('a'), 0
        if a_right == 0 or a_right == len(s):
            return 0

        for x in s:
            a_right -= x == 'a'
            ans = min(ans, b_left + a_right)
            b_left += x == 'b'

        return ans
```

Вдохновение

- Изучение Elixir
- Простые задачи на LeetCode

Вдохновение

- Изучение Elixir
- Простые задачи на LeetCode

```
class Solution:
   def minimumDeletions(self, s: str) -> int:
       return (
           0
           if (a_right := s.count('a')) in (0, len(s))
           else reduce(
               lambda prev, x: {
                   'ans': min(prev['ans'], prev['b_left'] + prev['a_right'] - (x == 'a')),
                   'a_right': prev['a_right'] - (x == 'a'),
                   'b_left': prev['b_left'] + (x == 'b'),
               },
               s,
               {'ans': inf, 'a_right': a_right, 'b_left': 0},
           )['ans']
       )
```

План

1. Введение
   - 1.1. 🤭 Одна строка VS одно выражение
   - 1.2. 🤓 Основные концепции функционального программирования
2. Красивая часть доклада
   - 2.1. 🤓 Функциональное программирование в Python
   - 2.2. 😮 Цепочка ленивых вычислений
   - 2.3. 🤔 Ленивость AND и OR
   - 2.4. 😏 Задачи
3. Ужасная часть доклада
   - 3.1. 🤮 Грязное функциональное программирование в Python
   - 3.2. 🤯 Задачи
4. Заключение

Введение

Терминология

- **Выражение** – код, который после исполнения возвращает какое-либо значение

Терминология

- **Выражение** – код, который после исполнения возвращает какое-либо значение
  - `(7 * a + 13) % 23  # a=2 -> 4`

Терминология

- **Выражение** – код, который после исполнения возвращает какое-либо значение
  - `(7 * a + 13) % 23` `# a=2 -> 4`
  - `[x ** 2 for x in range(1, 6, 2)]` `# [1, 9, 25]`

Терминология

- **Выражение** – код, который после исполнения возвращает какое-либо значение
  - `(7 * a + 13) % 23` `# a=2 -> 4`
  - `[x ** 2 for x in range(1, 6, 2)]` `# [1, 9, 25]`
  - `operator.add(2, 3)` `# 5`

Терминология

- **Выражение** – код, который после исполнения возвращает какое-либо значение
  - `(7 * a + 13) % 23` `# a=2 -> 4`
  - `[x ** 2 for x in range(1, 6, 2)]` `# [1, 9, 25]`
  - `operator.add(2, 3)` `# 5`
  - `print('Hello world')` `# None`

Терминология

- **Выражение** – код, который после исполнения возвращает какое-либо значение
  - `(7 * a + 13) % 23` `# a=2 -> 4`
  - `[x ** 2 for x in range(1, 6, 2)]` `# [1, 9, 25]`
  - `operator.add(2, 3)` `# 5`
  - `print('Hello world')` `# None`
- **Statement** (инструкция, оператор) – автономная часть, конкретное действие

Терминология

- **Выражение** – код, который после исполнения возвращает какое-либо значение
  - `(7 * a + 13) % 23` `# a=2 -> 4`
  - `[x ** 2 for x in range(1, 6, 2)]` `# [1, 9, 25]`
  - `operator.add(2, 3)` `# 5`
  - `print('Hello world')` `# None`
- **Statement** (инструкция, оператор) – автономная часть, конкретное действие
  - `return True`

Терминология

- **Выражение** – код, который после исполнения возвращает какое-либо значение
  - `(7 * a + 13) % 23` `# a=2 -> 4`
  - `[x ** 2 for x in range(1, 6, 2)]` `# [1, 9, 25]`
  - `operator.add(2, 3)` `# 5`
  - `print('Hello world')` `# None`
- **Statement** (инструкция, оператор) – автономная часть, конкретное действие
  - `return True`
  - `for _ in range(10): ...`

Терминология

- **Выражение** – код, который после исполнения возвращает какое-либо значение
  - `(7 * a + 13) % 23` `# a=2 -> 4`
  - `[x ** 2 for x in range(1, 6, 2)]` `# [1, 9, 25]`
  - `operator.add(2, 3)` `# 5`
  - `print('Hello world')` `# None`
- **Statement** (инструкция, оператор) – автономная часть, конкретное действие
  - `return True`
  - `for _ in range(10): ...`
  - `while True: ...`

Терминология

- **Выражение** – код, который после исполнения возвращает какое-либо значение
  - `(7 * a + 13) % 23` `# a=2 -> 4`
  - `[x ** 2 for x in range(1, 6, 2)]` `# [1, 9, 25]`
  - `operator.add(2, 3)` `# 5`
  - `print('Hello world')` `# None`
- **Statement** (инструкция, оператор) – автономная часть, конкретное действие
  - `return True`
  - `for _ in range(10): ...`
  - `while True: ...`
- Каждое *Выражение* – *Statement*, но не каждый *Statement* – *Выражение*

Терминология

- **Выражение** – код, который после исполнения возвращает какое-либо значение
  - `(7 * a + 13) % 23` `# a=2 -> 4`
  - `[x ** 2 for x in range(1, 6, 2)]` `# [1, 9, 25]`
  - `operator.add(2, 3)` `# 5`
  - `print('Hello world')` `# None`
- **Statement** (инструкция, оператор) – автономная часть, конкретное действие
  - `return True`
  - `for _ in range(10): ...`
  - `while True: ...`
- Каждое *Выражение* – *Statement*, но не каждый *Statement* – *Выражение*
- `x = yield 5` — *Выражение* или *Statement*?

Одна строчка VS одно выражение

- Одна строчка, несколько *Statement*

```
result = 0; for elem in range(1, n): result += elem
```

Одна строчка VS одно выражение

- Одна строчка, несколько *Statement*

