

Углублённое программирование на C++

Расширения на C

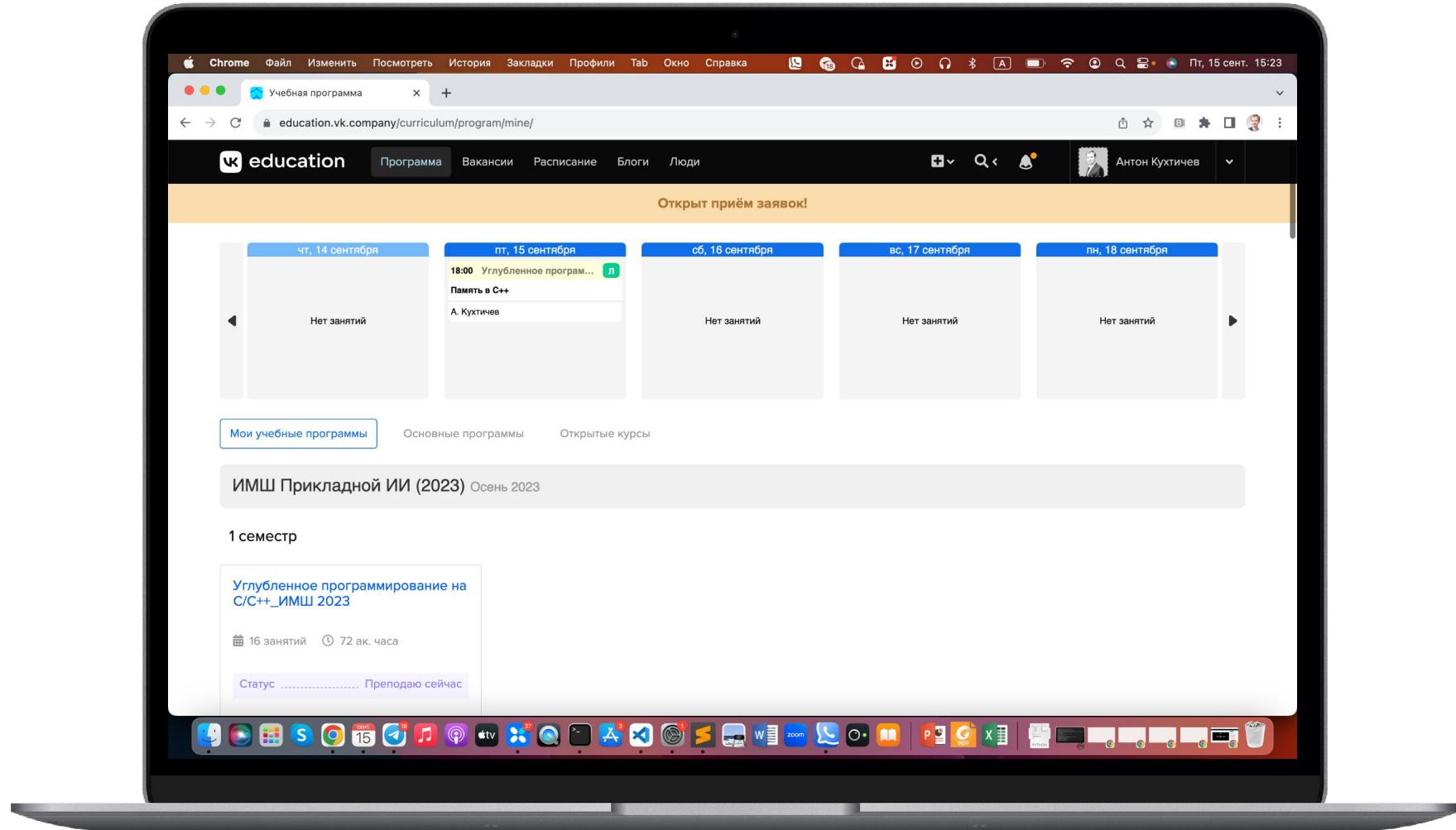
Кухтичев Антон



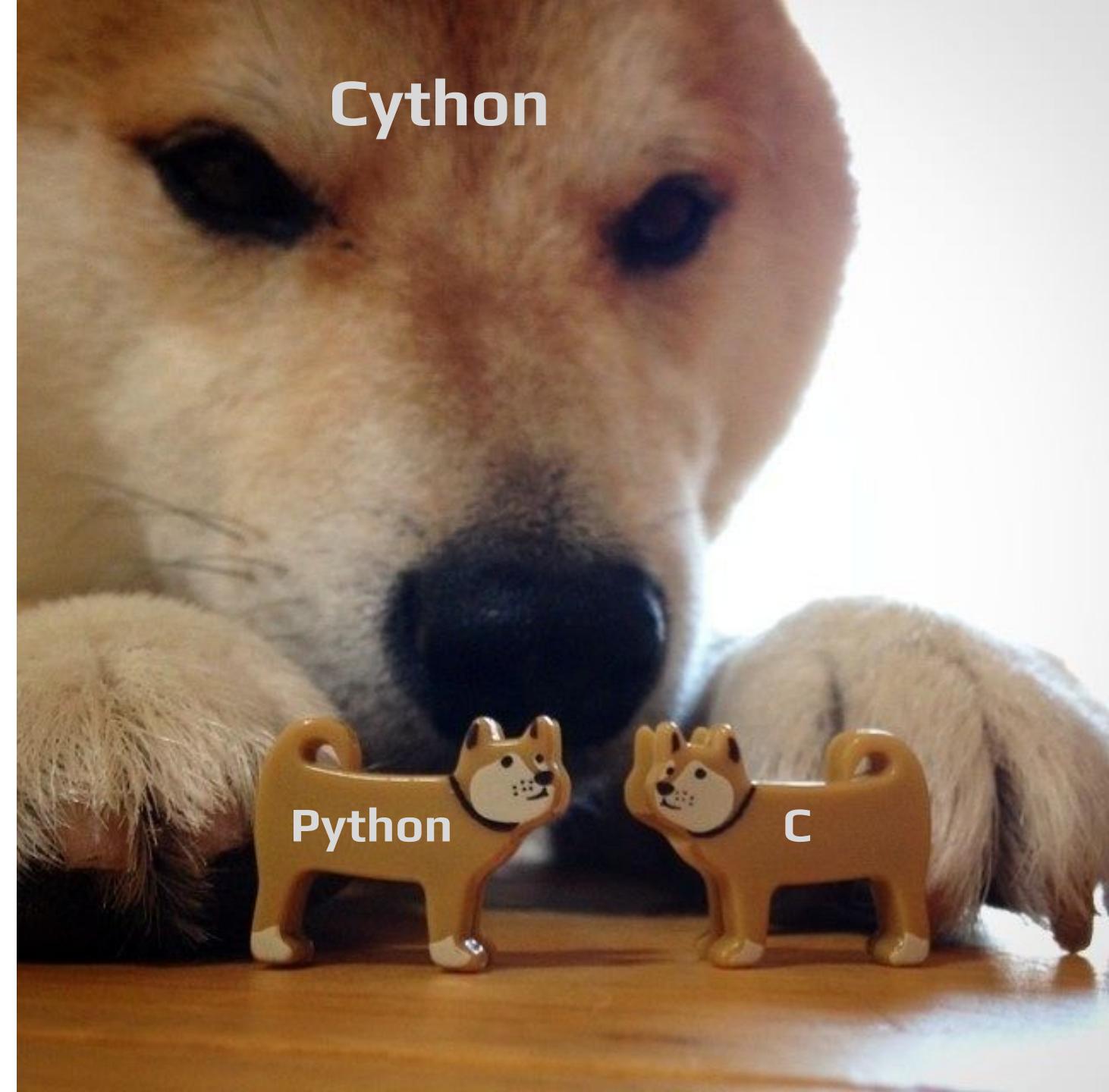
02 декабря 2025 года

Напоминание отметиться на портале

и оставить отзыв
после лекции



Мем недели



Содержание занятия

- Квиз
- ctypes;
- cffi;
- C API;
- Cython

Квиз



Для чего это всё???

