

Разработка кроссплатформенных
программных систем.
**Лекция №2. Встраивание и расширение
скриптовых языков**

П.Н. Советов

РТУ МИРЭА, 2022

Высокоуровневый язык программирования, предназначенный для **управления некоторой системой**, а также для **расширения ее функциональности**.

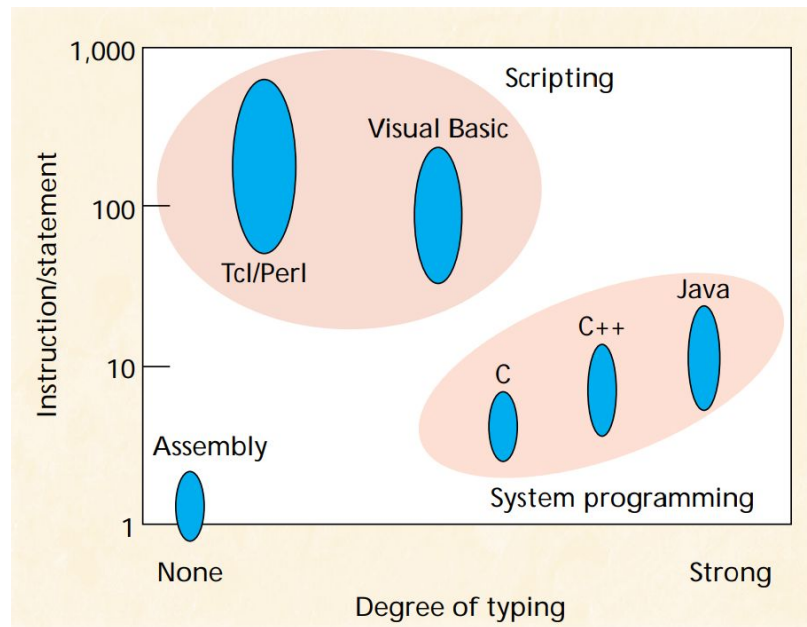
Скриптовые языки предназначены, зачастую, для программирующих **экспертов в предметной области**, а не для профессиональных программистов.

Скриптовые языки, в основном, используют **динамическую типизацию** и имеют реализацию на основе **интерпретатора**.

Другие названия:

- Язык сценариев.
- Язык-клей (glue language).

- Командные языки Bash и Powershell.
- Lua.
- Python.
- Lisp.
- Forth.
- Tcl.
- JavaScript.
- ...



Встраивание скриптового языка в приложение позволяет управлять этим приложением и расширять его функциональность.

Расширение скриптового языка предполагает добавление новых, пользовательских конструкций в базовый язык или добавление новых библиотек.

Архитектура на основе встраивания скриптового языка предполагает наличие:

- платформозависимого ядра, реализованного на системном языке,
- прикладной части, реализованной на скриптовом языке, встроенном в приложение.

Приложение

Встроенный
скриптовый язык
(прикладная часть)

Хост-язык (ядро)

- **Lisp** в САПР Autocad.
- **Python** в 3d редакторе Blender.
- **Tcl** во многих САПР из области микроэлектроники.
- **Lua** в большом количестве игр, Adobe Lightroom, СУБД. Tarantool, веб-сервере Nginx.
- Язык **1C** в системе 1C:Предприятие.

Прежде чем двигаться дальше, необходимо разобраться со статическими и динамическими библиотеками.

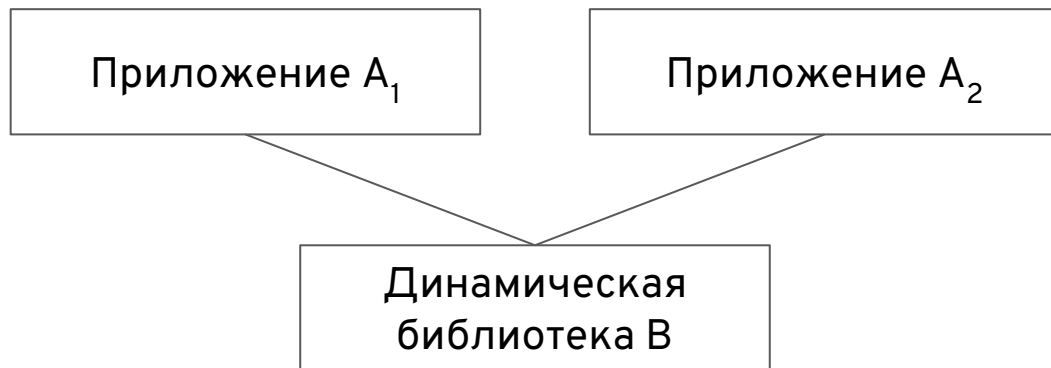
Библиотека представляет собой набор повторно используемых **скомпилированных программных модулей**.

