Лекция 12
Параллельное программирование
Подпроцессы
Расширения на C

10 мая 2017 г.

Введение

Последовательное выполнение

```
print 'hello, world!'
x = 1
y = 2
print x + y
```

Производительность

- Производительность $\sim$ частота процессора

Производительность

- Производительность $\sim$ частота процессора
- Ограничение роста частоты
- Есть возможность делать много процессоров

Блокирующие задачи

Ожидание действий

- Исполнение приостанавливается.
- Можно было бы что-нибудь делать.

Блокирующие задачи

Ожидание действий

- Исполнение приостанавливается.
- Можно было бы что-нибудь делать.

Примеры:

- Пользовательские интерфейсы (ожидание ввода).
- Работа с сетью.

Параллельное выполнение

Преимущества

- Ускорение $\approx$ число процессоров

Параллельное выполнение

Преимущества

- Ускорение $\approx$ число процессоров

Недостатки

- Нет «порядка» исполнения

```
text = 'hello, world!'
```

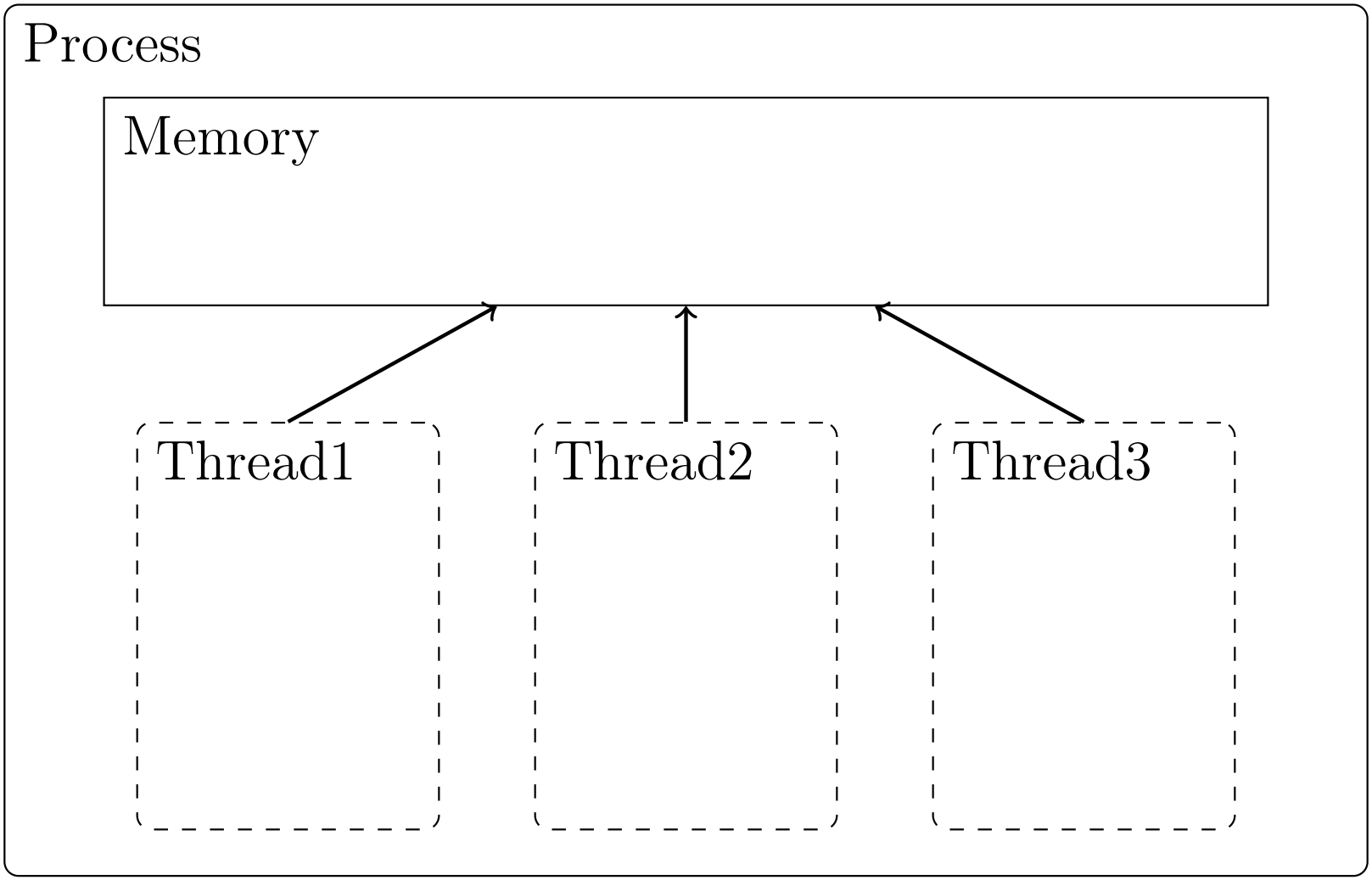
```
for s in text:
    do_smth()
    sys.stdout.write(s)
```

```
for s in text:
    do_smth()
    sys.stdout.write(s)
```

```
hheellllol,o w,or ld!world!
```