```
result = 0; for elem in range(1, n): result += elem
```

- Один *Expression*, одна строчка

```
sum(elem for elem in range(1, n))
```

Одна строчка VS одно выражение

- Одна строчка, несколько *Statement*

```
result = 0; for elem in range(1, n): result += elem
```

- Один *Expression*, одна строчка

```
sum(elem for elem in range(1, n))
```

- Один *Expression*, несколько строчек

```
sum(
    elem
    for elem in range(1, n)
)
```

Одна строчка VS одно выражение

- ~~Одна строчка, несколько *Statement*~~

  ```~~result = 0; for elem in range(1, n): result += elem~~```
- Один *Expression*, одна строчка

  ```
  sum(elem for elem in range(1, n))
  ```
- Один *Expression*, несколько строчек

  ```
  sum(
      elem
      for elem in range(1, n)
  )
  ```
  ```

Функциональное программирование

- Функции высших порядков
- Чистые функции
- Неизменяемое состояние

Функции высших порядков

Принимает или возвращает другую функцию

- `map(int, input().split())` `# принимает функцию int`
- `functools.cache(heavy_function)` `# возвращает новую функцию`

Чистые функции

Вход однозначно определяет выход. Отсутствуют побочные эффекты

Чистые функции

Вход однозначно определяет выход. Отсутствуют побочные эффекты

```
# грязная, работает с вводом пользователя
def square():
    return int(input()) ** 2
```

Чистые функции

Вход однозначно определяет выход. Отсутствуют побочные эффекты

```
# грязная, печатает результат
def square(x):
    print(x ** 2)
```

Чистые функции

Вход однозначно определяет выход. Отсутствуют побочные эффекты

```
# грязная, изменяет состояние
def square():
    global x
    x **= 2
    return x
```

Чистые функции

Вход однозначно определяет выход. Отсутствуют побочные эффекты

```
# чистая
def square(x):
    return x ** 2
```

Неизменяемое состояние

```
# изменяет состояние аргумента
def append_to_list(lst: list[int], elem: int) -> list:
    lst.append(elem)
    return lst
```

Неизменяемое состояние

```
# изменяет состояние аргумента
def append_to_list(lst: list[int], elem: int) -> list:
    lst.append(elem)
    return lst

# создает новый список
def append_to_list(lst: list[int], elem: int) -> list:
    return lst + [elem]
```

Функциональщина в Python

Моржовый оператор :=

- Появился в Python 3.8
- Присваивание, являющееся выражением

```
a = 5   # Statement,    a = 5
a := 5  # Expression,   a = 5
```

Моржовый оператор :=

- Появился в Python 3.8
- Присваивание, являющееся выражением

```
number = int(input())
print(number, number // 2)
```

Моржовый оператор :=

- Появился в Python 3.8
- Присваивание, являющееся выражением

```
number = int(input())
print(number, number // 2)


print(number := int(input()), number // 2)
```

Моржовый оператор :=

- Появился в Python 3.8
- Присваивание, являющееся выражением

```
while True:
    user_input = input()
    if user_input == 'X':
        break
    print(user_input)
```

Моржовый оператор :=

- Появился в Python 3.8
- Присваивание, являющееся выражением

```
while (user_input := input()) != 'X':
    print(user_input)
```

Моржовый оператор := и позиционные аргументы

- Появился в Python 3.8
- Присваивание, являющееся выражением

```
function_without_kwargs(name :=  'Alex', age := 22)
```

Моржовый оператор := в генераторных выражениях

- Появился в Python 3.8
- Присваивание, являющееся выражением

```
def string_to_number(string: str) -> int:
    return int(string)


[
    string_to_number(string)
    for string in input().split()
    if string_to_number(string) > 0
]
```

Моржовый оператор := в генераторных выражениях

- Появился в Python 3.8
- Присваивание, являющееся выражением

```
def string_to_number(string: str) -> int:
    return int(string)


[
    number := string_to_number(string)
    for string in input().split()
    if number > 0
]
```

Моржовый оператор := в генераторных выражениях

- Появился в Python 3.8
- Присваивание, являющееся выражением

```
def string_to_number(string: str) -> int:
    return int(string)


[
    number := string_to_number(string)
    for string in input().split()
    if number > 0                       # NameError: name 'number' is not defined
]
```

Моржовый оператор := в генераторных выражениях

- Появился в Python 3.8
- Присваивание, являющееся выражением

```
def string_to_number(string: str) -> int:
    return int(string)


[
    number
    for string in input().split()
    if (number := string_to_number(string)) > 0
]
```

- Сначала фильтрация, потом отображение!

```
    number if (number := string_to_number(string)) > 0 else None
```

Инструментарий для ФП в Python

- `map` – отображение

```
list(map(lambda x: x // 2, [10, 20, 30]))          # [5, 10, 15]
```

Инструментарий для ФП в Python

- `map` – отображение

```
list(map(lambda x: x // 2, [10, 20, 30]))          # [5, 10, 15]
```

- `filter` – фильтрация

```
list(filter(lambda x: x % 2 == 0, [1, 7, 6, 2]))   # [6, 2]
```

Инструментарий для ФП в Python

- `map` – отображение

```
list(map(lambda x: x // 2, [10, 20, 30]))          # [5, 10, 15]
```

- `filter` – фильтрация

```
list(filter(lambda x: x % 2 == 0, [1, 7, 6, 2]))   # [6, 2]
```

- `reduce` – свертка

```
from functools import reduce
reduce(lambda acc, x: acc + x, [1, 3, 3, 7])       # 14
```

Инструментарий для ФП в Python

- `map` – отображение

```
list(map(lambda x: x // 2, [10, 20, 30]))          # [5, 10, 15]
```

- `filter` – фильтрация

```
list(filter(lambda x: x % 2 == 0, [1, 7, 6, 2]))   # [6, 2]
```

- `reduce` – свертка

