A Cython Walkthrough

Nick Murdoch

Code will be displayed at this size on white. Slides: tinyurl.com/cywalk

tinyurl.com/cywalk

Hello

Nick Murdoch
@nickmurdoch

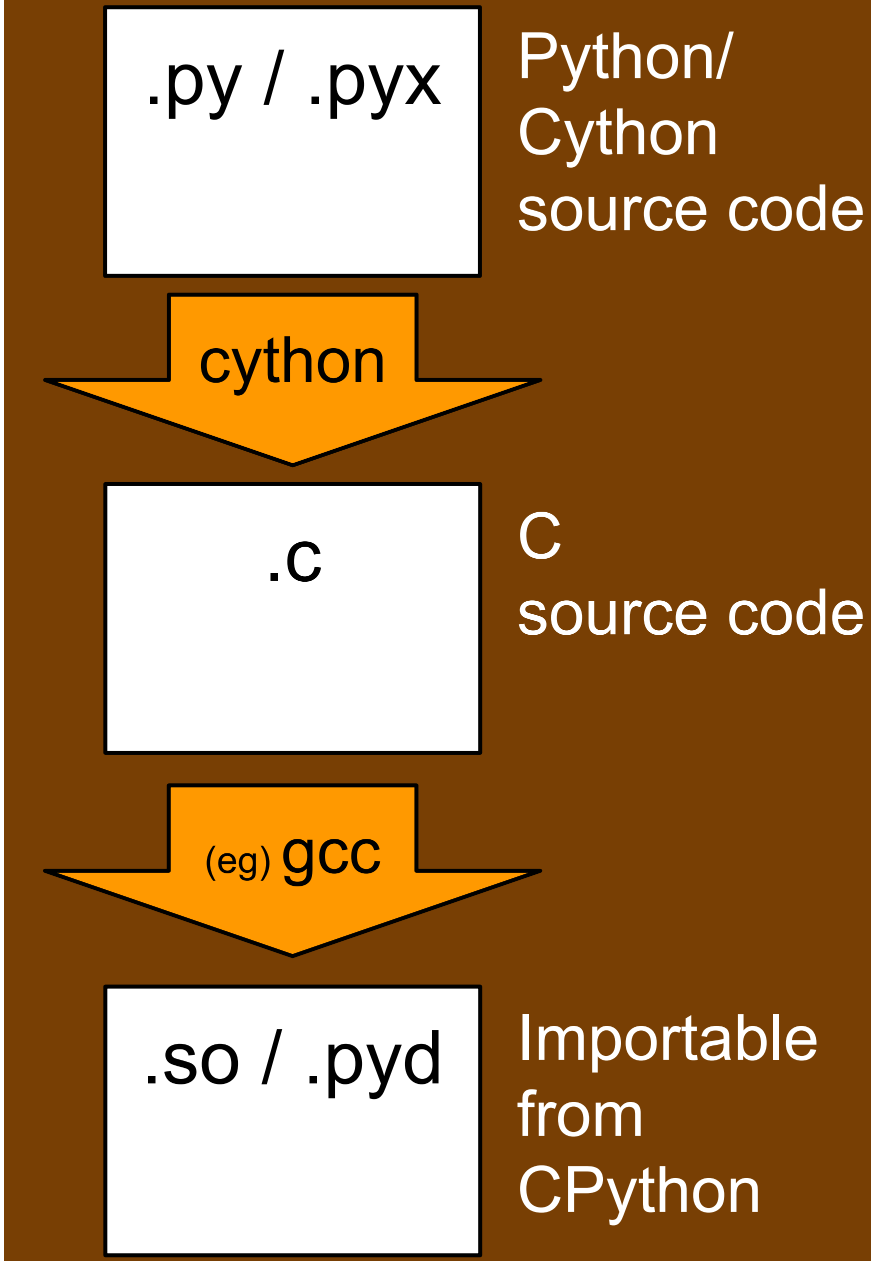
- Python - 12 years
- C - 9 years
- Pyrex/Cython - 9 y
- I just work here

So what is Cython?

- A programming language
- Looks like Python
- Based on Pyrex

tinyurl.com/cywalk

So what is Cython?



.py / .pyx

Python/
Cython
source code

cython

.c

C
source code

(eg) gcc

.so / .pyd

Importable
from
CPython

Uses for Cython

1. Faster Python code
2. Call Python from C
3. Call C from Python

Uses for Cython

1. Faster Python code
2. Call Python from C
3. Call C from Python

1. Faster Python code

tinyurl.com/cywalk

- Example problem
  - A function to find the maximum product of *n* consecutive digits in a list
  - Eg: digits = [6, 2, 3, 6, 5, 4]; n = 3
  - Result = 6 × 5 × 4 = 120

1. Faster Python code

```
# bigproduct.py:

def bigproduct(digits, n=13):
    '''Return the biggest product of n
       consecutive digits'''
    best = 0
    for i in range(len(digits) - (n - 1)):






    return best
```

1. Faster Python code

```python
# bigproduct.py:

def bigproduct(digits, n=13):
    '''Return the biggest product of n
       consecutive digits'''
    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            product *= digits[i + j]



    return best
```

1. Faster Python code

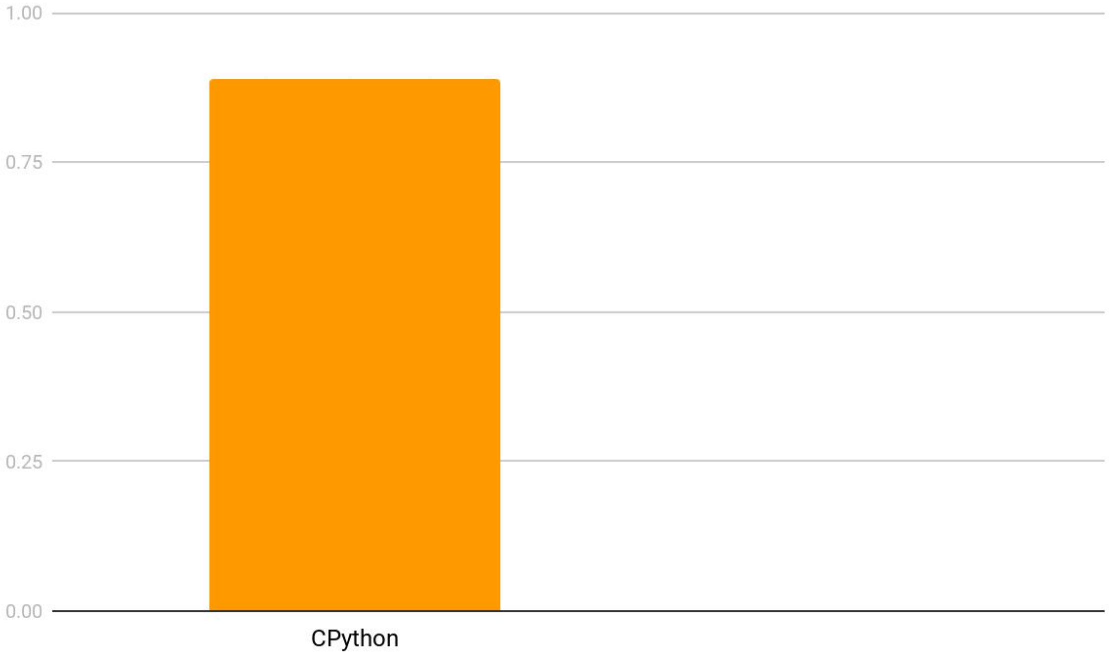
```python
# bigproduct.py:

def bigproduct(digits, n=13):
    '''Return the biggest product of n
       consecutive digits'''
    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            product *= digits[i + j]
        if product > best:
            best = product
    return best
```

1. Faster Python code

tinyurl.com/cywalk

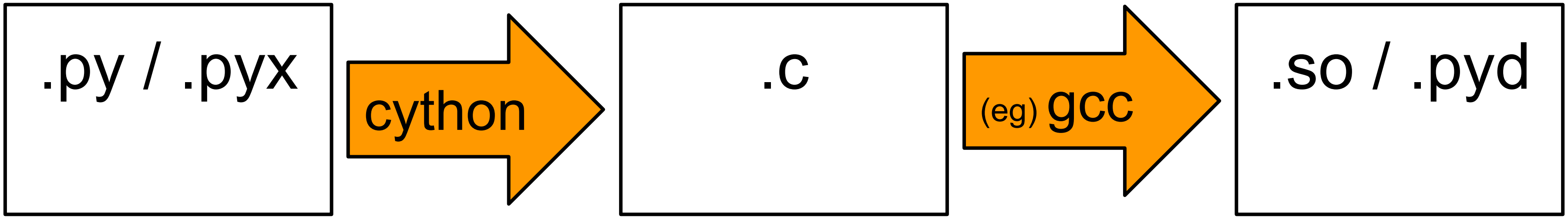
Timing, 1000 runs:

- Pure Python
  - 0.89s



1. Faster Python code

tinyurl.com/cywalk



.py / .pyx

cython

.c

(eg) gcc

.so / .pyd

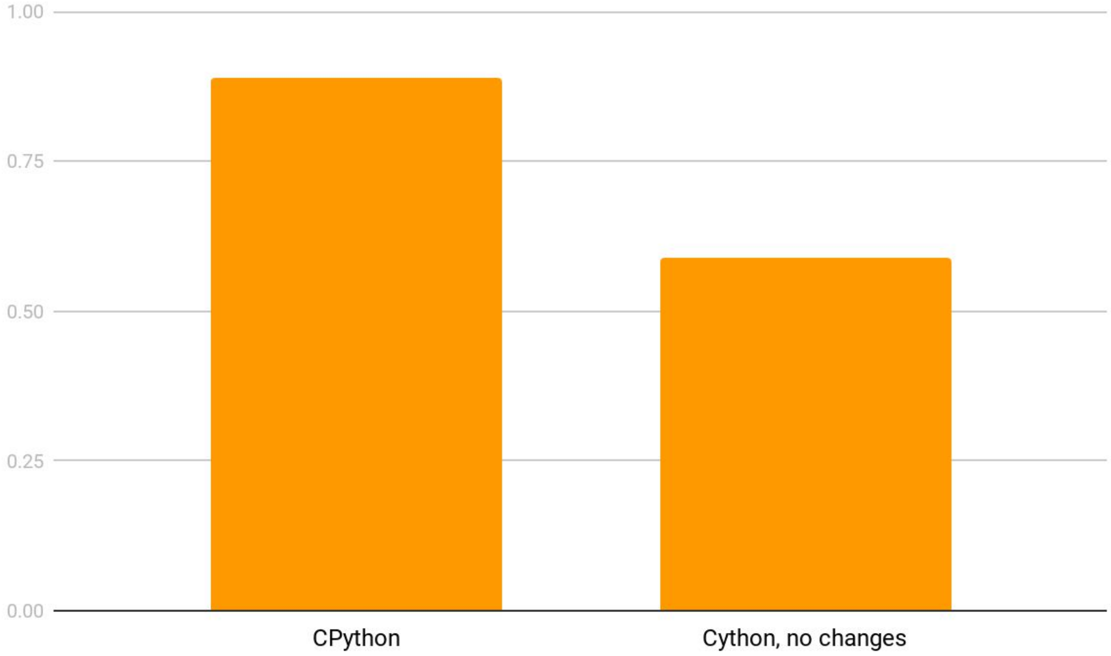
```
# setup.py:

from distutils.core import setup
    # or `from setuptools import setup`
from Cython.Build import cythonize


setup(
    # …
    ext_modules = cythonize("bigproduct.py")
)
```