Многопоточные программы

Несколько потоков



Подготовка

```
def test(part_index, inputs, outputs):
    output = inputs[part_index] ** 2
    outputs[part_index] = output


SIZE = 5
inputs = range(SIZE)
outputs = [None for i in inputs]
```

Модуль threading

```
from threading import Thread
```

Модуль threading

```
from threading import Thread

threads = []
for i in range(SIZE):
    t = Thread(target=test,
               args=(i, inputs, outputs))
    threads.append(t)
```

Модуль threading

```
from threading import Thread

threads = []
for i in range(SIZE):
    t = Thread(target=test,
               args=(i, inputs, outputs))
    threads.append(t)

for t in threads:
    t.start()
```

Модуль threading

```
from threading import Thread

threads = []
for i in range(SIZE):
    t = Thread(target=test,
               args=(i, inputs, outputs))
    threads.append(t)

for t in threads:
    t.start()

for t in threads:
    t.join()
```

Модуль threading

```
from threading import Thread

threads = []
for i in range(SIZE):
    t = Thread(target=test,
               args=(i, inputs, outputs))
    threads.append(t)

for t in threads:
    t.start()

for t in threads:
    t.join()

print outputs
```

```
[0, 1, 4, 9, 16]
```

Наследование от Thread

```
class TCalc(Thread):
    def __init__(self):
        super(TCalc, self).__init__()
        self.status = 'starting...'
    def run(self):
        time.sleep(1)
        self.status = 'done'
```

Наследование от Thread

```
class TCalc(Thread):
    def __init__(self):
        super(TCalc, self).__init__()
        self.status = 'starting...'
    def run(self):
        time.sleep(1)
        self.status = 'done'

t = TCalc()
t.start()
print t.status
t.join()
print t.status
```

```
starting...
done
```

Проверка состояния

Метод `Thread.is_alive()`

```
t = TCalc()
t.start()
print t.is_alive()
t.join()
print t.is_alive()
```

```
True
False
```

Принудительная остановка потока

- Поток может содержать ресурсы, которые нужно освободить

Принудительная остановка потока

- Поток может содержать ресурсы, которые нужно освободить
- Нет специальной функции для остановки

Принудительная остановка потока

- Поток может содержать ресурсы, которые нужно освободить
- Нет специальной функции для остановки
- Можно самостоятельно делать такую возможность

```
class KillableThread(Thread):
    def __init__(self):
        super(TCalc, self).__init__()
        self.killing = False
    def run(self):
        for i in range(5):
            if self.killing:
                break
            time.sleep(0.2)
    def kill(self):
        self.killing = True
```

Эксперимент скорости

- CPU-intense вычисления
- 4 процессора

Эксперимент	**Время**
Обычный запуск	42с
2 потока	65с
4 потока	80с

Что происходит?

GIL

(CPython)
Global Interpreter Lock

GIL

(CPython)
Global Interpreter Lock

- Код ядра некорректен для нескольких потоков (зато быстро работает для одного!)

