

Занятие 5. Функции. Декораторы



1

map, filter, reduce

map

Зачем?

- Используется для применения ф-ции к каждому элементу итерируемого объекта и возврата новых значений

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

```
list(map(lambda x: x**2, list_))
```

```
[1, 4, 9, 16]
```

filter

Зачем?

- Используется для фильтрации итерируемых объектов

```
list_ = [-10, 2, 3, 4, -1, -2, -3]  
list(filter(lambda x: x > 0, list_))
```

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

reduce

Зачем?

- Последовательно применяет ф-цию к элементам списка, возвращает единичное значение

```
from functools import reduce  
reduce(lambda a, x: a + x, [1, 2, 3], 0)
```

2

Функции высшего порядка. Декораторы

Функция высшего порядка

функции, которые принимают в себя другие функции или возвращают их, например map, reduce, filter

```
def func():  
    def inner_func():  
        print("Это внутренняя функция")  
    return inner_func  
  
result = func() # чему равен result?
```

result - это функция inner_func, которую мы можем вызвать (result())

Декоратор

Что это такое

- функция, которая позволяет обернуть другую функцию для расширения ее функциональности без изменения ее кода

Примеры декораторов:

- для кэширования (чтобы не приходилось при тех же значениях пересчитывать функцию)
- для логгирования (писать все вызовы функции)
- для того чтобы узнать, сколько выполнялась функция
- для того чтобы выполнить любую логику до и после выполнения функции

Пример декоратора

```
import time

def timer(func):
    def wrapper():
        start_time = time.time()
        func()
        print(f"Время выполнения функции {time.time() - start_time}")
    return wrapper

@timer
def say_hi():
    time.sleep(2)
    print("Hello")
```

```
say_hi()
```

Hello

Время выполнения функции 2.000582456588745

Декоратор с аргументами

```
def timer(max_time=2):  
    def wrapper(func):  
        def wrapped(*args, **kwargs):  
            start_time = time.time()  
            func(*args, **kwargs)  
            total_time = time.time() - start_time  
            print(f"Время выполнения функции {total_time}")  
            if total_time > max_time:  
                print("Время выполнения функции превысило 2 секунды!")  
        return wrapped  
    return wrapper  
  
@timer(2)  
def say_hi():  
    time.sleep(2)  
    print("Hello")
```

```
say_hi()
```

Hello

Время выполнения функции 2.0025479793548584

Время выполнения функции превысило 2 секунды!

Применение сразу нескольких декораторов

```
@timer(2)
@logger
def say_hi(name):
    time.sleep(2)
    print(f"Hello {name}")
```

3

Домашнее задание

Домашнее задание (часть 1)

Часть 1. Работа с `lambda`, `map`, `reduce`, `filter`:

1. Отфильтруйте список строк таким образом, чтобы остались строки, которые состоят только из букв
2. Посчитайте среднее арифметическое чисел `list_ = [10, 20, 30, 40]` (тут нужно разбить задачу на два этапа - посчитать сумму, а потом поделить на кол-во элементов)
3. Есть список людей, нужно каждому добавить возраст (посчитать исходя из года рождения)

```
persons = [  
    {"name": "Vasya", "birth_year": 1999},  
    {"name": "Valentin", "birth_year": 1934},  
    {"name": "Petr", "birth_year": 2005}]
```

Домашнее задание (часть 2)

Часть 2. Работа с декораторами:

1. Написать декоратор для функции, который будет возвращать Lena если пользователь введет lena, то есть делать capitalize:

```
def func():  
    return input("Введите имя: ")
```

Часть 3. (необязательное):

Написать декоратор, который реализовывает кэш для функции

```
def mul(a, b):  
    return a * b
```

The background is a vibrant yellow color, covered with a dense, repeating pattern of various geometric shapes. These shapes include circles, squares, triangles, and lines, some of which are filled with a fine grid pattern. The shapes are scattered across the entire surface, creating a dynamic and modern aesthetic.

Спасибо за внимание!