Python Advanced

Элементы функционального программирования

После урока обязательно

Повторите этот урок в видео формате на ITVDN.com

Проверьте как Вы усвоили данный материал
на TestProvider.com

Тема

Элементы функционального программирования

Понятие функционального программирования

Функциональное программирование – раздел дискретной математики и парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании последних (в отличие от функций как подпрограмм в процедурном программировании).

Функциональное программирование предполагает обходиться вычислением результатов функций от исходных данных и результатов других функций, и не предполагает явного хранения состояния программы. Соответственно, не предполагает оно и изменяемость этого состояния (в отличие от императивного, где одной из базовых концепций является переменная, хранящая своё значение и позволяющая менять его по мере выполнения алгоритма).

Элементы функционального программирования

Характерные черты функционального программирования

- Решение задачи записывается как совокупность независимых от внешнего состояния функций
- Функции как объекты первого класса
- Иммутабельность (неизменяемость) данных
- Использование функций высшего порядка
- Каррирование и частичное применение функций

Функция как объект первого класса

Объект называют **«объектом первого класса»**, если он:

- может быть сохранен в переменной или структурах данных;
- может быть передан в функцию как аргумент;
- может быть возвращен из функции как результат;
- может быть создан во время выполнения программы;
- внутренне самоидентифицируем (независим от именования).

Термин «объект» используется здесь в общем смысле, и не ограничивается объектами языка программирования.

В Python, как и в функциональных языках, функции являются объектами первого класса.

Лямбда-выражения

Обычное объявление функции:

```
def add(x, y):
    return x + y
```

Использование лямбда-выражения (лямбда-функции, анонимной функции):

```
add = lambda x, y: x + y
```

Замыкания

- **Замыкание** (англ. **closure**) в программировании – функция, в теле которой присутствуют ссылки на переменные, объявленные вне тела этой функции в окружающем коде и не являющиеся её параметрами.
- В случае замыкания ссылки на переменные внешней функции действительны внутри вложенной функции до тех пор, пока работает вложенная функция, даже если внешняя функция закончила работу, и переменные вышли из области видимости.
- Замыкание связывает код функции с её лексическим окружением (местом, в котором она определена в коде). Лексические переменные замыкания отличаются от глобальных переменных тем, что они не занимают глобальное пространство имён. От переменных в объектах они отличаются тем, что привязаны к функциям, а не объектам.
- В Python любые функции (в том числе и лямбда-выражения), объявленные внутри других функций, являются полноценными замыканиями.

Функции высшего порядка

Функция высшего порядка – функция, принимающая в качестве аргументов другие функции или возвращающая другую функцию в качестве результата. Основная идея состоит в том, что функции имеют тот же статус, что и другие объекты данных.

Каррирование или **карринг** (англ. **currying**) – преобразование функции от многих аргументов в функцию, берущую свои аргументы по одному. Это преобразование было введено М. Шейнфинкелем и Г. Фреге и получило свое название в честь Х. Карри.

Декораторы

Декоратор в Python – функция, которая принимает как параметр другую функцию (или класс) и возвращает новую, модифицированную функцию (или класс), которая её заменяет.

Кроме того, понятие функций высшего порядка часто применяется и для создания декораторов: часто требуется, чтобы декоратор принимал ещё какие-либо параметры, кроме модифицируемого объекта. В таком случае создаётся функция, создающая и возвращающая декоратор, а при применении декоратора вместо указания имени функции-декоратора данная функция вызывается.

Элементы функционального программирования

map, filter, reduce

Тремя классическими функциями высшего порядка, появившимися ещё в языке программирования Lisp, которые принимают функцию и последовательность, являются map, filter и reduce.

- **map** применяет функцию к каждому элементу последовательности. В Python 2 возвращает список, в Python 3 – объект-итератор.
- **filter** оставляет лишь те элементы последовательности, для которых заданная функция истинна. В Python 2 возвращает список, в Python 3 – объект-итератор.
- **reduce** (в Python 2 встроенная, в Python 3 находится в модуле functools) принимает функцию от двух аргументов, последовательность и опциональное начальное значение и вычисляет свёртку (fold) последовательности как результат последовательного применения данной функции к текущему значению (так называемому аккумулятору) и следующему элементу последовательности.

Модуль functools

Модуль *functools* содержит большое количество стандартных функций высшего порядка. Некоторые из них:

- **reduce** – рассмотрена ранее;
- **lru_cache** – декоратор, который кеширует значения функций, которые не меняют свой результат при неизменных аргументах; полезен для кеширования данных, мемоизации (сохранения результатов для возврата без вычисления функции) значений рекурсивных функций (например, такого типа, как функция вычисления n-го числа Фибоначчи) и т.д.;
- **partial** – частичное применение функции (вызов функции с меньшим количеством аргументов, чем она ожидает, и получение функции, которая принимает оставшиеся параметры).

Модуль itertools

Модуль *itertools* содержит функции для работы с итераторами и создания итераторов. Некоторые из них:

- **product** – декартово произведение итераторов (для избегания вложенных циклов for);
- **permutations** – генерация перестановок;
- **combinations** – генерация сочетаний;
- **combinations_with_replacement** – генерация размещений;
- **chain** – соединение нескольких итераторов в один;
- **takewhile** – получение значений последовательности, пока значение функции-предиката для её элементов истинно;
- **dropwhile** – получение значений последовательности начиная с элемента, для которого значение функции-предиката перестанет быть истинно.

Модуль operator

Модуль *operator* содержит функции, которые соответствуют стандартным операторам.

Примеры:

Функция модуля operator	Аналогичная лямбда-функция
add	lambda x, y: x + y
gt	lambda x, y: x > y
neg	lambda x: -x
…	

Q&A

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association