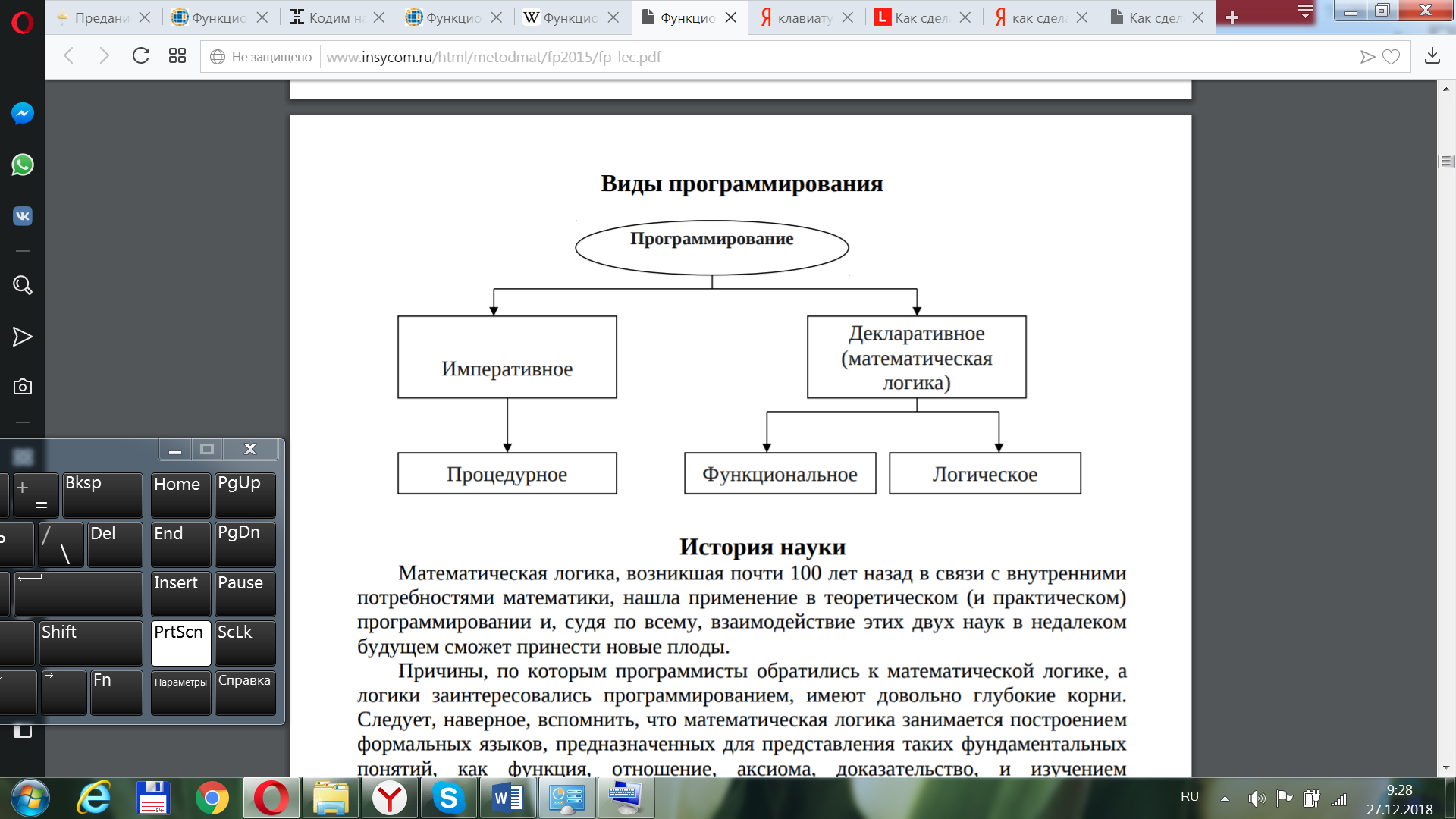
**Виды** [**программирования**](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)

По всей видимости Вы уже слышали о существовании в мире программирования трех основных парадигм - это логическое программирование, императивное и функциональное.



Императивное программирование, наиболее распространённым, заключается в формировании инструкций, последовательных команд, которые должна выполнять машина. За примерами далеко ходить не надо, просто откройте список самых популярных языков программирования: те, что сверху — императивные.

В логическое оперирует математической логикой для вывода новых фактов и состояний из уже известных.

Функциональным называется программирование при помощи функций в математическом их понимании.

[**Языки логического программирования**](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)

**Planner** — [функционально](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)-[логический](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) ([Карл Хьюит](https://ru.wikipedia.org/w/index.php?title=%D0%9A%D0%B0%D1%80%D0%BB_%D0%A5%D1%8C%D1%8E%D0%B8%D1%82&action=edit&redlink=1) , 1969г.) (от языка **Planner** произошли языки [**QA-4**](https://ru.wikipedia.org/w/index.php?title=QA-4&action=edit&redlink=1)**,**[**Popler**](https://ru.wikipedia.org/w/index.php?title=Popler&action=edit&redlink=1)**,**[**Conniver**](https://ru.wikipedia.org/w/index.php?title=Conniver&action=edit&redlink=1)**и**[**QLISP**](https://ru.wikipedia.org/w/index.php?title=QLISP_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)&action=edit&redlink=1)).

