Элементы функционального программирования

№ урока:	1	**Курс:**	Python Advanced

Средства обучения:	Python; интегрированная среда разработки (PyCharm или Microsoft Visual Studio + Python Tools for Visual Studio)

Обзор, цель и назначение урока

После завершения урока обучающиеся будут иметь представление об основах парадигмы функционального программирования, научатся использовать некоторые её принципы в программах на Python (например, создавать свои декораторы, которые, по сути, являются функциями высшего порядка), научатся использовать лямбда-выражения и стандартные функции и модули, связанные с функциональным программированием.

Изучив материал данного занятия, учащийся сможет:

- Иметь представление об основах парадигмы функционального программирования
- Использовать её принципы в программах на Python
- Использовать функции как объекты первого класса
- Использовать лямбда-выражения
- Создавать функции высшего порядка, собственные декораторы
- Использовать функции filter, map, reduce
- Использовать модули functools, operator, itertools

Содержание урока

- Функции как объекты первого класса (first-class citizens)
- Лямбда-выражения
- Замыкания
- Функции высшего порядка, каррирование функций
- Декораторы
- Функции filter, map, reduce
- Модули functools, operator, itertools

Резюме

В курсе Python Essential описана объектно ориентированная парадигма программирования и упоминается, что существует множество менее популярных парадигм, что находят свое применение. Одина из таких — **функциональная парадигма**. Заметим, что функциональная парадигма довольно популярна и составляет конкуренцию ООП.

Функциональное программирование (ФП) — парадигма программирования, в которой вычисления описываются как решение функций в математическом понимании.
ФП предполагает использовать вычисления результатов функций от исходных данных. Состояние программы явно не хранится. Соответственно, состояние не изменяется, в отличие от императивного, где одной из базовых концепций является переменная, хранящая своё значение и позволяющая менять его по мере выполнения алгоритма.
На практике отличие математической функции от понятия «функции» в императивном программировании заключается в том, что **императивные функции** могут опираться не только на аргументы, но и на состояние внешних по отношению к функции переменных, а также иметь побочные эффекты и менять состояние внешних переменных.
Таким образом, **в императивном программировании** при вызове одной и той же функции с одинаковыми параметрами, но на разных этапах выполнения алгоритма, **можно получить разные данные** на выходе из-за влияния на функцию состояния переменных.

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

В функциональном языке при вызове функции с одними и теми же аргументами **мы всегда получим одинаковый результат.** Выходные данные зависят только от входных. Это позволяет средам выполнения программ на функциональных языках кешировать результаты функций и вызывать их в порядке, не определяемом алгоритмом и распараллеливать их без каких-либо дополнительных действий со стороны программиста.
Python частично поддерживает парадигму функционального программирования и позволяет писать код в функциональном стиле. Кроме того, в нём присутствуют определённые возможности, характерные для функциональных языков (*списковые включения, лямбда-функции, функции высшего порядка и т.д.*).

Объектами первого класса (*англ. first-class object, first-class entity, first-class citizen*) в контексте конкретного языка программирования называются сущности, которые могут быть переданы как параметр, возвращены из функции и присвоены переменной.
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;
- 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**.
Обычное объявление функции в Python является оператором - def. Однако при написании кода в функциональном стиле часто бывает удобной возможность объявить анонимную функцию внутри выражения. В Python есть такая возможность: она реализуется при помощи лямбда-выражений

В анонимных функциях в Python существует ограничение: они могут состоять лишь из одного выражения. В некоторых языках такого ограничения нет.

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. Другими словами, замыкание — возможность функция, которая ссылается на переменную из своего контекста.
Замыкание, так же как и экземпляр объекта, есть способ представления функциональности и данных, связанных и упакованных вместе. **Замыкание** — это особый вид функции. Она определена в теле другой функции и создаётся каждый раз во время её выполнения.
Синтаксически это выглядит как функция, находящаяся целиком в теле другой функции. При этом вложенная внутренняя функция содержит ссылки на локальные переменные внешней функции. Каждый раз при выполнении внешней функции происходит создание нового экземпляра внутренней функции, с новыми ссылками на переменные внешней функции.
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.

