



*School of  
Computer  
Science*

# **ЭЛЕМЕНТЫ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ**

## **ПРОГРАММИРОВАНИЕ НА PYTHON**

**Лекции для IT-школы**



# ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ

- **Императивное** – последовательность инструкций, контекст, разбиение программы с помощью функций
- **Декларативное** – описывается задача и ожидаемый результат, но не её решение
- **ООП** – программа манипулирует набором объектов, имеющих своё состояние и набор методов для изменения этого состояния
- **Функциональное** – решение задачи разбивается на цепочку функций, которые только принимают параметры и возвращают значения, не изменяя состояния объектов



# ФУНКЦИОНАЛЬНАЯ ПАРАДИГМА (ФП)

## ОПРЕДЕЛЕНИЯ

- Процесс вычислений в ФП трактуется как вычисление значений функций в их математическом понимании
- В этом отличие от функций как подпрограмм в процедурном стиле
- В полностью функциональной парадигме нет переменных, вся программа состоит из последовательных вызовов функций
- В ФП не предполагается сохранение промежуточного состояния программы



# ФУНКЦИОНАЛЬНАЯ ПАРАДИГМА

## ПРЕДПОСЫЛКИ

- В 30-ых годах в Принстоне собрались Алан Тьюринг, Джон фон Нейман, Курт Гёдель и **Аллонзо Чёрч**
- Интересуясь формальными системами вычислений они пытались ответить на такие вопросы:
  - Какие задачи можно решать на машине с бесконечными вычислительными возможностями?
  - Можно ли решать эти задачи автоматически?
  - Существуют ли неразрешимые задачи и почему?
  - Будут ли машины с разной архитектурой одинаковыми по мощности?





# ФУНКЦИОНАЛЬНАЯ ПАРАДИГМА ТЕОРИЯ

- Использует математическую теорию лямбда-исчисления Алонзо Чёрча (1936)
- Основана на функциях, которые принимают в качестве аргументов функции, и возвращают функцию
- Такая функция была обозначена греческой буквой Лямбда, что дало название всей системе



# ФУНКЦИОНАЛЬНАЯ ПАРАДИГМА

## ПРАКТИЧЕСКОЕ ВОПЛОЩЕНИЕ

- В конце 50-ых Джон Маккарти стал проявлять интерес к работам Черча
- В 1958 году он представил язык обработки списков **List Processing Language (Lisp)**
- Lisp – имплементация Лямбда-исчисления Алонзо, которая работает на машинах с архитектурой фон Неймана
- 1973 г построена аппаратная Lisp-машина



# ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

## ПРИМЕР ПРОГРАММЫ НА LISP

```
(defun encr(l,a)
  (cond
    ((null l) 1)
    )
  ((atom (car l))
    (cond
      ( (> (car l) 0) (cons (+ (car l) a) (encr (cdr l) a) ) )
      )
    (t (encr (cdr l) a) )
    )
  )
  (t
    ( cons (encr (car l) a) (encr (cdr l) a))
    )
  )
)

(defun main()
  (setq l '(1 -2 (3 5) -2 4 (6 -2)))
  (setq ll (encr l 1))
)
(main)
```



# ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ КАК ЭТО БЫВАЕТ В «ЧИСТОМ» ВИДЕ:

- Вычисления программируются путем комбинирования функций, которые:
  - не изменяют свои аргументы
  - не обращаются к переменным, определяющим состояние программы и не изменяют эти переменные
  - не зависят от внешней среды
  - результаты своей работы поставляют только в виде возвращаемых значений





# ФУНКЦИОНАЛЬНЫЙ ПОДХОД

- Программа (её фрагмент) рассматривается как вычисление математических функций
- Не используются состояния (переменные)
- Функции являются «чистыми», т.е. они:
  - Не меняют глобальных переменных
  - Ничего никуда не посылают, не принимают извне, не сохраняют и не печатают
  - Делают вычисления, учитывая только аргументы, и возвращают новые данные
  - При этом отсутствуют какие-либо побочные эффекты, только возвращается результат
- ФП в Python включает:
  - Развитые средства работы с коллекциями
  - Использование **lambda**, **map**, **filter**, **reduce**



# ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

## ЗАЧЕМ?

- Возможность написания программ, работающих параллельно:
  - Конкурентность/параллелизм/многопоточность – одновременное выполнение нескольких вычислительных процессов, которые взаимодействуют друг с другом
- «Чистота» функций (нет побочных эффектов), позволяет кэшировать их результаты для последующего использования
- Используется в задачах с высокой вычислительной сложностью
- **ФП упрощает работу с коллекциями**



# ЯЗЫКИ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ

- LISP, диалекты Scheme (Racket) и Clojure
- Haskell – чисто функциональный язык
- Erlang – ФП с поддержкой процессов
- Kotlin, Scala – ФП + ООП
- XSLT, Xquery – для работы с XML
- R, Mathematica – ФП для мат. статистики



# ПРАКТИЧЕСКОЕ ЗАДАНИЕ

## ФУНКЦИОНАЛЬНАЯ ПРОГРАММА

- Запросите у пользователя 2 целых числа
- Выведите сумму этих чисел
- Программу нужно написать в стиле ФП:
  - Можно использовать только функции
  - Нельзя использовать промежуточные переменные
- Обработка ошибок ввода не требуется



# ЭЛЕМЕНТЫ ФП В PYTHON

Я никогда не считал, что на Python оказали заметное влияние функциональные языки, что бы кто об этом ни говорил или ни думал. Я был значительно лучше знаком с императивными языками типа C или Algol и, хотя **сделал функции полноправными объектами**, никогда не рассматривал Python как язык функционального программирования.

– Гвидо Ван Россум  
*пожизненный великодушный диктатор Python*

<http://bit.ly/1FHfhlo> – из блога Гвидо «История Python»





# ФУНКЦИИ КАК ПОЛНОПРАВНЫЕ ОБЪЕКТЫ

## ОПРЕДЕЛЕНИЕ

- Полноправный объект или **объект первого класса** – это элемент программы, который:
  - Может быть создан во время выполнения
  - Может быть присвоен переменной или полю структуры данных
  - **Может быть передан функции в качестве аргумента**
  - **Может быть возвращен функцией в качестве результата**
- См. скрипт `func_as_an_object.py`





# ПЕРЕДАЧА ФУНКЦИИ КАК АРГУМЕНТА

- Функции – такие же объекты как строки, числа, списки, и т.д.
- Функции можно присваивать переменным, хранить, и даже передавать как параметр другой функции
- Переменная, содержащая функцию, может инициировать ее вызов через ()
- Пример: [apply.py](#)





# ФУНКЦИИ ВЫСШЕГО ПОРЯДКА

- Принимают другие функции в качестве аргументов и/или
- Могут возвращать функцию как результат своей работы
- Примеры: `sort()`, `sorted()`, `max()`, `min()`
- Можно разрабатывать собственные функции высшего порядка





# ВКЛЮЧЕНИЕ СПИСКОВ

- Другие названия:
  - List comprehension
  - Списковые включения
  - Списочные выражения
  - Генератор списка
- **Включение – компактный способ создания списка, позволяющий объединять циклы и условные проверки**
- Синтаксис:
  - `[statement for var in iterable if predicate]`
- Примеры: `list_comrehen.py`



# ПРАКТИЧЕСКОЕ ЗАДАНИЕ

## ВИСОКОСНЫЕ ГОДЫ

- Год является високосным если:
  - его номер кратен 4, но не кратен 100
  - или же если он кратен 400
- Создайте список, содержащий все високосные годы текущего столетия
- Напишите этот код алгоритмически
- А затем перепишите с использованием спискового включения (list comprehension)
- Какой из этих вариантов более краткий?
- Какой из них более понятен для вас?