- Вам нужна скорость и вы знаете, что C в X раз быстрее Python;
- Вам нужна конкретная C-библиотека и вы не хотите писать “велосипед” на Python;
- Вам нужен низкоуровневый интерфейс управления ресурсами для работы с памятью и файлами;
- Просто потому что Вам так хочется

ctypes



ctypes (1)

- Работает с DLL (Dynamic link library);
- ctypes определяет типы данных, совместимых с языком С:
 - c_bool
 - c_char
 - c_int
 - c_char_p
 - c_void_p
- Чтобы подключить библиотеку нужно либо вызвать
 - `ctypes.cdll.LoadLibrary('<dll path>')`
 - `ctypes.CDLL('<dll path>')`

ctypes (2)

```
int sum(int *arr, int len)
{
    int res = 0;
    for (int i = 0; i < len; ++i)
    {
        res += arr[i];
    }
    return res;
}
```

```
$ gcc -fPIC -shared -o sumlib.so 1.c
```

ctypes (3)

```
import ctypes
from typing import List

lib1 = ctypes.CDLL('./lib1.so')
lib1.sum.argtypes = (ctypes.POINTER(ctypes.c_int), ctypes.c_int)
```

ctypes (3)

```
def sum(arr: List[int]) -> int:  
    arr_len = len(arr)  
    arr_type = ctypes.c_int * arr_len  
    result = lib1.sum(arr_type(*arr), ctypes.c_int(arr_len))  
    return int(result)
```

CFFI

C Foreign Function Interface



FFI (1)

```
# Установка  
pip install cffi
```

FFI (C Foreign Function Interface) генерирует поверх нашей библиотеки свою обвязку и компилирует её в библиотеку с которой мы и будем работать.

FFI (2)

```
from cffi import FFI

ffi = FFI()
lib = ffi.dlopen('../ctypes/lib1.so')

ffi.cdef('''int sum(int* arr, int len);'''')
arr = [1, 2, 3, 4]
c_arr = ffi.new('int[]', arr)

s = lib.sum(c_arr, len(arr))
print(s)
```

CFFI (3)

```
#include <stdlib.h>

struct Point {
    int x;
    int y;
};

int area(struct Point *p1, struct Point *p2) {
    return abs((p2->y - p1->y) * (p1->x - p2->x));
}

$ gcc -fPIC -shared -o lib2.so 2.c
```

FFI (4)

```
from cffi import FFI

ffi = FFI()
lib = ffi.dlopen('./lib2.so')

ffi.cdef('''
struct Point {
    int x;
    int y;
};
int area(struct Point *p1, struct Point *p2);
''')
```

CFFI (5)

```
p1 = ffi.new('struct Point*')
p2 = ffi.new('struct Point*')
```

```
p1.x = 0
p1.y = 0
```

```
p2.x = 10
p2.y = 10
```

```
s = lib.area(p1, p2)
print(s)
```

Плюсы и минусы CFFI

- + простой синтаксис при использовании в Python;
- + не нужно перекомпилировать исходную библиотеку.
- не удобная сборка, нужно прописывать пути до всех заголовочных файлов и библиотек;
- создается ещё одна динамическая библиотека, которая использует исходную.

CAPI



С API | Обзор

1. Подключаем Python.h

```
#include <Python.h>
```

2. Все видимые пользователю имена имеют один из префиксов Py или

```
_Py;
```

C API | PyObject

```
typedef struct _object {  
    _PyObject_HEAD_EXTRA  
    Py_ssize_t ob_refcnt;  
    struct _typeobject *ob_type;  
} PyObject;
```

- `ob_refcnt`: он подсчитывает количество ссылок. Когда значение равно 0, сборщик мусора может очистить объект.
- `ob_type`: указывает на объект класса текущего объекта экземпляра
- `_PyObject_HEAD_EXTRA`: макрос. Если определен параметр `Py_TRACE_REFS`, этот макрос будет предварительно обработан в виде двух указателей как реализация двусвязного списка, который отслеживает все объекты кучи.

C API | Полезные функции

1. PyArg_ParseTuple
2. PyDict_New
3. PyDict_SetItem
4. Py_BuildValue

C API | Пример

```
static PyObject* spam_system(PyObject *self, PyObject *args)
{
    const char *command;
    int sts;

    if (!PyArg_ParseTuple(args, "s", &command))
        return NULL;
    sts = system(command);
    if (sts < 0) {
        PyErr_SetString(SpamError, "System command failed");
        return NULL;
    }
    return PyLong_FromLong(sts);
}
```

Cython



Cython

```
# Установка  
pip install cython
```

При работе с функциями нам доступны следующие типы:

- `def` – обычная Python-функция, вызывается только из Python.
- `cdef` – Cython-функция, которую нельзя вызвать из обычного Python-кода. Такие функции можно вызывать только в пределах Cython-кода.
- `cpdef` – Функция, доступ к которой можно получить и из C, и из Python.

