Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Мануал по работе с ClosureScript

Москва — 2023 г.

Содержание

Введение

Что такое ClosureScript?

ClosureScript [1] — компилятор для Closure [2], выдающий в результате код на JavaScript.

Что такое Closure?

Closure — диалект языка LISP, являющийся динамическим компилируемым языком программирования, поддерживающий доступ к фреймворкам, написанным на Java. Из-за своего родства с LISP поддерживает функциональное программирование и использование макросов.

Что нужно для того, чтобы начать писать на Closure?

Во-первых, среда разработки или текстовый редактор для Closure — подходящих несколько, например Emacs, Intellij IDEA, VS Code. В рамках данного мануала будет рассмотрено использование текстового редактора VS Code для работы с Closure.

Во-вторых, сам Closure — он доступен для установки под MacOS, Linux и Windows.

В-третьих,

Установка

VS Code и Calva

Установить VS Code под свою платоформу можно по ссылке: `https://code.visualstudio.com/Download`

Для тех, кто не имеет опыта использования IDE или желает научиться использовать VS Code, рекомендуются к прочтению следующие статьи:

1) `https://habr.com/ru/post/490754/` — статья на русском языке

2) `https://code.visualstudio.com/docs/introvideos/basics` — статья на английском языке

Далее необходимо установить Calva — расширение для VS Code, поддерживающее ClojureScript и помогающее разрабатывать ПО на Clojure. Для этого в левом меню VS Code необходимо перейти в раздел "Расширения" (см. скриншот 1), сделать поиск "Calva" и установить найденное расширение (см. скриншот 2).

