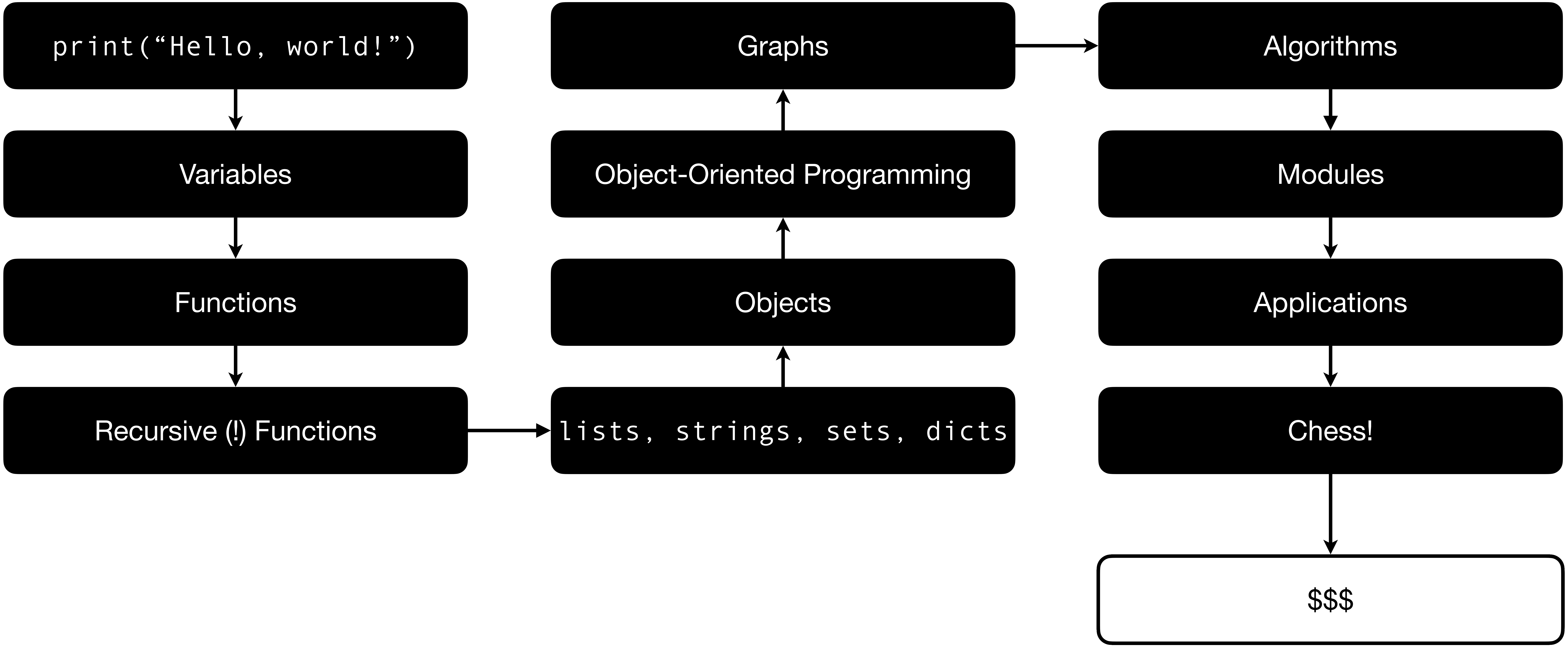
CMSC 14300

you’re in the right place

Byron Zhong

Your Journey in CS So Far



`print(“Hello, world!”)`

Variables

Functions

Recursive (!) Functions