Основной код приложения связывается со
статической библиотекой на этапе компиляции.

Приложение A_1
Статическая библиотека В

Приложение A_2
Статическая библиотека В

Основной код приложения связывается с **динамической библиотекой во время выполнения программы.**



```
gcc ... -c code.c -o code.o
```

```
ar rcs libcode.a code.o
```

Здесь gcc используется в режиме компиляции (-c) с созданием выходного **объектного модуля** code.o.

Затем объектный модуль добавляется с помощью **команды ar** в архив libcode.a. Этот архив и является статической библиотекой. В имени архива необходимо использовать префикс lib.

```
gcc ... -L. -lcode ...
```

Здесь gcc используется для компиляции основного кода приложения **совместно с кодом статической библиотеки**.

Ключ **-L** указывает на директорию, содержащую библиотеку.

Ключ **-l** задает имя подключаемой библиотеки. **Префикс lib указывать не нужно.**

```
gcc ... -shared -o libcode.dll
```

Здесь gcc используется в режиме создания динамической (разделяемой, -shared) библиотеки с **формированием dll-файла** (для Windows).

```
gcc ... libcode.dll ...
```

Здесь gcc используется для компиляции основного кода приложения с указанием динамической связи с **расположенным отдельно файлом libcode.dll**.

- Появился в 1993 году, разработан в Бразилии.
- Динамически типизированный, со сборщиком мусора.
- Реализован на C89.
- Компактный: насчитывает около 25000 строк кода.
- Легко настраивается и встраивается в приложение.



```
function fact(n)
    local r = 1 -- локальная переменная
    for i = 1, n do
        r = r * i
    end
    return r
end

print(fact(5))
```

- Шпаргалка: <https://tylerneylon.com/a/learn-lua/>
- Еще одна шпаргалка:
<http://thomaslauer.com/download/luarefv51.pdf>
- Маленькая книга о Lua (рус.):
<https://disce.ru/little-library/lua>
- Lua 5.3 Руководство (рус.):
http://lua.org.ru/contents_ru.html

Lua C API – интерфейс встраивания интерпретатора Lua в приложение на C.

Обмен данными между интерпретатором и программой на C происходит с помощью **стека**.

При запуске функции на C из интерпретатора функция получает **новый стек**, в котором хранятся переданные аргументы и в который функция записывает результаты.

в API-функциях используется два вида индексации в стеке:

- положительная – от **дна** стека (1 – первый элемент).
- отрицательная – относительно **вершины** стека (-1 – вершина).

- `lua_State *L = luaL_newstate()` – создать новый экземпляр интерпретатора.
- `lua_close(L)` – удалить созданный экземпляр интерпретатора.
- `luaL_openlibs(L)` – использовать стандартную библиотеку.
- `luaL_dofile(L, "file.lua")` – интерпретировать код из файла.
- `luaL_dostring(L, "string")` – интерпретировать строку.

```
lua_pushnumber(L, 2); -- положить 2 на стек  
lua_pushnumber(L, 3); -- положить 3 на стек  
lua_arith(L, LUA_OPADD); -- забрать со стека 2 и 3, положить 2+3  
printf("%f\n", lua_tonumber(L, -1)); -- напечатать вершину стека
```

Языковое связывание (language binding) – API, позволяющий использовать библиотеку, реализованную на другом языке.

Архитектура приложения на основе расширения скриптового языка предполагает наличие:

- базовой переносимой части, реализованной на скриптовом языке,
- набора расширений, реализованных на системных языках.

Приложение

Расширение на системном языке
Связующий API (прослойка)
Скриптовый язык (ядро)

Такие модули могут использоваться, к примеру, для работы с различными **аппаратными устройствами**, **API ОС** или для увеличения **быстродействия** приложения.

Наиболее простой способ подключения динамической библиотеки на С — использование модуля **ctypes**.

Документация: <https://docs.python.org/3/library/ctypes.html>

Функция `LoadLibrary` загружает динамическую библиотеку и возвращает объект-ссылку на загруженные функции.

Для каждой функции необходимо указать аргументы и результат. Пример использования импортированной функции сложения `add`:

```
add_module = ctypes.cdll.LoadLibrary(...)
add = add_module.add
add.argtypes = (ctypes.c_int, ctypes.c_int)
add.restype = ctypes.c_int
print(add(2, 2))
```