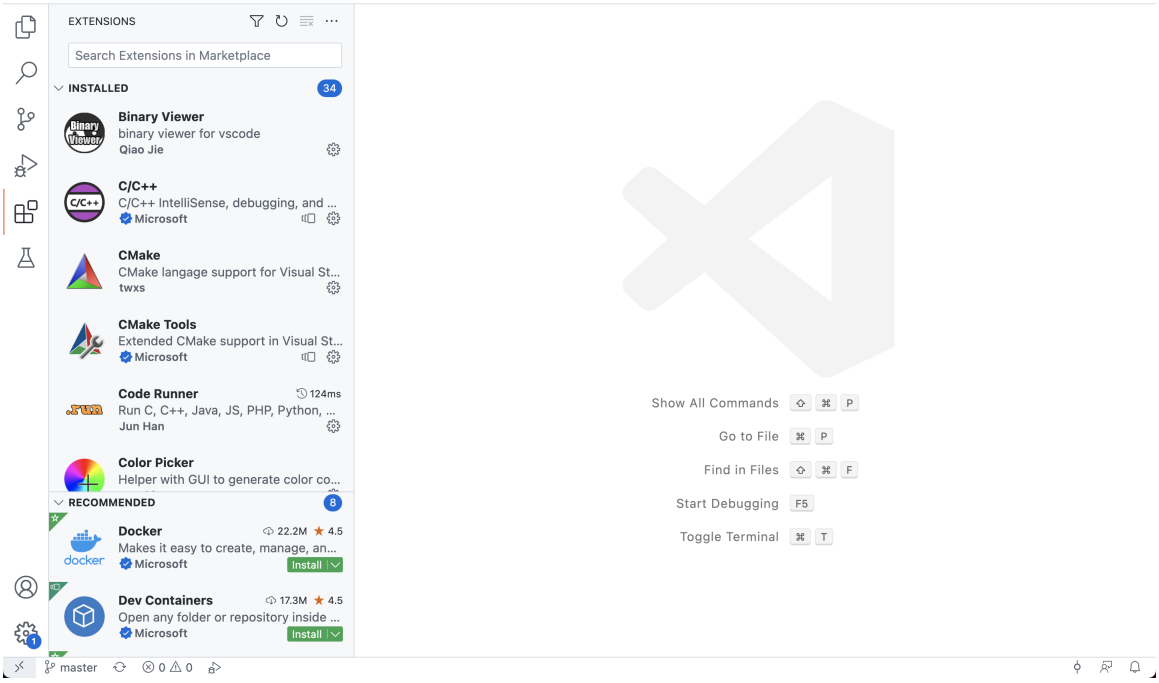

Рисунок 1 – Раздел "Расширения" VS Code

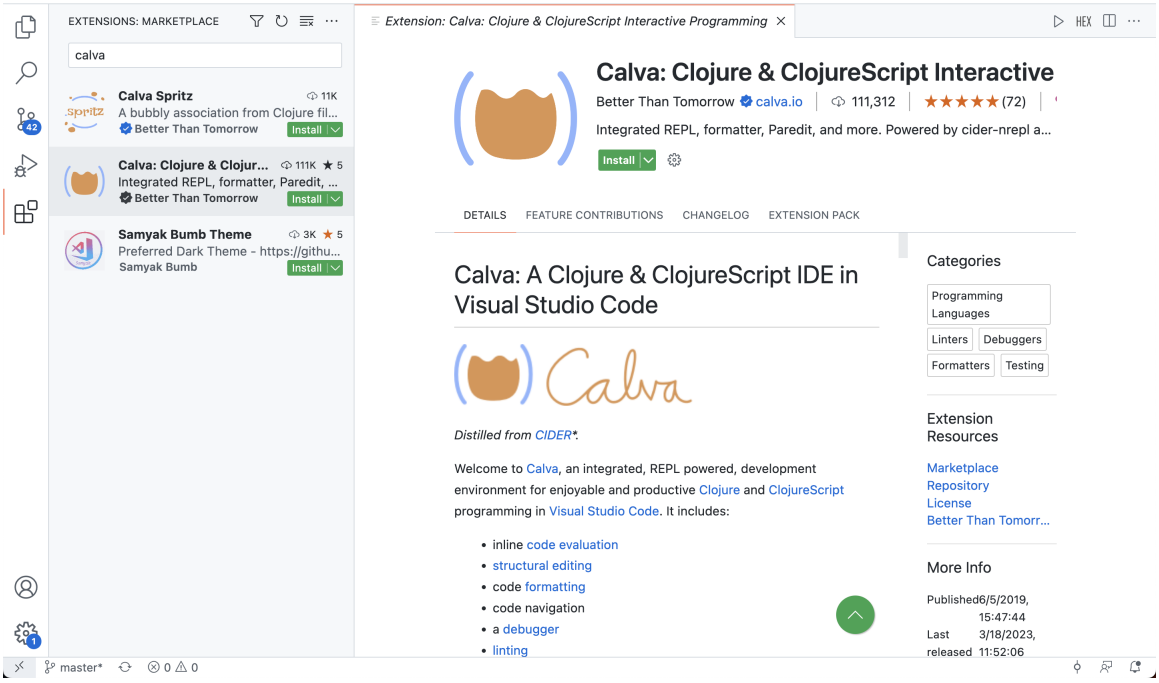

Рисунок 2 – Расширение Calva

Clojure

Для того, чтобы установить Clojure, воспользуемся инструкциями с официального сайта: `https://clojure.org/guides/install_clojure`.

В приведённой статье (на английском языке) присутствую инструкции по установке Clojure на ОС MacOS, Linux-подобные ОС (Ubuntu, Debian), а также Windows. В данном мануале будет рассмотрена установка Clojure на Linux-подобные ОС посредством установщика пакетов brew.

