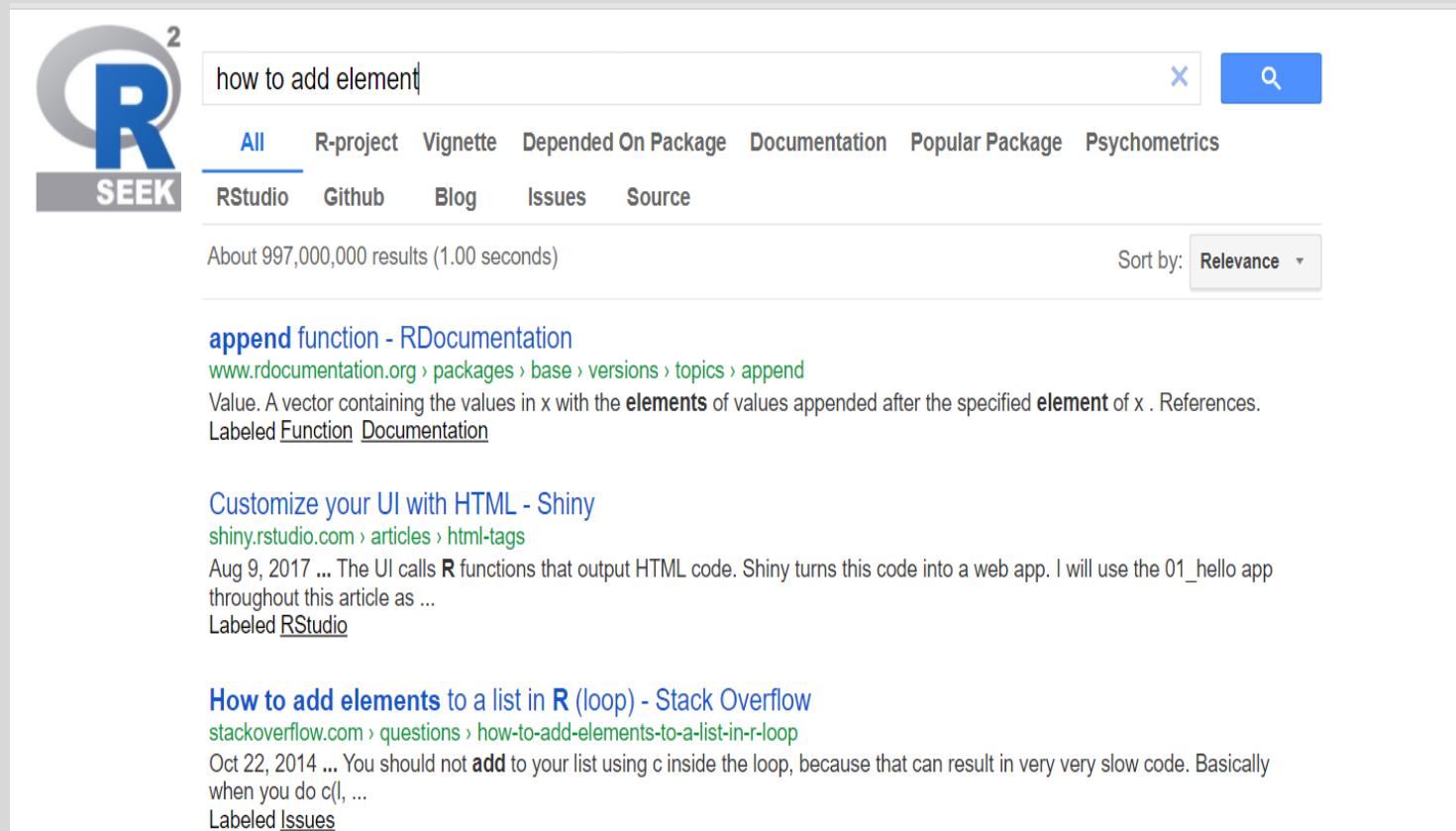




РЕКУРСИЯ

Занятие 3

Но для начала полезный сайт: rseek.org



The screenshot displays the Rseek search engine interface. At the top left is the Rseek logo, featuring a stylized 'R' with a superscript '2' and the word 'SEEK' below it. To the right of the logo is a search bar containing the text 'how to add element'. Below the search bar are several tabs: 'All', 'R-project', 'Vignette', 'Depended On Package', 'Documentation', 'Popular Package', and 'Psychometrics'. The 'All' tab is currently selected. Below these tabs are links for 'RStudio', 'Github', 'Blog', 'Issues', and 'Source'. To the right of the search bar is a blue button with a magnifying glass icon. Below the search bar, it says 'About 997,000,000 results (1.00 seconds)'. To the right of this is a 'Sort by:' dropdown menu set to 'Relevance'. The search results are listed below. The first result is titled 'append function - RDocumentation' and includes a breadcrumb trail: 'www.rdocumentation.org > packages > base > versions > topics > append'. The description states: 'Value. A vector containing the values in x with the **elements** of values appended after the specified **element** of x . References. Labeled [Function](#) [Documentation](#)'. The second result is titled 'Customize your UI with HTML - Shiny' and includes a breadcrumb trail: 'shiny.rstudio.com > articles > html-tags'. The description states: 'Aug 9, 2017 ... The UI calls **R** functions that output HTML code. Shiny turns this code into a web app. I will use the 01_hello app throughout this article as ... Labeled [RStudio](#)'. The third result is titled 'How to add elements to a list in R (loop) - Stack Overflow' and includes a breadcrumb trail: 'stackoverflow.com > questions > how-to-add-elements-to-a-list-in-r-loop'. The description states: 'Oct 22, 2014 ... You should not **add** to your list using c inside the loop, because that can result in very very slow code. Basically when you do c(l, ... Labeled [Issues](#)'.

Написание функций