`lists, strings, sets, dicts`

Objects

Object-Oriented Programming

Graphs

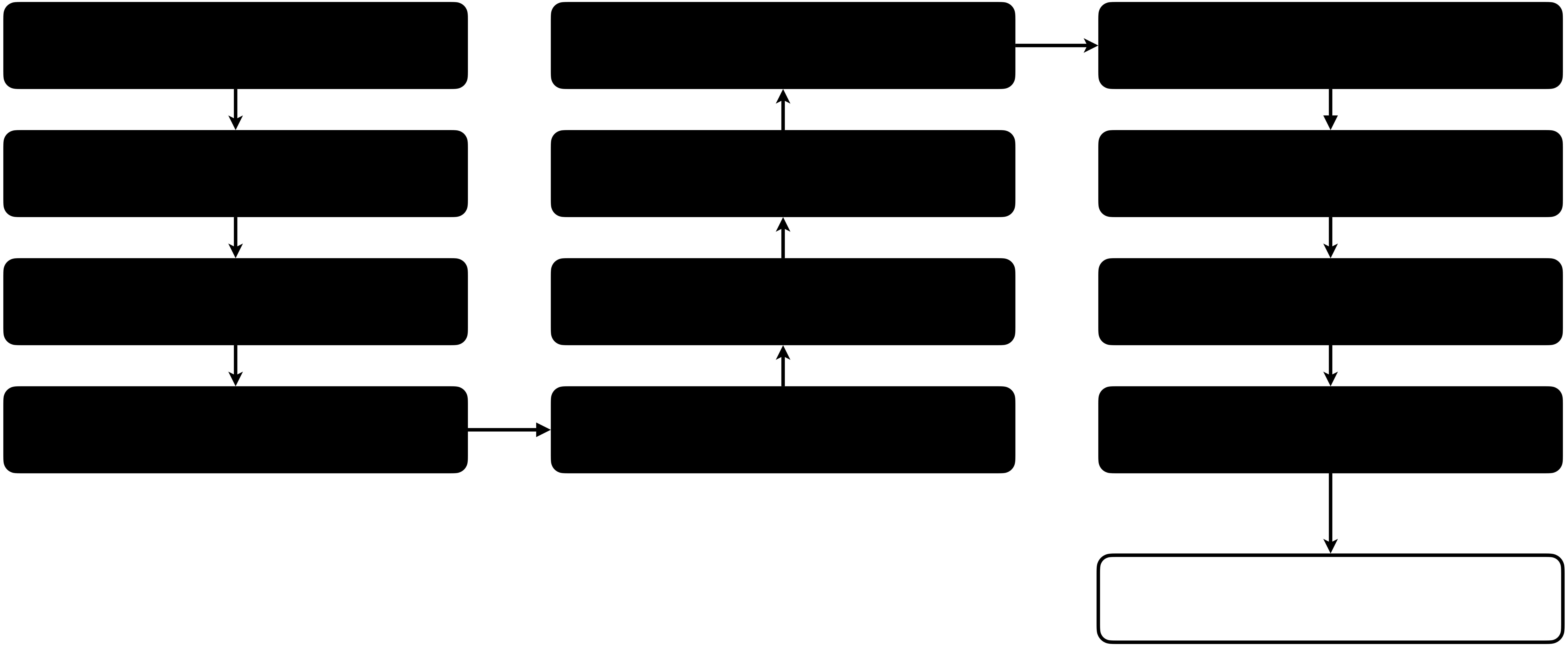
Algorithms

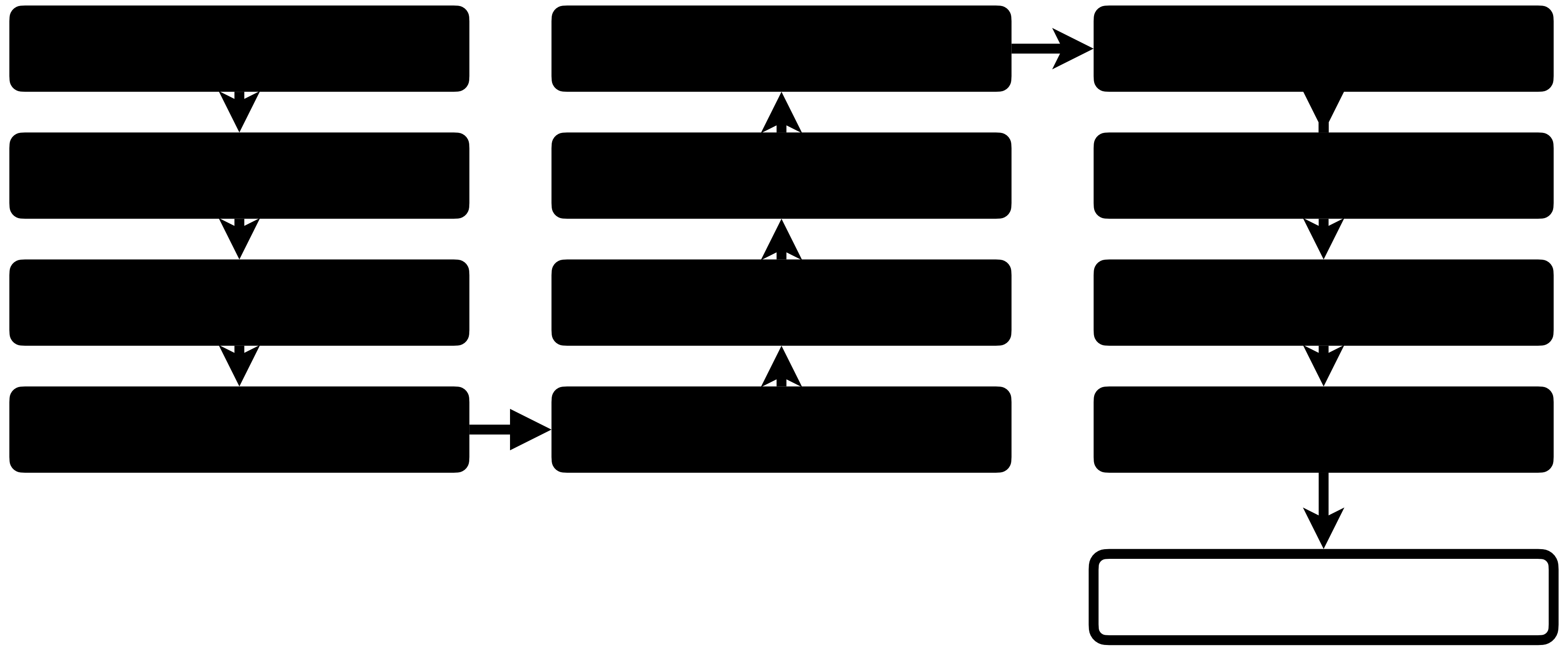
Modules