```
from functools import reduce
reduce(lambda acc, x: acc + x, [1, 3, 3, 7])       # 14
```

- `zip` – упаковка

```
names = ['Alice', 'Bob']
ages = [20, 28]
list(zip(names, ages))              # [('Alice', 20), ('Bob', 28)]
```

Бесконечные последовательности

```
from itertools import count, cycle, repeat

count(1)           # 1, 2, 3, 4, ...
```

Бесконечные последовательности

```
from itertools import count, cycle, repeat

count(1)           # 1, 2, 3, 4, ...

cnt = 1
while True:
   if something:
       print(cnt)
       break
   cnt += 1
```

Бесконечные последовательности

```
from itertools import count, cycle, repeat

count(1)           # 1, 2, 3, 4, ...

counter = count(1)
while cnt := next(counter):
   if something:
       print(cnt)
       break
```

Бесконечные последовательности

```
from itertools import count, cycle, repeat

count(1)           # 1, 2, 3, 4, ...

cycle([1, 2, 3])   # 1, 2, 3, 1, 2, 3, ...
```

Бесконечные последовательности

```
from itertools import count, cycle, repeat

count(1)           # 1, 2, 3, 4, ...

cycle([1, 2, 3])   # 1, 2, 3, 1, 2, 3, ...

repeat(8)          # 8, 8, 8, 8, ...
```

Еще несколько интересных функций

```
from itertools import accumulate, compress, groupby, pairwise, takewhile

list(accumulate([1, 2, 3, 4], initial=0))  # [0, 1, 3, 6, 10]
```

Еще несколько интересных функций

```
from itertools import accumulate, compress, groupby, pairwise, takewhile

list(accumulate([1, 2, 3, 4], initial=0))  # [0, 1, 3, 6, 10]

gain = [5, -8, 4, -2, 3, -2, 5]
current_gain = max_gain = 0
for x in gain:
   current_gain += x                       # 5, -3, 1, -1, 2, 0, 5
   max_gain = max(max_gain, current_gain
max_gain                                   # 5
```

Еще несколько интересных функций

```
from itertools import accumulate, compress, groupby, pairwise, takewhile

list(accumulate([1, 2, 3, 4], initial=0))  # [0, 1, 3, 6, 10]

gain = [5, -8, 4, -2, 3, -2, 5]
max_gain = max(accumulate(gain, initial=0))  # max(0, 5, -3, 1, -1, 2, 0, 5) -> 5
```

Еще несколько интересных функций

```
from itertools import accumulate, compress, groupby, pairwise, takewhile

list(accumulate([1, 2, 3, 4], initial=0))  # [0, 1, 3, 6, 10]

''.join(f'{k}{len(list(v))}' for k, v in groupby('AAAABBBCCDAABBB')) # A4B3C2D1A2B3
```

Еще несколько интересных функций

```
from itertools import accumulate, compress, groupby, pairwise, takewhile

list(accumulate([1, 2, 3, 4], initial=0))  # [0, 1, 3, 6, 10]

''.join(f'{k}{len(list(v))}' for k, v in groupby('AAAABBBCCDAABBB')) # A4B3C2D1A2B3

list(compress([1, 2, 3, 4], [0, 1, 0, 1])) # [2, 4]
```

Еще несколько интересных функций

```
from itertools import accumulate, compress, groupby, pairwise, takewhile

list(accumulate([1, 2, 3, 4], initial=0))  # [0, 1, 3, 6, 10]

''.join(f'{k}{len(list(v))}' for k, v in groupby('AAAABBBCCDAABBB')) # A4B3C2D1A2B3

list(compress([1, 2, 3, 4], [0, 1, 0, 1])) # [2, 4]

values = (2, 38, 32)
combinations = [(0, 0, 1), (1, 0, 1), (0, 1, 1)]
for combination in combinations:
   selected_values = compress(values, combination)
```

Еще несколько интересных функций

```
from itertools import accumulate, compress, groupby, pairwise, takewhile

list(accumulate([1, 2, 3, 4], initial=0))  # [0, 1, 3, 6, 10]

''.join(f'{k}{len(list(v))}' for k, v in groupby('AAAABBBCCDAABBB')) # A4B3C2D1A2B3

list(compress([1, 2, 3, 4], [0, 1, 0, 1])) # [2, 4]

list(pairwise('ABCDEFG'))  # ['AB', 'BC', 'CD', 'DE', 'EF', 'FG']
```

Еще несколько интересных функций

```
from itertools import accumulate, compress, groupby, pairwise, takewhile

list(accumulate([1, 2, 3, 4], initial=0))  # [0, 1, 3, 6, 10]

''.join(f'{k}{len(list(v))}' for k, v in groupby('AAAABBBCCDAABBB')) # A4B3C2D1A2B3

list(compress([1, 2, 3, 4], [0, 1, 0, 1])) # [2, 4]

list(pairwise('ABCDEFG'))  # ['AB', 'BC', 'CD', 'DE', 'EF', 'FG']

numbers = [5, 6, 0, 4, 8, 13]
min(b - a for a, b in pairwise(sorted(numbers)))  # 1
```

Еще несколько интересных функций

```
from itertools import accumulate, compress, groupby, pairwise, takewhile

list(accumulate([1, 2, 3, 4], initial=0))  # [0, 1, 3, 6, 10]

''.join(f'{k}{len(list(v))}' for k, v in groupby('AAAABBBCCDAABBB')) # A4B3C2D1A2B3

list(compress([1, 2, 3, 4], [0, 1, 0, 1])) # [2, 4]

list(pairwise('ABCDEFG'))  # ['AB', 'BC', 'CD', 'DE', 'EF', 'FG']

list(takewhile(lambda x: x < 5, [1, 4, 6, 3, 8]))  # [1, 4]
```

Еще несколько интересных функций

```
from itertools import accumulate, compress, groupby, pairwise, takewhile

list(accumulate([1, 2, 3, 4], initial=0))  # [0, 1, 3, 6, 10]