- Мы с вами уже знакомы с функциями в R.
Например, когда использовали функцию `print()` или `sum()`.
- Мы также знаем, что функция работает следующим образом: она берет что-то на вход, потом с этим что-то делает (мы называем это “черный ящик”) и в конце что-то выводит.
Например функция `sum()` может брать массив из чисел и выводить их сумму.
- Но как она это делает? Что происходит в “черном ящике”? На эти вопросы мы и ответим с вами в этом разделе, а также научимся писать свои собственные функции.

Предположим перед нами стоит задача вычислить 5!, 6!, 2!.

```
# 6!  
otv = 1  
for(i in 1:6){  
  otv <- otv * i  
}  
print(otv)
```

```
# 5!  
otv2 = 1  
for(i in 1:5){  
  otv2 <- otv2 * i  
}  
print(otv2)
```

```
# 2!  
otv2 = 1  
for(i in 1:2){  
  otv2 <- otv2 * i  
}  
print(otv2)
```

- Уже можно заметить, что код занимает много места, да и глупо каждый раз писать заново практически одно и то же. Если же задача была сложнее, то проблем возникло бы больше.
- Для таких ситуаций существуют функции. Функции — это такие участки кода, которые изолированы от остальной программы и выполняются только тогда, когда вызываются. Функция задается следующим образом

Напишем функцию для факториала

```
# эта часть кода не будет выполняться
fact <- function(x){
  res <- 1
  for(i in 1:x){
    if(x == 0){
      return(res)
    }
    res <- res * i
  }
  return(res)
}

# теперь вызываем функцию
fact(0)
```

Видно, что код довольно упростился.

Теперь поговорим о нескольких замечаний в данном коде.

- Мы сначала называли функцию, потом написали “черный ящик”, а только потом вызвали ее. (соблюдайте только этот порядок)
- Если бы мы функцию не стали вызывать и нажали бы кнопку source (выполнение всего кода), то кусок кода с функцией бы никак не отобразился.
- Инструкция **return** может встречаться в произвольном месте функции, ее исполнение завершает работу функции и возвращает указанное значение в место вызова. По сути, это что-то вроде print() и break одновременно. В return мы указываем переменную, которая в итоге должна появиться на выходе.