# ВКЛЮЧЕНИЕ ДРУГИХ КОЛЛЕКЦИЙ

- Включение словаря:  
`{key:value for var in iterable if predicate}`
- Примеры: `dict_comprehen.py`
- Включение множества:  
`{statement for var in iterable if predicate}`
- Примеры: `set_comprehen.py`



# ЛЯМБДА-ФУНКЦИИ (СМ. [LAMBDA.PY](https://lambda.py))

## АНОНИМНЫЕ «ФУНКЦИИ ПО МЕСТУ»

- Объявляются так:
  - **lambda [parameters]: expression**
  - parameters – необязательный список параметров через запятую
  - expression не может содержать присваивания, условные инструкции и циклы, а также return
- Возвращают значение expression
- Нужны для создания однострочных функций, как правило с целью их моментального и единоразового использования
- Используются обычно в качестве аргументов функций высшего порядка



# СМ. `MAP_FILTER_REDUCE.PY`

## ОТОБРАЖЕНИЕ – `MAP()`

- **Отображение** – это запуск функции для каждого элемента итерируемого объекта
- Отображение возвращает новый итерируемый объект, каждый элемент которого представляет результат вызова функции для соответствующего элемента в оригинальном объекте
- Отображение реализуется функцией `map()` и выражением-генератором



# СМ. MAP\_FILTER\_REDUCE.PY

## ФИЛЬТРАЦИЯ – FILTER()

- **Фильтрация** – совместное использование функции и коллекции для получения нового итерируемого объекта, в состав которого включаются все те элементы оригинальной коллекции, для которых функция вернула значение **True**
- Это понятие поддерживается встроенной функцией **filter()** и с помощью условного выражения-генератора



# СМ. `MAP_FILTER_REDUCE.PY`

## УПРОЩЕНИЕ – `REDUCE()`

- **Упрощение** – совместное использование функции и коллекции для получения в качестве результата отдельного значения
- Это понятие поддерживается функцией `functools.reduce()`
- Идея `reduce()` – применить некую операцию к каждому элементу последовательности с аккумулярованием результатов и тем самым, свести (редуцировать) последовательность значений к одному



# УПРОЩЕНИЕ. ДРУГИЕ РЕДУЦИРУЮЩИЕ ФУНКЦИИ

Вызов	Описание
<b>all(it)</b>	True, если все элементы итерируемого объекта it в логическом контексте оцениваются как значение True
<b>any(it)</b>	True, если хотя бы один элемент итерируемого объекта it в логическом контексте оценивается как значение True
<b>max(it [, key]), min(it [, key])</b>	Возвращает наибольший/наименьший элемент в итерируемом объекте it или элемент с наибольшим/наименьшим значением key(item), если функция key определена
<b>sum(...)</b>	Сумма значений итерируемого объекта





# ПРАКТИЧЕСКОЕ ЗАДАНИЕ

## ПОДСЧЕТ СУММЫ МОДУЛЕЙ

- Нужно в одну строку кода посчитать сумму модулей чисел, сохраненных в списке
- Используйте функциональный подход
- Не используйте присваивания, `print()`, ...
- Напишите в Shell только выражение, вычисляющее сумму в одной строке
- Из нескольких возможных решений используйте самое короткое



# ПРАКТИЧЕСКОЕ ЗАДАНИЕ

## **`sort()` КАК ФУНКЦИЯ ВЫСШЕГО ПОРЯДКА**

- Дан список `[128, 11, 47, 564, 2, 90]`
- Используйте `sort()` как функцию высшего порядка для сортировки по последней цифре каждого элемента списка
- Используйте функциональный подход
- Не используйте присваивания, `print()`, ...
- Напишите в Shell только сортирующее выражение, в одной строке

**СПАСИБО ЗА ВНИМАНИЕ !**  
**ВОПРОСЫ ?**



*School of  
Computer  
Science*