Cython

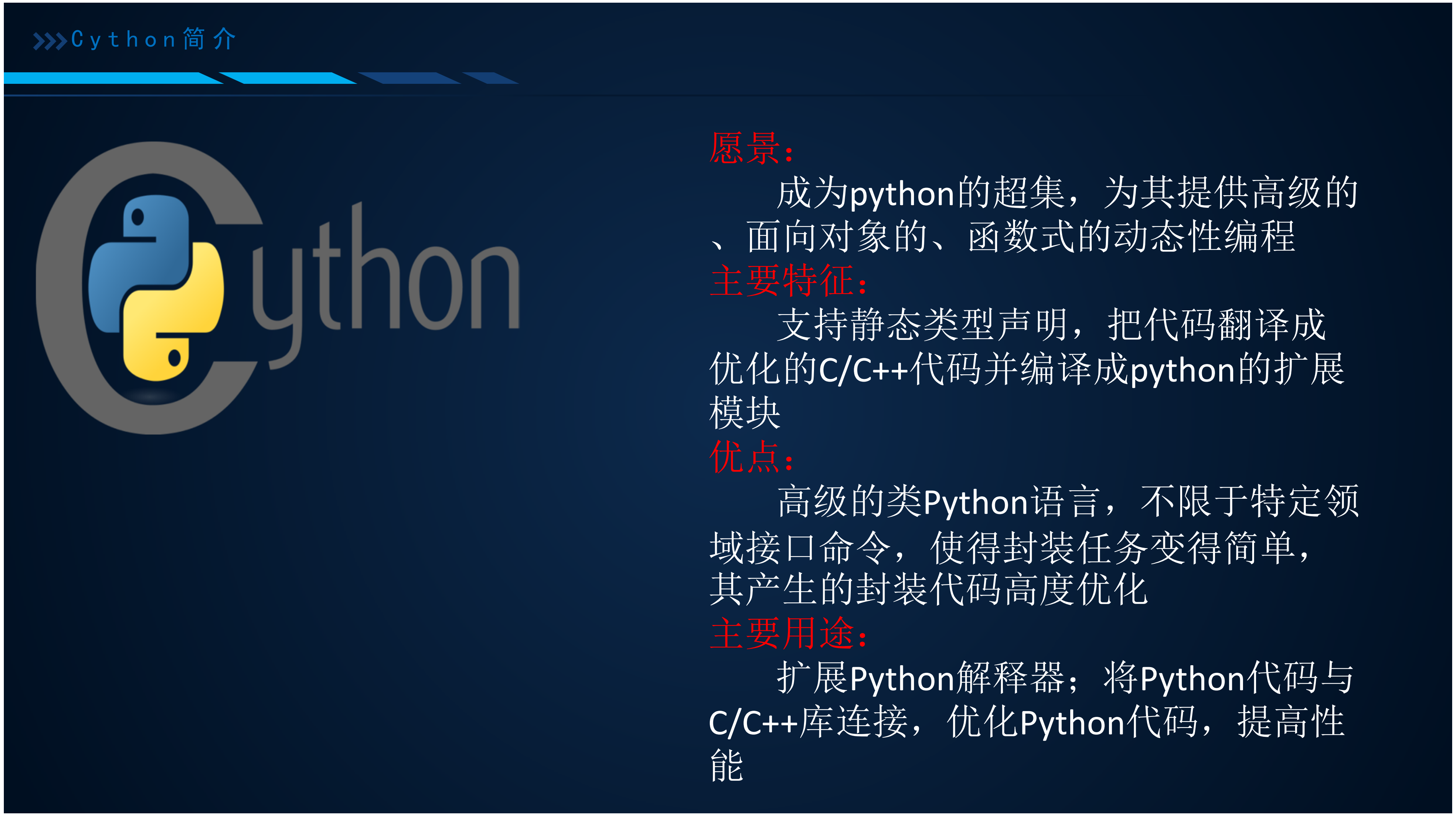
Python + C/C++

——littley

目录Catalog

01.

Cython简介



Cython简介

愿景：

成为python的超集，为其提供高级的、面向对象的、函数式的动态性编程

主要特征：

支持静态类型声明，把代码翻译成优化的C/C++代码并编译成python的扩展模块

优点：

高级的类Python语言，不限于特定领域接口命令，使得封装任务变得简单，其产生的封装代码高度优化

主要用途：

扩展Python解释器；将Python代码与C/C++库连接，优化Python代码，提高性能

>>>Cython简介

Cython

Sage

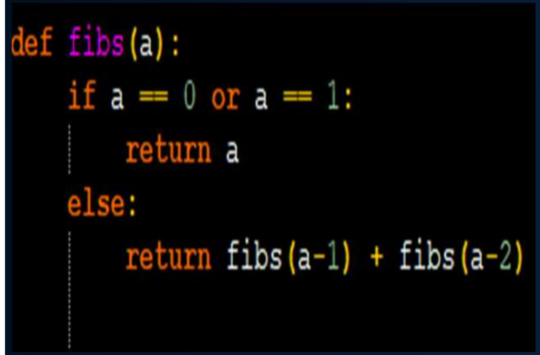
Scipy

mpi4py

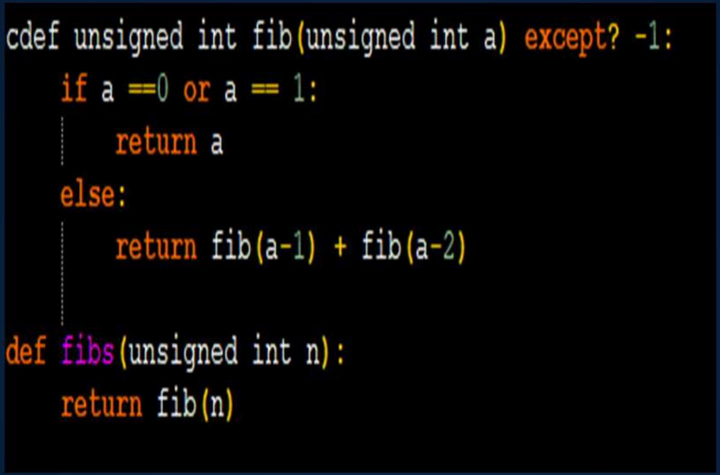
spaCy

lxml

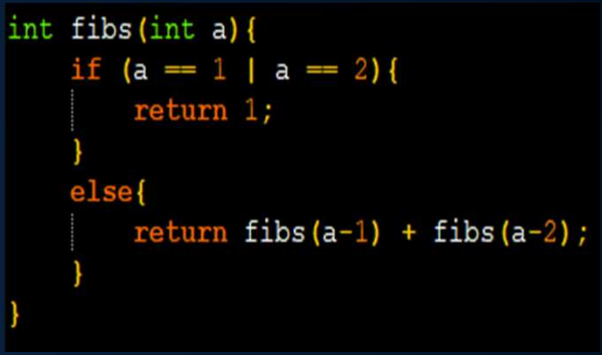
>>>Cython简介



```python
def fibs(a):
    if a == 0 or a == 1:
        return a
    else:
        return fibs(a-1) + fibs(a-2)
```



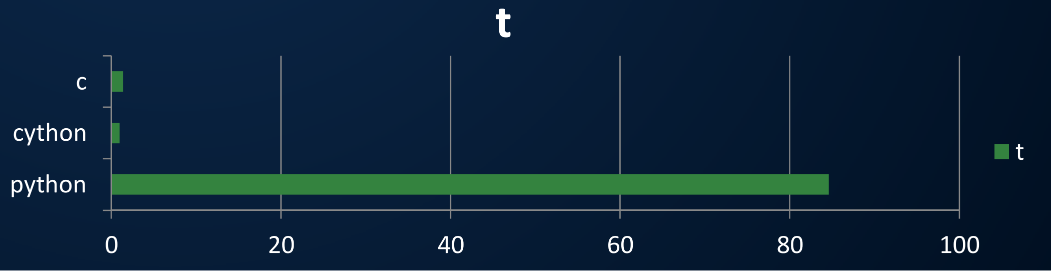
```cython
cdef unsigned int fib(unsigned int a) except? -1:
    if a ==0 or a == 1:
        return a
    else:
        return fib(a-1) + fib(a-2)

def fibs(unsigned int n):
    return fib(n)
```



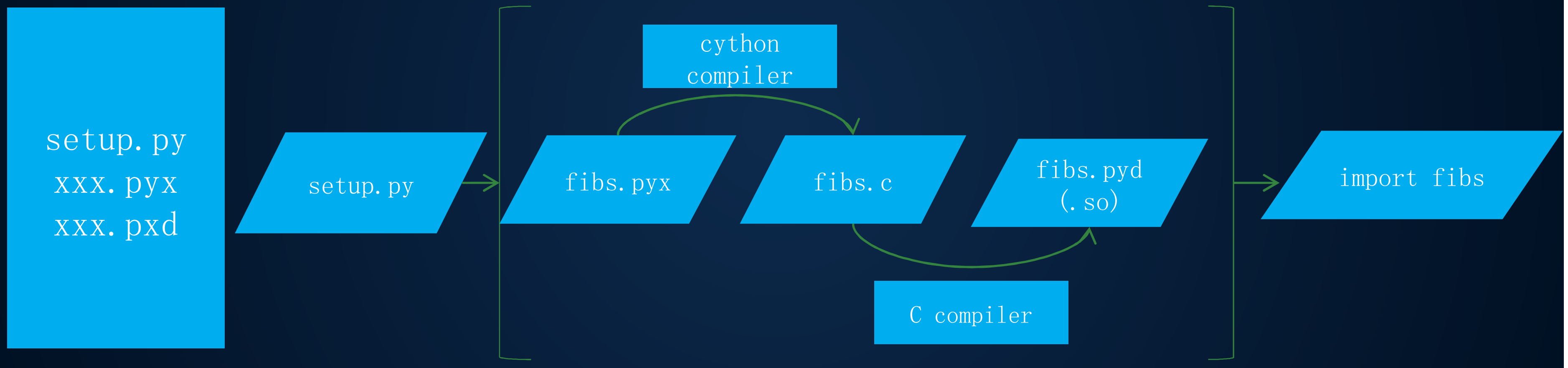
```c
int fibs(int a){
    if (a == 1 | a == 2){
        return 1;
    }
    else{
        return fibs(a-1) + fibs(a-2);
    }
}
```