**Prolog** - (Ален Колмероэ, 1972г.) – язык [логического программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)

является актуальным и свежим и на сегодняшний день. ( [Mercury](https://ru.wikipedia.org/wiki/Mercury_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [Visual Prolog](https://ru.wikipedia.org/wiki/Visual_Prolog), [Oz](https://ru.wikipedia.org/wiki/Oz_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) и [Fril](https://ru.wikipedia.org/w/index.php?title=Fril&action=edit&redlink=1" \o "Fril (страница отсутствует)))

**Достоинства:**

* операции, совершаемые в логическом программировании всегда понятны. Он довольно прост, в понимании обычного пользователя;
* результат практически всегда не зависит от выбранного пути реализации;
* может быть использован в качестве невычислительного языка используя только выражения и факты.

**Недостатки**:

* если брать за пример логического языка программирования Prolog, то на лицо невозможность создания комплексных задач. То есть в реальности логический язык может идти дополнением к процедурному, но самостоятельно используется крайне редко;
* из-за недостатка в инвестициях и простом внимании, логические языки слабо развиваются;
* если предстоит иметь дело с вычислительными операциями, то логические языки программирования - не лучший выбор.

**Вывод:**

Языком логического программирования является язык, который обрабатывает выражения и факты, а побочным эффектом выдает результат работы данной программы. Такие языки весьма полезны в создании искусственного интеллекта и работы с данными, однако редко применяются без сторонних языков программирования. - Читайте подробнее на

[**Языки функционального программирования**](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)

Наиболее известными [языками функционального программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) являются:

* [**Лисп**](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%81%D0%BF) — ([Джон Маккарти](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D0%BA%D0%BA%D0%B0%D1%80%D1%82%D0%B8,_%D0%94%D0%B6%D0%BE%D0%BD), [1958](https://ru.wikipedia.org/wiki/1958)) и множество его диалектов, наиболее современные из которых:
  + [Scheme](https://ru.wikipedia.org/wiki/Scheme)
  + [Clojure](https://ru.wikipedia.org/wiki/Clojure)
  + [Common Lisp](https://ru.wikipedia.org/wiki/Common_Lisp)
* [**Erlang**](https://ru.wikipedia.org/wiki/Erlang) — ([Joe Armstrong](https://ru.wikipedia.org/w/index.php?title=Joe_Armstrong&action=edit&redlink=1" \o "Joe Armstrong (страница отсутствует)), [1986](https://ru.wikipedia.org/wiki/1986)) функциональный язык с поддержкой процессов.
  + [Elixir](https://ru.wikipedia.org/wiki/Elixir_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))
* [APL](https://ru.wikipedia.org/wiki/APL_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) — предшественник современных научных вычислительных сред, таких как [MATLAB](https://ru.wikipedia.org/wiki/MATLAB).
* [ML](https://ru.wikipedia.org/wiki/ML) ([Робин Милнер](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BB%D0%BD%D0%B5%D1%80,_%D0%A0%D0%BE%D0%B1%D0%B8%D0%BD), [1979](https://ru.wikipedia.org/wiki/1979), из ныне используемых диалектов известны [Standard ML](https://ru.wikipedia.org/wiki/SML" \o "SML) и [Objective CAML](https://ru.wikipedia.org/wiki/OCaml" \o "OCaml)).
* [F#](https://ru.wikipedia.org/wiki/F_Sharp) — функциональный язык семейства ML для платформы [.NET](https://ru.wikipedia.org/wiki/.NET_Framework)
* [Scala](https://ru.wikipedia.org/wiki/Scala_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F))
* [Miranda](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D1%80%D0%B0%D0%BD%D0%B4%D0%B0_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) ([Дэвид Тёрнер](https://ru.wikipedia.org/wiki/%D0%A2%D1%91%D1%80%D0%BD%D0%B5%D1%80,_%D0%94%D1%8D%D0%B2%D0%B8%D0%B4), [1985](https://ru.wikipedia.org/wiki/1985), который впоследствии дал развитие языку Haskell).
* [Nemerle](https://ru.wikipedia.org/wiki/Nemerle) — гибридный функционально/императивный язык.
* [XSLT](https://ru.wikipedia.org/wiki/XSLT)[[3]](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5#cite_note-3)[[4]](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5#cite_note-4) и [XQuery](https://ru.wikipedia.org/wiki/XQuery" \o "XQuery)
* [Haskell](https://ru.wikipedia.org/wiki/Haskell) — [чистый](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D1%82%D0%BE%D1%82%D0%B0_%D1%8F%D0%B7%D1%8B%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) функциональный. Назван в честь [Хаскелла Карри](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D1%80%D0%B8,_%D0%A5%D0%B0%D1%81%D0%BA%D0%B5%D0%BB%D0%BB" \o "Карри, Хаскелл).