GIL

(CPython)
Global Interpreter Lock

- Код ядра некорректен для нескольких потоков (зато быстро работает для одного!)
- GIL не дает одновременно исполнять несколько потоков ядра.

GIL

(CPython)
Global Interpreter Lock

- Код ядра некорректен для нескольких потоков (зато быстро работает для одного!)
- GIL не дает одновременно исполнять несколько потоков ядра.
- Поток «захватывает» интерпертатор на короткое время.

GIL

(CPython)
Global Interpreter Lock

- Код ядра некорректен для нескольких потоков (зато быстро работает для одного!)
- GIL не дает одновременно исполнять несколько потоков ядра.
- Поток «захватывает» интерпертатор на короткое время.
- После освобождения это может сделать другой поток.

GIL

(CPython)
Global Interpreter Lock

- Код ядра некорректен для нескольких потоков (зато быстро работает для одного!)
- GIL не дает одновременно исполнять несколько потоков ядра.
- Поток «захватывает» интерпертатор на короткое время.
- После освобождения это может сделать другой поток.

Зачем нужен GIL?

- Скорость однопоточных программ $\longleftrightarrow$ GIL

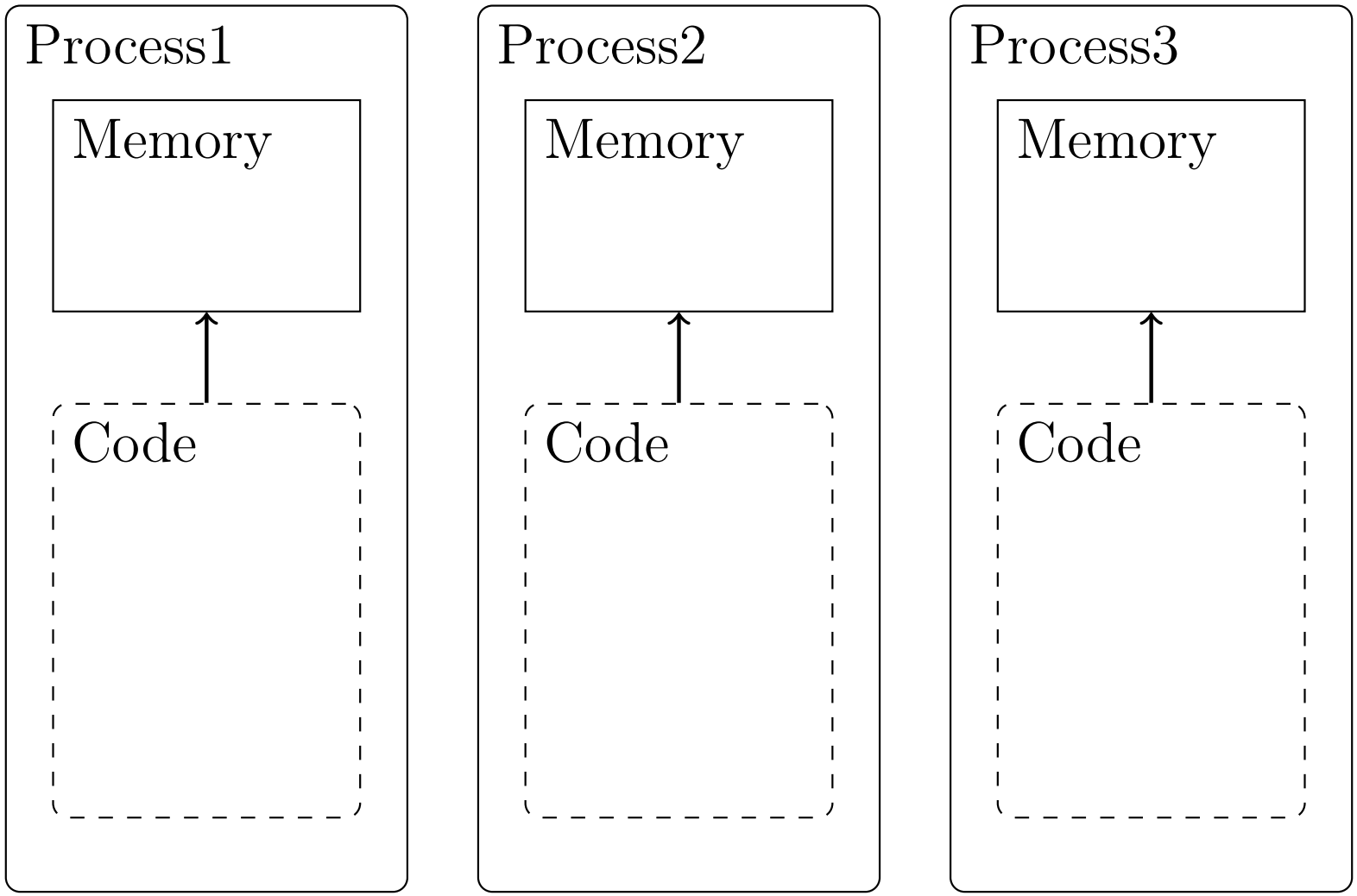
Использование потоков

Задачи, которые проводят мало времени «внутри ядра».

- Интерфейсы
- Работа с сетью
- Ввод/вывод
- Внешние библиотеки

Параллельные процессы

Несколько процессов



Модуль multiprocessing

```
from multiprocessing import Process

def f(name):
    print 'hello', name

p = Process(target=f, args=('world',))
p.start()
p.join()
```

```
hello world
```

Модуль multiprocessing

```
from multiprocessing import Process

def f(name):
    print 'hello', name

p = Process(target=f, args=('world',))
p.start()
p.join()
```

```
hello world
```

Обмен данными:

- mulitprocessing.Queue
- mulitprocessing.Pipe

multiprocessing.Pool

```
from multiprocessing import Pool

def calc(value):
    return value ** 2
```

multiprocessing.Pool

```
from multiprocessing import Pool

def calc(value):
    return value ** 2

p = Pool(processes=4)
```

multiprocessing.Pool

```