Для установки необходимо открыть терминал и выполнить в нём следующие команды:

Листинг 1 – Установка brew

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/
    Homebrew/install/HEAD/install.sh)"
(echo; echo 'eval "$(/home/linuxbrew/.linuxbrew/bin/brew shellenv)"'
    ) >> /home/<YOURUSERNAME>/.profile
eval "$(/home/linuxbrew/.linuxbrew/bin/brew shellenv)"
sudo apt-get install build-essentials
```

После установки brew необходимо установить и сам Clojure:

Листинг 2 – Установка Clojure

```
1 brew install clojure/tools/clojure
```

LISP

Поскольку Clojure является диалектом LISP, стоит также установить и базовый интерпретатор LISP, например, Common Lisp. Для ОС семейства Linux это можно сделать следующей командой:

Листинг 3 – Установка Common Lisp

```
1 sudo apt-get install sbcl
```

Начало работы и примеры

Hello World, Clojure

Откроем VS Code и нажмём Ctrl + Shift + P (данное сочетание клавиш откроет паллет управления в VS Code) и введём следующий текст: Calva: Fire up the Getting Started REPL Среди появившихся результатов выберем единственный:

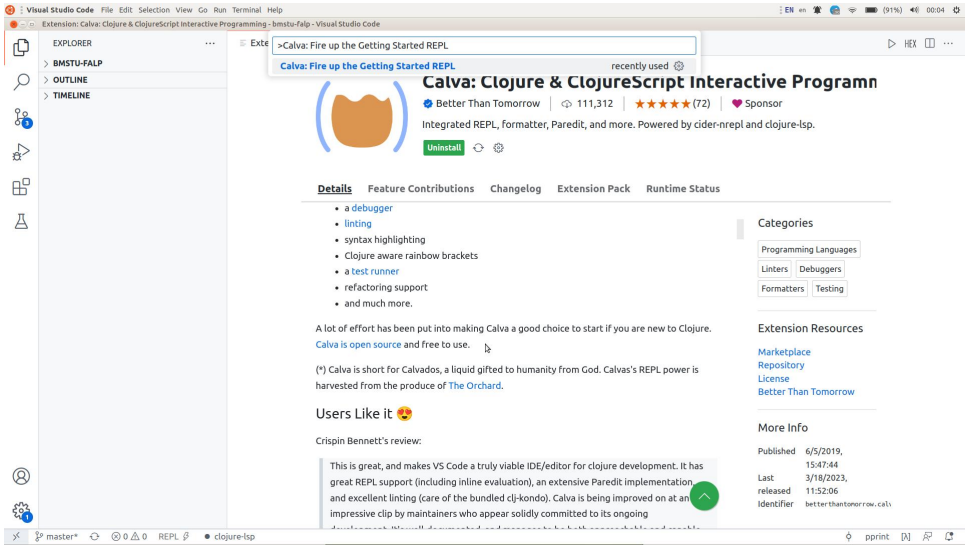

Рисунок 3 – Запуск Calva

После запуска Calva мы увидим примерно следующее:

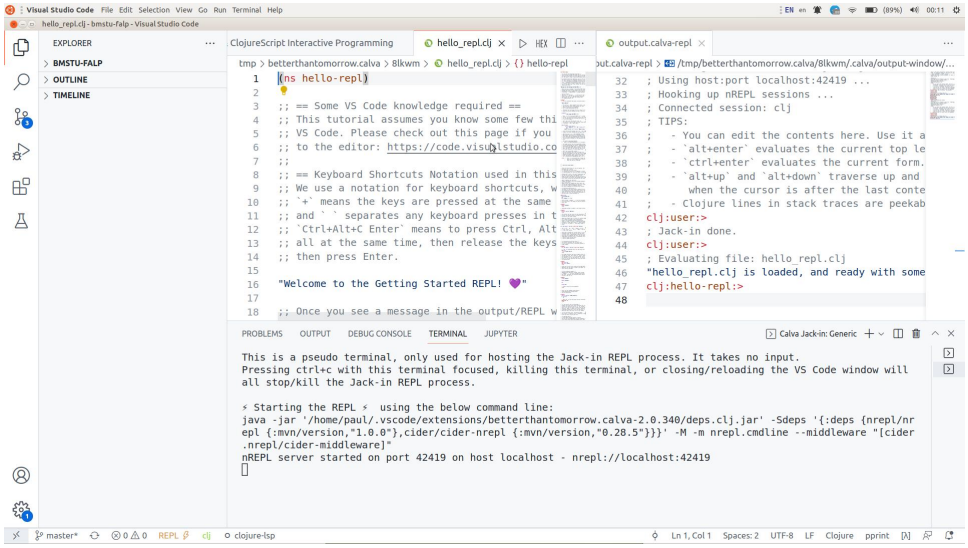

Рисунок 4 – После запуска Calva

Очистим содержимое левого файла и напишем в него ”Hello, World!”:

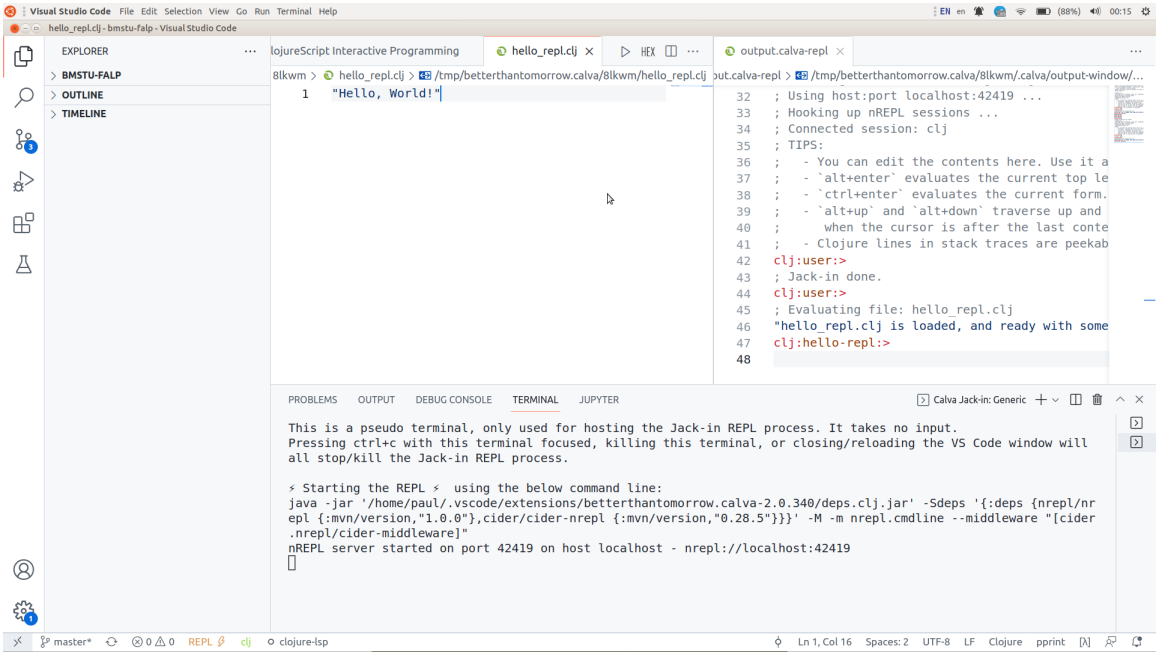

Рисунок 5 – Напишем ”Hello, World!”

Нажмём Alt + Enter и увидим результат:

Рисунок 6 – Hello, World!

Hello World, ClojureScript

Чтобы запустить Hello World с использованием ClojureScript, мы будем выполнять действия по следующему гайду:

`https://clojurescript.org/guides/quick-start`

Для этого создадим папку hello-world и настроим её содержимое следующим образом:

Листинг 4 – ClojureScirpt