''.join(f'{k}{len(list(v))}' for k, v in groupby('AAAABBBCCDAABBB')) # A4B3C2D1A2B3

list(compress([1, 2, 3, 4], [0, 1, 0, 1])) # [2, 4]

list(pairwise('ABCDEFG'))  # ['AB', 'BC', 'CD', 'DE', 'EF', 'FG']

list(takewhile(lambda x: x < 5, [1, 4, 6, 3, 8]))  # [1, 4]

command_generator = ('hello', 'world', ':wq', 'bye vim')
takewhile(lambda x: x != ':wq', command_generator)  # hello, world
```

Цепочка ленивых вычислений

```
numbers = range(10)
odd_numbers = filter(lambda x: x % 2, numbers)
squared_numbers = map(lambda x: x ** 2, odd_numbers)
result = sum(squared_numbers)
print(result)
```

Цепочка ленивых вычислений

```
numbers = range(10)
odd_numbers = filter(lambda x: x % 2, numbers)
squared_numbers = map(lambda x: x ** 2, odd_numbers)
result = sum(squared_numbers)
print(result)
```

Цепочка ленивых вычислений

```
numbers = range(10)
odd_numbers = filter(lambda x: x % 2, numbers)
squared_numbers = map(lambda x: x ** 2, odd_numbers)
result = sum(squared_numbers)
print(result)
```

Цепочка ленивых вычислений

```
numbers = range(10)
odd_numbers = filter(lambda x: x % 2, numbers)
squared_numbers = map(lambda x: x ** 2, odd_numbers)
result = sum(squared_numbers)
print(result)
```

Цепочка ленивых вычислений

```
numbers = range(10) # 0
odd_numbers = filter(lambda x: x % 2, numbers)
squared_numbers = map(lambda x: x ** 2, odd_numbers)
result = sum(squared_numbers)
print(result)
```

Цепочка ленивых вычислений

```
numbers = range(10) # 0
odd_numbers = filter(lambda x: x % 2, numbers) # 0 % 2 = 0
squared_numbers = map(lambda x: x ** 2, odd_numbers)
result = sum(squared_numbers)
print(result)
```

Цепочка ленивых вычислений

```
numbers = range(10) # 1
odd_numbers = filter(lambda x: x % 2, numbers)
squared_numbers = map(lambda x: x ** 2, odd_numbers)
result = sum(squared_numbers)
print(result)
```

Цепочка ленивых вычислений

```
numbers = range(10) # 1
odd_numbers = filter(lambda x: x % 2, numbers) # 1 % 2 = 1
squared_numbers = map(lambda x: x ** 2, odd_numbers)
result = sum(squared_numbers)
print(result)
```

Цепочка ленивых вычислений

```
numbers = range(10) # 1
odd_numbers = filter(lambda x: x % 2, numbers) # 1 % 2 = 1
squared_numbers = map(lambda x: x ** 2, odd_numbers)
result = sum(squared_numbers)
print(result)
```

Цепочка ленивых вычислений

```
numbers = range(10) # 1
odd_numbers = filter(lambda x: x % 2, numbers) # 1 % 2 = 1
squared_numbers = map(lambda x: x ** 2, odd_numbers) # 1 ** 2 = 1
result = sum(squared_numbers)
print(result)
```

Цепочка ленивых вычислений

```
numbers = range(10) # 1
odd_numbers = filter(lambda x: x % 2, numbers) # 1 % 2 = 1
squared_numbers = map(lambda x: x ** 2, odd_numbers) # 1 ** 2 = 1
result = sum(squared_numbers) # _result += 1
print(result)
```

Ленивость AND и OR

- Логические операции: `and`, `or`
- Побитовые операции: `&`, `|`
- Зачем тогда нужны логические?

Ленивость AND и OR

- `1 & 1 & 0 & 0`
- `1 and 1 and 0 and 0`

Ленивость AND и OR

- `1 & 1 & 0 & 0`
- `1 and 1 and 0 and 0`

Ленивость AND и OR

- `1 & 1 & 0 & 0`
- `1 and 1 and 0 and 0`

Ленивость AND и OR

- Логические И/ИЛИ возвращают не `bool`!
- ИЛИ: первый истинный операнд **или последний**
- И: первый ложный операнд **или последний**

Ленивость AND и OR

- Логические И/ИЛИ возвращают не `bool`!
- ИЛИ: первый истинный операнд **или последний**
- И: первый ложный операнд **или последний**

```
True or 1                # True
1 or True                # 1
False or 'Hello World'  # 'Hello world'

True and 5               # 5
5 and True               # True
False and 'Hello world' # False
```

Ленивость AND и OR

- Логические И/ИЛИ возвращают не `bool`!
- ИЛИ: первый истинный операнд **или последний**
- И: первый ложный операнд **или последний**

JavaScript

```
pageTitle && <Header>{pageTitle}<Header/> || <Loader/>
```

Ленивость AND и OR

- Логические И/ИЛИ возвращают не `bool`!
- ИЛИ: первый истинный операнд **или последний**
- И: первый ложный операнд **или последний**

JavaScript

```
pageTitle && <Header>{pageTitle}<Header/> || <Loader/>
```

Unix

```
mkdir test && echo "Something" || echo "Failed to create folder"
```

Ленивость AND и OR

- Логические И/ИЛИ возвращают не `bool`!
- ИЛИ: первый истинный операнд **или последний**
- И: первый ложный операнд **или последний**

JavaScript

```
pageTitle && <Header>{pageTitle}<Header/> || <Loader/>
```

Unix

```
mkdir test && echo "Something" || echo "Failed to create folder"
```

Python

```
age = user and user.age or -1
```

Ленивость AND и OR

Используется в функциях `all` и `any`

```
all([True, True, False, True]) # False
any([False, True, True, True]) # True
```

Задача 1

Найти сумму четных чисел подаваемого на вход списка

Задача 1

Найти сумму четных чисел подаваемого на вход списка