from multiprocessing import Pool

def calc(value):
    return value ** 2

p = Pool(processes=4)

results = p.map(calc, range(10))
print results
```

multiprocessing.Pool

```
from multiprocessing import Pool

def calc(value):
    return value ** 2

p = Pool(processes=4)

results = p.map(calc, range(10))
print results
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Ограничения multiprocessing

```
def calc_parallel(values):
    def calc(value):
        return value ** 2
    p = Pool(processes=2)
    return p.map(calc, values)
```

Ограничения multiprocessing

```
def calc_parallel(values):
    def calc(value):
        return value ** 2
    p = Pool(processes=2)
    return p.map(calc, values)

calc_parallel(range(10))
```

(Ошибка, скорее всего с зависанием)

Ограничения `multiprocessing`

```
def calc_parallel(values):
    def calc(value):
        return value ** 2
    p = Pool(processes=2)
    return p.map(calc, values)

calc_parallel(range(10))
```

(Ошибка, скорее всего с зависанием)

- Импортирует текущий файл
- Все параметры должны быть «picklable» (см. модуль `pickle`)

Внешние процессы

Простой способ

```
import os

print os.system('ls')
```

Простой способ

```
import os

print os.system('ls')
```

```
0
```

Модуль subprocess

```
import subprocess

subprocess.check_output(
    ['echo', 'hello, world'])
```

```
'hello, world\n'
```

Модуль subprocess

```
import subprocess

subprocess.check_output(
    ['echo', 'hello, world'])
```

```
'hello, world\n'
```

```
subprocess.check_output('exit 1', shell=True)
```

```
CalledProcessError: Command 'exit 1' returned
non-zero exit status 1
```

Конструктор Popen

```
p = subprocess.Popen(['echo', 'hi'])
```

Конструктор Popen

```
p = subprocess.Popen(['echo', 'hi'])
p.communicate()
```

```
(None, None)
```

Конструктор Popen

```
p = subprocess.Popen(['echo', 'hi'])
p.communicate()
```

```
(None, None)
```

```
p = subprocess.Popen(['echo', 'hi'],
                     stdout=subprocess.PIPE)
p.communicate()
```