Глобальные и локальные переменные

- Переменные, которые вы задаете за пределами функции называются **глобальными**. В RStudio они отображаются в правом верхнем углу. Если же вы инициализируете переменную внутри функции (например, переменная `s` внутри функции `SUM`), то такие переменные называются **локальными**. Если мы попробуем вывести локальную переменную за пределами функции, мы получим ошибку.
- Если переменная задана за функцией, то есть она глобальная, ее все равно можно использовать в функции

```
# Интересным получается результат, если мы используем локальную переменную с  
# таким же названием, как и глобальная.  
a <- 0  
f <- function(){  
  a <- 1  
  return(a)  
}  
f()
```

```
> f()  
[1] 1
```

```
a <- 0  
f <- function(){  
  return(a)  
}  
f()
```

```
> f()  
[1] 0
```

- Видим, что функция учла ту переменную, которая была локальной, проигнорировав глобальную.
- Также можно внутри функции задать глобальную переменную знаком `<<-`, но так лучше никогда не делать.

Переменные по умолчанию

- Функция необязательно может принимать только один аргумент, например, функция `matrix` принимает несколько аргументов (все их можно посмотреть, вызвав `help(matrix)`). Некоторые из этих аргументов уже чему-то равны по умолчанию. Это сделано, чтобы не вводить какие-то значения для них постоянно. Обычно это аргументы, которые могут принимать небольшое число значений (например, `TRUE` или `FALSE`). Так, например, в функции `matrix` есть аргумент `byrow`, который по умолчанию стоит в значении `FALSE` и означает, что элементы заполняются в матрице по столбцам.
- Создавая свои функции, мы тоже можем указывать подобного рода аргументы.
- Рассмотрим следующую задачу. Пусть у нас есть какой-то числовой вектор, и нам надо в каких-то случаях вывести максимальное число в этом векторе, а в каких-то случаях минимальное. Мы можем сделать это следующим образом.

Собственно функция

- Type – переменная, заданная по умолчанию

```
max_or_min <- function(x, type = 'max'){  
  if(type == 'max'){  
    k <- max(x)  
  } else {  
    k <- min(x)  
  }  
  return(k)  
}
```

```
max_or_min(c(1,10,2,4,100))  
max_or_min(c(1,10,2,4,100), type = 'min')
```

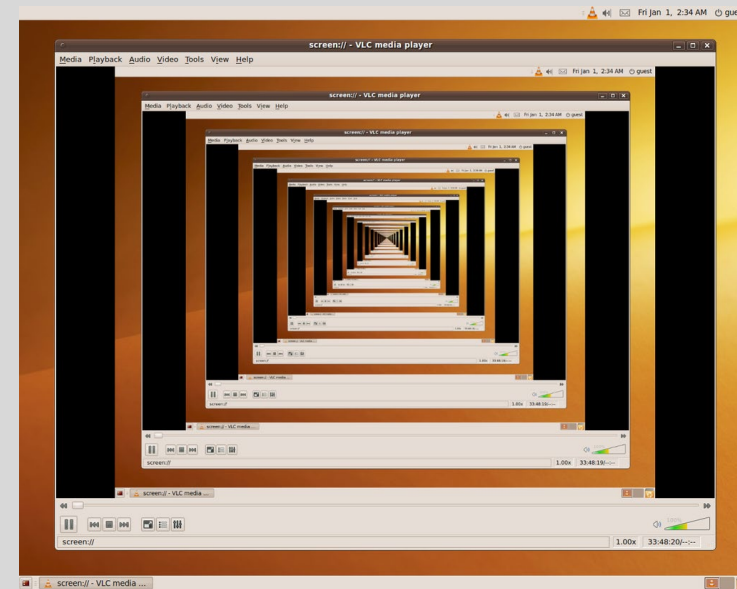

Рекурсия

- Рекурсивная функция это та функция, которая вызывает **саму себя**
- Рекурсия используется, когда задачу можно решить, разбив её на подзадачи
- В рекурсии есть базовый случай, который не делится на другие. Это наименьшая подзадача в задаче. К базовому случаю мы сводим нашу задачу, используя рекуррентный случай
- Далее приведу пример с матрешками и будет понятнее, но...