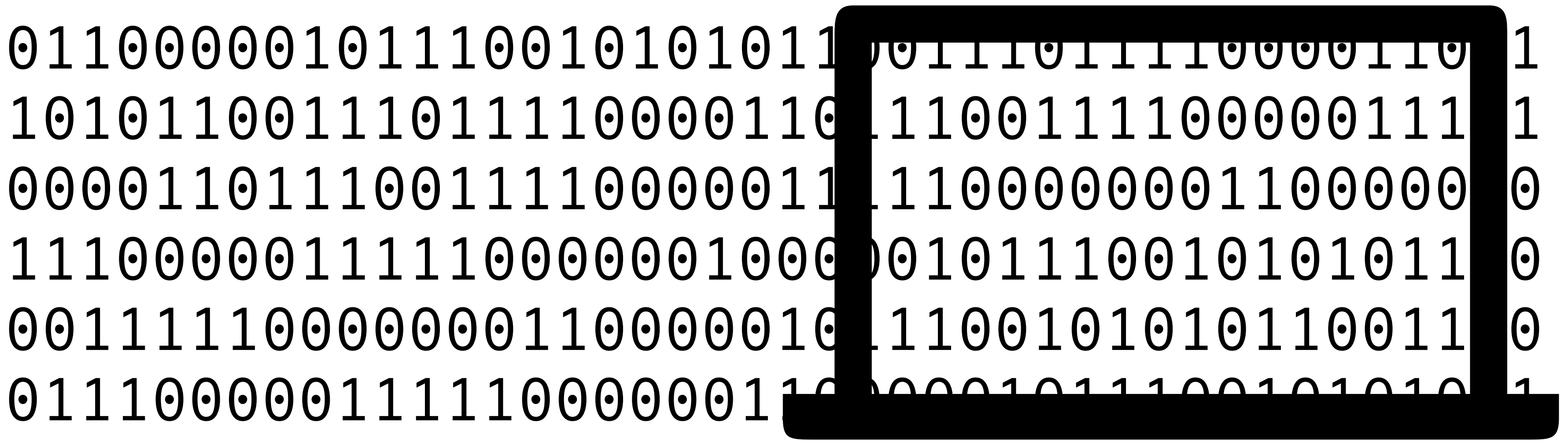
Applications

Chess!

$$$







But what really is a variable?

… what really is a function?

… what does CPU do exactly?

… how does anything work?