Fibs(40)

python	84.595
cython	0.97998
c	1.373

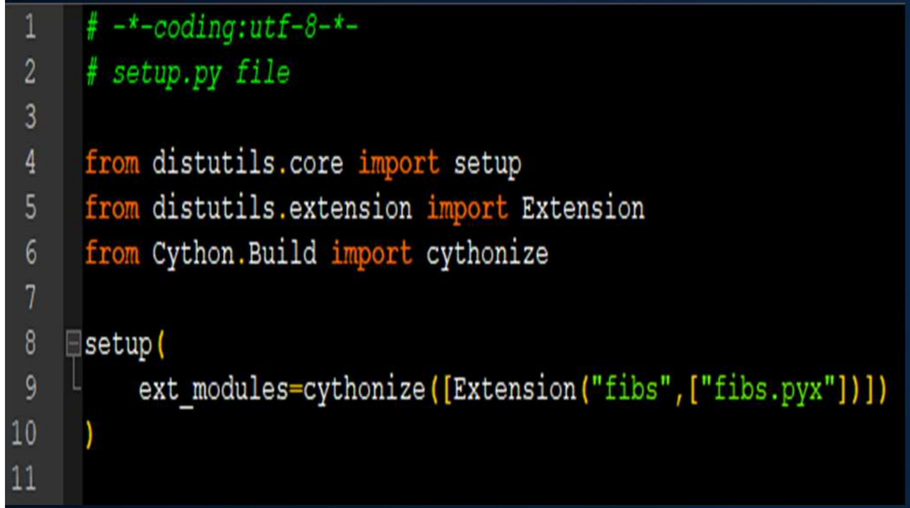


Cython工作流程

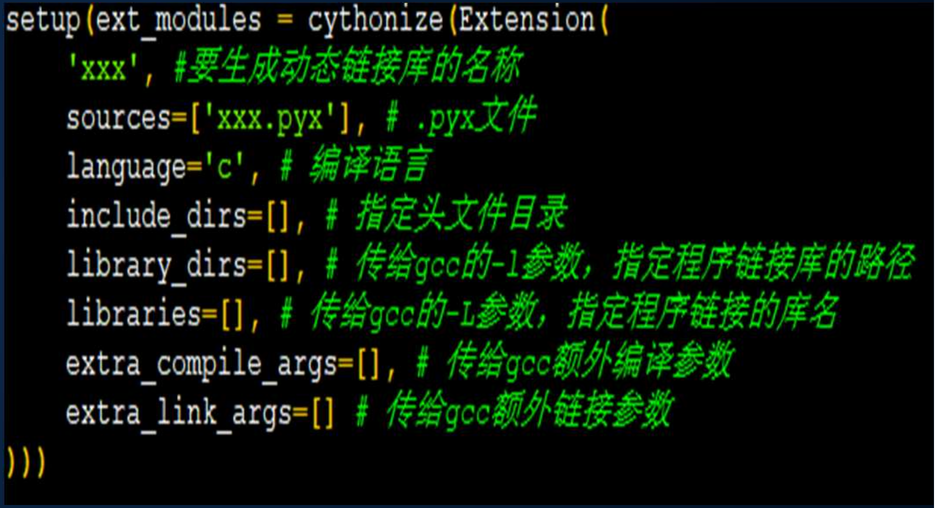

setup.py
xxx.pyx
xxx.pxd
setup.py
cython compiler
fibs.pyx
fibs.c
fibs.pyd (.so)
C compiler
import fibs

Cython简介

setup.py



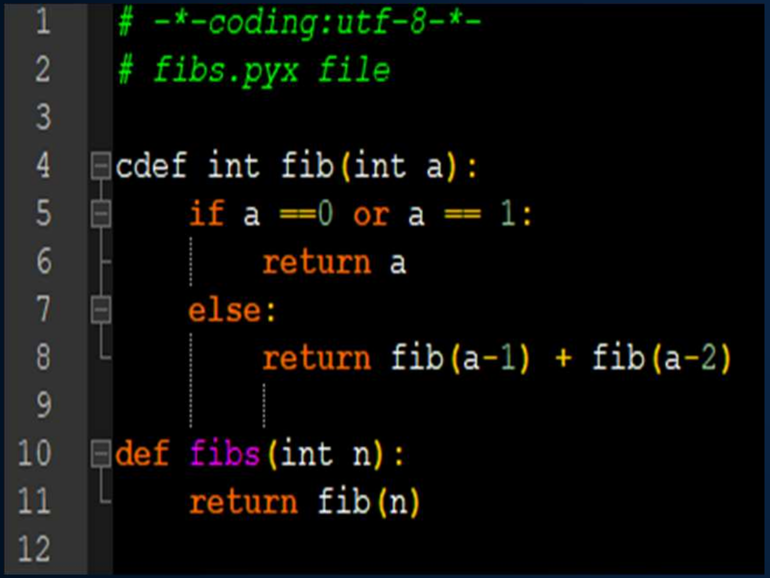
```python
# -*-coding:utf-8-*-
# setup.py file

from distutils.core import setup
from distutils.extension import Extension
from Cython.Build import cythonize

setup(
    ext_modules=cythonize([Extension("fibs",["fibs.pyx"])])
)
```



```python
setup(ext_modules = cythonize(Extension(
    'xxx', #要生成动态链接库的名称
    sources=['xxx.pyx'], # .pyx文件
    language='c', # 编译语言
    include_dirs=[], # 指定头文件目录
    library_dirs=[], # 传给gcc的-l参数，指定程序链接库的路径
    libraries=[], # 传给gcc的-L参数，指定程序链接的库名
    extra_compile_args=[], # 传给gcc额外编译参数
    extra_link_args=[] # 传给gcc额外链接参数
)))
```

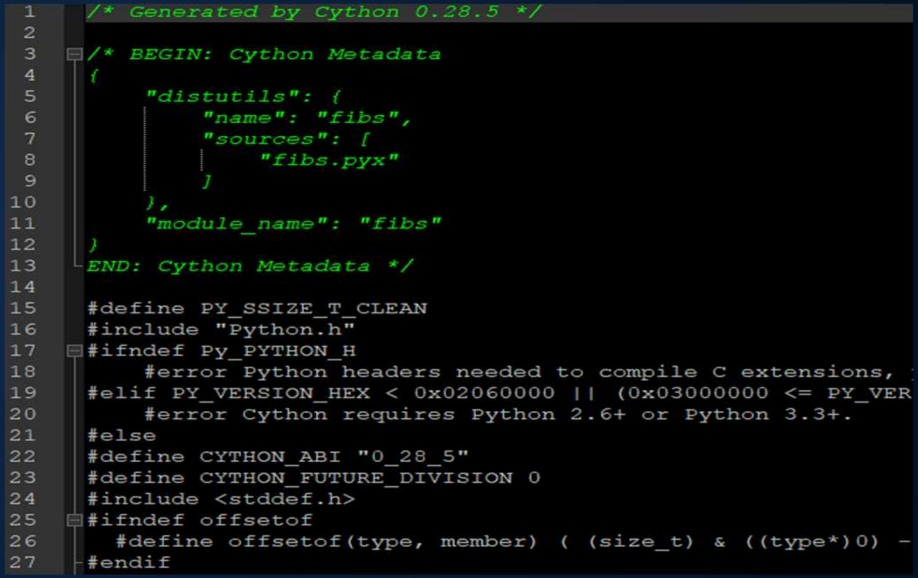
Cython简介

fibs.pyx



```python
# -*-coding:utf-8-*-
# fibs.pyx file

cdef int fib(int a):
    if a ==0 or a == 1:
        return a
    else:
        return fib(a-1) + fib(a-2)


def fibs(int n):
    return fib(n)