```
('hi\n', None)
```

Конструктор Popen

```
p = subprocess.Popen(['echo', 'hi'])
p.communicate()
```

```
(None, None)
```

```
p = subprocess.Popen(['echo', 'hi'],
                     stdout=subprocess.PIPE)
p.communicate()
```

```
('hi\n', None)
```

```
p = subprocess.Popen(['cat'],
    stdout=subprocess.PIPE,
    stdin=subprocess.PIPE)
p.communicate('sample text')
```

Конструктор Popen

```
p = subprocess.Popen(['echo', 'hi'])
p.communicate()
```

```
(None, None)
```

```
p = subprocess.Popen(['echo', 'hi'],
                     stdout=subprocess.PIPE)
p.communicate()
```

```
('hi\n', None)
```

```
p = subprocess.Popen(['cat'],
    stdout=subprocess.PIPE,
    stdin=subprocess.PIPE)
p.communicate('sample text')
```

```
('sample text', None)
```

Эмуляция pipeline

```
p1 = Popen(['echo', 'qwerty'], stdout=PIPE)
p2 = Popen(['tr', 'q', 'z'], stdin=p1.stdout,
           stdout=PIPE)
```

Эмуляция pipeline

```
p1 = Popen(['echo', 'qwerty'], stdout=PIPE)
p2 = Popen(['tr', 'q', 'z'], stdin=p1.stdout,
           stdout=PIPE)

p1.stdout.close()
print p2.communicate()
```

```
('zwerty\n', None)
```

Расширения на C

Зачем писать расширения?

- Библиотеки, уже написанные на C/C++
- Эффективность

Функция инкремента

```
def increment(x):
    return x + 1
```

Инкремент на C++

examplemodule.cpp

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

Инкремент на C++

examplemodule.cpp

```
#include <Python.h>

static PyObject* example_increment(
    PyObject* self, PyObject* args)
{
```

Инкремент на C++

examplemodule.cpp

```
#include <Python.h>

static PyObject* example_increment(
    PyObject* self, PyObject* args)
{
    int value;
    if (!PyArg_ParseTuple(args, "i", &value)) {
        return NULL;
    }
```

Инкремент на C++

examplemodule.cpp

```
#include <Python.h>


static PyObject* example_increment(
    PyObject* self, PyObject* args)
{
    int value;
    if (!PyArg_ParseTuple(args, "i", &value)) {
        return NULL;
    }

    int result = value + 1;
    return Py_BuildValue("i", result);
}
```

Регистрация функции

```
static PyMethodDef ExampleMethods[] = {
    {"increment",
        example_increment,
        METH_VARARGS,
        "Increment an integer value by one."},
    {NULL, NULL, 0, NULL}        /* Sentinel */
};
```

Регистрация функции

```
static PyMethodDef ExampleMethods[] = {
    {"increment",
        example_increment,
        METH_VARARGS,
        "Increment an integer value by one."},
    {NULL, NULL, 0, NULL}        /* Sentinel */
};

PyMODINIT_FUNC initexample() {
    Py_InitModule("example", ExampleMethods);
}
```

Сборка модуля

setup.py

```
from distutils.core import setup, Extension

module = Extension('example',
                   ['examplemodule.cpp'])
setup(ext_modules=[module])
```

Сборка модуля

setup.py

```
from distutils.core import setup, Extension

module = Extension('example',
                   ['examplemodule.cpp'])
setup(ext_modules=[module])
```

Запуск сборки (в текущую директорию):

```
$ python setup.py build_ext -i
```

Пример работы

```
>>> from example import increment
>>> increment(1)
2
```

Пример работы

```
>>> from example import increment
>>> increment(1)
2
>>> increment('hi')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: an integer is required
>>> increment(1, 2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: function takes exactly 1 argument
(2 given)
```

Ошибки