1. Faster Python code

tinyurl.com/cywalk

Timing, 1000 runs:

- Pure Python:
  - 0.89s
- Cython, no code changes:
  - 0.59s

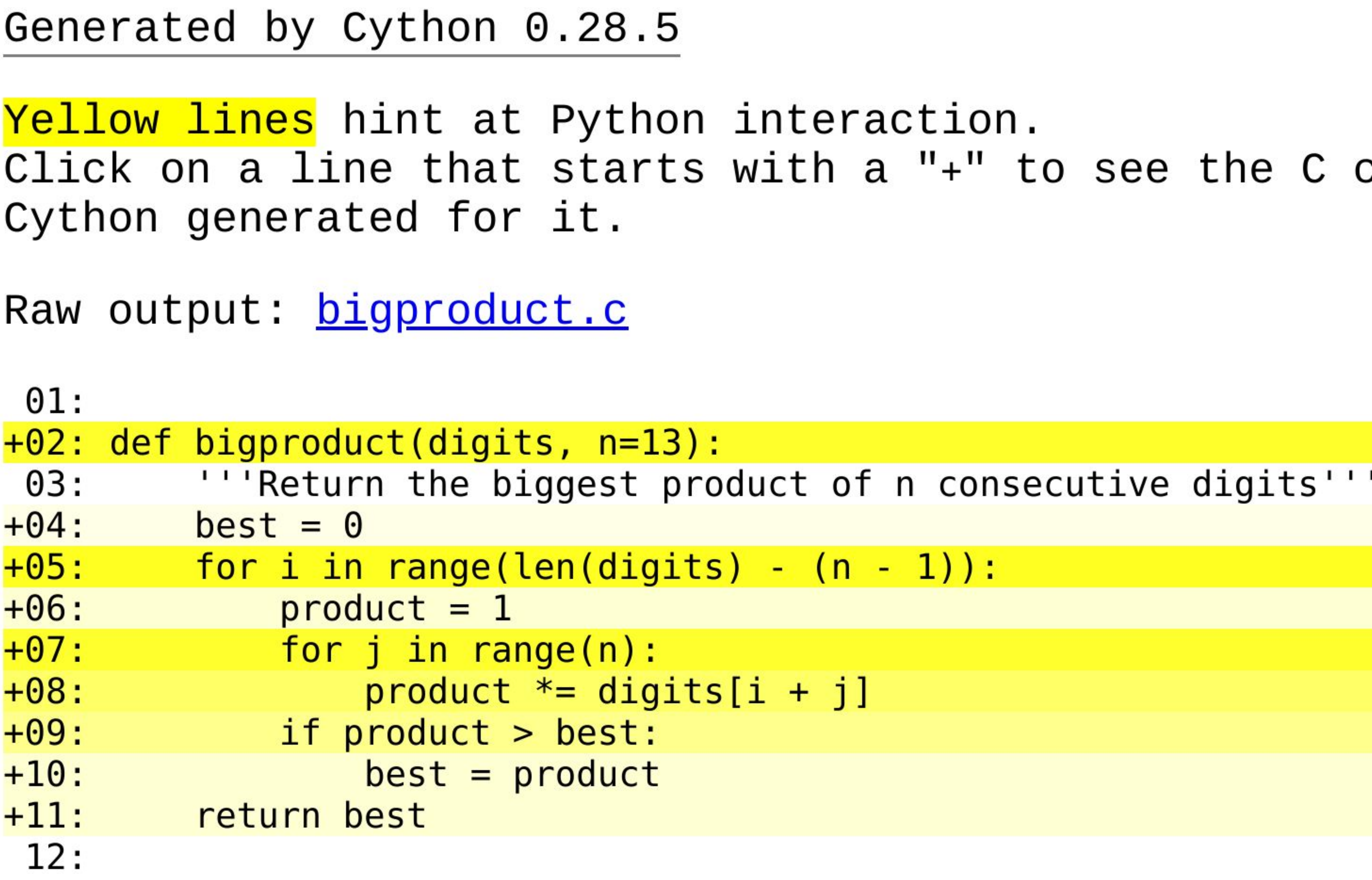


1. Faster Python code

tinyurl.com/cywalk

- What’s taking the most time?
- Python!
- Where are the most calls into Python?
- `$ cython -a bigproduct.py`

  ➜ bigproduct.html

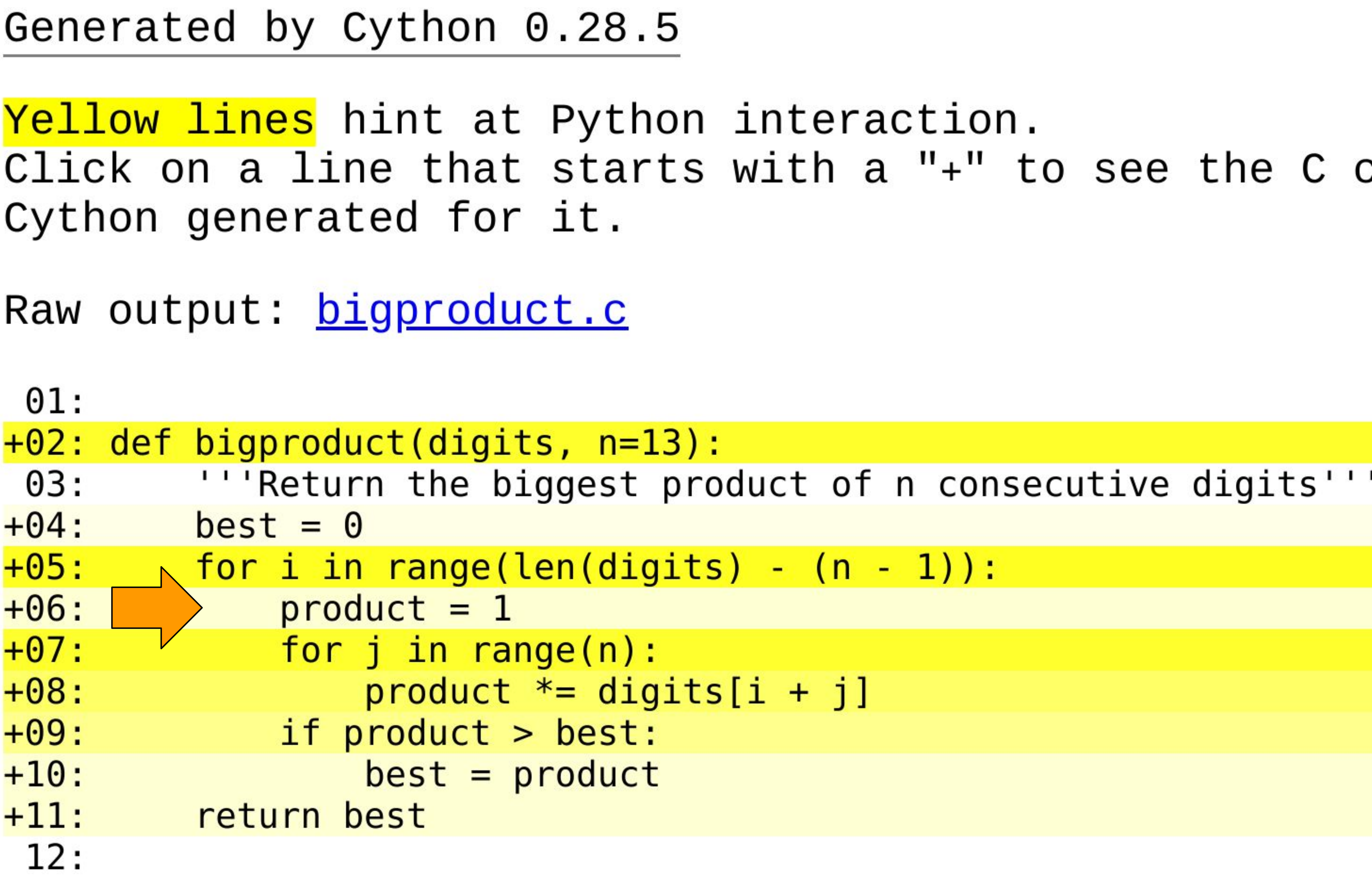


Generated by Cython 0.28.5

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C c
Cython generated for it.

Raw output: bigproduct.c

```
 01: 
+02: def bigproduct(digits, n=13):
 03:     '''Return the biggest product of n consecutive digits'''
+04:     best = 0
+05:     for i in range(len(digits) - (n - 1)):
+06:         product = 1
+07:         for j in range(n):
+08:             product *= digits[i + j]
+09:         if product > best:
+10:             best = product
+11:     return best
 12: 
```

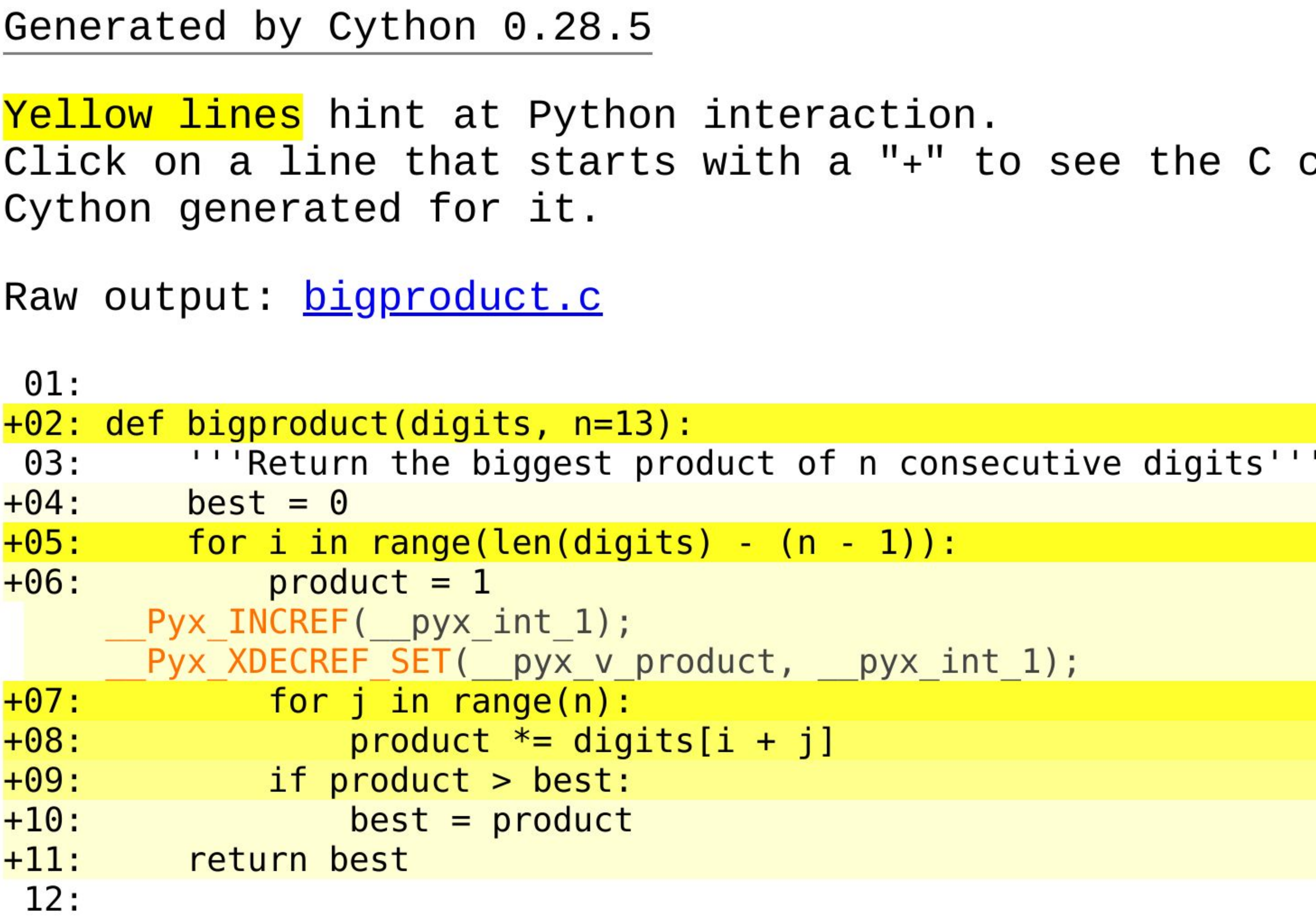


Generated by Cython 0.28.5

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C c
Cython generated for it.