Традиционные процедурные языки предназначаются для описания процессов вычислений на Фоннеймоновской машине.

Каждое действие программы – это изменение оперативной памяти.

Мы описываем не формулы для решения задачи, мы описываем процесс, т.е.

как задача должна решаться. Подобные программы сложны для понимания и

доказательств.

Функциональное программирование было задумано как альтернатива

процедурному программированию (в конце 50-х годов).

Функциональная программа представляет собой некоторое выражение (в математическом смысле); выполнение программы означает вычисление значения этого выражения. Другими словами, функциональные программы соответствуют математическим объектам (позволяют производить строгие суждения – правильно/неправильно).

Функциональное программирование основано на идее, что, в

результате каждого действия возникает значение, которое может быть аргументом следующего действия. Программы строятся из логически расчлененных определений функций. Каждое определение функции состоит из организующих вычисления управляющих структур и из вложенных, в том числе вызывающих самих себя (рекурсивных) вызовов функций

|  |  |  |
| --- | --- | --- |
| Процедурное программирование | Функциональное программирование | |
| Переменные | | |
| Именованные ячейки памяти,  изменяемые константы. | | Настоящие переменные (в частности не  используется оператор присваивания). |
| Циклы | | |
| Циклы есть. | | Не может быть циклов (нет счетчика), используются рекурсивные функции  вместо циклов. |
| Программа – это … | | |
| Последовательное изменение памяти. | | Вычисление функции. Последовательное выполнение команд бессмысленно, поскольку одна команда не может повлиять на выполнение следующей (т.к. негде сохранять промежуточные  значения). |
| Функции | | |
| При вызове функции возможны побочные эффекты (два вызова одной и той же функции с одними и теми же аргументами могут дать разные результаты; изменение значений глобальных переменных). | | Функции используются широко, могут даже использоваться в качестве аргументов для других функций и возвращать в качестве результата; побочных эффектов нет. |

**Функциональный подход к решению задачи**

Задача: Найти максимальное из 6-ти значений : a,b,c,d,e,f.

1.Создать Функцию “максимум из двух” :

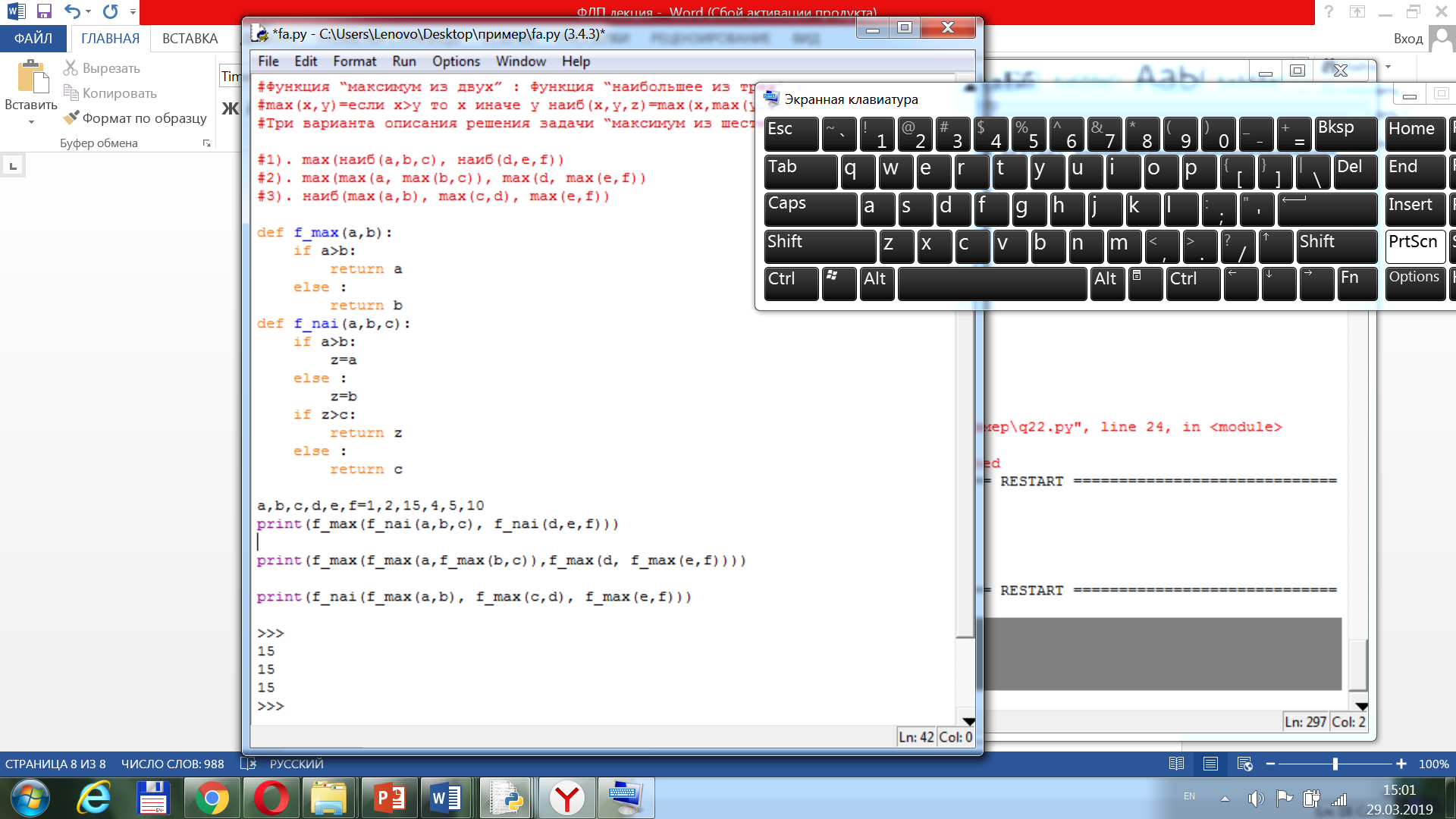
2. Создать Функция “наибольшее из трех” :