```

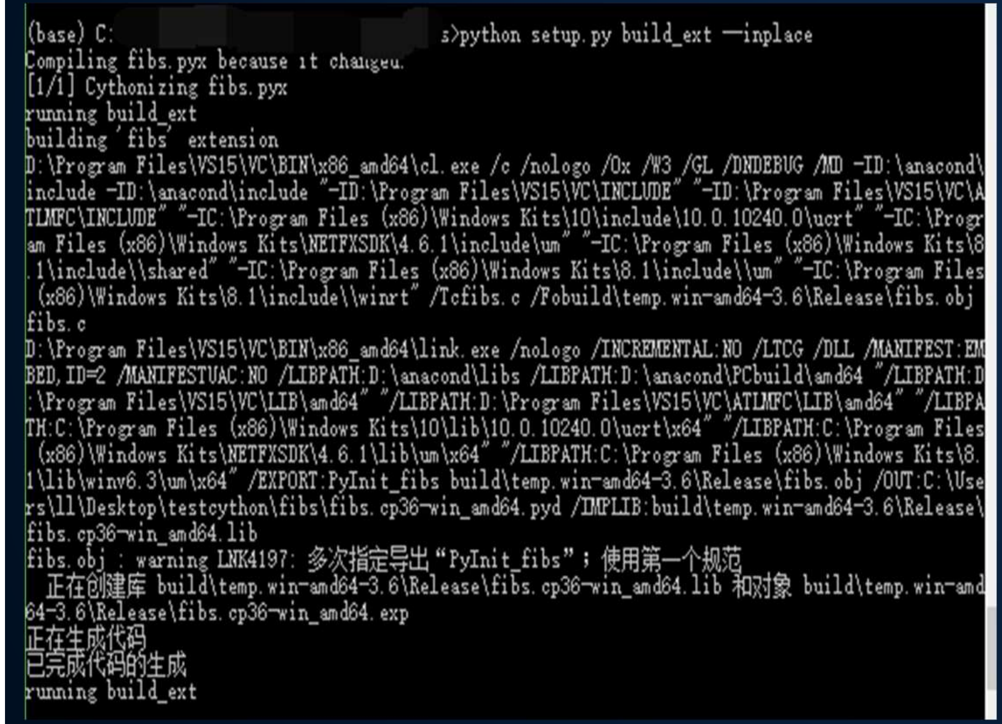
fibs.c



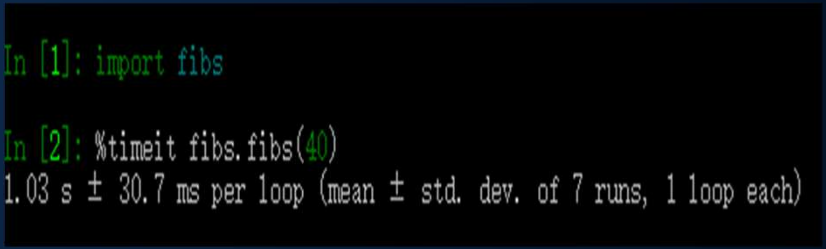
```c
/* Generated by Cython 0.28.5 */

/* BEGIN: Cython Metadata
{
    "distutils": {
        "name": "fibs",
        "sources": [
            "fibs.pyx"
        ]
    },
    "module_name": "fibs"
}
END: Cython Metadata */

#define PY_SSIZE_T_CLEAN
#include "Python.h"
#ifndef Py_PYTHON_H
    #error Python headers needed to compile C extensions,
#elif PY_VERSION_HEX < 0x02060000 || (0x03000000 <= PY_VER
    #error Cython requires Python 2.6+ or Python 3.3+.
#else
#define CYTHON_ABI "0_28_5"
#define CYTHON_FUTURE_DIVISION 0
#include <stddef.h>
#ifndef offsetof
  #define offsetof(type, member) ( (size_t) & ((type*)0) -
#endif
```

›››Cython简介

python setup.py build_ext --inplace



```
(base) C:[redacted]s>python setup.py build_ext —inplace
Compiling fibs.pyx because it changed.
[1/1] Cythonizing fibs.pyx
running build_ext
building 'fibs' extension
D:\Program Files\VS15\VC\BIN\x86_amd64\cl.exe /c /nologo /Ox /W3 /GL /DNDEBUG /MD -ID:\anacond\
include -ID:\anacond\include "-ID:\Program Files\VS15\VC\INCLUDE" "-ID:\Program Files\VS15\VC\A
TLMFC\INCLUDE" "-IC:\Program Files (x86)\Windows Kits\10\include\10.0.10240.0\ucrt" "-IC:\Progr
am Files (x86)\Windows Kits\NETFXSDK\4.6.1\include\um" "-IC:\Program Files (x86)\Windows Kits\8
.1\include\\shared" "-IC:\Program Files (x86)\Windows Kits\8.1\include\\um" "-IC:\Program Files
 (x86)\Windows Kits\8.1\include\\winrt" /Tcfibs.c /Fobuild\temp.win-amd64-3.6\Release\fibs.obj
fibs.c
D:\Program Files\VS15\VC\BIN\x86_amd64\link.exe /nologo /INCREMENTAL:NO /LTCG /DLL /MANIFEST:EM
BED,ID=2 /MANIFESTUAC:NO /LIBPATH:D:\anacond\libs /LIBPATH:D:\anacond\PCbuild\amd64 "/LIBPATH:D
:\Program Files\VS15\VC\LIB\amd64" "/LIBPATH:D:\Program Files\VS15\VC\ATLMFC\LIB\amd64" "/LIBPA
TH:C:\Program Files (x86)\Windows Kits\10\lib\10.0.10240.0\ucrt\x64" "/LIBPATH:C:\Program Files
 (x86)\Windows Kits\NETFXSDK\4.6.1\lib\um\x64" "/LIBPATH:C:\Program Files (x86)\Windows Kits\8.
1\lib\winv6.3\um\x64" /EXPORT:PyInit_fibs build\temp.win-amd64-3.6\Release\fibs.obj /OUT:C:\Use
rs\ll\Desktop\testcython\fibs\fibs.cp36-win_amd64.pyd /IMPLIB:build\temp.win-amd64-3.6\Release\
fibs.cp36-win_amd64.lib
fibs.obj : warning LNK4197: 多次指定导出"PyInit_fibs"；使用第一个规范
  正在创建库 build\temp.win-amd64-3.6\Release\fibs.cp36-win_amd64.lib 和对象 build\temp.win-amd
64-3.6\Release\fibs.cp36-win_amd64.exp
正在生成代码
已完成代码的生成
running build_ext
```

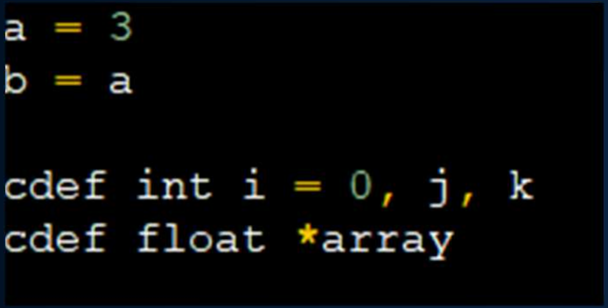


```
In [1]: import fibs

In [2]: %timeit fibs.fibs(40)
1.03 s ± 30.7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

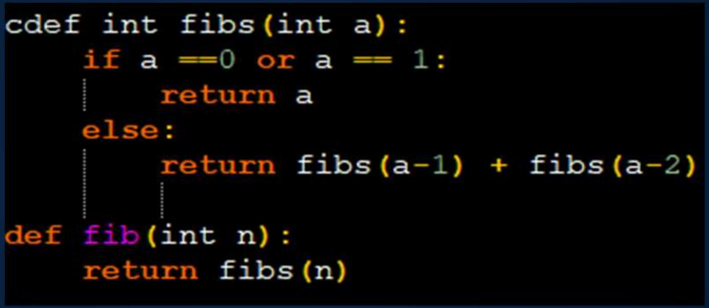
02.

Cython基础

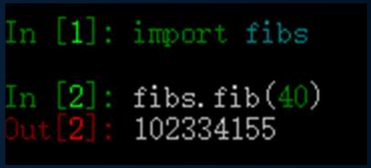
静态类型声明：cdef和cpdef



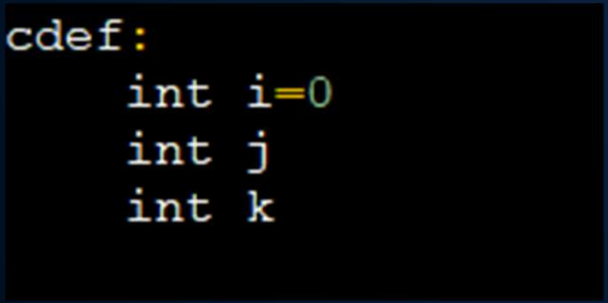
```
a = 3
b = a

cdef int i = 0, j, k
cdef float *array
```



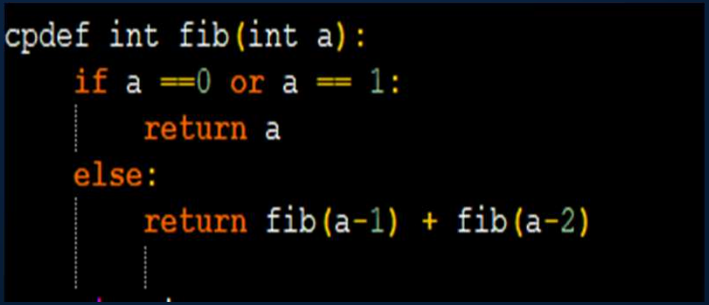
```
cdef int fibs(int a):
    if a ==0 or a == 1:
        return a
    else:
        return fibs(a-1) + fibs(a-2)

def fib(int n):
    return fibs(n)
```



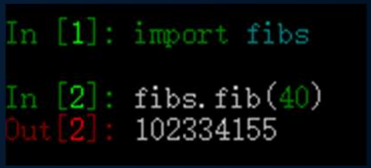
```
In [1]: import fibs

In [2]: fibs.fib(40)
Out[2]: 102334155
```



```
cdef:
    int i=0
    int j
    int k
```



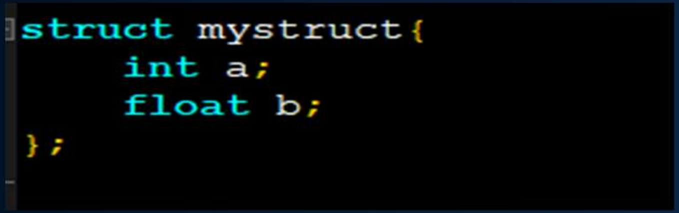
```
cpdef int fib(int a):
    if a ==0 or a == 1:
        return a
    else:
        return fib(a-1) + fib(a-2)
```



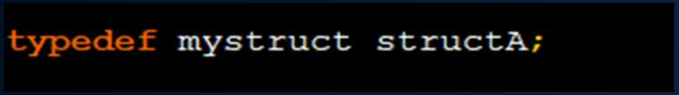
```
In [1]: import fibs

In [2]: fibs.fib(40)
Out[2]: 102334155
```

struct

C

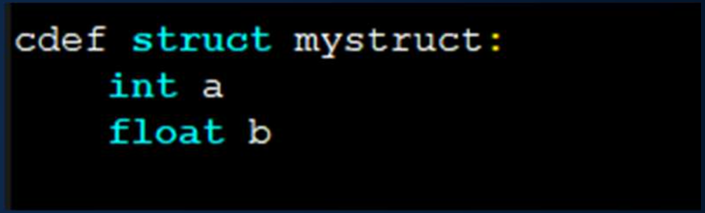


```
struct mystruct{
     int a;
     float b;
};
```

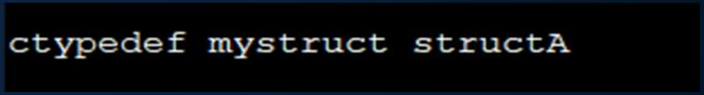