Raw output: bigproduct.c

```
 01:
+02: def bigproduct(digits, n=13):
 03:     '''Return the biggest product of n consecutive digits''
+04:     best = 0
+05:     for i in range(len(digits) - (n - 1)):
+06:         product = 1
+07:         for j in range(n):
+08:             product *= digits[i + j]
+09:         if product > best:
+10:             best = product
+11:     return best
 12:
```

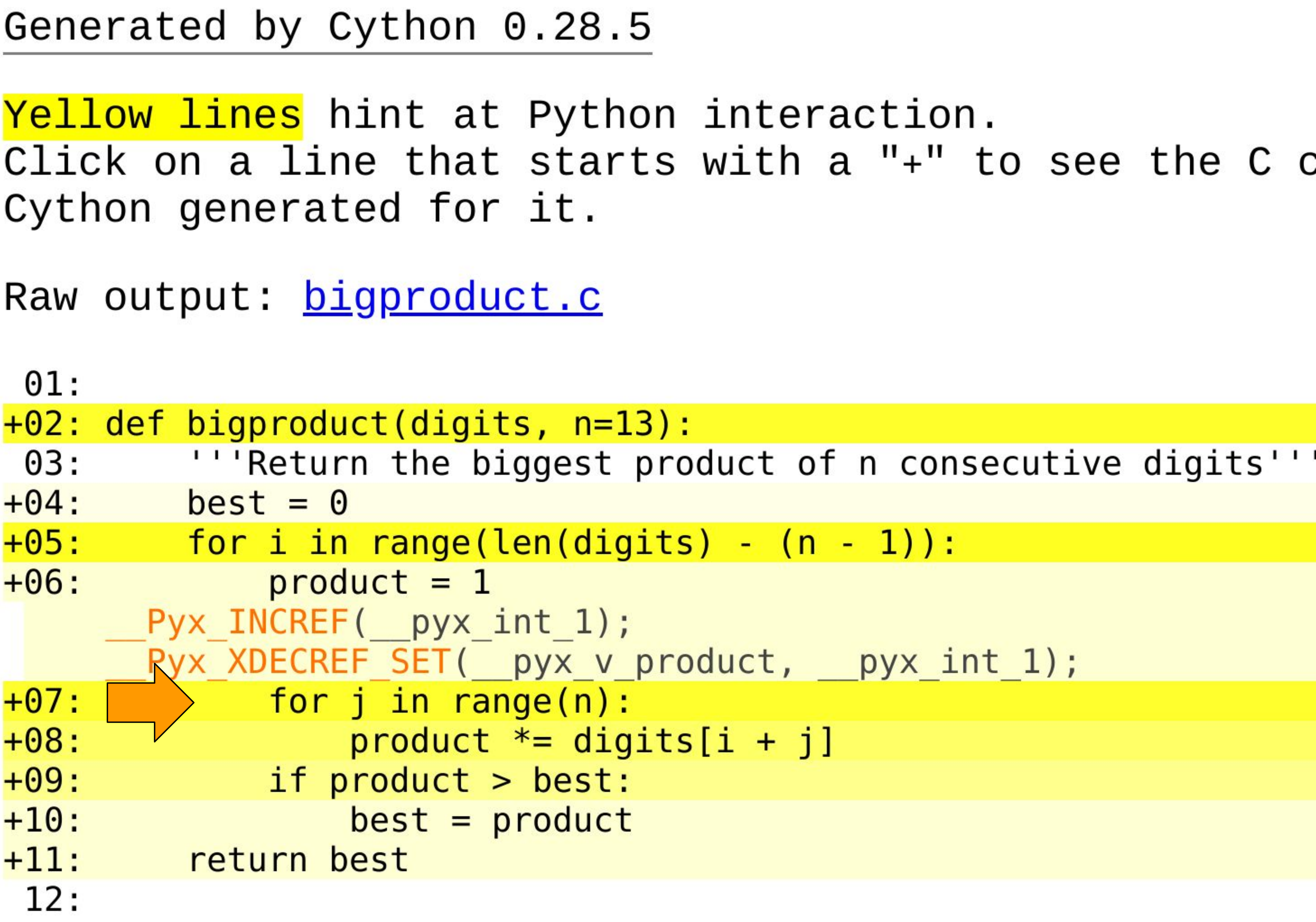


Generated by Cython 0.28.5

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C c
Cython generated for it.

Raw output: bigproduct.c

```
 01: 
+02: def bigproduct(digits, n=13):
 03:     '''Return the biggest product of n consecutive digits''
+04:     best = 0
+05:     for i in range(len(digits) - (n - 1)):
+06:         product = 1
    __Pyx_INCREF(__pyx_int_1);
    __Pyx_XDECREF_SET(__pyx_v_product, __pyx_int_1);
+07:         for j in range(n):
+08:             product *= digits[i + j]
+09:         if product > best:
+10:             best = product
+11:     return best
 12: 
```

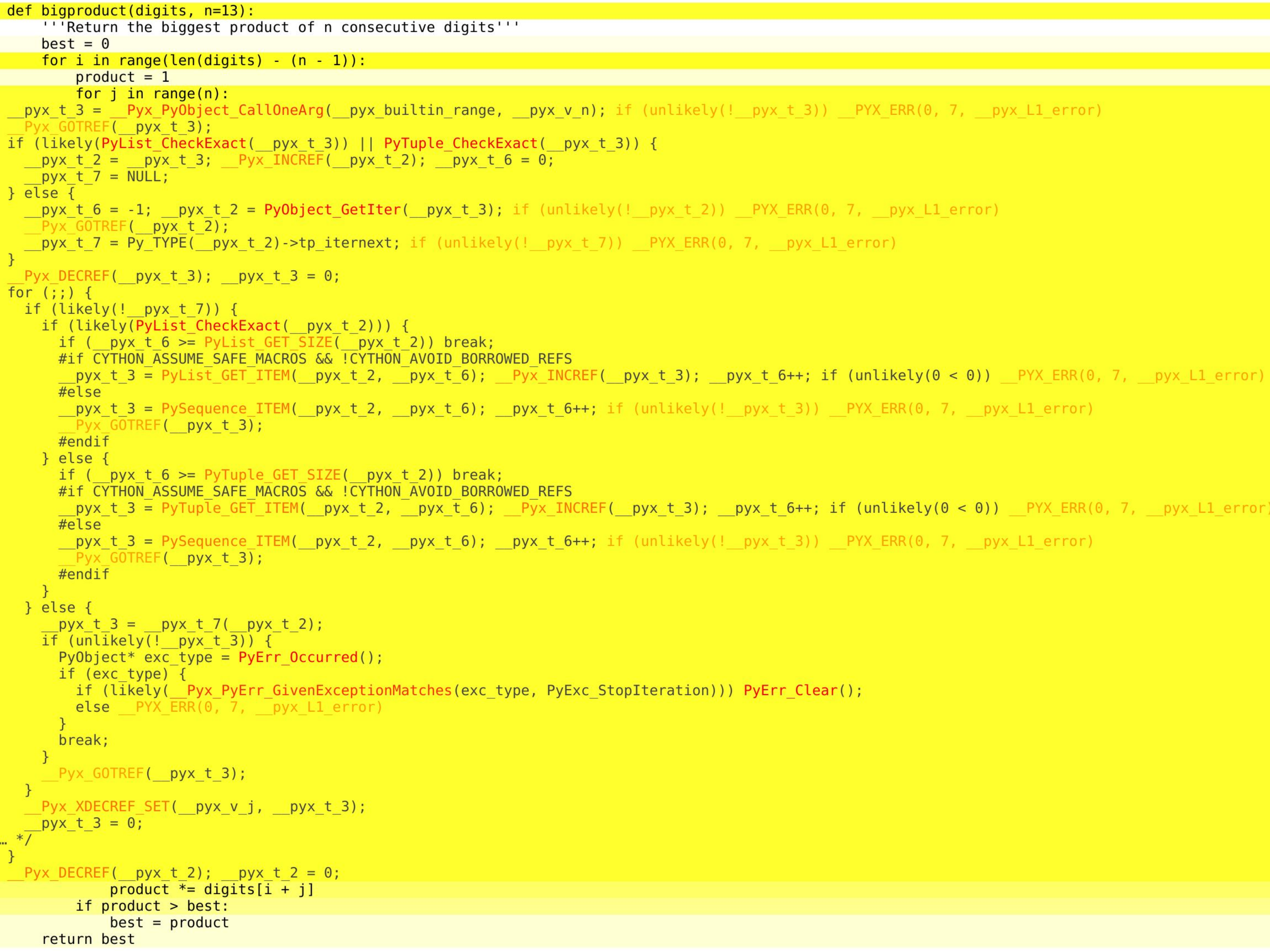


Generated by Cython 0.28.5

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C c
Cython generated for it.