Три варианта решения задачи “максимум из шести” :

1). max(наиб(a,b,c), наиб(d,e,f))

2). max(max(a, max(b,c)), max(d, max(e,f))

3). наиб(max(a,b), max(c,d), max(e,f))



**Нахождение факториала на языке процедурного и функционального программирования**

|  |  |
| --- | --- |
| https://cf.ppt-online.org/files1/slide/u/u7XOogmTSQt61PVUKLYiWcFvkh08IEpx4fe3N5MZd/slide-2.jpg | https://cf.ppt-online.org/files1/slide/u/u7XOogmTSQt61PVUKLYiWcFvkh08IEpx4fe3N5MZd/slide-2.jpg |
| def fact(n):  x = 1;  while (n > 0):  x = x \* n;  n = n - 1;  return (x); | def fact(n):  if n==0:  return 1  return n\*fact(n-1) |
| https://cf.ppt-online.org/files1/slide/u/u7XOogmTSQt61PVUKLYiWcFvkh08IEpx4fe3N5MZd/slide-2.jpg | https://cf.ppt-online.org/files1/slide/u/u7XOogmTSQt61PVUKLYiWcFvkh08IEpx4fe3N5MZd/slide-2.jpg |

Как правило, когда говорят о элементах функционального программировании в Python, то подразумеваются следующие функции: **lambda**, **map**, **filter**, **reduce**, **zip**.

**Функция Lambda**

Это маленькая не имеющая имени (анонимная) функция. Такие функции можно назвать одноразовыми, они используются только при создании.

lambda [<*Аргументы*>]]: <*Возвращаемое значение*>

В качестве значения анонимная функция возвращает ссылку на объект-функцию, которую можно сохранить в переменной или передать в качестве параметра в другую функцию

f1 = lambda: 10+20 # Функция без параметров

f2 = lambda x,y: x + y # Функция с двумя параметрами

f3 = lambda x,y,z: x + y + z # Функция с тремя параметрами

print(f1()) # Выведет: 30

print(f2(5, 10)) # Выведет: 15

print(f3(5, 10, 30)) # Выведет: 45

Функция map() в Python:

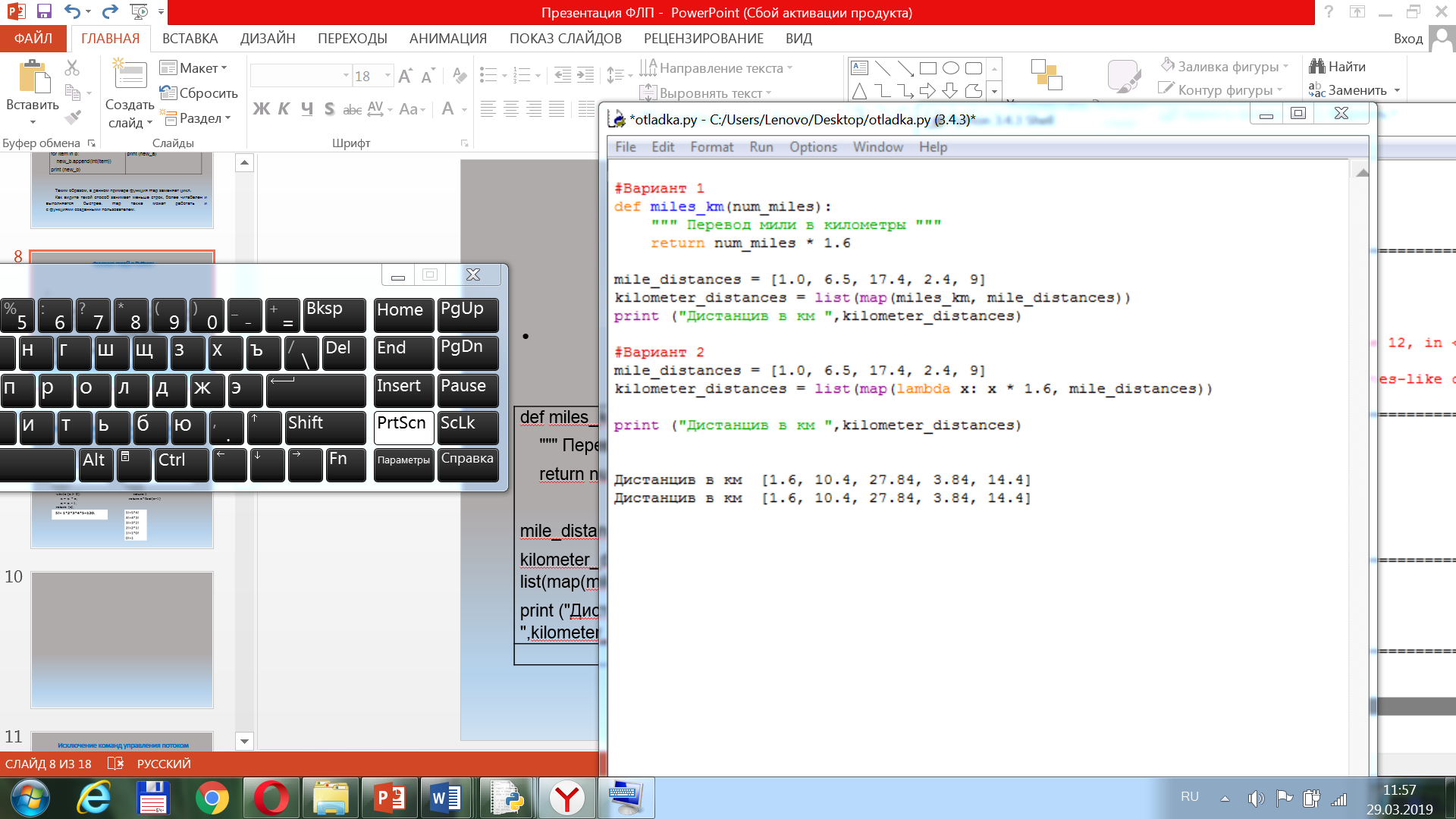
В **Python функция map** принимает два аргумента: функцию и аргумент составного типа данных, например, список. map применяет к каждому элементу списка переданную функцию. Например, вы прочитали из файла список чисел, изначально все эти числа имеют строковый тип данных, чтобы работать с ними - нужно превратить их в целое число:

|  |  |
| --- | --- |
| b = ['1', '2', '3', '4', '5', '6', '7']  new\_b = []  for item in b:  new\_b.append(int(item))    print (new\_b) | a = ['1', '2', '3', '4', '5', '6', '7']  new\_a = list(map(int, a))  print (new\_a) |

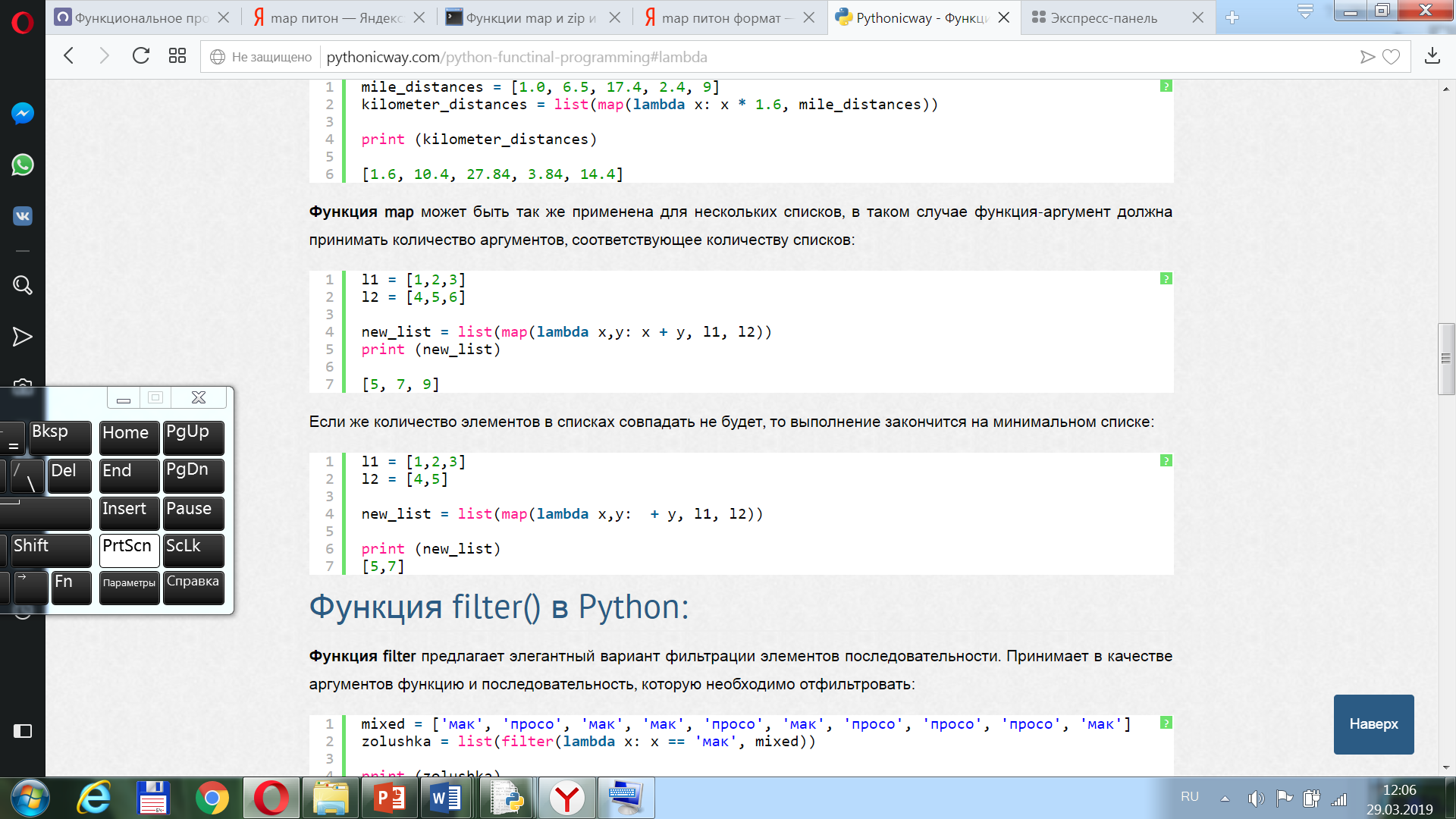
В данном примере функция **map** заменяет цикл.