```
typedef mystruct structA;
```

Cython



```
cdef struct mystruct:
    int a
    float b
```



```
ctypedef mystruct structA
```

Struct初始化

1. 作为参数传进

```
cdef mystruct m = mystruct(20,11.1)
cdef mystruct m = mystruct(a=20, b=11.1)
```

2. 作为元素传进

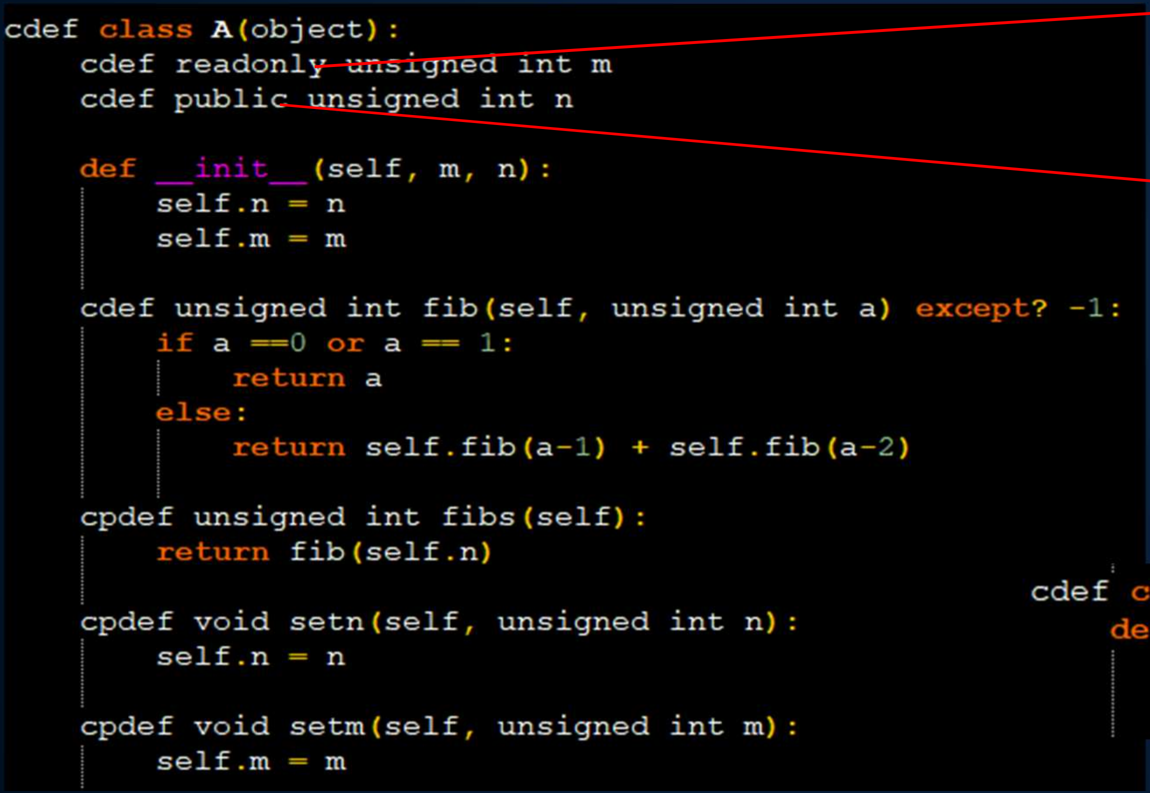
```
cdef mystruct zz
zz.a = 20
zz.b = 11.1
```

元素赋值与取值使用”.”

3. 作为字典传进

```
cdef mystruct zz = {'a':20,'b':11.1}
```

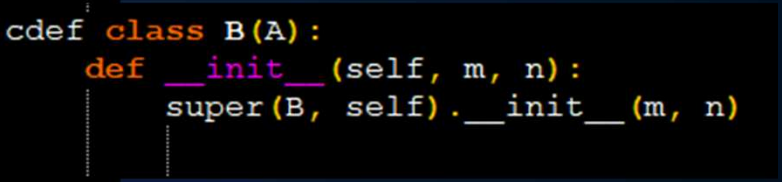
定义类



```
cdef class A(object):
    cdef readonly unsigned int m
    cdef public unsigned int n

    def __init__(self, m, n):
        self.n = n
        self.m = m

    cdef unsigned int fib(self, unsigned int a) except? -1:
        if a ==0 or a == 1:
            return a
        else:
            return self.fib(a-1) + self.fib(a-2)

    cpdef unsigned int fibs(self):
        return fib(self.n)

    cpdef void setn(self, unsigned int n):
        self.n = n

    cpdef void setm(self, unsigned int m):
        self.m = m
```

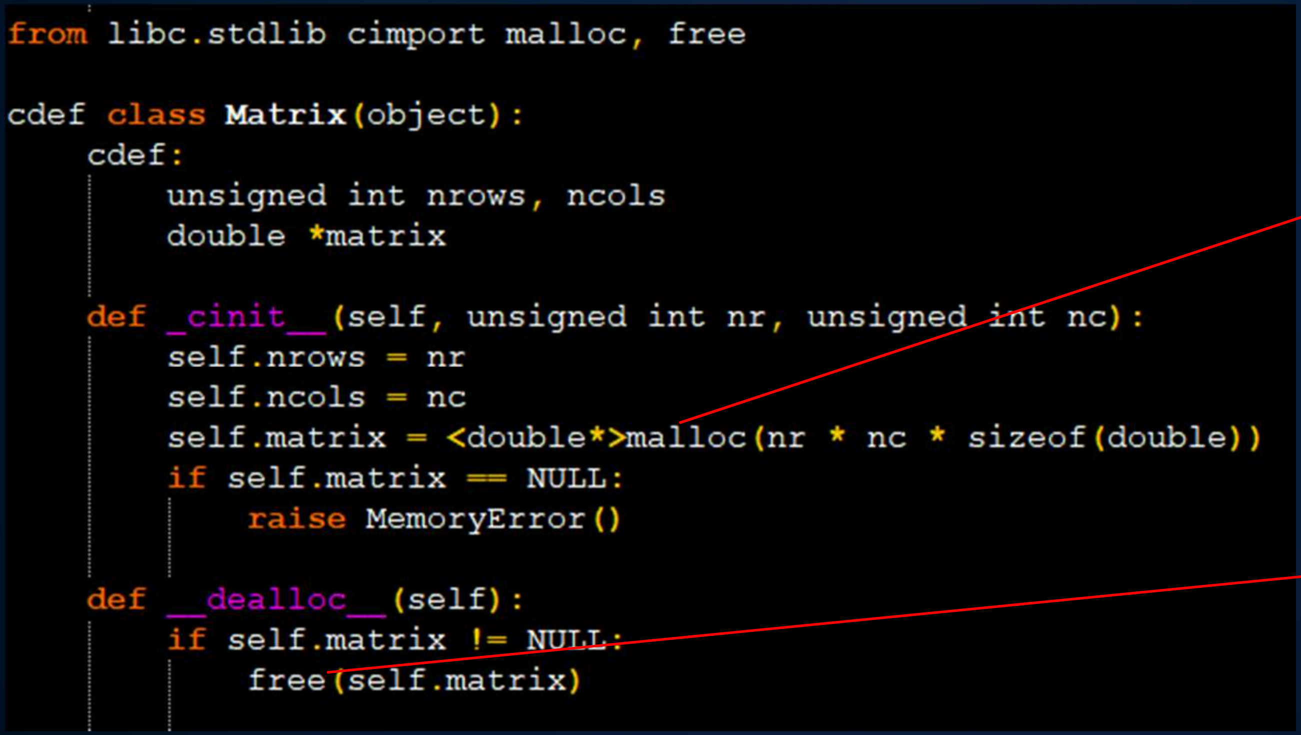
只读数据，外部不可更改

外部可读可写



```
cdef class B(A):
    def __init__(self, m, n):
        super(B, self).__init__(m, n)
```

__cinit__和__dealloc__



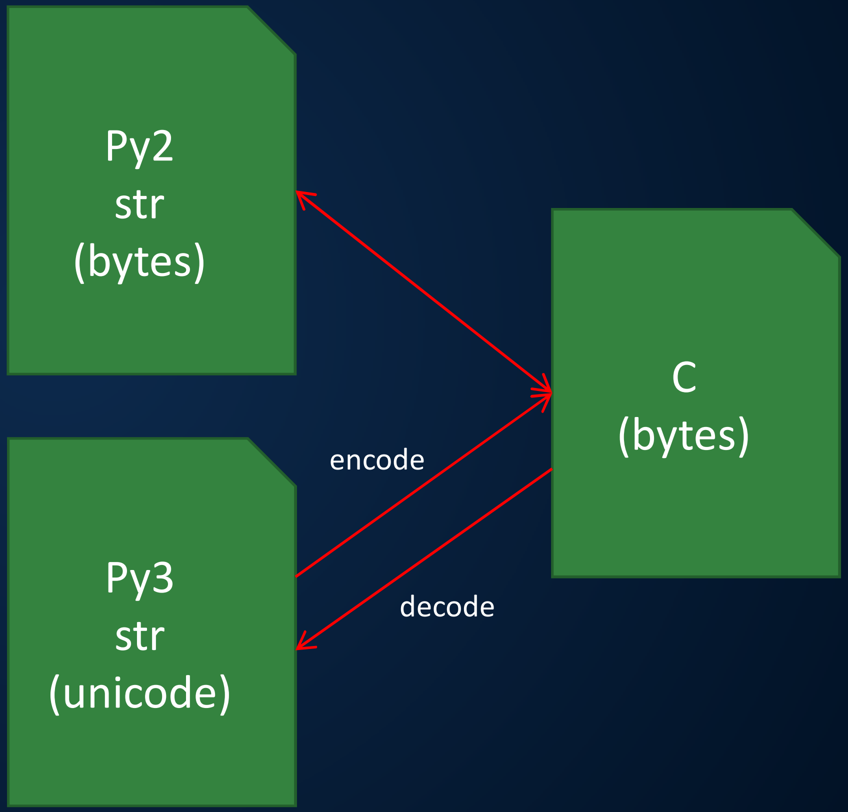
```
from libc.stdlib cimport malloc, free