```
result = 0
for x in nums:
    if x % 2 == 0:
        result += x
```

Задача 1

Найти сумму четных чисел подаваемого на вход списка

```
result = sum(x for x in nums if x % 2 == 0)
```

Задача 1

Найти сумму четных чисел подаваемого на вход списка

```
result = sum(x for x in nums if x % 2 == 0)
```

```
result = sum(filter(lambda x: x % 2 == 0, nums))
```

Задача 2

Количество уникальных элементов в списке

Задача 2

Количество уникальных элементов в списке

```
unique = set()
for x in nums:
    unique.add(x)
result = len(unique)
```

Задача 2

Количество уникальных элементов в списке

```
result = len(set(nums))
```

Задача 3

Составить словарь частот

Задача 3

Составить словарь частот

```
result = {}
for x in nums:
    result[x] = result.get(x,  0) + 1
```

Задача 3

Составить словарь частот

```
from collections import Counter
result = Counter(nums)
```

Задача 4

Являются ли две строки анаграммами?

```
is_anagram('abcaa', 'bcaaa')  # True
is_anagram('abcaa', 'bcaad')  # False
```

Задача 4

Являются ли две строки анаграммами?

```
def is_anagram(s1: str, s2: str) -> bool:
    freq = {}
    for i in range(len(s1)):
        freq[s1[i]] = freq.get(s1[i],0) + 1
        freq[s2[i]] = freq.get(s2[i],0) - 1

    for key in freq:
        if freq[key] != 0:
            return False

    return True
```

Задача 4

Являются ли две строки анаграммами?

```
def is_anagram(s1: str, s2: str) -> bool:
    freq = {}
    for c1, c2 in zip(s1, s2):
        freq[c1] = freq.get(c1, 0) + 1
        freq[c2] = freq.get(c2, 0) - 1

    for key in freq:
        if freq[key] != 0:
            return False

    return True
```

Задача 4

Являются ли две строки анаграммами?

```
from collections import defaultdict

def is_anagram(s1: str, s2: str) -> bool:
    freq = defaultdict(int)
    for c1, c2 in zip(s1, s2):
        freq[c1] += 1
        freq[c2] -= 1

    for key in freq:
        if freq[key] != 0:
            return False

    return True
```

Задача 4

Являются ли две строки анаграммами?

```
from collections import defaultdict

def is_anagram(s1: str, s2: str) -> bool:
    freq = defaultdict(int)
    for c1, c2 in zip(s1, s2):
        freq[c1] += 1
        freq[c2] -= 1

    return all(freq[key] == 0 for key in freq)
```

Задача 4

Являются ли две строки анаграммами?

```
from collections import Counter


def is_anagram(s1: str, s2: str) -> bool:
    return Counter(s1) == Counter(s2)
```

Задача 4

Являются ли две строки анаграммами?

```
def is_anagram(s1: str, s2: str) -> bool:
    return sorted(s1) == sorted(s2)
```

Так и как решить любую задачу однострочником?

- ~~Чистый код~~
- Грязный рот

Грязная функциональщина

Задача 3

Составить словарь частот

Задача 3

Составить словарь частот

```
{x: nums.count(x)  for x in nums}
```

Задача 3

Составить словарь частот

```
result = {x: result.get(x,  0) + 1 for x in nums}   # NameError
```

Задача 3

Составить словарь частот

```
[result.__setitem__(x, result.get(x, 0) + 1) for x in nums]
```

Задача 3

Составить словарь частот

```
(result := {}) or [result.__setitem__(x, result.get(x, 0) + 1) for x in nums]
```

Задача 3

Составить словарь частот

```
(
   (result := {})
   or [result.__setitem__(x, result.get(x, 0) + 1) for x in nums]
   and False
   or result
)
```

Задача 5

Вернуть первый положительный элемент списка. Если нет, вернуть None

Задача 5

Вернуть первый положительный элемент списка. Если нет, вернуть None

```
def find_first_positive_or_none (nums):
 for x in nums:
   if x > 0:
     return x
 return None
```

Задача 5

Вернуть первый положительный элемент списка. Если нет, вернуть None

```
lst[0] if (lst := [x for x in nums if x > 0]) else None
```

Задача 5

Вернуть первый положительный элемент списка. Если нет, вернуть None

```
def find_first_positive_or_none (nums):
    return next(filter(lambda x: x > 0, nums), None)
```

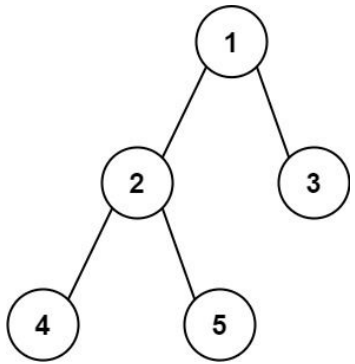
next(*iterator*)
next(*iterator*, *default*)

Retrieve the next item from the iterator by calling its `__next__()` method. If *default* is given, it is returned if the iterator is exhausted, otherwise `StopIteration` is raised.

Задача 6

Реализовать обратный обход дерева поиском в глубину

```
@dataclass
class Node:
    val: int
    left: Optional['Node'] = None
    right: Optional['Node'] = None
```

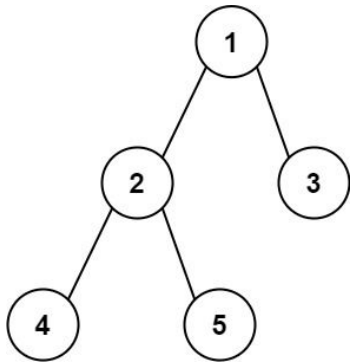


[4, 5, 2, 3, 1]

Задача 6

Реализовать обратный обход дерева поиском в глубину