|  |  |
| --- | --- |
|  |  |

Как видите такой способ занимает меньше строк, более читабелен и выполняется быстрее. map также работает и с [функциями](http://pythonicway.com/python-functions) созданными пользователем:



**Функция map** может быть так же применена для нескольких списков, в таком случае функция-аргумент должна принимать количество аргументов, соответствующее количеству списков:



Если же количество элементов в списках совпадать не будет, то выполнение закончится на минимальном списке:

|  |  |
| --- | --- |
|  |  |

**Функция filter() - фильтрация последовательностей**

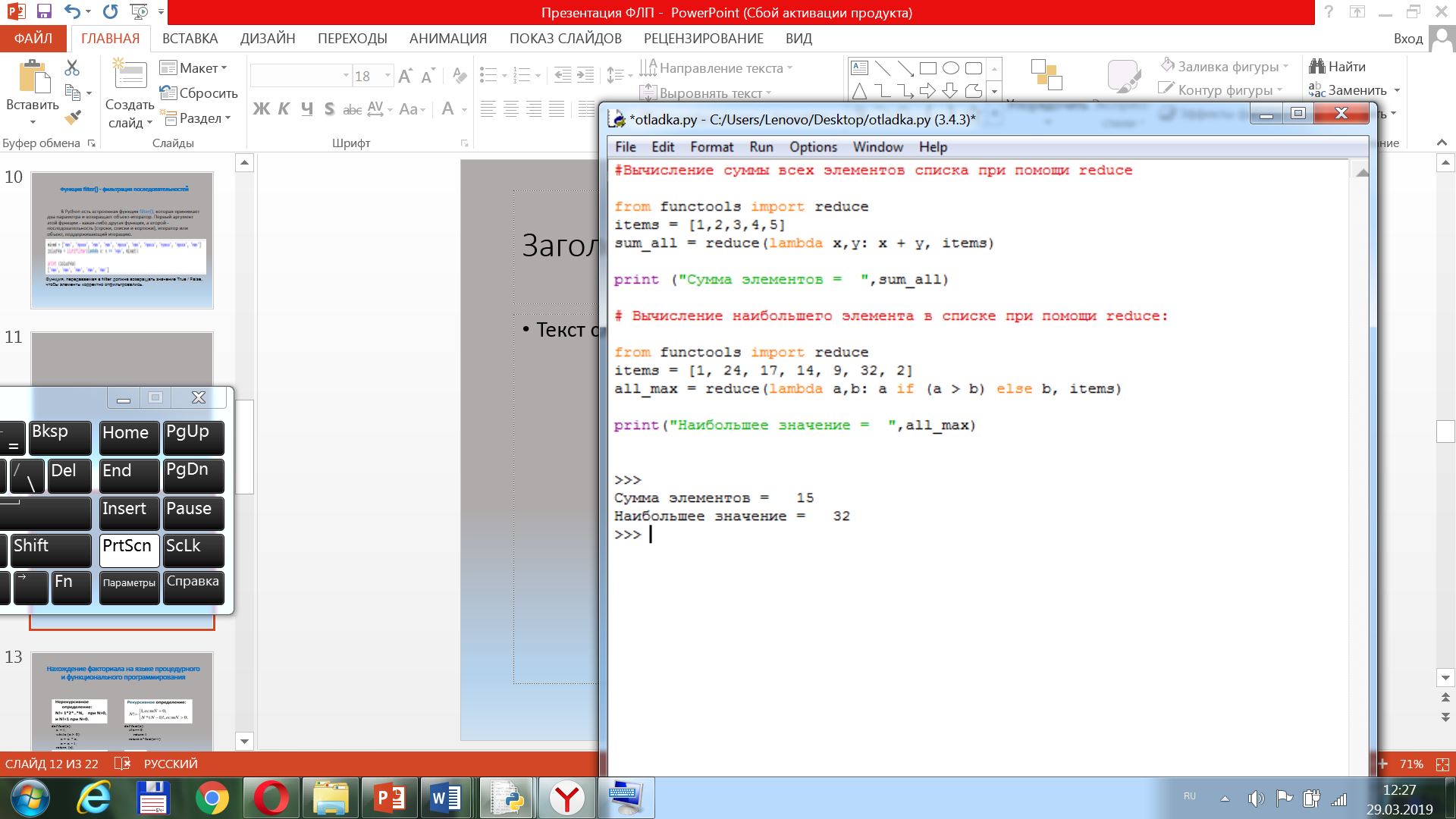
В Python есть встроенная функция filter(), которая принимает два параметра и возвращает объект-итератор. Первый аргумент этой функции - какая-либо другая функция, а второй - последовательность (строки, списки и кортежи), итератор или объект, поддерживающий итерацию.