В примере выше мы выполнили функцию make_closure. Казалось бы, она выполнилась и все, что было создано внутри неё перестало существовать. Но лексическое окружение родительской функции доступно дочерней функции и, соответственно, может получить доступ к родительским переменным.
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.
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.

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Каррирование или карринг (англ. currying) в информатике — 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.

Существует такой паттерн проектирования как **Декоратор**: структурный шаблон проектирования, предназначенный для динамического подключения дополнительного поведения к объекту. Шаблон Декоратор предоставляет гибкую альтернативу практике создания подклассов с целью расширения функциональности.
Декоратор в Python – функция, которая принимает другую функцию (или класс) и возвращает новую функцию (или класс).
Этот механизм позволяет обернуть другую функцию для расширения её функциональности без непосредственного изменения её кода.

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.

Тремя классическими функциями высшего порядка, появившимися ещё в языке программирования Lisp, которые принимают функцию и последовательность, являются **map, filter и reduce**.
Функция map применяет функцию к каждому элементу последовательности. В Python 2 возвращает список, в Python 3 – объект-итератор.

Функция filter оставляет лишь те элементы последовательности, для которых заданная функция истинна. В Python 2 возвращает список, в Python 3 – объект-итератор.

Функция 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** содержит большое количество стандартных функций высшего порядка. Среди них особенно полезны:

- reduce – рассмотрена выше;
- lru_cache – декоратор, который кеширует значения функций, что 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 содержит функции для работы с итераторами и создания итераторов. Некоторые из его функций:

- 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 содержит функции, которые соответствуют стандартным операторам. Таким образом, вместо lambda x, y: x + y можно использовать уже готовую функцию operator.add и т.д.

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Закрепление материала

- Что такое функциональное программирование?
- Что такое объект первого класса?
- Что такое лямбда-выражение?
- Что такое замыкание?
- Что такое функция высшего порядка?
- Что такое каррирование?
- Что такое декоратор?
- Что делает функция map?
- Что делает функция filter?
- Что делает функция reduce?
- Что такое частичное применение функции?

Дополнительное задание

Задание
Создайте обычную функцию умножения двух чисел. Частично примените её к одному аргументу. Создайте каррированную функцию умножения двух чисел. Частично примените её к одному аргументу.

Самостоятельная деятельность учащегося

Задание 1
Ещё раз разберите все примеры к уроку, повторите теорию и ознакомьтесь с документацией по рассмотренным модулям.

Задание 2
Создайте список целых чисел. Получите список квадратов нечётных чисел из этого списка.

Задание 3
Создайте функцию-генератор чисел Фибоначчи. Примените к ней декоратор, который будет оставлять в последовательности только чётные числа.

Рекомендуемые ресурсы

Документация Python
https://docs.python.org/3/library/functions.html#filter
https://docs.python.org/3/library/functions.html#map
https://docs.python.org/2/library/functions.html#reduce
https://docs.python.org/3/library/functools.html
https://docs.python.org/3/library/itertools.html
https://docs.python.org/3/library/operator.html
https://www.python.org/dev/peps/pep-0318/

Статьи в Википедии о ключевых понятиях, рассмотренных на этом уроке
https://ru.wikipedia.org/wiki/Функциональное_программирование
https://ru.wikipedia.org/wiki/Объект_первого_класса
https://ru.wikipedia.org/wiki/Замыкание_(программирование)
https://ru.wikipedia.org/wiki/Функция_высшего_порядка
https://ru.wikipedia.org/wiki/Каррирование
https://ru.wikipedia.org/wiki/Декоратор_(шаблон_проектирования)
https://ru.wikipedia.org/wiki/Функциональное_программирование_на_Python

CyberBionic systematics
CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com