cdef class Matrix(object):
    cdef:
        unsigned int nrows, ncols
        double *matrix

    def __cinit__(self, unsigned int nr, unsigned int nc):
        self.nrows = nr
        self.ncols = nc
        self.matrix = <double*>malloc(nr * nc * sizeof(double))
        if self.matrix == NULL:
            raise MemoryError()

    def __dealloc__(self):
        if self.matrix != NULL:
            free(self.matrix)
```

动态分配内存

垃圾回收

字符串处理

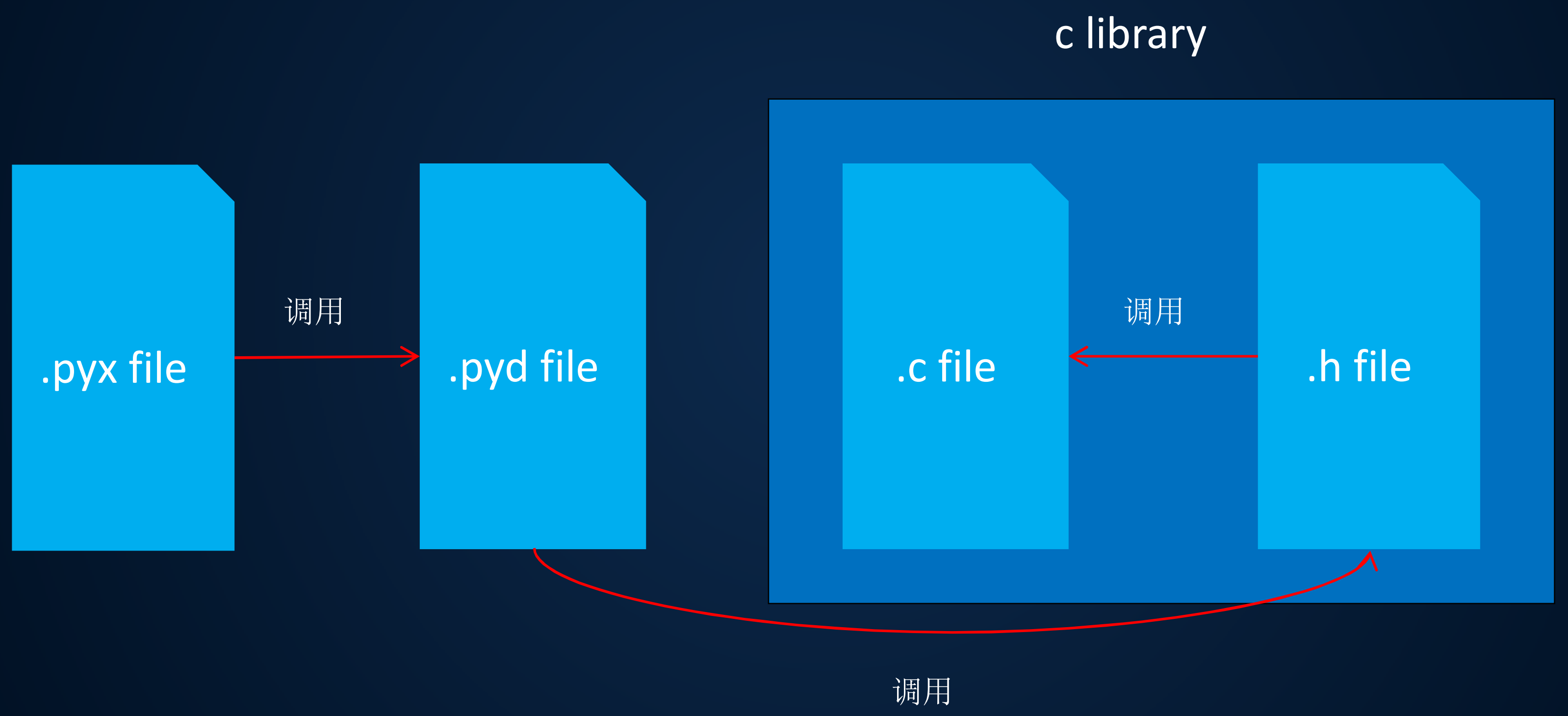
Py2: 字符串类型为bytes
Py3: 字符串类型为unicode



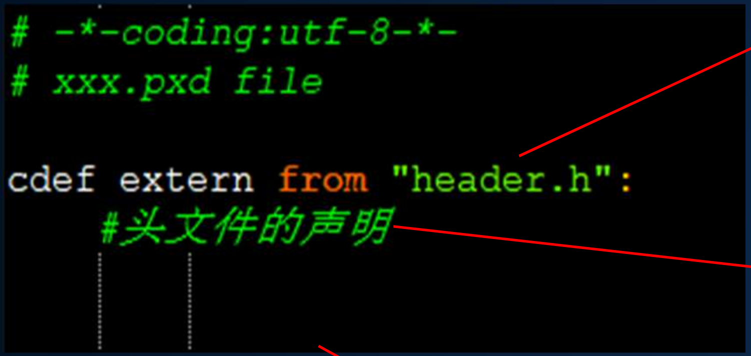
03.

Cython与C

Cython与C



»»Cython与C

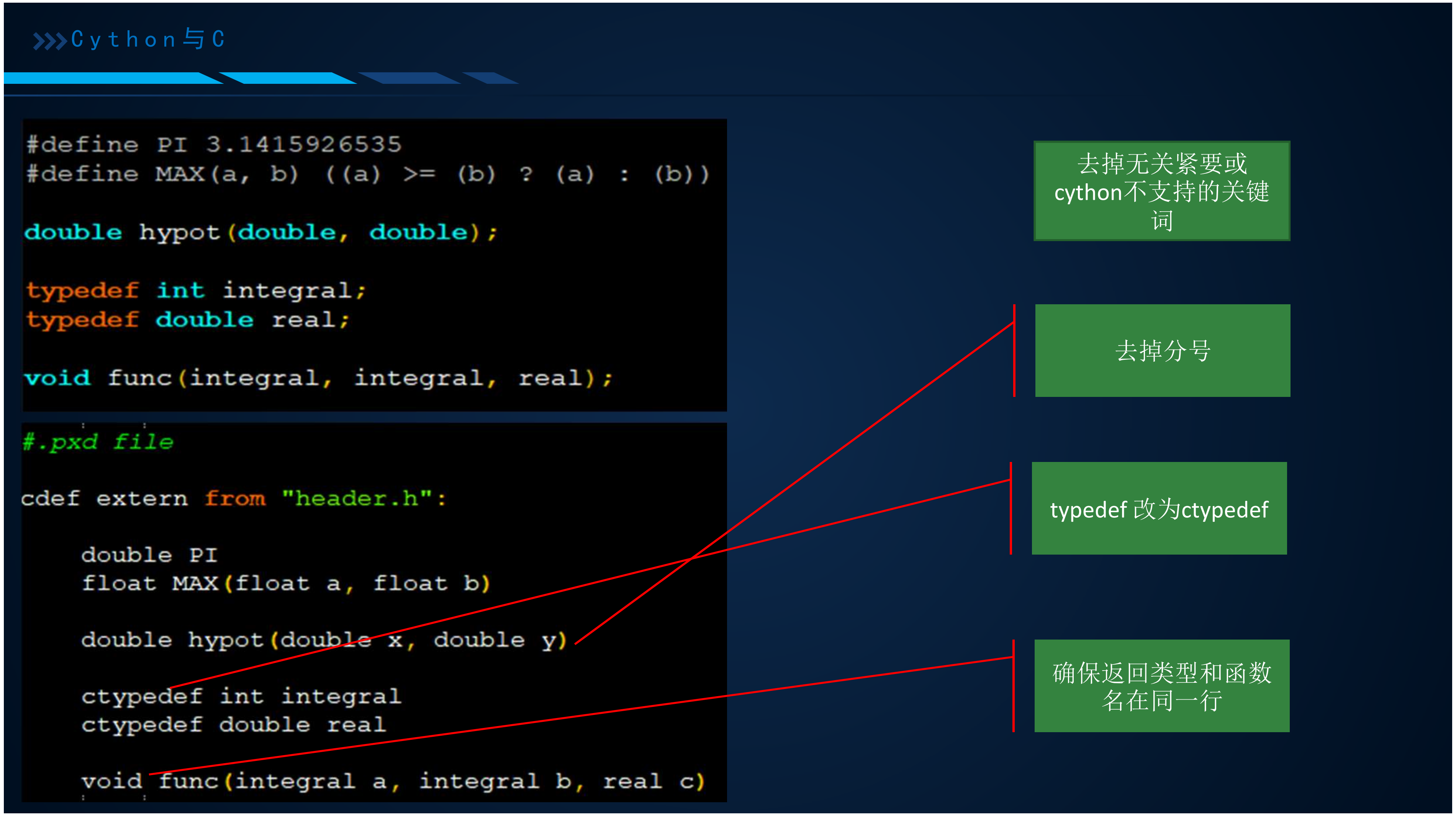


```
# -*-coding:utf-8-*-
# xxx.pxd file

cdef extern from "header.h":
    #头文件的声明
```

Cython编译器在原文件中生成一个#include “header.h”

块中的类型，函数与其他声明在Cython中都可用

Cython在编译时检查C声明是否用于正确的方式，如果不是会编译错误



»»Cython与C