Raw output: bigproduct.c

```
 01:
+02: def bigproduct(digits, n=13):
 03:     '''Return the biggest product of n consecutive digits''
+04:     best = 0
+05:     for i in range(len(digits) - (n - 1)):
+06:         product = 1
    __Pyx_INCREF(__pyx_int_1);
    __Pyx_XDECREF_SET(__pyx_v_product, __pyx_int_1);
+07:         for j in range(n):
+08:             product *= digits[i + j]
+09:         if product > best:
+10:             best = product
+11:     return best
 12:
```

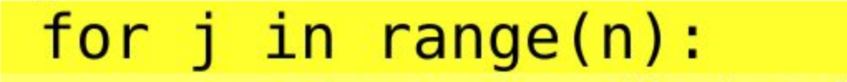


```
def bigproduct(digits, n=13):
    '''Return the biggest product of n consecutive digits'''
    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
__pyx_t_3 = __Pyx_PyObject_CallOneArg(__pyx_builtin_range, __pyx_v_n); if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 7, __pyx_L1_error)
__Pyx_GOTREF(__pyx_t_3);
if (likely(PyList_CheckExact(__pyx_t_3)) || PyTuple_CheckExact(__pyx_t_3)) {
  __pyx_t_2 = __pyx_t_3; __Pyx_INCREF(__pyx_t_2); __pyx_t_6 = 0;
  __pyx_t_7 = NULL;
} else {
  __pyx_t_6 = -1; __pyx_t_2 = PyObject_GetIter(__pyx_t_3); if (unlikely(!__pyx_t_2)) __PYX_ERR(0, 7, __pyx_L1_error)
  __Pyx_GOTREF(__pyx_t_2);
  __pyx_t_7 = Py_TYPE(__pyx_t_2)->tp_iternext; if (unlikely(!__pyx_t_7)) __PYX_ERR(0, 7, __pyx_L1_error)
}
__Pyx_DECREF(__pyx_t_3); __pyx_t_3 = 0;
for (;;) {
  if (likely(!__pyx_t_7)) {
    if (likely(PyList_CheckExact(__pyx_t_2))) {
      if (__pyx_t_6 >= PyList_GET_SIZE(__pyx_t_2)) break;
      #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
      __pyx_t_3 = PyList_GET_ITEM(__pyx_t_2, __pyx_t_6); __Pyx_INCREF(__pyx_t_3); __pyx_t_6++; if (unlikely(0 < 0)) __PYX_ERR(0, 7, __pyx_L1_error)
      #else
      __pyx_t_3 = PySequence_ITEM(__pyx_t_2, __pyx_t_6); __pyx_t_6++; if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 7, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_3);
      #endif
    } else {
      if (__pyx_t_6 >= PyTuple_GET_SIZE(__pyx_t_2)) break;
      #if CYTHON_ASSUME_SAFE_MACROS && !CYTHON_AVOID_BORROWED_REFS
      __pyx_t_3 = PyTuple_GET_ITEM(__pyx_t_2, __pyx_t_6); __Pyx_INCREF(__pyx_t_3); __pyx_t_6++; if (unlikely(0 < 0)) __PYX_ERR(0, 7, __pyx_L1_error)
      #else
      __pyx_t_3 = PySequence_ITEM(__pyx_t_2, __pyx_t_6); __pyx_t_6++; if (unlikely(!__pyx_t_3)) __PYX_ERR(0, 7, __pyx_L1_error)
      __Pyx_GOTREF(__pyx_t_3);
      #endif
    }
  } else {
    __pyx_t_3 = __pyx_t_7(__pyx_t_2);
    if (unlikely(!__pyx_t_3)) {
      PyObject* exc_type = PyErr_Occurred();
      if (exc_type) {
        if (likely(__Pyx_PyErr_GivenExceptionMatches(exc_type, PyExc_StopIteration))) PyErr_Clear();
        else __PYX_ERR(0, 7, __pyx_L1_error)
      }
      break;
    }
    __Pyx_GOTREF(__pyx_t_3);
  }
  __Pyx_XDECREF_SET(__pyx_v_j, __pyx_t_3);
  __pyx_t_3 = 0;
… */
}
__Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
            product *= digits[i + j]
        if product > best:
            best = product
    return best
```

1. Faster Python code

tinyurl.com/cywalk

... but briefly:



```
for j in range(n):
```

- Pyx_PyObject_CallOneArg(builtin_range, n)
- PyObject_GetIter()
- for (;;) {
  - j = tp_iternext()
  - PyErr_Occurred() (StopIteration? ➔ break)
  - [rest of block] }
- Also some optimisations for lists and tuples

1. Faster Python code

tinyurl.com/cywalk

```
# bigproduct.py:
def bigproduct(digits, n=13):


    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            digit = digits[i + j]
            product *= digit
        if product > best:
            best = product
    return best
```

1. Faster Python code

```
# bigproduct.pyx:
def bigproduct(digits, n=13):



    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            digit = digits[i + j]
            product *= digit
        if product > best:
            best = product
    return best
```

1. Faster Python code

tinyurl.com/cywalk

```
# bigproduct.pyx:
def bigproduct(digits, unsigned int n=13):


    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            digit = digits[i + j]
            product *= digit
        if product > best:
            best = product
    return best
```

1. Faster Python code

tinyurl.com/cywalk

```
# bigproduct.pyx:
def bigproduct(digits, unsigned int n=13):
    cdef size_t i, j

    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            digit = digits[i + j]
            product *= digit
        if product > best:
            best = product
    return best
```

1. Faster Python code

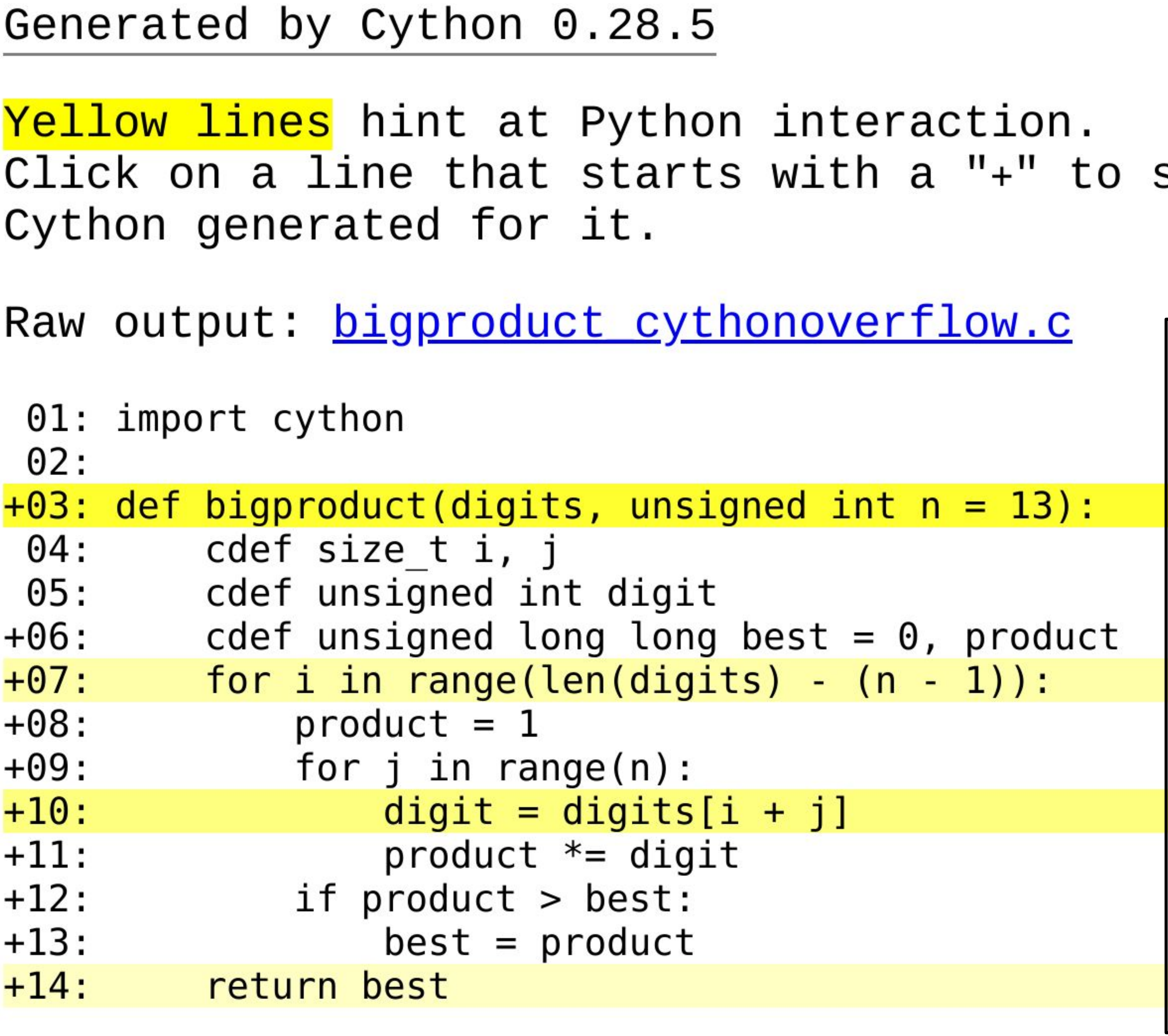
tinyurl.com/cywalk

```
# bigproduct.pyx:
def bigproduct(digits, unsigned int n=13):
    cdef size_t i, j
    cdef unsigned int digit
    best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            digit = digits[i + j]
            product *= digit
        if product > best:
            best = product
    return best
```

1. Faster Python code

tinyurl.com/cywalk

```
# bigproduct.pyx:
def bigproduct(digits, unsigned int n=13):
    cdef size_t i, j
    cdef unsigned int digit
    cdef unsigned long long product, best = 0
    for i in range(len(digits) - (n - 1)):
        product = 1
        for j in range(n):
            digit = digits[i + j]
            product *= digit
        if product > best:
            best = product
    return best
```

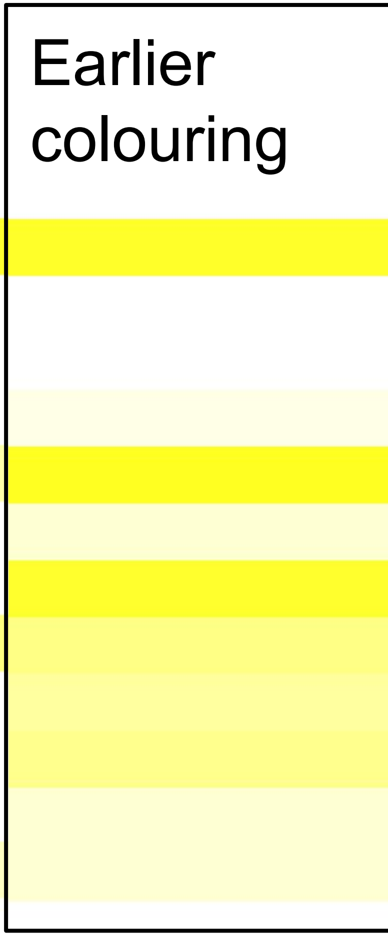


Generated by Cython 0.28.5

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C co
Cython generated for it.

