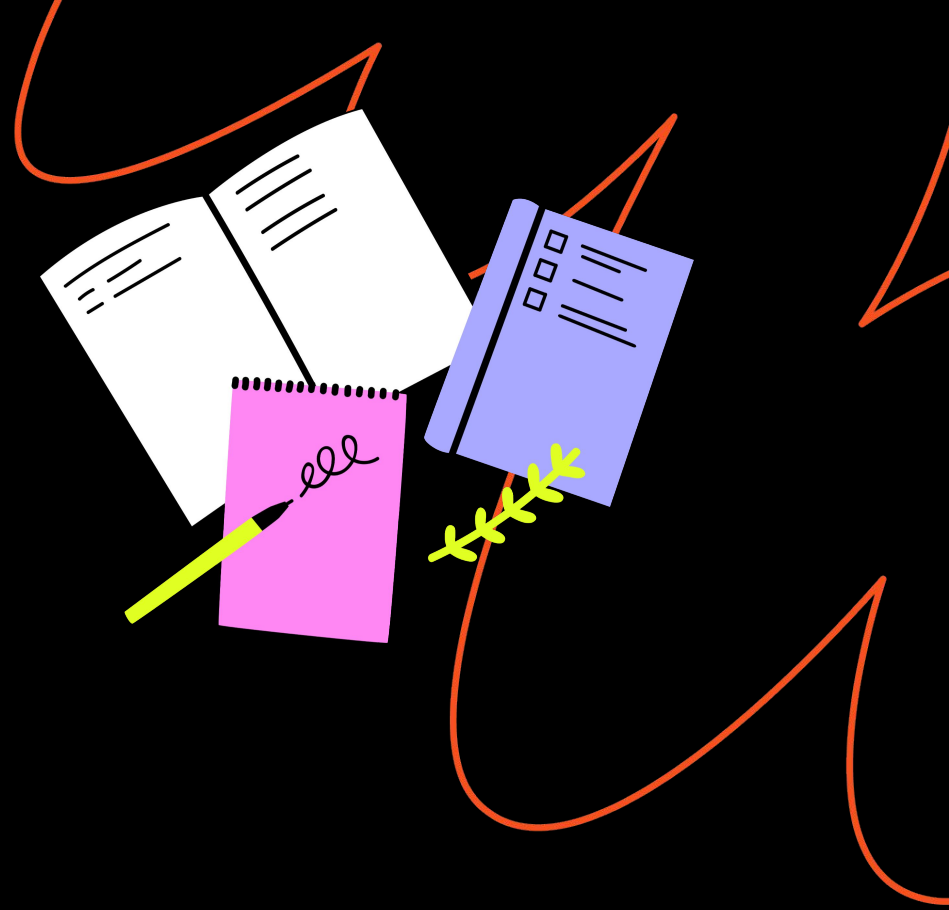
Ускоренная обработка данных: lambda, filter, map, zip, enumerate, List Comprehension

Простое и любопытное



Анонимные, lambda функции

Анонимные, lambda функции

Идея: часто описывать функцию некогда и незачем

Анонимные, lambda функции

```
def sum(x):
    return x+10

def mult(x):
    return x**2

sum(mult(2))
```

```
def sum1(x):
    return x+22

def mult2(x):
    return x**3

sum1(mult2(2))
```