```
#define PI 3.1415926535
#define MAX(a, b) ((a) >= (b) ? (a) : (b))

double hypot(double, double);

typedef int integral;
typedef double real;

void func(integral, integral, real);
```

```
#.pxd file

cdef extern from "header.h":

    double PI
    float MAX(float a, float b)

    double hypot(double x, double y)

    ctypedef int integral
    ctypedef double real

    void func(integral a, integral b, real c)
```

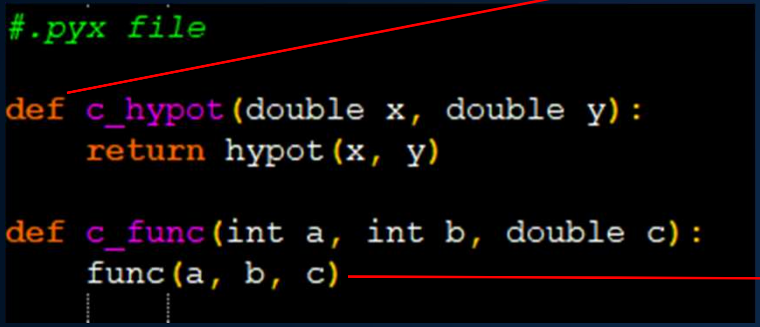
去掉无关紧要或cython不支持的关键词

去掉分号

typedef 改为ctypedef

确保返回类型和函数名在同一行

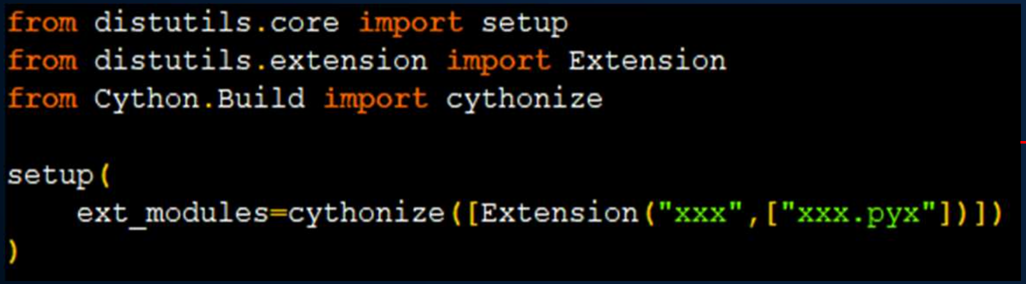
»»Cython与C



```
#.pyx file

def c_hypot(double x, double y):
    return hypot(x, y)

def c_func(int a, int b, double c):
    func(a, b, c)
```

def或cpdef

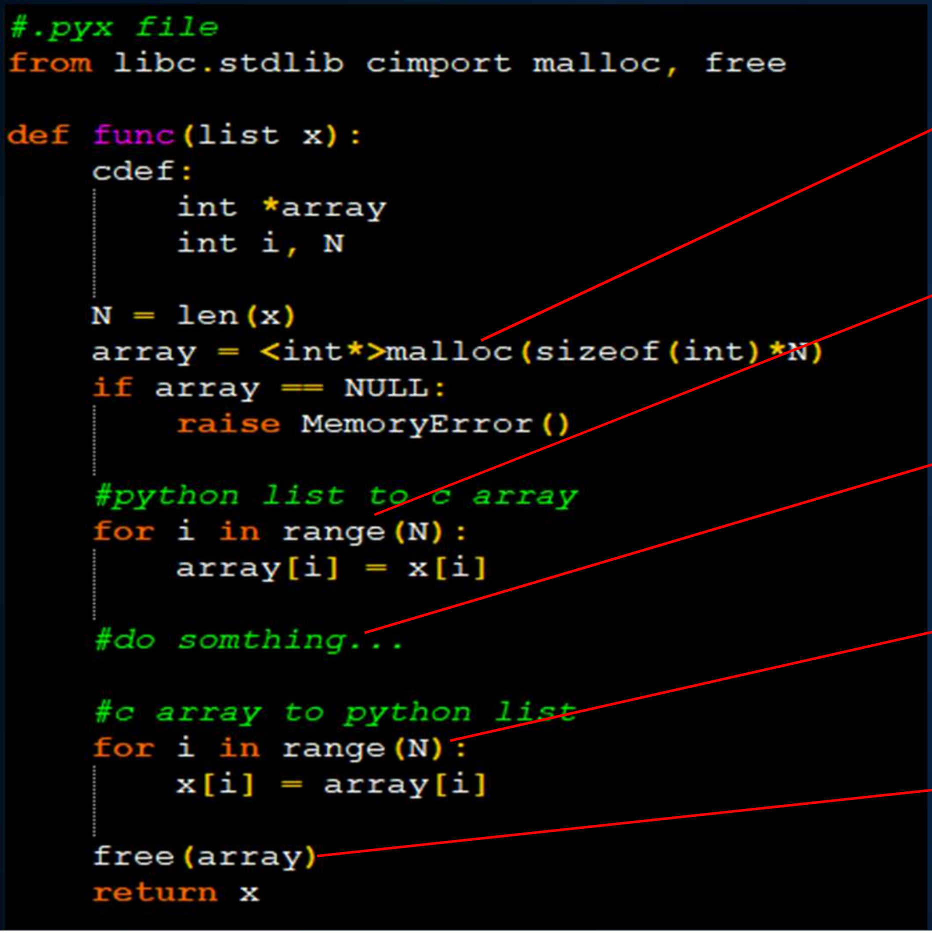
无返回值



```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Build import cythonize

setup(
    ext_modules=cythonize([Extension("xxx",["xxx.pyx"])])
)
```

python setup.py build_ext --inplace

>>>Cython与C



```
#.pyx file
from libc.stdlib cimport malloc, free

def func(list x):
    cdef:
        int *array
        int i, N

    N = len(x)
    array = <int*>malloc(sizeof(int)*N)
    if array == NULL:
        raise MemoryError()

    #python list to c array
    for i in range(N):
        array[i] = x[i]

    #do somthing...

    #c array to python list
    for i in range(N):
        x[i] = array[i]