Сначала немного мемов!



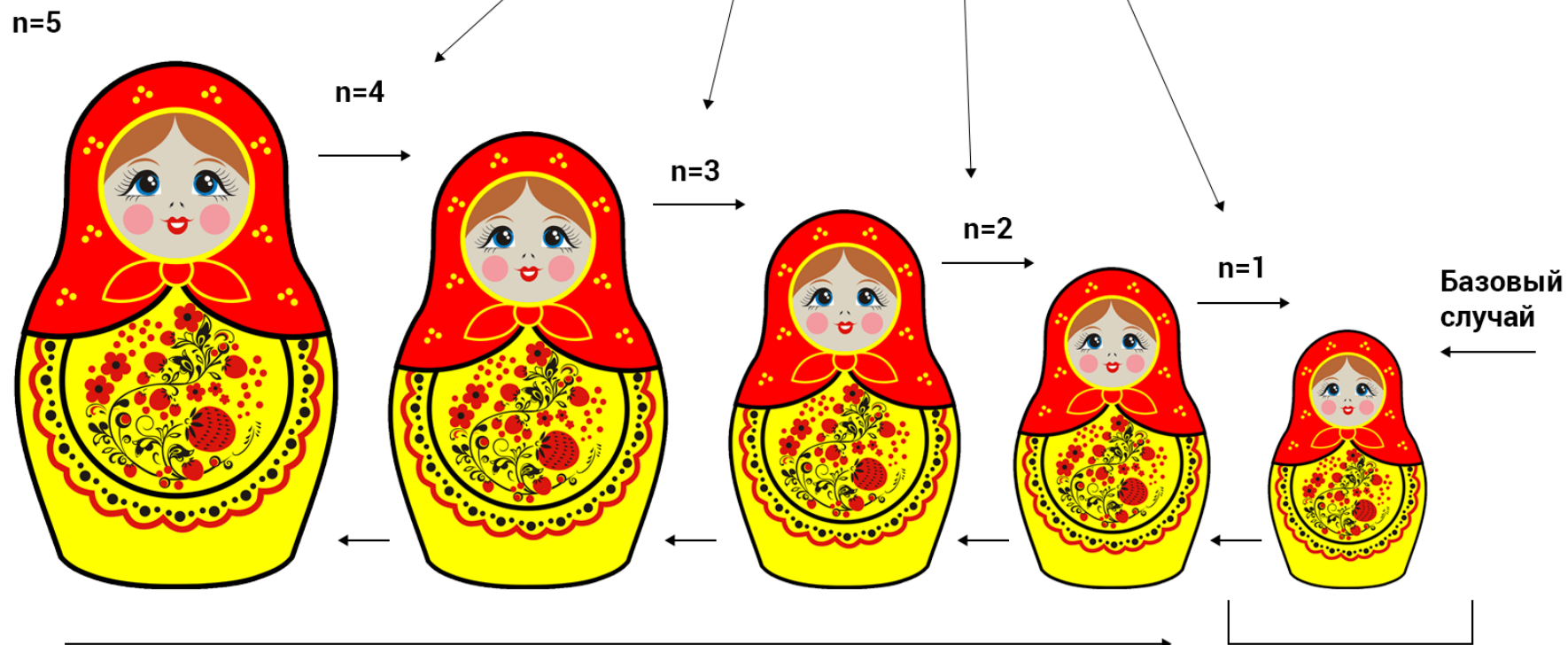
«Чтобы понять рекурсию, ты должен понять рекурсию!»



Популярный видос про матрёшку
(картинка кликабельна)



Рекуррентный случай



Сводим задачу к вычислению конкретного значения

В этот момент происходит непосредственное вычисление, и результат возвращается обратно