Raw output: bigproduct_cythonoverflow.c

```
 01: import cython
 02:
+03: def bigproduct(digits, unsigned int n = 13):
 04:     cdef size_t i, j
 05:     cdef unsigned int digit
+06:     cdef unsigned long long best = 0, product
+07:     for i in range(len(digits) - (n - 1)):
+08:         product = 1
+09:         for j in range(n):
+10:             digit = digits[i + j]
+11:             product *= digit
+12:         if product > best:
+13:             best = product
+14:     return best
```



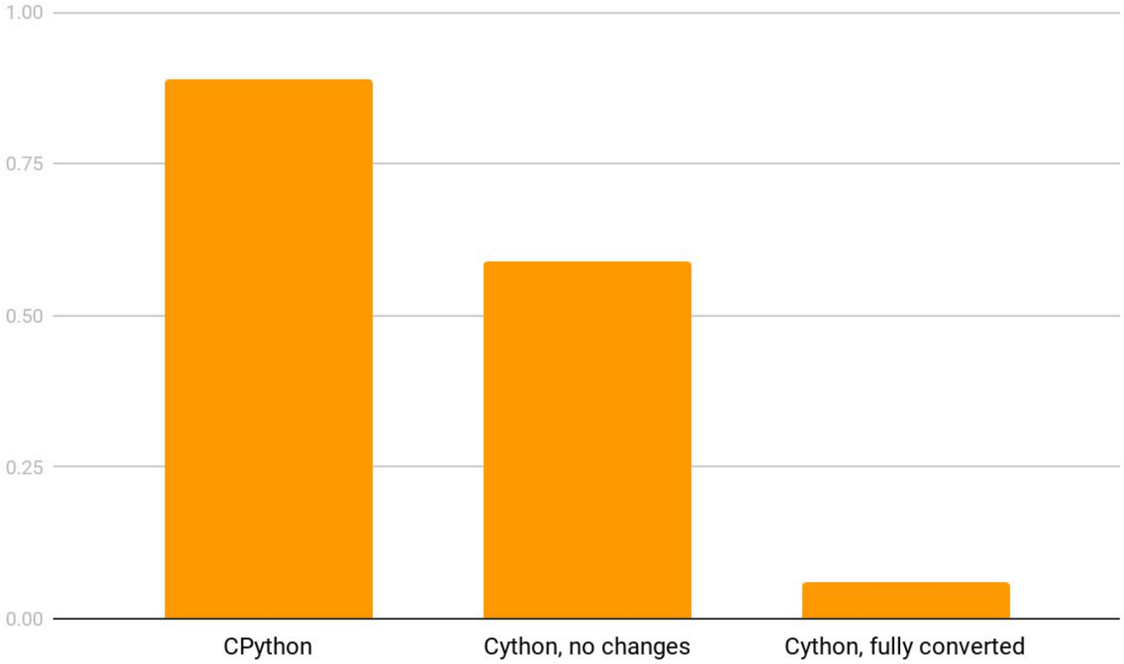
Earlier colouring

1. Faster Python code

tinyurl.com/cywalk

Timing, 1000 runs:

- Pure Python:
  - 0.89s
- Cython, no code changes:
  - 0.59s
- Cython, fully converted:
  - 0.06s



Uses for Cython →

1. Faster Python code
2. Call Python from C
3. Call C from Python

2. Call Python from C

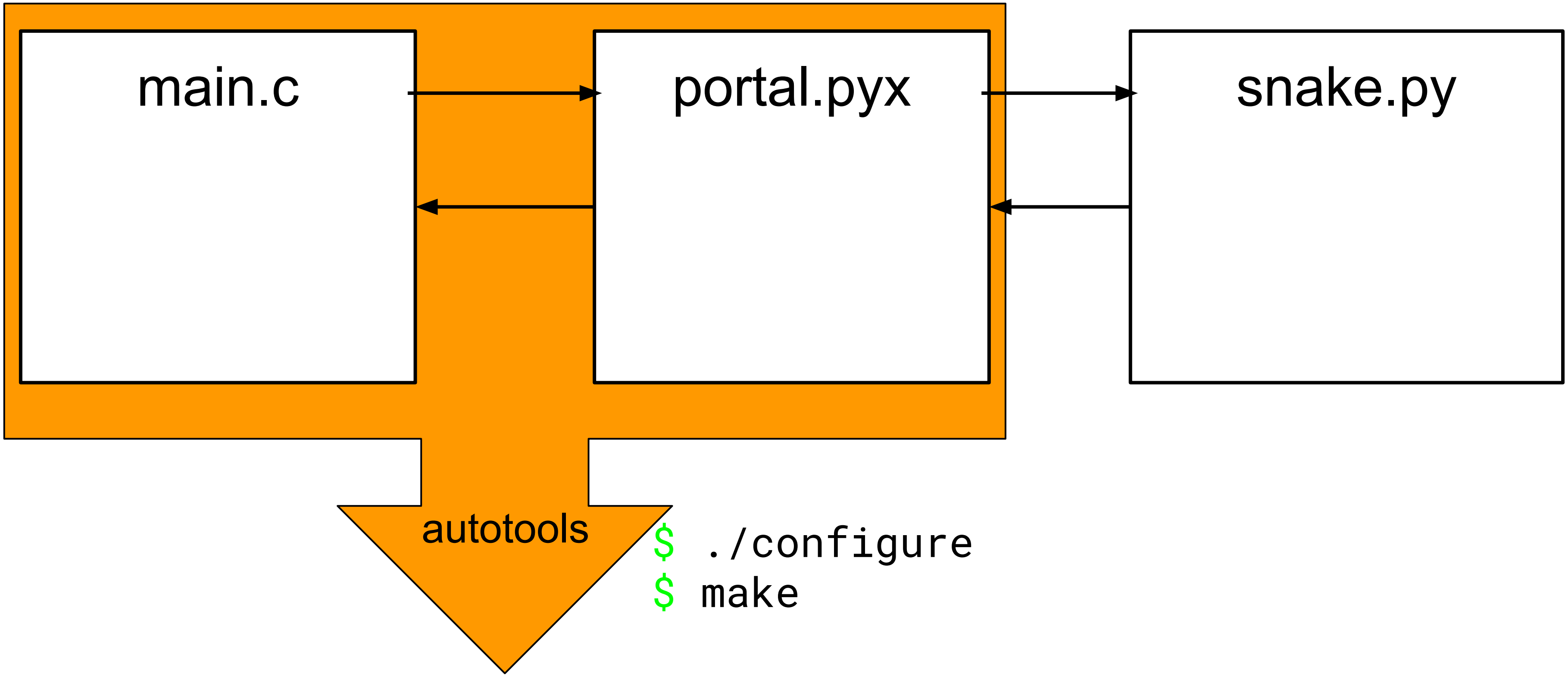
tinyurl.com/cywalk

But why?

- Script customisable tasks with Python
- C program with Python plugins
- Fast prototyping of features

2. Call Python from C

tinyurl.com/cywalk

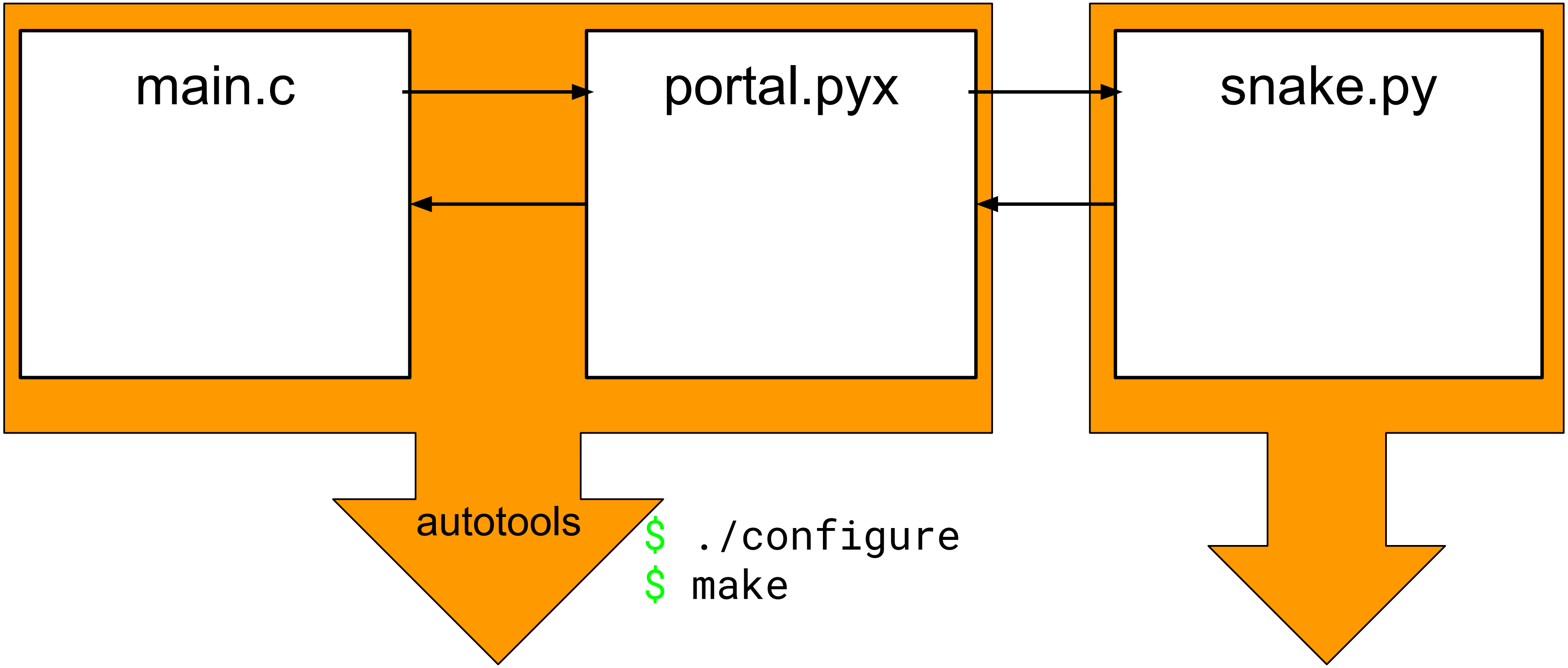


```
$ ./configure
$ make
```

```
$ ./cythonwalkthrough
```

2. Call Python from C

tinyurl.com/cywalk



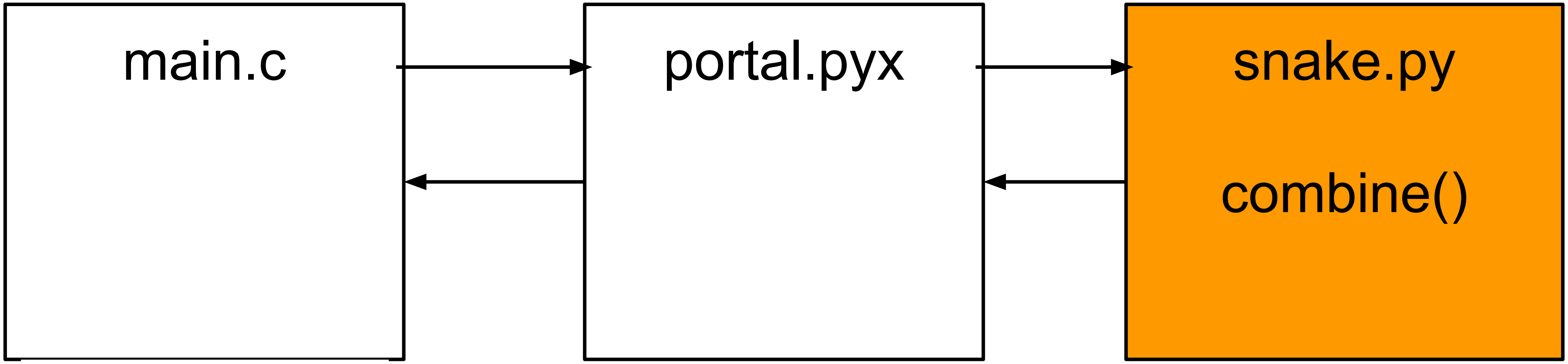
```
$ ./configure
$ make
```

```
$ ./cythonwalkthrough
```