    free(array)
    return x
```

分配合适大小的内存

Python列表转为C array

用C处理任务

C array转为Python列表

释放内存

04.

Cython与C++

Cython支持C++内容

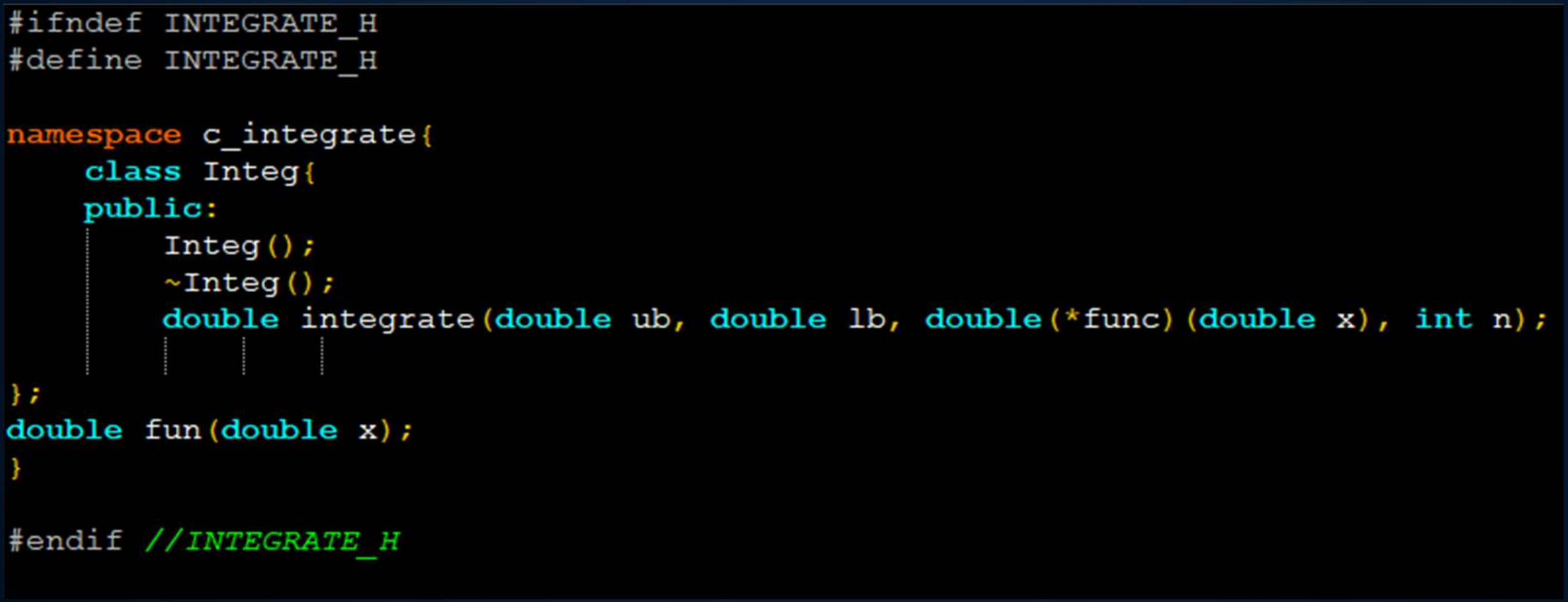
1. C++对象可以使用new和del关键字进行动态内存分配
2. C++对象能够使用栈分配内存
3. C++类能使用关键字cppclass进行声明
4. 支持模板类和函数
5. 支持C++操作符重载

封装C++步骤

1. 在setup.py里设定语言为C++语言
2. 创建一个或多个.pxd文件，其内有cdef extern from 块和C++命名空间，在这些块中：
   ① 声明类（cdef cppclass）
   ② 声明公共名称（变量，方法， 构造器等）

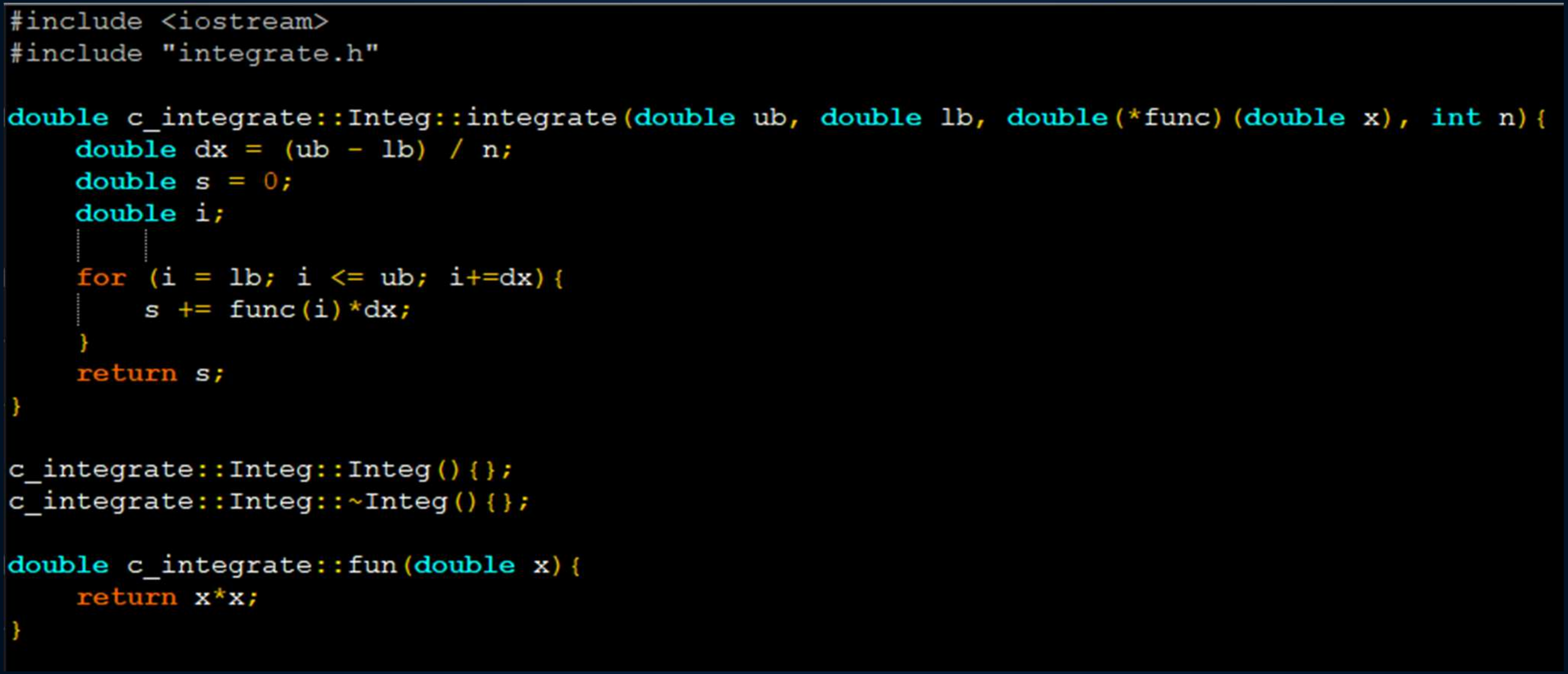
3. 把.pxd文件的内容cimport到.pyx文件中

Integrate.h文件



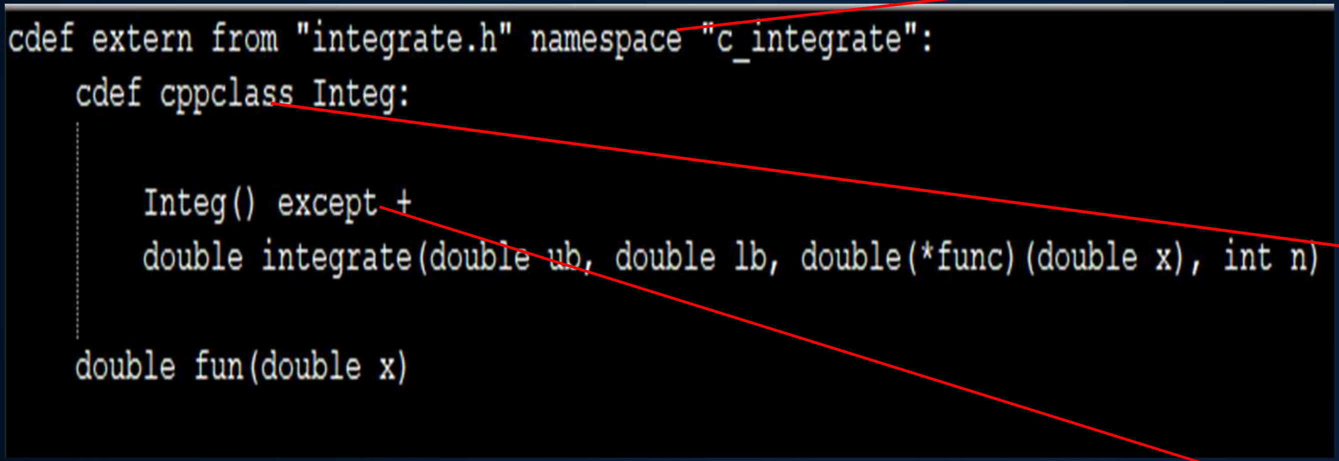
```cpp
#ifndef INTEGRATE_H
#define INTEGRATE_H

namespace c_integrate{
    class Integ{
    public:
        Integ();
        ~Integ();
        double integrate(double ub, double lb, double(*func)(double x), int n);

};
double fun(double x);
}

#endif //INTEGRATE_H
```

Integrate.cpp文件



```
#include <iostream>
#include "integrate.h"

double c_integrate::Integ::integrate(double ub, double lb, double(*func)(double x), int n){
    double dx = (ub - lb) / n;
    double s = 0;
    double i;

    for (i = lb; i <= ub; i+=dx){
        s += func(i)*dx;
    }
    return s;
}

c_integrate::Integ::Integ(){};
c_integrate::Integ::~Integ(){};

double c_integrate::fun(double x){
    return x*x;
}
```

Integrate.pxd文件



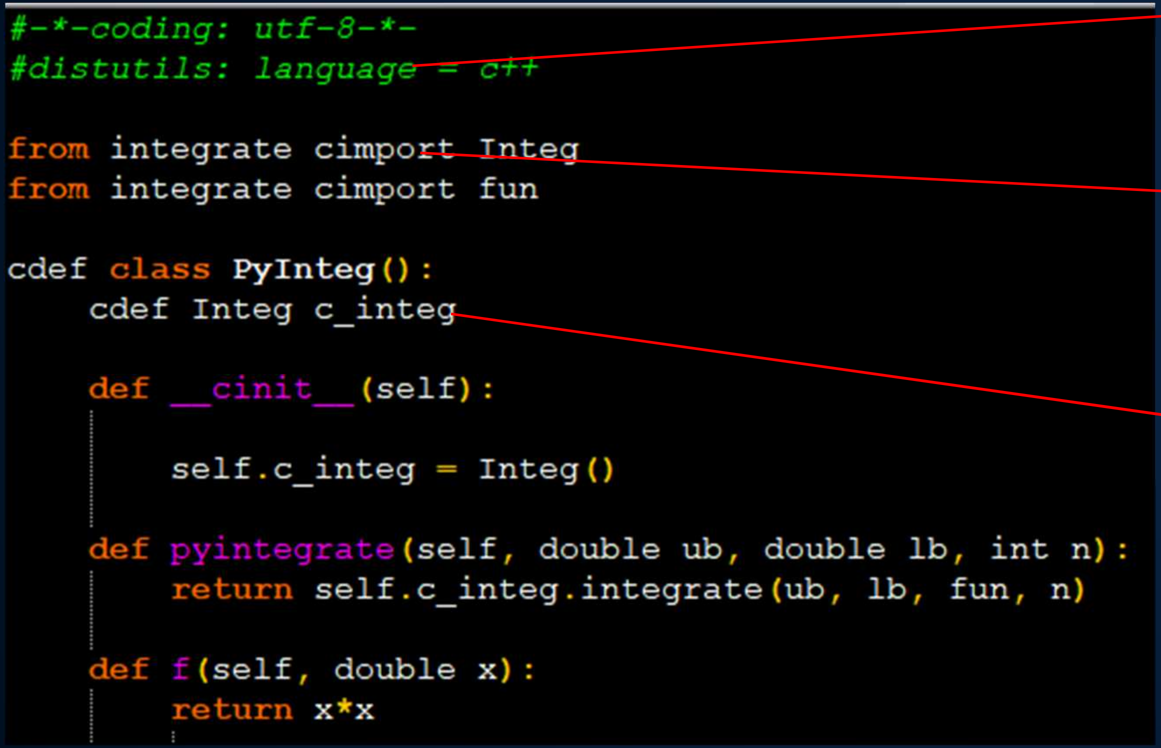
```
cdef extern from "integrate.h" namespace "c_integrate":
    cdef cppclass Integ:

        Integ() except +
        double integrate(double ub, double lb, double(*func)(double x), int n)

    double fun(double x)
```

包含类的命名空间

定义类使用cppclass关键词

构造器添加except+

itgr.pyx文件1



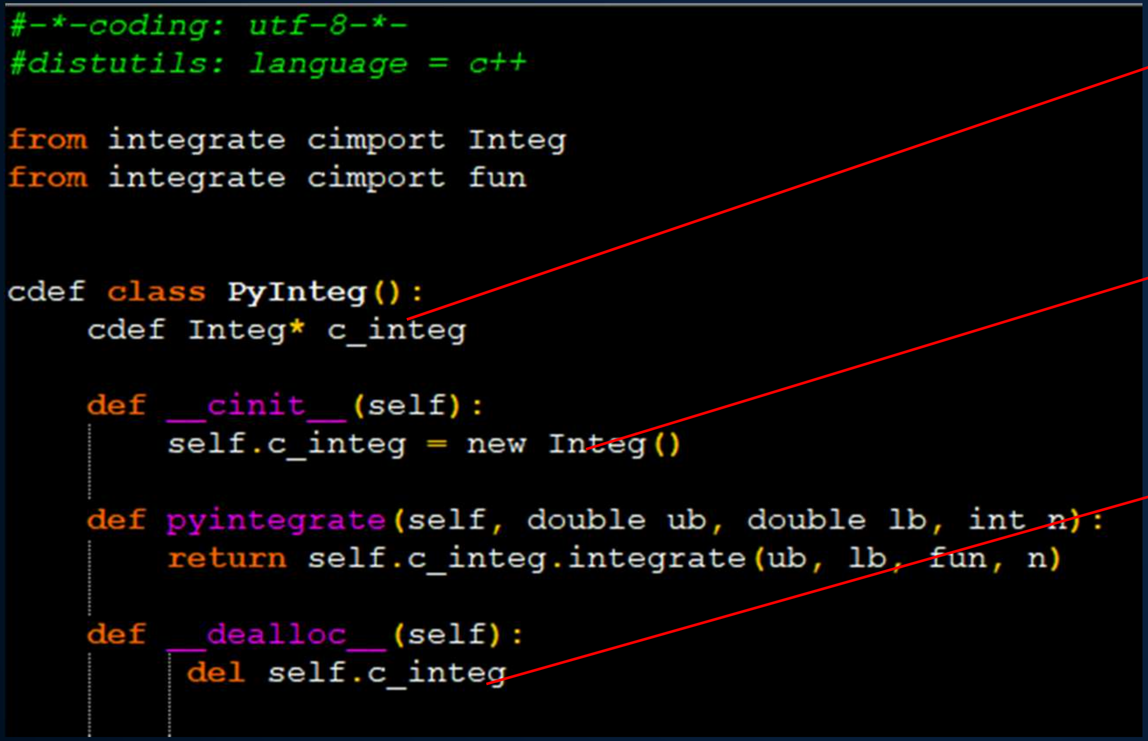
```
#-*-coding: utf-8-*-
#distutils: language = c++

from integrate cimport Integ
from integrate cimport fun

cdef class PyInteg():
    cdef Integ c_integ

    def __cinit__(self):

        self.c_integ = Integ()

    def pyintegrate(self, double ub, double lb, int n):
        return self.c_integ.integrate(ub, lb, fun, n)

    def f(self, double x):
        return x*x
```

设定语言为C++

cimport pxd文件的方法

使用栈分配内存的方法声明对象

itgr.pyx文件2



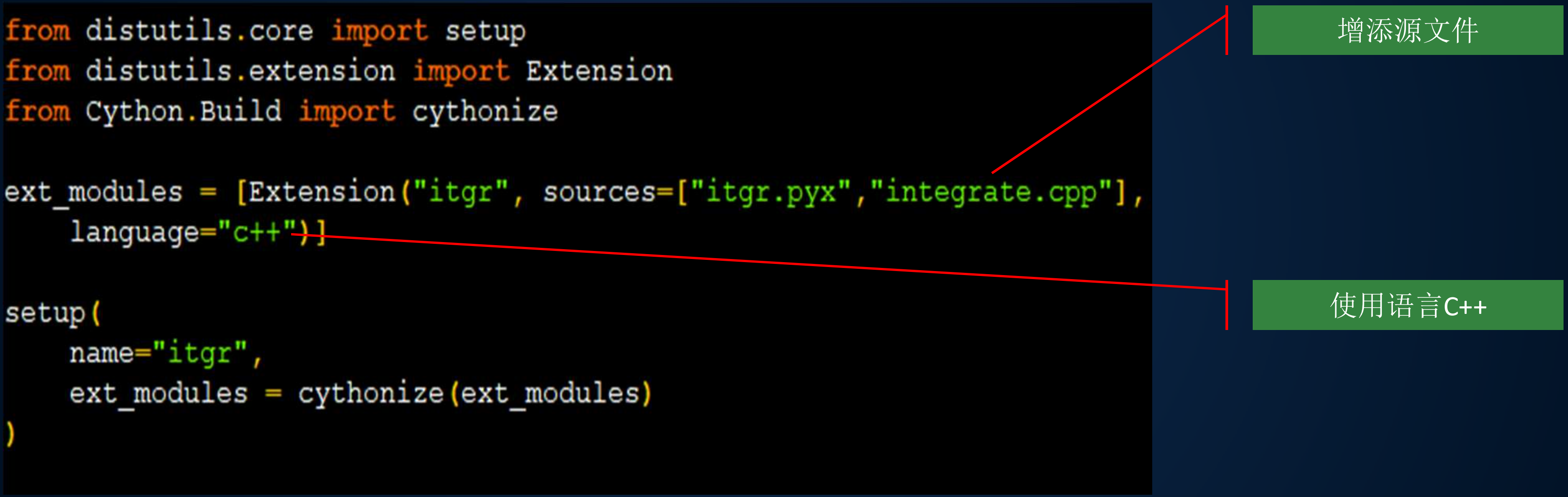
```
#-*-coding: utf-8-*-
#distutils: language = c++

from integrate cimport Integ
from integrate cimport fun


cdef class PyInteg():
    cdef Integ* c_integ

    def __cinit__(self):
        self.c_integ = new Integ()

    def pyintegrate(self, double ub, double lb, int n):
        return self.c_integ.integrate(ub, lb, fun, n)

    def __dealloc__(self):
        del self.c_integ
```

使用指针进行堆内存分配对象

使用new关键字进行初始化

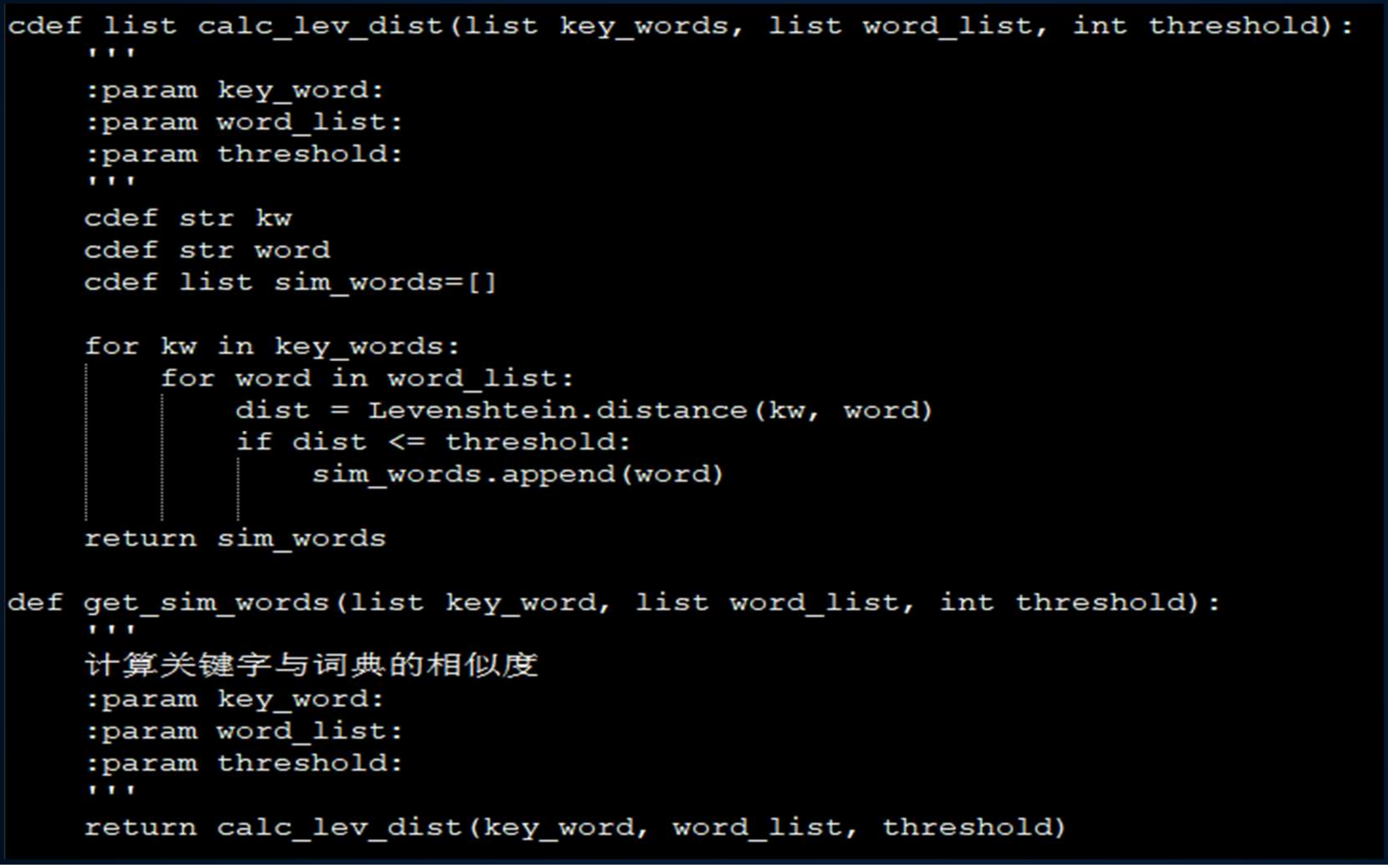
使用del关键字进行垃圾回收，del关键字只能删除在堆上建立的对象，如果对象建立在栈上会出错

Setup.py文件



```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Build import cythonize

ext_modules = [Extension("itgr", sources=["itgr.pyx","integrate.cpp"],
    language="c++")]

setup(
    name="itgr",
    ext_modules = cythonize(ext_modules)
)
```

Cython与C++



```
cdef list calc_lev_dist(list key_words, list word_list, int threshold):
    '''
    :param key_word:
    :param word_list:
    :param threshold:
    '''
    cdef str kw
    cdef str word
    cdef list sim_words=[]

    for kw in key_words:
        for word in word_list:
            dist = Levenshtein.distance(kw, word)
            if dist <= threshold:
                sim_words.append(word)

    return sim_words

def get_sim_words(list key_word, list word_list, int threshold):
    '''
    计算关键字与词典的相似度
    :param key_word:
    :param word_list:
    :param threshold:
    '''
    return calc_lev_dist(key_word, word_list, threshold)
```

2018

演示完毕 感谢聆听

Thanks for listening