```
hello-world          # Our project folder
    src              # The CLJS source code for our project
        hello_world  # Our hello_world namespace folder
            core.cljs # Our main file
    cljs.jar         # (Windows only) The standalone Jar you
      downloaded earlier
    deps.edn         # (macOS/Linux only) A file for listing our
      dependencies
```

Содержимое deps.edn:

Листинг 5 – Содержимое deps.edn

```
{:deps {org.clojure/clojurescript {:mvn/version "1.11.54"}}}
```

Содержимое src/hello_world/core.cljs:

Листинг 6 – Содержимое src/hello_world/core.cljs

```
(ns hello-world.core)

(println "Hello world!")
```

После чего запустим в терминал следующую команду:

Листинг 7 – Запуск в терминале

```
clj -M --main cljs.main --compile hello-world.core --repl
```

В браузере при этом должна открыться следующая страница:

После этого в терминале, из которого произходил запуск, должно появиться сообщение Hello World!

Попробуем добавить несколько функций. Для этого изменим содержимое src/hello_world/core.cljs:

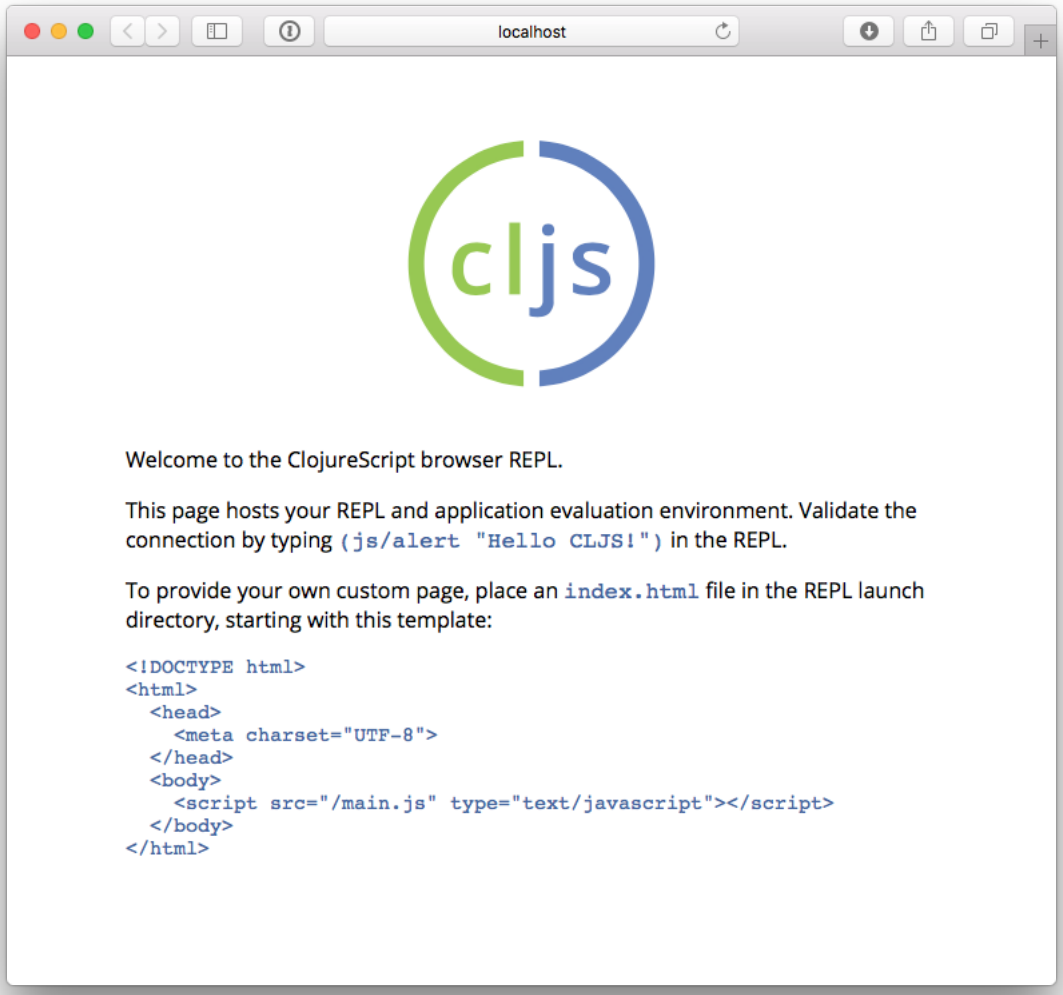

Рисунок 7 – Hello, World!

Листинг 8 – Содержимое src/hello_world/core.cljs