Функция, передаваемая в **filter** должна возвращать значение True / False, чтобы элементы корректно отфильтровались.

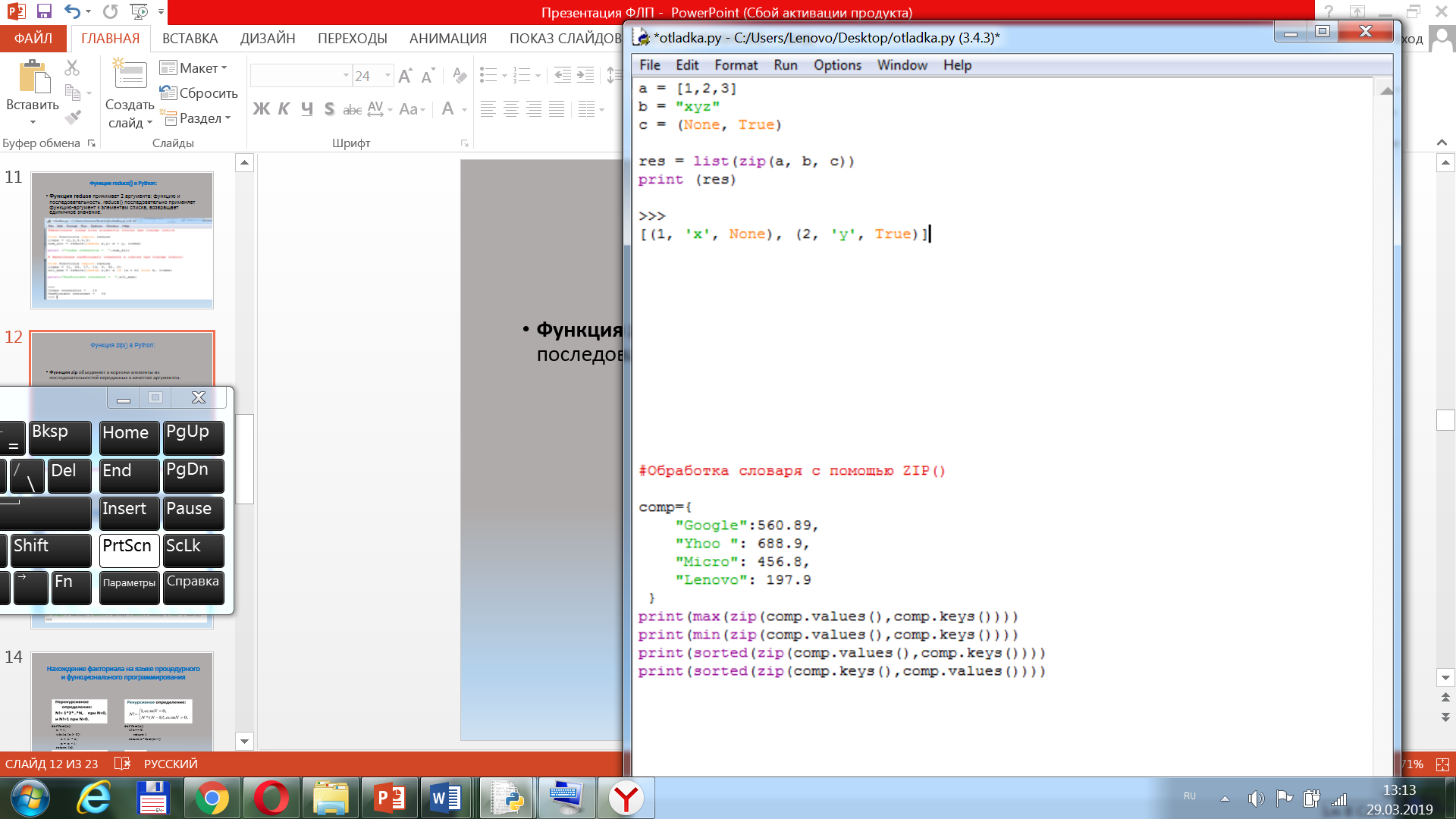
**Функция reduce() в Python:**

**Функция reduce** принимает 2 аргумента: функцию и последовательность. reduce() (ред’юс)последовательно применяет функцию-аргумент к элементам списка, возвращает единичное значение.