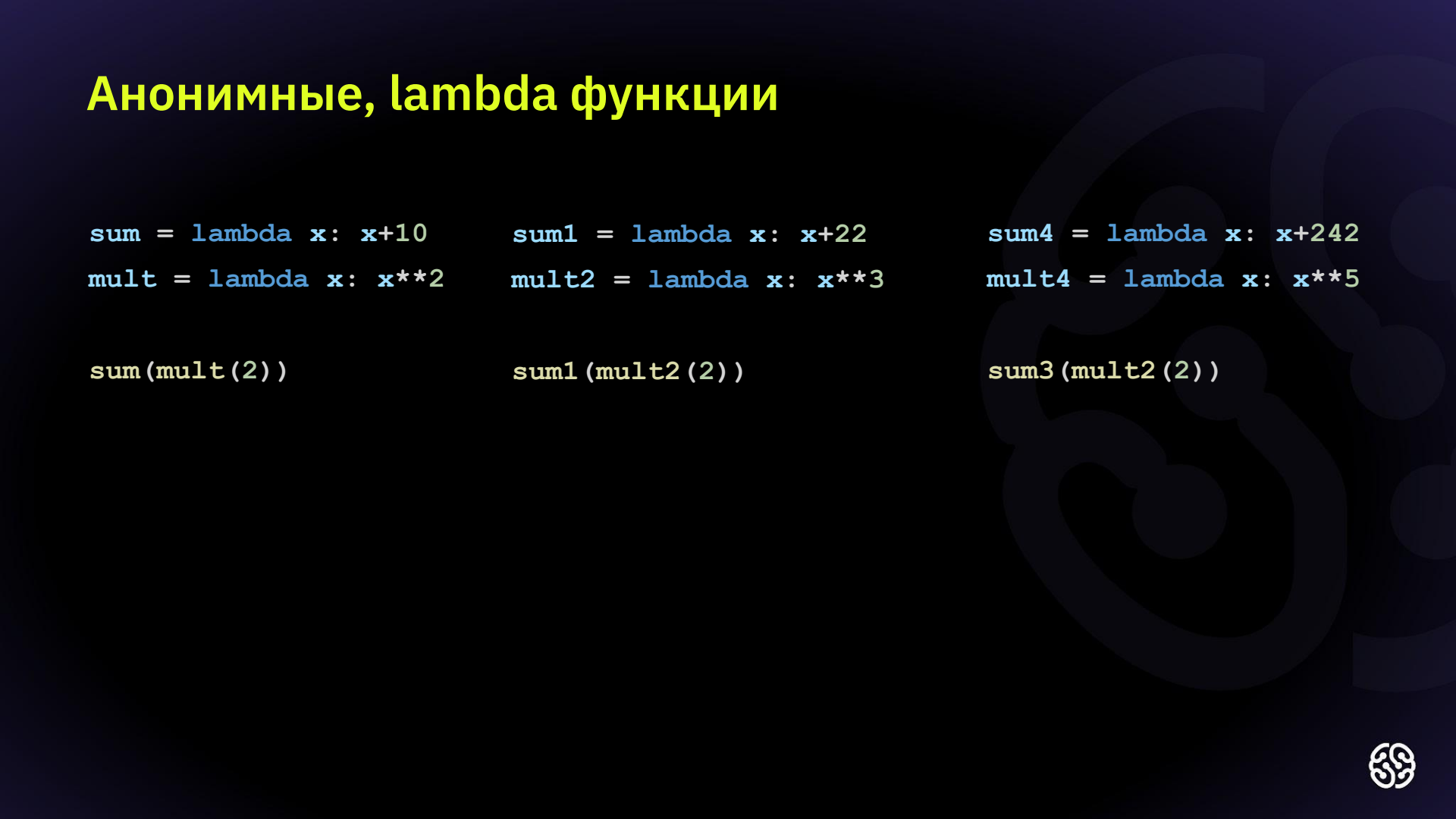
```
def sum3(x):
    return x+242

def mult4(x):
    return x**5

sum3(mult2(2))
```

Анонимные, lambda функции

```
sum = lambda x: x+10
mult = lambda x: x**2

sum(mult(2))
```

```
sum1 = lambda x: x+22
mult2 = lambda x: x**3

sum1(mult2(2))
```

```
sum4 = lambda x: x+242
mult4 = lambda x: x**5

sum3(mult2(2))
```



Анонимные, lambda функции

```
sum = lambda x: x+10
mult = lambda x: x**2

sum1 = lambda x: x+22
mult2 = lambda x: x**3

sum4 = lambda x: x+242
mult4 = lambda x: x**5

sum(mult(2))

sum1(mult2(2))

sum3(mult2(2))

f(g(x))

def h(f, g, x): return f(g(x))h = lambda f, g, x: f(g(x))
h(sum, mult, 5)

h(lambda x: x+10, lambda x: x**2, 5)
```