Freely editable

2. Call Python from C

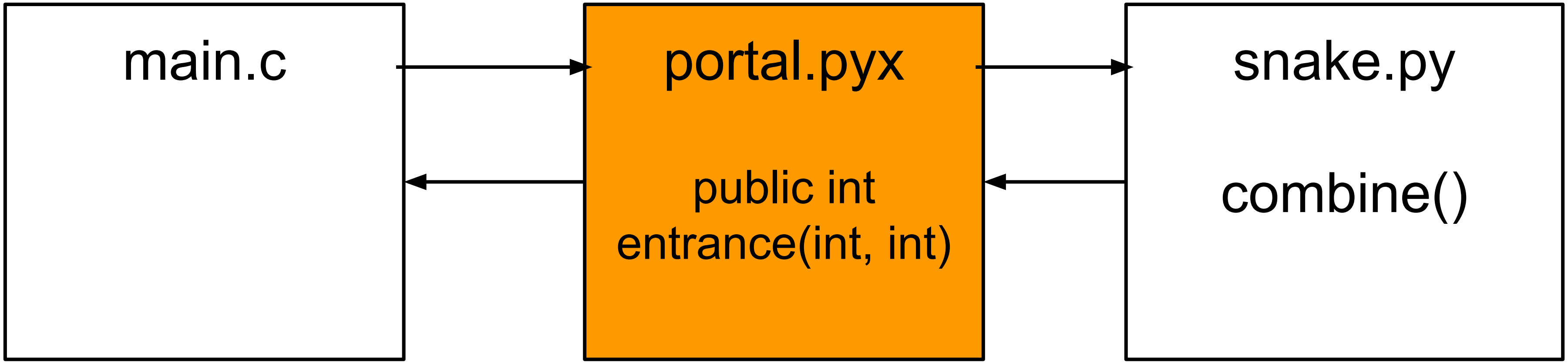
tinyurl.com/cywalk



```
# snake.py:

def combine(a, b):
    print('hello from Python')
    return a ** b
```

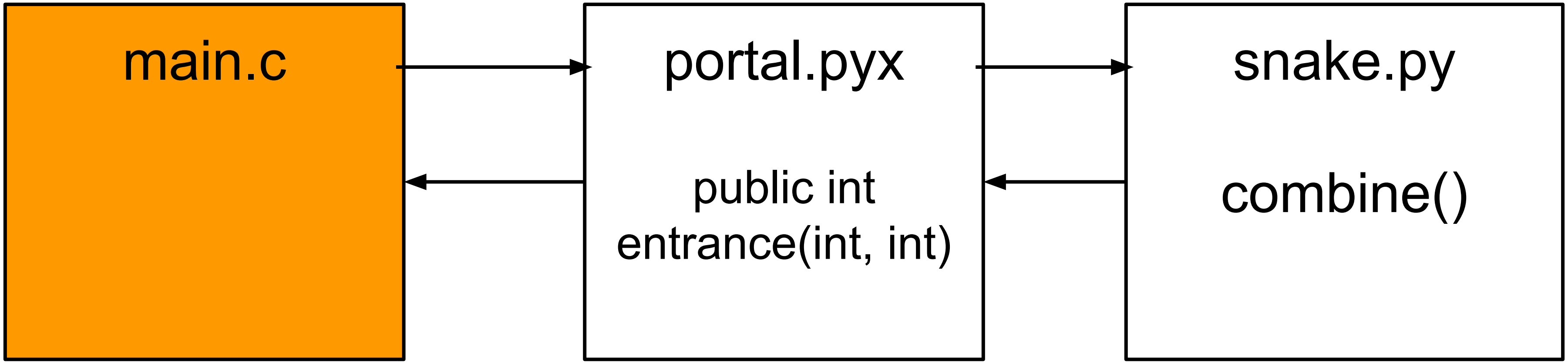
2. Call Python from C

tinyurl.com/cywalk



```
# portal.pyx:

import snake


cdef public int entrance(int a, int b) except? -1:
    print('hello from Cython')
    return snake.combine(a, b)
```

2. Call Python from C



main.c

portal.pyx

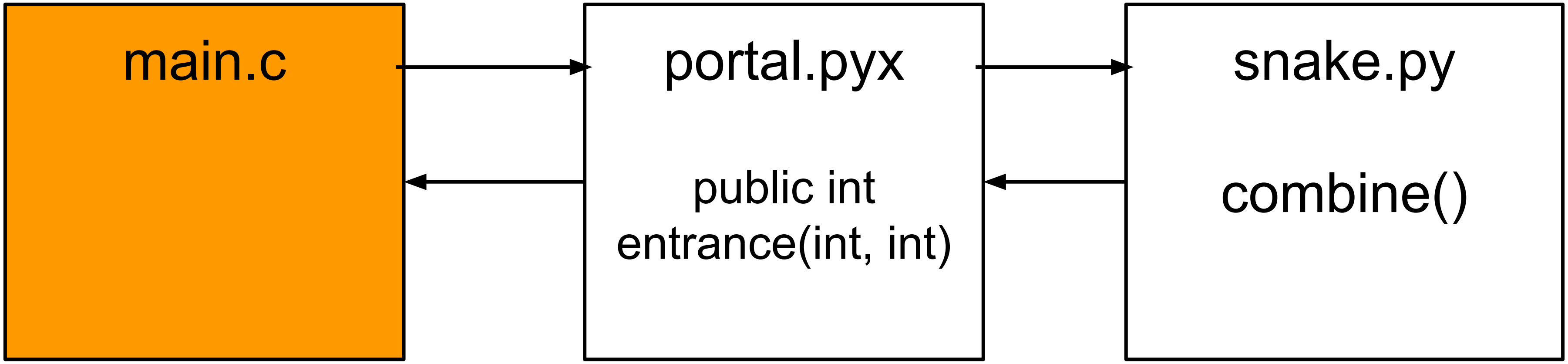
public int
entrance(int, int)

snake.py

combine()

```
/* main.c: */  /* NB: error handling removed */
#include <Python.h>
#include "portal.h"
int main(int argc, char ** argv) {
    int result;
    printf("hello from C\n");
    PyImport_AppendInittab("portal",
                          PyInit_portal);
    Py_Initialize();
```

2. Call Python from C



main.c

portal.pyx

public int entrance(int, int)

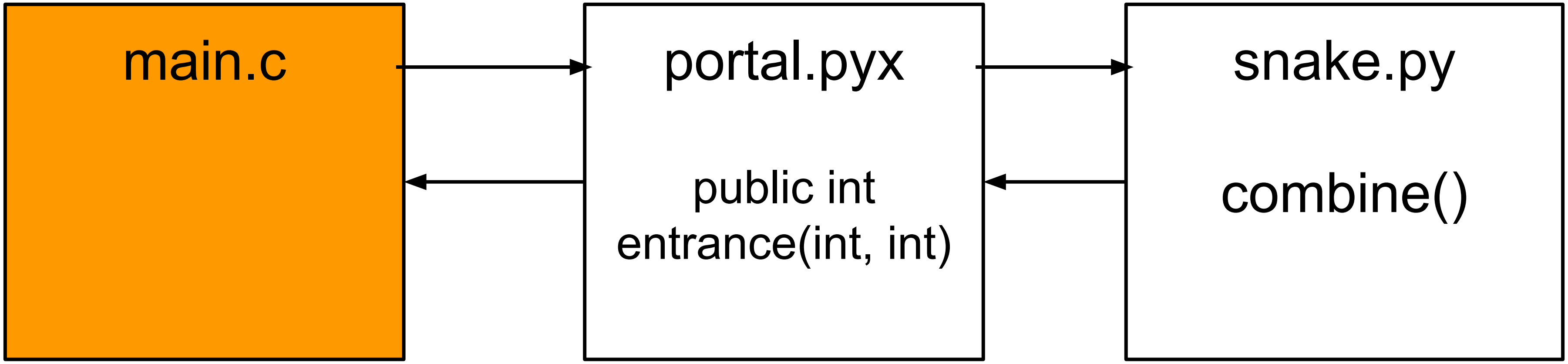
snake.py

combine()

```
/* main.c: */  /* NB: error handling removed */
#include <Python.h>
#include "portal.h"
int main(int argc, char ** argv) {
    int result;
    printf("hello from C\n");
    PyImport_AppendInittab("portal",
                           PyInit_portal);
    Py_Initialize();
```

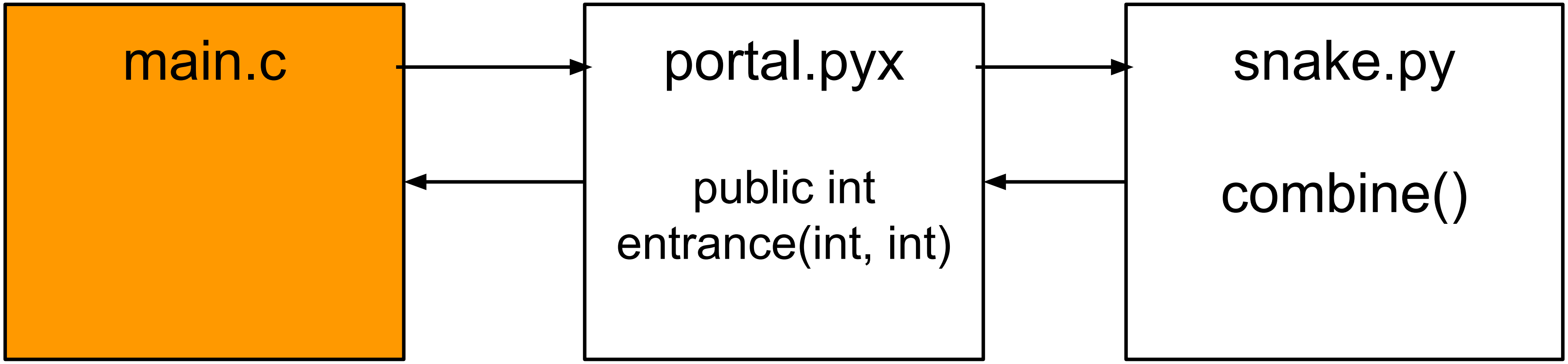
2. Call Python from C

tinyurl.com/cywalk



```c
#include <Python.h>
#include "portal.h"
int main(int argc, char ** argv) {
    int result;
    printf("hello from C\n");
    PyImport_AppendInittab("portal",
                           PyInit_portal);
    Py_Initialize();
    PyImport_ImportModule("portal");
```

2. Call Python from C



main.c

portal.pyx

public int entrance(int, int)