Cython

```
# setup.py
from setuptools import setup, Extension
from Cython.Build import cythonize

setup(
    ext_modules= cythonize(['cutils.pyx'])
)

# Выполним компиляцию
$ python setup.py build_ext --inplace
```

Домашнее задание



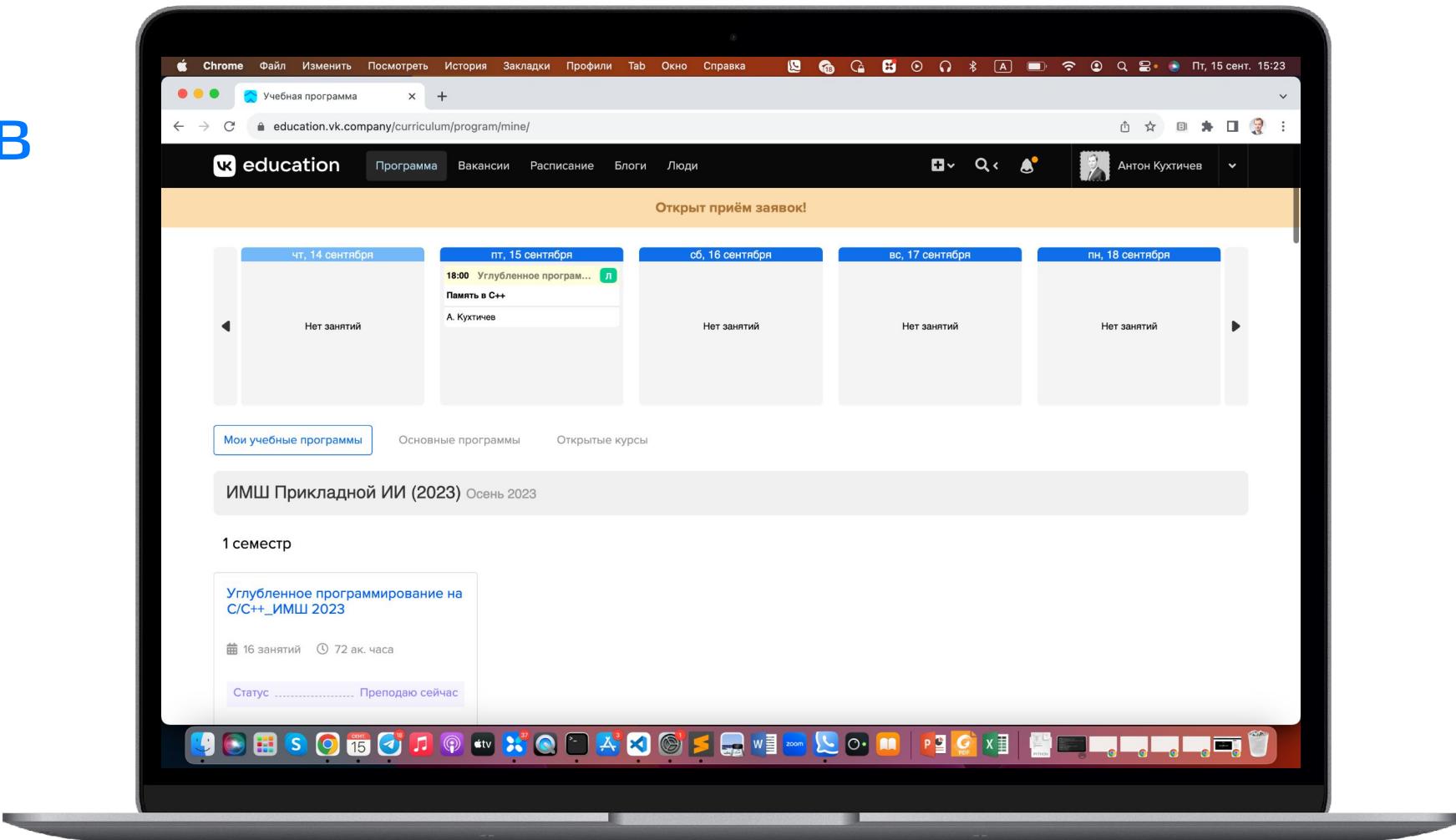
Домашнее задание

- Реализовать свой модуль cJSON для сериализации/десериализации json
- Значения – строк или числа int64_t

```
import cJSON  
  
obj = {"hello": "world", "key1": 100500}  
  
s = cJSON.dumps(obj)  
  
assert obj == cJSON.loads(s)
```

Напоминание оставить отзыв

Это правда важно





Спасибо
за внимание!