```
static PyObject* example_divide(
    PyObject* self, PyObject* args)
{
    int numerator, denominator;
    if (!PyArg_ParseTuple(args, "ii",
        &numerator, &denominator))
    {
        return NULL;
    }

    int result = numerator / denominator;
    return Py_BuildValue("i", result);
}
```

Ошибки

```
>>> from example import divide
>>> divide(5, 2)
2
>>> divide(1, 0)
```

ошибка с падением интерпретатора

Обработка ошибок

```
static PyObject* example_divide(
    PyObject* self, PyObject* args)
{
    ...

    if (denominator == 0) {
        PyErr_SetString(PyExc_ZeroDivisionError,
                        "Cannot divide by zero.");
        return NULL;
    }

    ...
}
```

Обработка ошибок

```
>>> from example import divide
>>> divide(1, 0)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: Cannot divide by zero.
```

Работа со списками

```
static PyObject* example_sort(
    PyObject* self, PyObject* args)
{
    PyObject* input_list;
    if (!PyArg_ParseTuple(args, "O",
                          &input_list))
    {
        return NULL;
    }
```

Работа со списками

```
static PyObject* example_sort(
    PyObject* self, PyObject* args)
{
    PyObject* input_list;
    if (!PyArg_ParseTuple(args, "O",
                          &input_list))
    {
        return NULL;
    }
    if (!PyList_Check(input_list)) {
        PyErr_SetString(PyExc_TypeError,
                        "Expected a list.");
        return NULL;
    }
    ...
}
```

Работа со списками

```
Py_ssize_t size = PyList_Size(input_list);
std::vector<double> values;
```

Работа со списками

```
Py_ssize_t size = PyList_Size(input_list);
std::vector<double> values;

for (Py_ssize_t i = 0; i < size; ++i) {
    PyObject* val = PyList_GetItem(
        input_list, i);
```

Работа со списками

```
Py_ssize_t size = PyList_Size(input_list);
std::vector<double> values;

for (Py_ssize_t i = 0; i < size; ++i) {
    PyObject* val = PyList_GetItem(
        input_list, i);

    if (!PyFloat_Check(val)) {
        PyErr_SetString(
            PyExc_TypeError,
            "The list contains non-double.");
        return NULL;
    }

    values.push_back(PyFloat_AsDouble(val));
}
```

Работа со списками

```
static PyObject* example_sort(
    PyObject* self, PyObject* args)
{
    ...

    std::sort(values.begin(), values.end());
```

Работа со списками

```
static PyObject* example_sort(
    PyObject* self, PyObject* args)
{
    ...

    std::sort(values.begin(), values.end());

    PyObject* result_list = PyList_New(size);
    for (Py_ssize_t i = 0; i < size; ++i) {
        PyObject* val = PyFloat_FromDouble(
            values[i]);
        PyList_SetItem(result_list, i, val);
    }

    return result_list;
}
```

Возможные проблемы

```
void bug(PyObject *list) {
    PyObject *item = PyList_GetItem(list, 0);
    PyList_SetItem(list, 1, PyInt_FromLong(0L));
    PyObject_Print(item, stdout, 0); /* BUG! */
}
```

Владение ссылками на объекты

Владение

- Нужно удалить ссылку по окончании работы
- Объект не удалится, пока есть ссылка

Владение ссылками на объекты

Владение

- Нужно удалить ссылку по окончании работы
- Объект не удалится, пока есть ссылка

Одалживание

- Не нужно удалять
- Можно пользоваться только пока “одалживающая” ссылка жива

Удаление ссылок

```
PyList_SetItem(list, 1, PyInt_FromLong(0L));
```

Захват владения

```
void no_bug(PyObject *list) {
    PyObject *item = PyList_GetItem(list, 0);
    Py_INCREF(item);

    PyList_SetItem(list, 1, PyInt_FromLong(0L));
    PyObject_Print(item, stdout, 0);
    Py_DECREF(item);
}
```

Расширения на C/C++: другие способы

- Библиотека `ctypes` (стандартная библиотека)
- SWIG [ссылка]
- Boost.Python [ссылка]