```
def dfs(node) -> list[int]:
    ans = []
    if node.left:
        ans.extend(dfs(node.left))
    if node.right:
        ans.extend(dfs(node.right))
    return ans + [node.val]
```

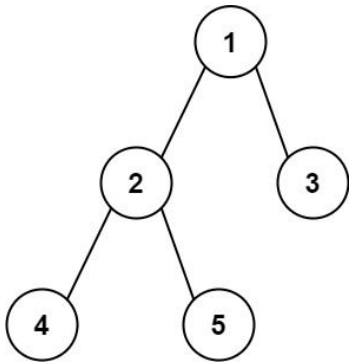


[4, 5, 2, 3, 1]

Задача 6

Реализовать обратный обход дерева поиском в глубину

```
dfs = lambda node: (
    (node.left and dfs(node.left) or []) +
    (node.right and dfs(node.right) or []) +
    [node.val]
)
```



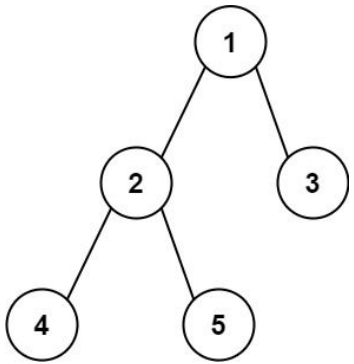
[4, 5, 2, 3, 1]

Задача 6

Реализовать обратный обход дерева поиском в глубину

```
dfs = lambda node: (
    (node.left and dfs(node.left) or []) +
    (node.right and dfs(node.right) or []) +
    [node.val]
)
```

- Операнды вычисляются лениво!

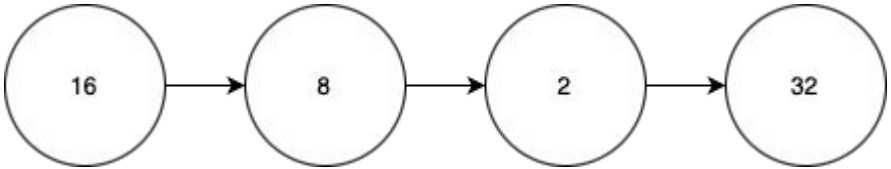


[4, 5, 2, 3, 1]

Задача 7

Итерация по связному списку **без рекурсии**

```
@dataclass
class Node:
    val: int
    next: Optional['Node'] = None
```

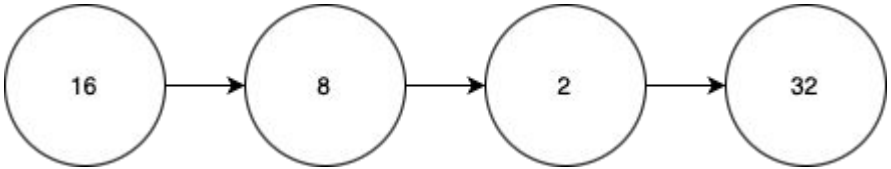


Задача 7

Итерация по связному списку **без рекурсии**

```
@dataclass
class Node:
    val: int
    next: Optional['Node'] = None


def imperative(node):
    while node:
        yield node.val
        node = node.next
```



[16, 8, 2, 32]

Задача 7

- В Python нет while-comprehension
- `filter`, `map`, `reduce` принимают на вход итератор
- И как быть?

Задача 7

- Нужно генерировать бесконечную последовательность

`itertools.`**`count`**`(`*`start=0, step=1`*`)`

Make an iterator that returns evenly spaced values beginning with *start*. Can be used with `map()` to generate consecutive data points or with `zip()` to add sequence numbers.

- Нужно в какой-то момент остановиться

`itertools.`**`takewhile`**`(`*`predicate, iterable`*`)`

Make an iterator that returns elements from the *iterable* as long as the *predicate* is true.

Задача 7

Изменение состояния внутри генератора

```
(node := node.next  for  _  in  count())
```

Задача 7

Останавливаемся, если вышли за пределы списка

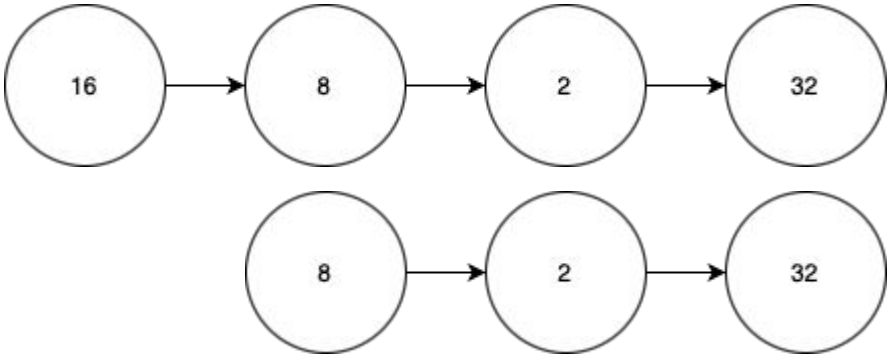
```
takewhile(bool, (node := node.next for _ in count()))
```

Задача 7

Останавливаемся, если вышли за пределы списка

```
takewhile(bool, (node := node.next for _ in count()))
```

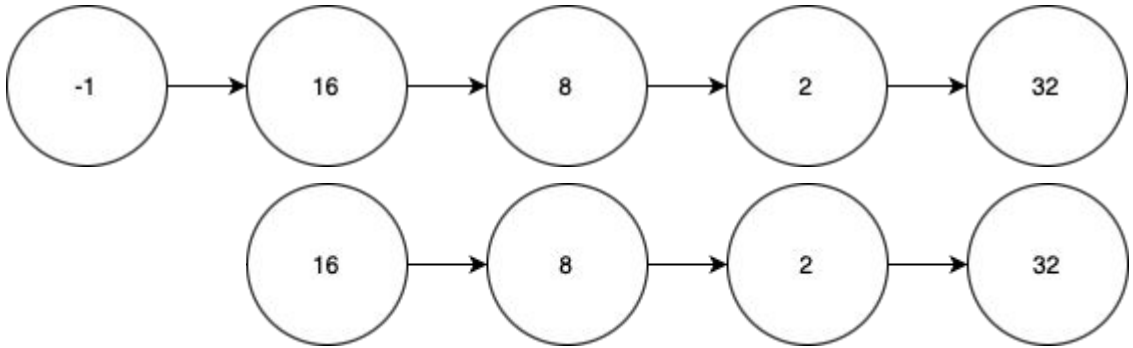
А голову ты дома не забыл?



Задача 7

Добавляем липовый узел как в роли предшествующего состояния