Анонимные, lambda функции

К чему это всё?
В файле хранятся числа, нужно выбрать четные и составить список пар (число; квадрат числа).
Пример:
1 2 3 5 8 15 23 38
Получить:
[(2, 4), (8, 64), (38, 1444)]

Анонимные, lambda функции

```
f = open('f.txt', 'r')
data = f.read() + ' '
f.close()

numbers = []

while data != '':
    space_pos = data.index(' ')
    numbers.append(int(data[:space_pos]))
    data = data[space_pos+1:]

out = []
for e in numbers:
    if not e % 2:
        out.append((e,e **2))
print(out)
```

Анонимные, lambda функции

```
def select(f, col):
    return [f(x) for x in col]

def where(f, col):
    return [x for x in col if f(x)]

data = '1 2 3 5 8 15 23 38'.split()
data = select(int, data)
data = where(lambda e: not e % 2, data)
data = list(select(lambda e: (e, e**2), data))
```

Анонимные, lambda функции

```
def select(f, col):
    return [f(x) for x in col]

def where(f, col):
    return [x for x in col if f(x)]

data = '1 2 3 5 8 15 23 38'.split()
data = select(int, data)
data = where(lambda e: not e % 2, data)
data = list(select(lambda e: (e, e**2), data))
```

Анонимные, lambda функции

```
data = '1 2 3 5 8 15 23 38'.split()
data = list(map(int, data))
data = list(filter(lambda e: not e % 2, data))
data = list(map(lambda e: (e, e**2), data))
print(data)
```

Функция map

Функция map

Функция map() применяет указанную функцию к каждому элементу итерируемого объекта и возвращает итератор с новыми объектами.

```
f(x) ⇒ x + 10
map(f, [  1,  2,  3,  4, 5])
          ↓   ↓   ↓   ↓   ↓
        [ 11, 12, 13, 14, 15]
```

Нельзя пройтись дважды

Функция filter

Функция filter

Функция filter() применяет указанную функцию к каждому элементу итерируемого объекта и возвращает итератор с теми объектами, для которых функция вернула True.

```
f(x) ⇒ x - чётное
filter(f, [ 1, 2, 3, 4,5])
                  ↓
          [    2,    4  ]
```

Нельзя пройтись дважды

Функция zip

Функция zip

Функция zip() применяется к набору итерируемых объектов и возвращает итератор с кортежами из элементов входных данных.

Количество элементов в результате равно минимальному количеству элементов входного набора

```
zip ([1, 2, 3], [ `o`, `д`, `т`], [`f',`s',`t'])
                          ↓
[(1, 'o', 'f'), (2, 'д', 's'), (3, 'т', 't')]
```

Нельзя пройтись дважды

Функция enumerate

Функция enumerate

Функция enumerate() применяется к итерируемому объекту и возвращает новый итератор с кортежами из индекса и элементов входных данных.

```
enumerate(['Казань', 'Смоленск', 'Рыбки', 'Чикаго'])
                          ↓
[(0, 'Казань'), (1, 'Смоленск'), (2, 'Рыбки'), (3, 'Чикаго')]
```

Нельзя пройтись дважды

List Comprehension

List Comprehension

```
[exp for item in iterable]
```

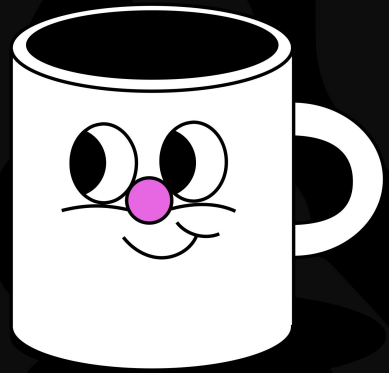
```
[exp for item in iterable (if conditional)]
```

```
[exp <if conditional> for item in iterable (if conditional)]
```

Итоги

Итоги

Как упростить работу с данными
Научились **map**, **filter**, **zip**
List comprehensions,
enumerate,
анонимные \ lambda функции



Как улучшить
программу, написанную в предыдущей лекции?