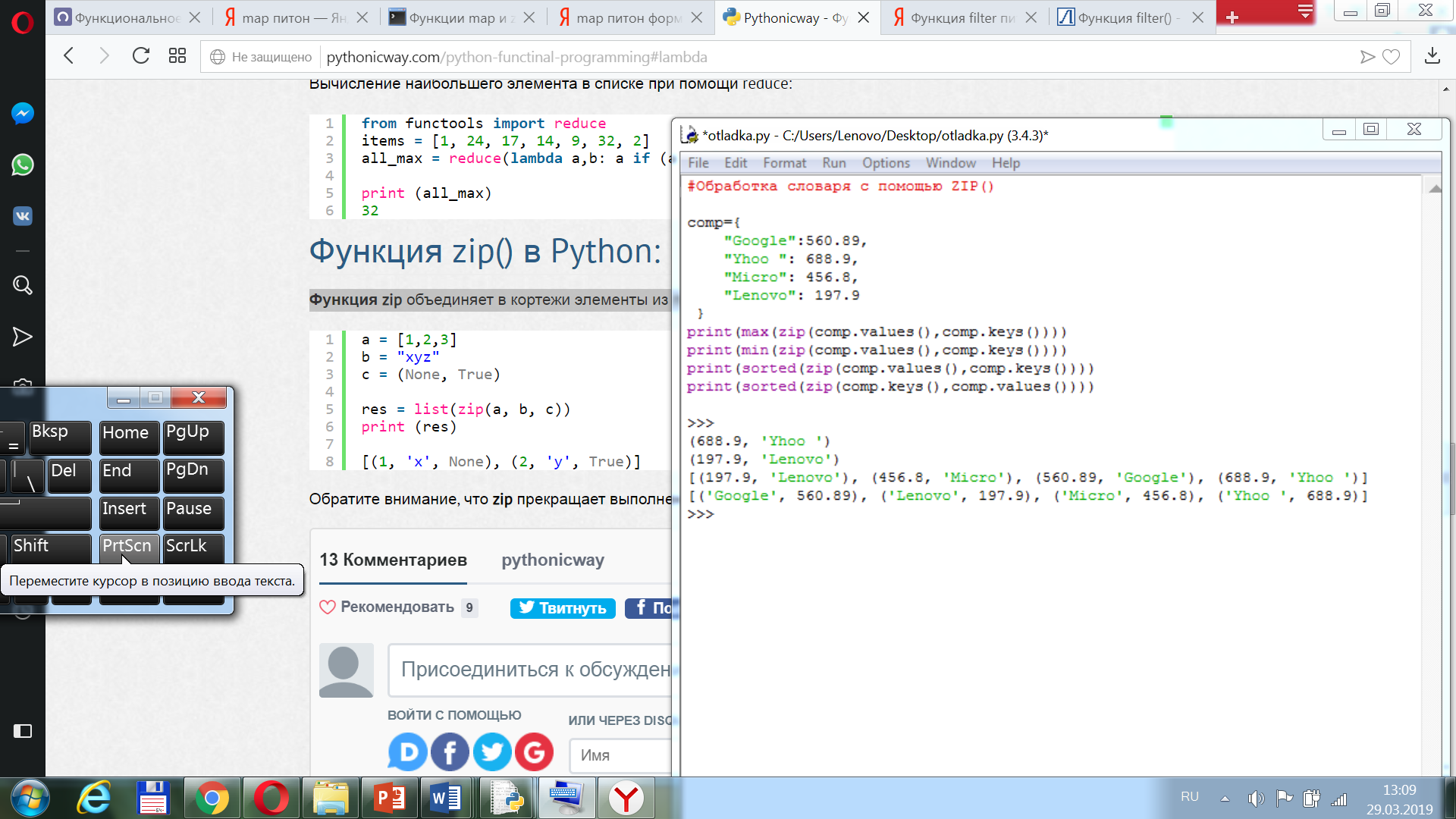


Функция zip() в Python:

**Функция zip ()**объединяет в кортежи элементы из последовательностей переданных в качестве аргументов.



**zip ()**  прекращает выполнение, как только достигнут конец самого короткого списка.



**Исключение команд управления потоком**

Используя приемы функционального программирования в Python применять эквиваленты блоков ‘if’/’elif’/’else’ в виде выражения.