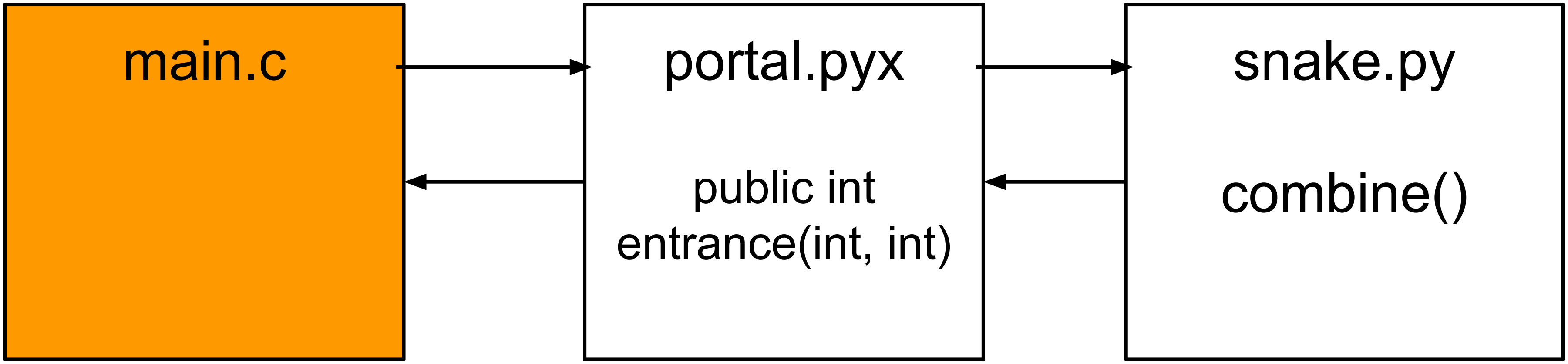
snake.py

combine()

```c
#include "portal.h"
int main(int argc, char ** argv) {
    int result;
    printf("hello from C\n");
    PyImport_AppendInittab("portal",
                           PyInit_portal);
    Py_Initialize();
    PyImport_ImportModule("portal");
    result = entrance(1, 2);  /* defined in
```

2. Call Python from C

tinyurl.com/cywalk



main.c

portal.pyx

public int entrance(int, int)

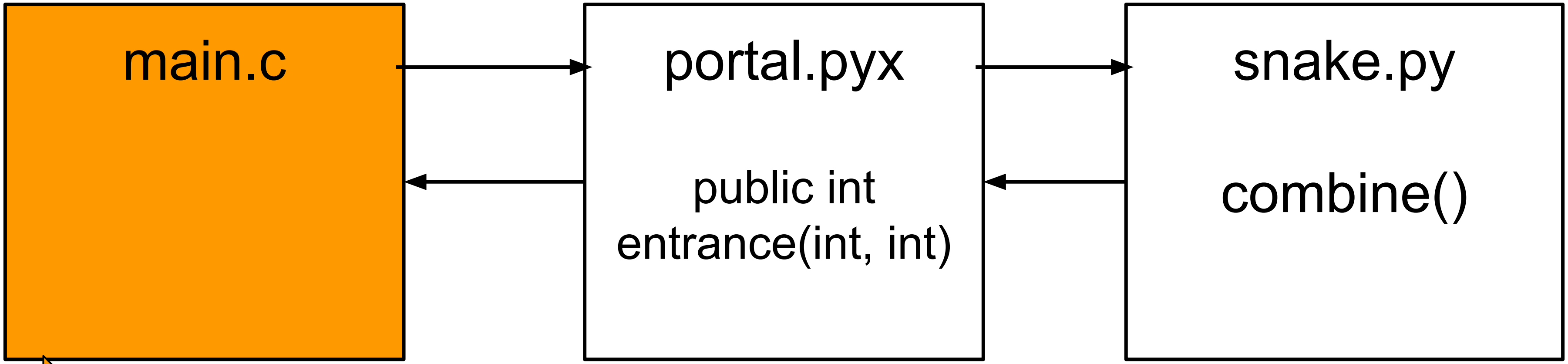
snake.py

combine()

```c
int main(int argc, char ** argv) {
    int result;
    printf("hello from C\n");
    PyImport_AppendInittab("portal",
                           PyInit_portal);
    Py_Initialize();
    PyImport_ImportModule("portal");
    result = entrance(1, 2);  /* portal.h */
    printf("Result is %d\n", result);
```

2. Call Python from C

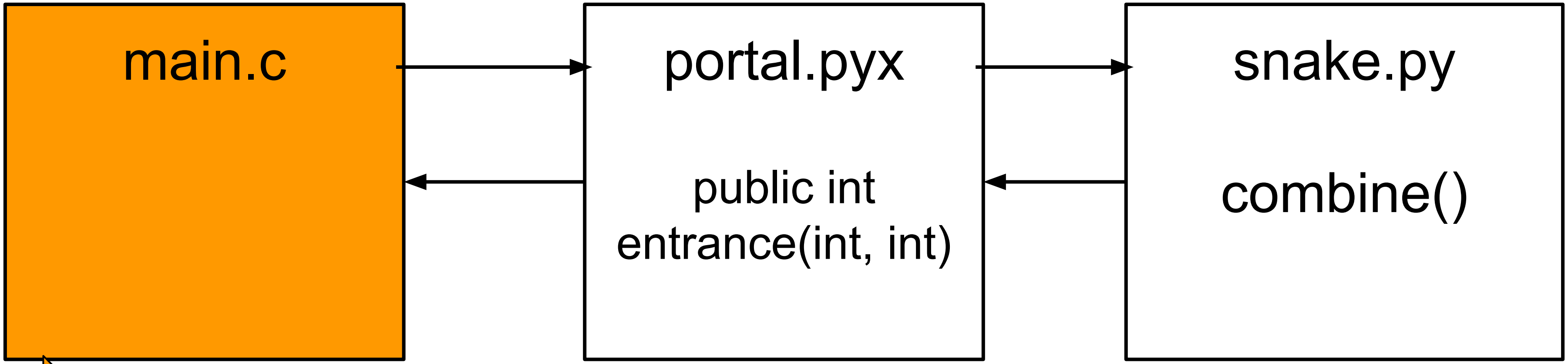
tinyurl.com/cywalk



```
int result;
printf("hello from C\n");
PyImport_AppendInittab("portal",
                       PyInit_portal);
Py_Initialize();
PyImport_ImportModule("portal");
result = entrance(1, 2);  /* portal.h */
printf("Result is %d\n", result);
Py_Finalize();
```

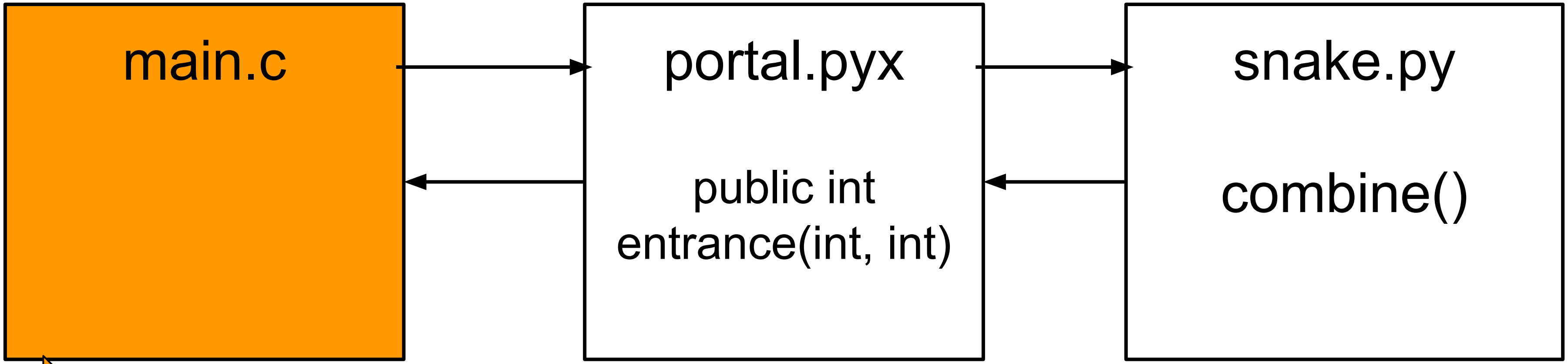
2. Call Python from C

tinyurl.com/cywalk



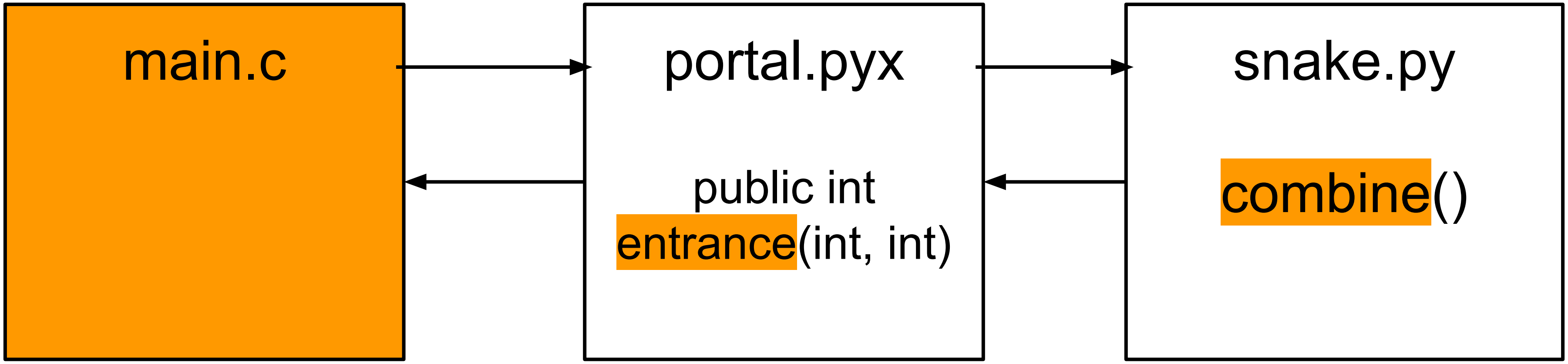
```
    printf("hello from C\n");
    PyImport_AppendInittab("portal",
                           PyInit_portal);
    Py_Initialize();
    PyImport_ImportModule("portal");
    result = entrance(1, 2);  /* portal.h */
    printf("Result is %d\n", result);
    Py_Finalize();
}
```

2. Call Python from C



```
    PyImport_AppendInittab("portal",
                           PyInit_portal);
    Py_Initialize();
    PyImport_ImportModule("portal");
    result = entrance(1, 2);  /* portal.h */
    printf("Result is %d\n", result);
    Py_Finalize();
}
```

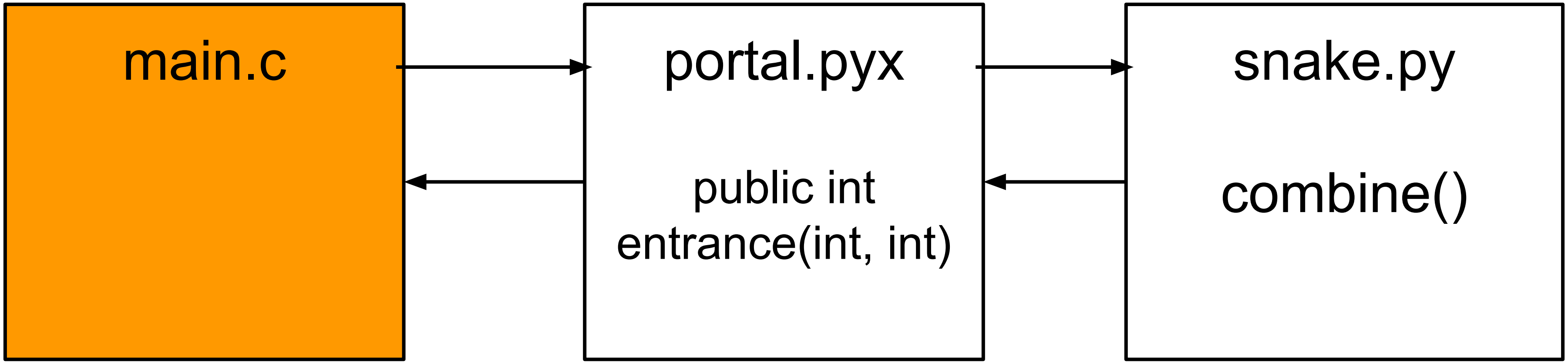
2. Call Python from C



```
    PyImport_AppendInittab("portal",
                           PyInit_portal);
    Py_Initialize();
    PyImport_ImportModule("portal");
    result = entrance(1, 2);  /* portal.h */
    printf("Result is %d\n", result);
    Py_Finalize();
}
```