```
1  (ns hello-world.core)
2
3  (println "Hello world!")
4
5  ;; ADDED
6  (defn average [a b]
7      (/ (+ a b) 2.0))
8
9  (defn plus [a b]
10     (+ a b))
```

Перекомпилируем рабочее пространство, выполнив в терминале следующие команды:

Листинг 9 – Обновление страницы проверка новых функций

```
1 (require '[hello-world.core :as hello] :reload)
2 (hello/average 20 13)
3 (hello/plus 1 21)
```

В терминале должны отобразиться результаты работы функций - числа 16.5 и 22.

Отдельно отметим, что в папке out можно найти скомпилированный JavaScript код. Исследовав её содержимое, можно придти к выводу, что полученный код является ни слишком оптимизированным. Для того, чтобы получить более оптимизированную версию кода, необходимо запустить ClojureScript компилятор со следующими опциями:

Листинг 10 – Получение оптимизированной сборки

```
1 clj -M -m cljs.main --optimizations advanced -c hello-world.core
```

Оптимизированная сборка (со значением advanced) может быть использована для получения итоговой сборки, которая будет использована не разработчиками, а пользователями.

Примеры

Рассмотрим подробнее основные синтаксические конструкции ClojureScript на примерах.

Создание переменных и констант

Создать переменные и константы в ClojureScript можно несколькими способами, при помощи ключевых слов def, let, а также конструкции with-local-vars. Рассмотрим каждую из них подробнее.

Ключевое слово def по своему действию похоже на var в JavaScript — оно создаёт глобальную переменную, связывая его со значением:

Листинг 11 – Использование var

```
1 (def my-name "Fred")
2
3 my-name
4 ;; "Fred"
```

При этом даже если def используется внутри функции, создаваемая переменная всё равно будет глобальной:

Листинг 12 – Использование var внутри функции

```
1 (defn mk-global [value]
2   (def i-am-global value))
3
4 mk-global
5
6 (mk-global [4 8 15 16 23 42])
7
8 i-am-global
9 ;; [4 8 15 16 23 42]
```

Для того, чтобы использовать локальные константы и локальные переменые, существуют let и with-local-vars. Использование let осуществляется следующим образом:

Листинг 13 – Правило использования let

```
1 (let [bindings]
2   expr1
3   expr2
4   ...
5   expr-n)
```

Так, для того, чтобы создать локальную константу с именем a со значением 10 и тут же вычислить её, нужно написать следующее:

Листинг 14 – Пример использования let

```
1 (let [y 10]  y) ;; 10
```

Локальные переменые же создаются при помощи конструкции with-local-vars, которая имеет абсолютно аналогичный синтаксис:

Листинг 15 – Пример использования with-local-vars

```
1 (defn example [x]
2     (with-local-vars [y 10 z x]
3         (+ @y @z)))
```

Данная функция складывает переданое ей число и 10. Для того, чтобы изменить значение переменной, необходимо использовать функцию var-set, а для того, чтобы получить её значение - функцию var-get или символ @:

Листинг 16 – Пример использования with-local-vars

```
1 (defn example [x]
2     (with-local-vars [y 10 z x]
3         (var-set y 20)
4         (+ @y (var-get z))))
```

Данная функция складывает переданное ей число и 20, при этом осуществляет запись значения 20 в локальную переменную.

Осуществление выбора

Как условные конструкции в ClojureScript рекомендуется использовать ключевые слова if, cond и then. Использование if осуществляется следующим образом:

Листинг 17 – Синтаксис использования if

```
1 (if test-expr then-expr else-expr)
```

Таким образом, if принимает 3 выражения: условие, выражение, которое будет вычислено в случае истинности условия и выражение, вычисленое в случае, если условие окажется ложным. По своему действию if в ClojureScript похоже на работу тернарного оператора в JavaScript.

Использование if удобно при небольшой вложенности вычисляемых выражений. Если уровень вложенности больше трёх, рекомендуется использовать

функцию cond, которая работает, аналогично switch-case конструкции в других языках программирования:

Листинг 18 – Пример использования cond

