Что интересного есть в видео:

**Управляющие структуры:**

**Введение**

В R есть все те же операторы цикла, что и в, например, Сях. Разве что называются чутка по-другому.

**If-else**

Суть фактически та же, что и в других ЯП, но есть одна магическая конструкция:

y <- if(x > 3) {

10

} else {

0

}

Которая присваивает значение в зависимости от условия

**For**

Суть та же. Синтаксис:

for(i in 1:10) {

print(i)

}

**While**

Вообще ничего нового

**Repeat, Next, Break**

Repeat – это While, но без условия, т.е. ломается только break’ом. Это делает его достаточно опасным в использовании ввиду, например, того, что выполнение цикла может затянуться.

Next = continue из Сей, ломает только тот цикл, в который вложен непосредственно, не трогая те, в которые вложен этот цикл. Можно юзать для возвращения значения функции, например.

Под конец резюмирует циклы, рекомендуя какой где использовать, и какие баги при их использовании могут встретиться.

**Функции:**

**Ваша первая функция в R**

Начинает знакомить нас с функциями путём написания простенького скрипта. Рекомендует писать их в текстовых файлах, а не в консоли. Потом расскажет, как их в библиотеки R пихать.

\*Минутка фапа на MacOS\*

Напишем функцию, которая тупо складывает два принимаемых числа

**> add2<-function(x,y){**

**+ x+y //Функции всё равно возвращают последнее выполненное выражение**

**+ }**

**> add2(3,5)**

**[1] 8**

**>**

Следующая функция принимает вектор и выводит из него только те числа, что больше 10

**above10<-function(x){**

**use <- x > 10**

**x[use]**

**}**

А теперь та же функция, но юзверь ещё и вводит значение потолка

**> above<-function(x,n){**

**+ use <- x > n**

**+ x[use]**

**+ }**

**> c <- 1:20**

**> above(c,15)**

**[1] 16 17 18 19 20**

**>**

А если указать «**above<-function(x,n = 10)»**, то n будет 10 по-дефолту, если забыть её указать при вызове, например.

Последняя функция считает среднее значение для каждого столбца матрицы. Ничего нового с точки зрения синтаксиса, разве что он задаёт выражение «**removeNA = TRUE**» в списке переменных функции, дабы неопределённые в столбце элементы автоматически заменялись нулями и можно было вычислить значение по оставшимся.

**Работа с функциями, часть 1.**

А вот теперь он начинает рассказывать о функциях с нуля, лол. Эта часть посвящена основам написания и принципов работы. Далее по порядку:

1. Синтаксис написания функций
2. Как писать в функции аргументы и вызвать её. Там достаточно много вариантов, можно даже порядок переменных тасовать как угодно при вызове. Многое из этого полезно, например, для функций с длиннющим списком переменных.

**Работа с функциями, часть 2.**

Начинает он с описания т.н. «Ленивых» вычислений. Суть такова: если в аргументах функции объявлены переменные, которые вообще никак не используются в ходе работы этой функции, то если при вызове не указывать их, ошибки тоже не будет.

Ещё в R есть особый аргумент «…» (многоточие), которое позволяет переносить аргументы из других функций, переписывая только нужные в данный момент. Ещё это будет полезно в ООП, но потом. Также просто необходим, когда точное кол-во переменных в функции неизвестно, например, в функции, которая скрепляет любое кол-во полученных строк в одну. Если писать многоточие перед переменными, то все переменные после нужно обязательно по имени вызывать.

**Правила ограничения области видимости в R:**

**Привязка к символам**

Здесь он на примере собственнонаписанной функции lm(), которая также есть и в статистических библиотеках R, объясняет как среда отличит одну от другой. Прежде всего, даётся определение глобальной среды – это ваша рабочая среда со всем тем, что вы успели в ней повызывать и пообъявлять.

Фича ограничения области видимости является очень сильным отличием R от S.

Порядок поиска вызываемой функции:

1. Сначала тщательно обыскивается глоб среда
2. Если в глобальной среде нет, то идём в пространство имён подключённой – родительской. Явно подключённые имеют больший приоритет.
3. Продолжает, пока не просмотрит всё подключенное, а потом и не подключенное. Если не найдёт, то ALARM, ERROR и далее по списку

R поддерживает несколько способов вычисления области видимости: статический, динамический и лексический.

Правила лексического способа:

1. Среда – это набор пар «символ-значение»
2. У каждой среды есть родительская среда, у которой, в свою очередь, мб несколько детей
3. Средой без родителя мб только пустая среда
4. Каждая библиотека имеет пространство имён, которое тоже среда. Если мы берём функцию и связываем её со средой, то происходит «замыкание функции» (closure).

**Продолжение пред лекции**

Теперь он показывает весь этот кошмар на примерах.

Функция, определённая в другой функции, относится не к глобальной среде, а к среде родительской функции. Что это значит? Разные области видимости переменных, конечно.

**make.power <- function(n) {**

**pow <- function(x) {**

**x^n**

**}**

**pow**

**}** n определена в make.power, а значит видна и в pow.

На примере этой функции он показывает разницу между лексическим и динамическим способами поиска:

**y <- 10**

**f <- function(x) {**

**y <- 2**

**y^2+ g(x)**

**}**

**g <- function(x) {**

**x\*y**

**}**

С лексическим в g переменная y будет равна 10, т.к. он полезет искать её в родит среду.

С динамическим, g возьмёт значение y из той среды, откуда была вызвана, т.е. y будет равна 2.

Лексическое ограничения области видимости также исп в Python, Perl, Scheme и Common Lisp.

Последствия лексического способа:

- Все объекты суются в оперативку

- Каждая функция должна иметь указатель на среду, в которой была определена, а она мб в заднице.

**Грамотное использование правил ограничения (необязательная лекция, да и нудный пример он там разбирает)**

**Стандарты написания кода в R:**

Здесь он перечислит минимально требуемые правила написания хорошего, годного (не как у Лёхи) и читабельного кода.

1. Пишите весь код в текстовых файлах, ибо совместимость со всем
2. Делайте отступы у всяких вложенных конструкций…
3. …но не увлекайтесь – не более 80 колонок вправо. Если не влезает, то заканчивай вложенные циклы пихать и на функции разбивай.

Он описал примерные правила для отступов, но мне влом их перечислять, ибо это всё равно дело вкуса. Функции он рекомендует использовать строго для одного, определённого в названии, базового действия.

Ещё разбитие на маленькие функции ощутимо упростит дебаг, ибо среда говорит только в какой функции ошибка, а не в какой строке конкретно.

**Дата и время в R:**

В R Предусмотрены особые типы данных:

1. Date для даты
2. POSIXct и POSIXlt для времени
3. Задаются в формате ГГГГ.ММ.ДД

Как и в Сях, даты хранятся в формате «сколько дней прошло с 1-го января 1970» (в Сях с 1900, правда)

POSIX – группа стандартов для вычисления времени на компе

POSIXct – время представлено, как очень длинное целое число/большой целочисленный вектор

POSIXlt – время представлено в виде списка и хранит всякую муру в духе «день недели», «номер месяца» и пр.

Конвертировать из одного в другой, используя as. , не возбраняется

Ещё есть функции, которые работают с датами и временем одновременно

Существует функция strptime(), которая может конвертировать время, написанное в строке, в человеческие временные форматы