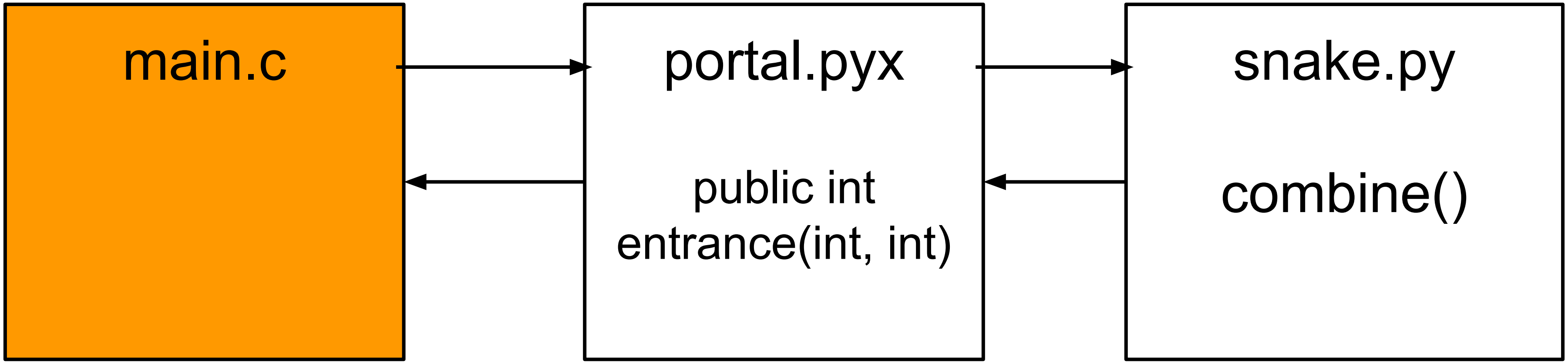
2. Call Python from C

tinyurl.com/cywalk



```
    PyImport_AppendInittab("portal",
                           PyInit_portal);
    Py_Initialize();
    PyImport_ImportModule("portal");
    result = entrance(1, 2);  /* portal.h */
    printf("Result is %d\n", result);
    Py_Finalize();
}
```

2. Call Python from C



```
    PyImport_AppendInittab("portal",
                           PyInit_portal);
    Py_Initialize();
    PyImport_ImportModule("portal");
    result = entrance(1, 2);  /* portal.h */
    printf("Result is %d\n", result);
    Py_Finalize();
}
```

2. Call Python from C

```
$ ./configure

$ make

$ ./cythonwalkthrough

hello from C
hello from Cython
hello from Python
Result is 81
```

Uses for Cython

1. Faster Python code
2. Call Python from C
3. Call C from Python

3. Call C from Python

But why?

- C files included in your Python project
- External libraries
- Anything that provides C/C++ header files

3. Call C from Python

Example:

- Let’s draw a picture with libgd (“graphics draw”)
- https://libgd.github.io/

3. Call C from Python

```
# setup.py:

from setuptools import setup, Extension
from Cython.Build import cythonize

setup(
    # …
    ext_modules=cythonize(
        Extension('*', ['pygd/*.pyx'],
                  libraries=['gd'])),
)
```

3. Call C from Python

```
# pygd/gd.pyx:

cdef extern from "gd.h":
```

- Definitions so Cython can generate correct C code
- Should match (reasonably) with actual C headers

3. Call C from Python

```
# pygd/gd.pyx:

cdef extern from "gd.h":
    ctypedef struct gdImage
    ctypedef struct gdPoint:
        int x
        int y
```

3. Call C from Python

```
# pygd/gd.pyx:

cdef extern from "gd.h":
    ctypedef struct gdImage
    ctypedef struct gdPoint:
        int x
        int y
```

Not interested in contents

3. Call C from Python

```
# pygd/gd.pyx:

cdef extern from "gd.h":
    ctypedef struct gdImage
    ctypedef struct gdPoint:
        int x
        int y
```

Only need to include members we use

3. Call C from Python

tinyurl.com/cywalk

```
# pygd/gd.pyx:

cdef extern from "gd.h":
    ctypedef struct gdImage
    ctypedef struct gdPoint:
        int x
        int y
    cdef gdImage * gdImageCreate(
        int sx, int sy)
    cdef void gdImageDestroy(
        gdImage * im)
    cdef void gdImageFilledPolygon(
        gdImage * im, gdPoint *p, int n, int c)
    # …
```

3. Call C from Python

```
# pygd/gd.pyx:

cdef extern from "gd.h":
    ctypedef struct gdImage
    ctypedef struct gdPoint:
        int x
        int y
    cdef gdImage * gdImageCreate(
        int sx, int sy)
    cdef void gdImageDestroy(
        gdImage * im)
    cdef void gdImageFilledPolygon(
        gdImage * im, gdPoint *p, int n, int c)
    # …
```

3. Call C from Python

```
# pygd/gd.pyx:

cdef extern from "gd.h":
    ctypedef struct gdImage
    ctypedef struct gdPoint:
        int x
        int y
    cdef gdImage * gdImageCreate(
        int sx, int sy)
    cdef void gdImageDestroy(
        gdImage * self?)
    cdef void gdImageFilledPolygon(
        gdImage * self?, gdPoint *p, int n, int
    # …
```

3. Call C from Python

```
# pygd/gd.pyx:

cdef class Image:
    cdef gdImage * cobj
```

3. Call C from Python

tinyurl.com/cywalk

```
# pygd/gd.pyx:

cdef class Image:
    cdef gdImage * cobj

    def __cinit__(self,
                  int size_x, int size_y,
                  bgcolor=(255, 255, 255)):
        self.cobj = gdImageCreate(size_x,
                                  size_y)
        if self.cobj is NULL:
            raise MemoryError(
                'Unable to create gdImage')
```

3. Call C from Python

```
# pygd/gd.pyx:

cdef class Image:
    cdef gdImage * cobj

    def __cinit__(self, int size_x, …):
        # …

    def __dealloc__(self):
        if self.cobj:
            gdImageDestroy(self.cobj)
```

3. Call C from Python

```
# pygd/gd.pyx:

cdef class Image:
    cdef gdImage * cobj

    def __cinit__(self, int size_x, …):
        # …

    def __dealloc__(self):
        # …

    def filled_polygon(self,
                       *points,
                       color=(0, 0, 0)):
        # points: eg [(1,2), (4,6), (9,3)]
```

3. Call C from Python

```
def filled_polygon(self, *points, color=…):
    cdef gdPoint * cpoints = <gdPoint *>calloc(
        len(points), sizeof(gdPoint))
```

3. Call C from Python

tinyurl.com/cywalk

```
def filled_polygon(self, *points, color=…):
    cdef gdPoint * cpoints = <gdPoint *>calloc(
        len(points), sizeof(gdPoint))
    if cpoints is NULL:
        raise MemoryError()
```

3. Call C from Python

```
def filled_polygon(self, *points, color=…):
    cdef gdPoint * cpoints = <gdPoint *>calloc(
        len(points), sizeof(gdPoint))
    if cpoints is NULL:
        raise MemoryError()
    try:



    finally:
        free(cpoints)
```

3. Call C from Python

tinyurl.com/cywalk

```
def filled_polygon(self, *points, color=…):
    cdef gdPoint * cpoints = <gdPoint *>calloc(
        len(points), sizeof(gdPoint))
    if cpoints is NULL:
        raise MemoryError()
    try:
        for i, point in enumerate(points):
            cpoints[i].x = point[0]
            cpoints[i].y = point[1]


    finally:
        free(cpoints)
```

3. Call C from Python

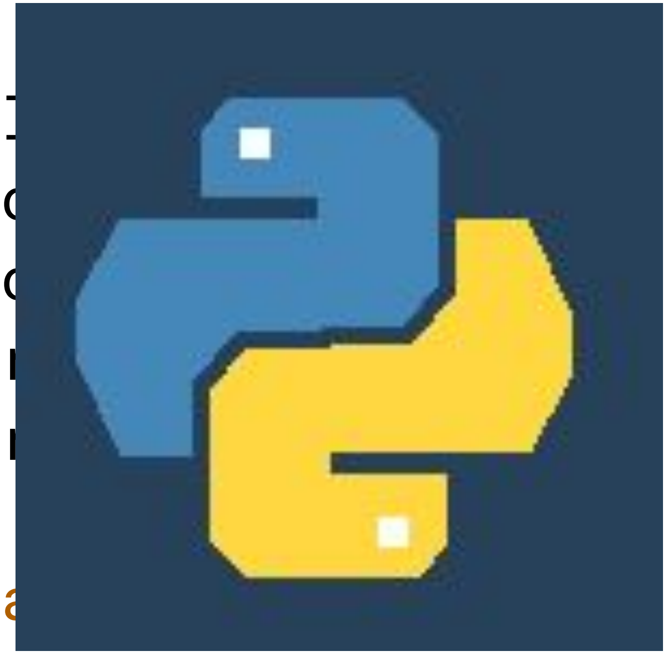
```
def filled_polygon(self, *points, color=…):
    cdef gdPoint * cpoints = <gdPoint *>calloc(
        len(points), sizeof(gdPoint))
    if cpoints is NULL:
        raise MemoryError()
    try:
        for i, point in enumerate(points):
            cpoints[i].x = point[0]
            cpoints[i].y = point[1]
        gdImageFilledPolygon(
            self.cobj, cpoints, len(points), …)
    finally:
        free(cpoints)
```

3. Call C from Python

```
# example.py:

import pygd

img = pygd.Image(150, 150, bgcolor=(40, 65, 90))
img.filled_polygon(*s_blue, color=(70, 135, 185))
img.filled_polygon(*s_yellow, color=(255, 215, 65))
img.filled_polygon(*eye_1, color=(255, 255, 255))
img.filled_polygon(*eye_2, color=(255, 255, 255))

with open('example.png', 'wb') as f:
    img.dump(f)
```

3. Call C from Python

```
# example.py:

import pygd

img = pygd.Image(150, 150, bgcol
img.filled_polygon(*s_blue, colo
img.filled_polygon(*s_yellow, co                      )
img.filled_polygon(*eye_1, colo
img.filled_polygon(*eye_2, colo

with open('example.png', 'wb') a
    img.dump(f)
```



Final thoughts

- Mix Python and C
- Alternatives
  - PyPy
  - cffi
  - ctypes

tinyurl.com/cywalk

Thank you

Nick Murdoch
@nickmurdoch

Code and slides:

github.com/flexo/
cythonwalkthrough