```
(node := Node(-1, node)) and takewhile(
    bool,
    (node := node.next for _ in count())
)
```



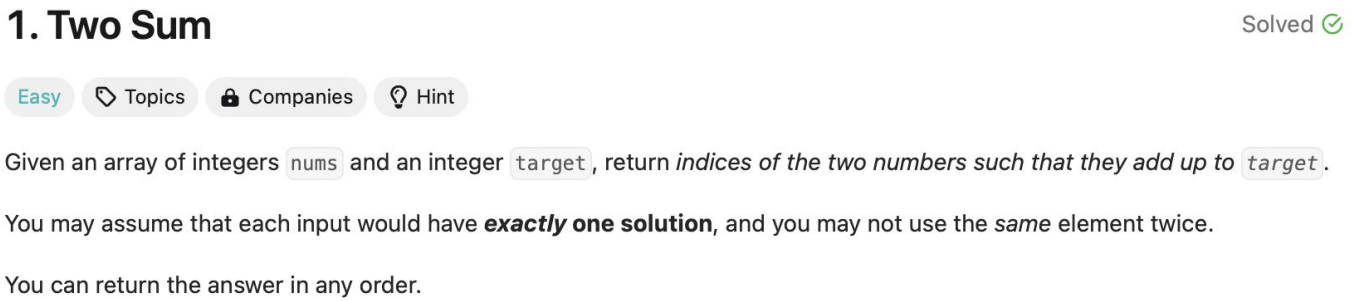
Задача 7

Итерация по связному списку **без рекурсии**

```
def functional(node):
    return (node := Node(-1, node)) and map(
        lambda x: x.val,
        takewhile(
            bool,
            (node := node.next  for _ in count())
        )
    )
```

Задача 8

Найти индексы элементов массива `nums`, сумма которых равна `target`



1. Two Sum

Solved

Easy | Topics | Companies | Hint

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to* `target`.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Задача 8

Найти индексы элементов массива `nums`, сумма которых равна `target`

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        indices = {}
        for i, num in enumerate(nums):
            if target - num in indices:
                return [i, indices[target - num]]
            indices[num] = i
        return [-1, -1]
```

Задача 8

Найти индексы элементов массива `nums`, сумма которых равна `target`

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        indices = {}
        for i, num in enumerate(nums):
            if target - num in indices:
                return [i, indices[target - num]]
            indices[num] = i
        return [-1, -1]
```

Задача 8

Найти индексы элементов массива `nums`, сумма которых равна `target`

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        indices = {}
        for i, num in enumerate(nums):
            if target - num in indices:
                return [i, indices[target - num]]
            indices[num] = i
        return [-1, -1]
```

Задача 8

Найти индексы элементов массива `nums`, сумма которых равна `target`

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        return (indices := {}) or next(
            (
                [i, indices[target - num]]
                for i, num in enumerate(nums)
                if target - num in indices or indices.__setitem__(num, i)
            ),
            [-1, -1],
        )
```

Задача 8

Найти индексы элементов массива `nums`, сумма которых равна `target`

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        return (indices := {}) or next(
            (
                [i, indices[target - num]]
                for i, num in enumerate(nums)
                if target - num in indices or indices.__setitem__(num, i)
            ),
            [-1, -1],
        )
```

Задача 8

Найти индексы элементов массива `nums`, сумма которых равна `target`

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        return (indices := {}) or next(
            (
                [i, indices[target - num]]
                for i, num in enumerate(nums)
                if target - num in indices or indices.__setitem__(num, i)
            ),
            [-1, -1],
        )
```

Задача 8

Найти индексы элементов массива `nums`, сумма которых равна `target`

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        return (indices := {}) or next(
            (
                [i, indices[target - num]]
                for i, num in enumerate(nums)
                if target - num in indices or indices.__setitem__(num, i)
            ),
            [-1, -1],
        )
```

Задача 9 (последняя)

Максимум нулей слева и единиц справа

1422. Maximum Score After Splitting a String

Easy | Topics | Companies | Hint

Given a string `s` of zeros and ones, *return the maximum score after splitting the string into two **non-empty** substrings* (i.e. **left** substring and **right** substring).

The score after splitting a string is the number of **zeros** in the **left** substring plus the number of **ones** in the **right** substring.

Example 1:

```
Input: s = "011101"
Output: 5
Explanation:
All possible ways of splitting s into two non-empty substrings are:
left = "0" and right = "11101", score = 1 + 4 = 5
left = "01" and right = "1101", score = 1 + 3 = 4
left = "011" and right = "101", score = 1 + 2 = 3
left = "0111" and right = "01", score = 1 + 1 = 2
left = "01110" and right = "1", score = 2 + 1 = 3
```

Задача 9 (последняя)

Максимум нулей слева и единиц справа

```
class Solution:
    def maxScore(self, string: str) -> int:
        zeros_left, ones_right = 0, string.count('1')
        answer = 0
        for symbol in string[:-1]:
            if symbol == '0':
                zeros_left += 1
            else:
                ones_right -= 1
            answer = max(answer, zeros_left + ones_right)
        return answer
```

Задача 9 (последняя)

