УРОК 28. ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ НА PYTHON

ВСТРОЕННЫЕ ФУНКЦИИ MAP И FILTER

В Python есть встроенные функции map и filter, которые позволяют применять функцию к каждому элементу итерируемого объекта и фильтровать элементы соответственно.

Функция map возвращает итератор с преобразованными значениями.

Функция filter возвращает итератор, содержащий только элементы, для которых функция-предикат возвращает True.

```python
numbers = [1, 2, 3, 4, 5]
squared = map(lambda x: x ** 2, numbers) # [1, 4, 9, 16, 25]

even = filter(lambda x: x % 2 == 0, numbers) # [2, 4]
```

Во многих простых случаях comprehensions, map-filter и обычные циклы с условиями могут быть взаимозаменяемыми для создания нового списка на основе существующего. Каждый из этих подходов имеет свои преимущества и может быть выбран в зависимости от предпочтений программиста или читабельности кода.

```python
numbers = [1, 2, 3, 4, 5]

# Comprehension
squared = [x ** 2 for x in numbers] # [1, 4, 9, 16, 25]

# Map-Filter
squared = list(map(lambda x: x ** 2, numbers)) # [1, 4, 9, 16, 25]

# Loop with condition
```


```
squared = []
for x in numbers:
  squared.append(x ** 2) # [1, 4, 9, 16, 25]
```

Каждый из трех подходов - списковые включения (comprehensions), map-filter и обычные циклы с условиями - имеет свои достоинства и недостатки.

- Comprehensions обычно считаются более краткими и читабельными, особенно для простых операций на списках.
- Map-filter предлагает более гибкую и обобщенную функциональность, особенно если требуется применять пользовательские функции.
- Обычные циклы с условиями дают больше гибкости и контроля, особенно при сложных манипуляциях с данными.

ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ

1. Встроенная функция map() возвращает:

    a. строку
    b. список
    c. кортеж
    d. словарь
    e. множество
    f. итератор

2. Встроенная функция filter() возвращает:

    a. кортеж
    b. итератор
    c. множество
    d. строку
    e. словарь
    f. список

ПОСТРОЕНИЕ ЦЕПОЧЕК ИЗ ЭЛЕМЕНТОВ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ

В Python можно строить цепочки из различных элементов функционального программирования, таких как map, filter, reduce и другие функции. Это позволяет последовательно применять функции к данным и преобразовывать их.

```python
from functools import reduce

numbers = [1, 2, 3, 4, 5]

result = reduce(lambda x, y: x + y, filter(lambda x: x % 2 == 0, map(lambda x: x ** 2, numbers)))
# Результат: 20
```

ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ

1. Что выведет указанный ниже код?

```python
iterable = ['1', '2', '3']
result = list(map(len, iterable))
print(result)
```

   a. произойдет ошибка во время выполнения программы
   b. 1
   c. [1, 2, 3]
   d. [[1], [1], [1]]
   e. [[1], [2], [3]]
   f. [1, 1, 1]

2. Что выведет указанный ниже код?

```python
iterable = [[1], [2], [3]]
result = list(map(len, iterable))
print(result)
```

   a. [[1], [2], [3]]
   b. [1, 1, 1]
   c. [[1], [1], [1]]
   d. [1, 2, 3]
   e. произойдет ошибка во время выполнения программы
   f. 1

3. Что выведет указанный ниже код?


```python
iterable = [1, 2, 3]
result = list(map(len, iterable))
print(result)
```

a. [1, 2, 3]
b. произойдет ошибка во время выполнения программы
c. [1, 1, 1]
d. [[1], [1], [1]]
e. [[1], [2], [3]]
f. 1

МОДУЛЬ ITERTOOLS

reduce - это функция, которая применяет указанную функцию к элементам последовательности, сводя их к одному значению.

Однако, в Python 3 reduce был вынесен из числа встроенных функций в модуль functools в связи с читаемостью и ясностью кода. Использование цикла for или comprehensions обычно считается более понятным и предпочтительным способом свертки данных.

Модуль itertools предлагает функцию accumulate, которая выполняет свертку данных, накапливая промежуточные результаты. Это позволяет поэтапно применять указанную функцию к элементам последовательности и сохранять каждый промежуточный результат. Кроме того, модуль operator предоставляет функции для использования в reduce и accumulate, что делает код более ясным и компактным.

```python
import itertools
import operator

numbers = [1, 2, 3, 4, 5]
result = itertools.accumulate(numbers, operator.mul)
# Результат: [1, 2, 6, 24, 120]
```

Модуль itertools предлагает функции для генерации комбинаторных объектов, таких как комбинации (combinations), перестановки (permutations) и декартово произведение (product). Эти функции позволяют генерировать все возможные комбинации элементов из итерируемых объектов.


```python
import itertools

letters = ['a', 'b', 'c']
numbers = [1, 2, 3]

combinations = list(itertools.combinations(letters, 2))
# Результат: [('a', 'b'), ('a', 'c'), ('b', 'c')]

permutations = list(itertools.permutations(numbers))
# Результат: [(1, 2, 3), (1, 3, 2

), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)]

product = list(itertools.product(letters, numbers))
# Результат: [('a', 1), ('a', 2), ('a', 3), ('b', 1), ('b', 2), ('b', 3), ('c',
1), ('c', 2), ('c', 3)]
```

Метод zip_longest из модуля itertools позволяет создавать итератор, который сопоставляет элементы нескольких последовательностей, даже если они имеют разную длину.

Это особенно полезно, когда требуется обработать данные, учитывая разные длины или пропуски в последовательностях.

```python
import itertools

letters = ['a', 'b', 'c']
numbers = [1, 2]

zipped = list(itertools.zip_longest(letters, numbers, fillvalue='-'))
# Результат: [('a', 1), ('b', 2), ('c', '-')]
```

ЗАДАНИЕ ДЛЯ ЗАКРЕПЛЕНИЯ

Что выведет указанный ниже код?

```python
from operator import add
from itertools import accumulate

result = list(accumulate([[1, 2, 3], [4, 5, 6], [7, 8, 9]], add))
print(result)
```

a. [6, 15, 24]
b. [1, 2, 3, 4, 5, 6, 7, 8, 9]
c. 45
d. [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
e. [12, 15, 18]
f. [[1, 2, 3], [1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6, 7, 8, 9]]

ПРАКТИЧЕСКАЯ РАБОТА

1. Сгенерировать все пароли, состоящие из 4 символов чтобы среди них была хотя бы одна большая латинская буква, одна маленькая, одна цифра и никаких иных типов символов.
   При этом запрещено, чтобы символ повторялся в пароле или две соседние по алфавиту буквы стояли рядом. Записать полученные пароли в файл.

2. Напишите программу, которая принимает список чисел от пользователя и использует функцию map, чтобы преобразовать каждый элемент списка в его квадрат.
   Затем программа должна использовать функцию filter, чтобы отфильтровать только те элементы, которые являются четными числами.
   В результате программа должна вывести новый список, содержащий квадраты только четных чисел.

ПОЛЕЗНЫЕ МАТЕРИАЛЫ

1. Введение в функциональное программирование на Python
2. Python/Функциональное программирование на Python