```
(let [grade 85]
        (cond
          (>= grade 90) "A"
          (>= grade 80) "B"
          (>= grade 70) "C"
          (>= grade 60) "D"
          :else "F"))
```

Использование слова when похоже на if, но без вычисления выражений в случае, когда условие ложно:

Листинг 19 – Синтаксис использования when

```
(when test-expr some-value)
```

Использование циклов

В ClojureScript существуют циклы с счётчиком (for), циклы с предусловием (while), а также конструкции loop, recur и doseq.

Цикл с счётчиком for работает следующим образом:

Листинг 20 – Синтаксис использования for для 2-х последовательностей

```
(for [elem1 sequence1
      elem2 sequence2]
  expr)
```

Этот цикл for состоит из трёх основных этапов,

Он связывает каждый элемент из последовательности sequence1 к счётчику elem1, делает то же самое для последовательности sequence2, после чего вычисляет в очередной итерации выражение expr с использованием текущей комбинации элементов elem1 и elem2.

Так, чтобы вычислить квадраты всех чисел от 0 до 10, необходимо написать следующее:

Листинг 21 – Пример использования for для одной последовательности

```
(for [n (range 10)]
  (* n n))
;; (0 1 4 9 16 25 36 49 64 81)
```

Цикл с счётчиком for может быть использован для получения, например, декатового произведения набора последовательностей:

Листинг 22 – Пример использования for для трёх последовательностей

```
(let [colors [:magenta :chartreuse :taupe]
      sizes [:sm :md :lg :xl]
      styles [:budget :plain :fancy]]
  (for [color colors
        size sizes
        style styles]
    [color size style]))
;; ([:magenta :sm :plain] [:magenta :sm :regular] [:magenta :sm :
   fancy]
;; ... [:taupe :xl :plain] [:taupe :xl :regular] [:taupe :xl :fancy
   ])
```

Также у ключевого слова for имеются параметры, позволяющие лучше настраивать его работу: let, when и while, принцип работы которых можно понять на следующем примере:

Листинг 23 – Пример использования for с let

```
(for [n (range 100)
      :let [square (* n n)]
      :when (even? n)
      :while (< n 20)]
  (str "n␣is␣" n "␣and␣its␣square␣is␣" square))

;; ("n␣is␣0␣and␣its␣square␣is␣0"
;; "n␣is␣2␣and␣its␣square␣is␣4"
;; "n␣is␣4␣and␣its␣square␣is␣16"
;; ...
;; "n␣is␣18␣and␣its␣square␣is␣324")
```

Таким образом, let позволяет создать локальные переменные для тела цикла for, when и while определить, для каких значений вычислить это тело. Найти различие между when и while предлагается самостоятельно.

Цикл с предусловием while работает так же, как и в других языках программирования:

Листинг 24 – Синтаксис использования while

```
1 (while test & body)
```

Предлагается рассмотреть работу цикла while на примере функции, создающей вектор переданной размерности, состоящий из элементов с переданным значением (функция conj позволяет добавить значение в конец списка, а функция inc — увеличить значение на 1):

Листинг 25 – Пример использования while

```
1 (defn vecMake
2   [n val]
3   (with-local-vars [result [], i 0]
4     (while (< @i n)
5       (var-set result (conj @result val))
6       (var-set i (inc @i)))
7     @result))
```

Полезные ссылки для дальнейшего изучения

`https://clojurescript.org/` — официальный сайт ClojureScript

`https://www.learn-clojurescript.com/` — учебник по ClojureScript на английском

`https://clojuredocs.org/` — документация ClojureScript с примерами работы

`https://ericnormand.me/guide/clojurescript-tutorial` — краткий учебник по созданию простого web-приложения на ClojureScript

`https://lambdaisland.com/guides/clojure-repls/clojurescript-repls` — краткий справочник по использованию ClojureScript в разных окружениях