Максимум нулей слева и единиц справа

```
class Solution:
    def maxScore(self, string: str) -> int:
        zeros_left, ones_right = 0, string.count('1')
        answer = 0
        for symbol in string[:-1]:
            if symbol == '0':
                zeros_left +=1
            else:
                ones_right -=1
            answer = max(answer, zeros_left + ones_right)
        return answer
```

Задача 9 (последняя)

Максимум нулей слева и единиц справа

```
class Solution:
    def maxScore(self, string: str) -> int:
        zeros_left, ones_right = 0, string.count('1')
        answer = 0
        for symbol in string[:-1]:
            if symbol == '0':
                zeros_left += 1
            else:
                ones_right -= 1
            answer = max(answer, zeros_left + ones_right)
        return answer
```

Задача 9 (последняя)

Максимум нулей слева и единиц справа

```
class Solution:
    def maxScore(self, string: str) -> int:
        zeros_left, ones_right = 0, string.count('1')
        answer = 0
        for symbol in string[:-1]:
            if symbol == '0':
                zeros_left += 1
            else:
                ones_right -= 1
            answer = max(answer, zeros_left + ones_right)
        return answer
```

Задача 9 (последняя)

Максимум нулей слева и единиц справа

```
class Solution:
    def maxScore(self, string: str) -> int:
        zeros_left, ones_right = 0, string.count('1')
        answer = 0
        for symbol in string[:-1]:
            if symbol == '0':
                zeros_left += 1
            else:
                ones_right -= 1
            answer = max(answer, zeros_left + ones_right)
        return answer
```

Задача 9 (последняя)

Максимум нулей слева и единиц справа

```
class Solution:
    def maxScore(self, string: str) -> int:
        zeros_left, ones_right =0, string.count('1')
        answer = 0
        for symbol in string[:-1]:
            if symbol == '0':
                zeros_left +=1
            else:
                ones_right -=1
            answer = max(answer, zeros_left + ones_right)
        return answer
```

Задача 9 (последняя)

Максимум нулей слева и единиц справа

```
class Solution:
    def maxScore(self, string: str) -> int:
        return reduce(
            lambda state, symbol: {
                'zeros_left':   (zeros_left := state['zeros_left'] + (symbol == '0')),
                'ones_right':   (ones_right := state['ones_right'] - (symbol == '1')),
                'answer':       max(state['answer'], zeros_left + ones_right),
            },
            string[:-1],
            {
                'zeros_left':   0,
                'ones_right':   string.count('1'),
                'answer':       0,
            },
        )['answer']
```

Задача 9 (последняя)

Максимум нулей слева и единиц справа

```
class Solution:
    def maxScore(self, string: str) -> int:
        return reduce(
            lambda state, symbol: {
                'zeros_left':   (zeros_left := state['zeros_left'] + (symbol == '0')),
                'ones_right':   (ones_right := state['ones_right'] - (symbol == '1')),
                'answer':       max(state['answer'], zeros_left + ones_right),
            },
            string[:-1],
            {
                'zeros_left':   0,
                'ones_right':   string.count('1'),
                'answer':       0,
            },
        )['answer']
```

Задача 9 (последняя)

Максимум нулей слева и единиц справа

```
class Solution:
    def maxScore(self, string: str) -> int:
        return reduce(
            lambda state, symbol: {
                'zeros_left':   (zeros_left := state['zeros_left'] + (symbol == '0')),
                'ones_right':   (ones_right := state['ones_right'] - (symbol == '1')),
                'answer':       max(state['answer'], zeros_left + ones_right),
            },
            string[:-1],
            {
                'zeros_left':   0,
                'ones_right':   string.count('1'),
                'answer':       0,
            },
        )['answer']
```

Задача 9 (последняя)

Максимум нулей слева и единиц справа

```
class Solution:
    def maxScore(self, string: str) -> int:
        return reduce(
            lambda state, symbol: {
                'zeros_left':   (zeros_left := state['zeros_left'] + (symbol == '0')),
                'ones_right':   (ones_right := state['ones_right'] - (symbol == '1')),
                'answer':       max(state['answer'], zeros_left + ones_right),
            },
            string[:-1],
            {
                'zeros_left':   0,
                'ones_right':   string.count('1'),
                'answer':       0,
            },
        )['answer']
```

Задача 9 (последняя)

Максимум нулей слева и единиц справа

```
class Solution:
    def maxScore(self, string: str) -> int:
        return reduce(
            lambda state, symbol: {
                'zeros_left':   (zeros_left := state['zeros_left'] + (symbol == '0')),
                'ones_right':   (ones_right := state['ones_right'] - (symbol == '1')),
                'answer':       max(state['answer'], zeros_left + ones_right),
            },
            string[:-1],
            {
                'zeros_left':   0,
                'ones_right':   string.count('1'),
                'answer':       0,
            },
        )['answer']
```

Задача 9 (последняя)

Максимум нулей слева и единиц справа

```
class Solution:
    def maxScore(self, string: str) -> int:
        return reduce(
            lambda state, symbol: {
                'zeros_left':   (zeros_left := state['zeros_left'] + (symbol == '0')),
                'ones_right':   (ones_right := state['ones_right'] - (symbol == '1')),
                'answer':       max(state['answer'], zeros_left + ones_right),
            },
            string[:-1],
            {
                'zeros_left':   0,
                'ones_right':   string.count('1'),
                'answer':       0,
            },
        )['answer']
```

Любую ли задачу можно так решить?

- `eval('код с несколькими statement')`

Любую ли задачу можно так решить?

- `eval('код с несколькими statement')`
- Вопрос остается открытым

Домашнее задание:

- Решить дейлик на LeetCode одним выражением
- Закинуть однострочник в Merge Request своему ревьюеру
- Доказать полноту по тьюрингу :)

Заключение

- Python обладает богатыми возможностями
- Решения из первой части можно и нужно использовать в работе
- Решения из второй части забудьте как страшный сон
- Хороший однострочник описывает сам себя

Вопросы

Александр Харламов, CodeScoring
Telegram: @wignorbo