CMSC 14300
Systems Programming I

Introduction

Byron Zhong

Today's Plan

1. Administrivia
2. A whirlwind tour of C
3. Terminal and coding environment

Administrivia

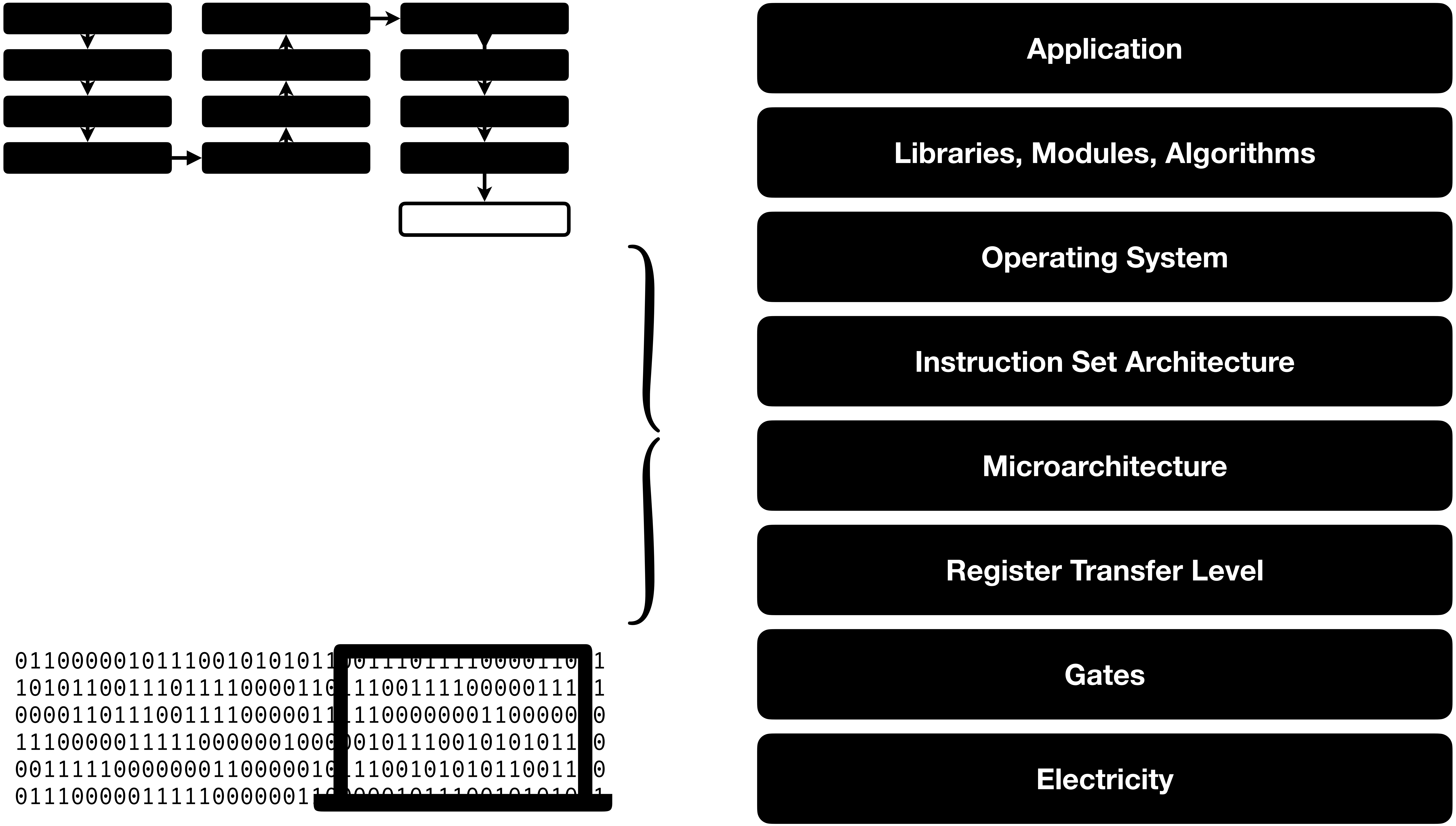
Staff

1. Me
2. Víctor (Course Coordinator)

Administrivia

143’s goals

1. Develop a deep understanding of how computers work
2. Transition from introductory programming to programming as a professional



Application

Libraries, Modules, Algorithms

Operating System

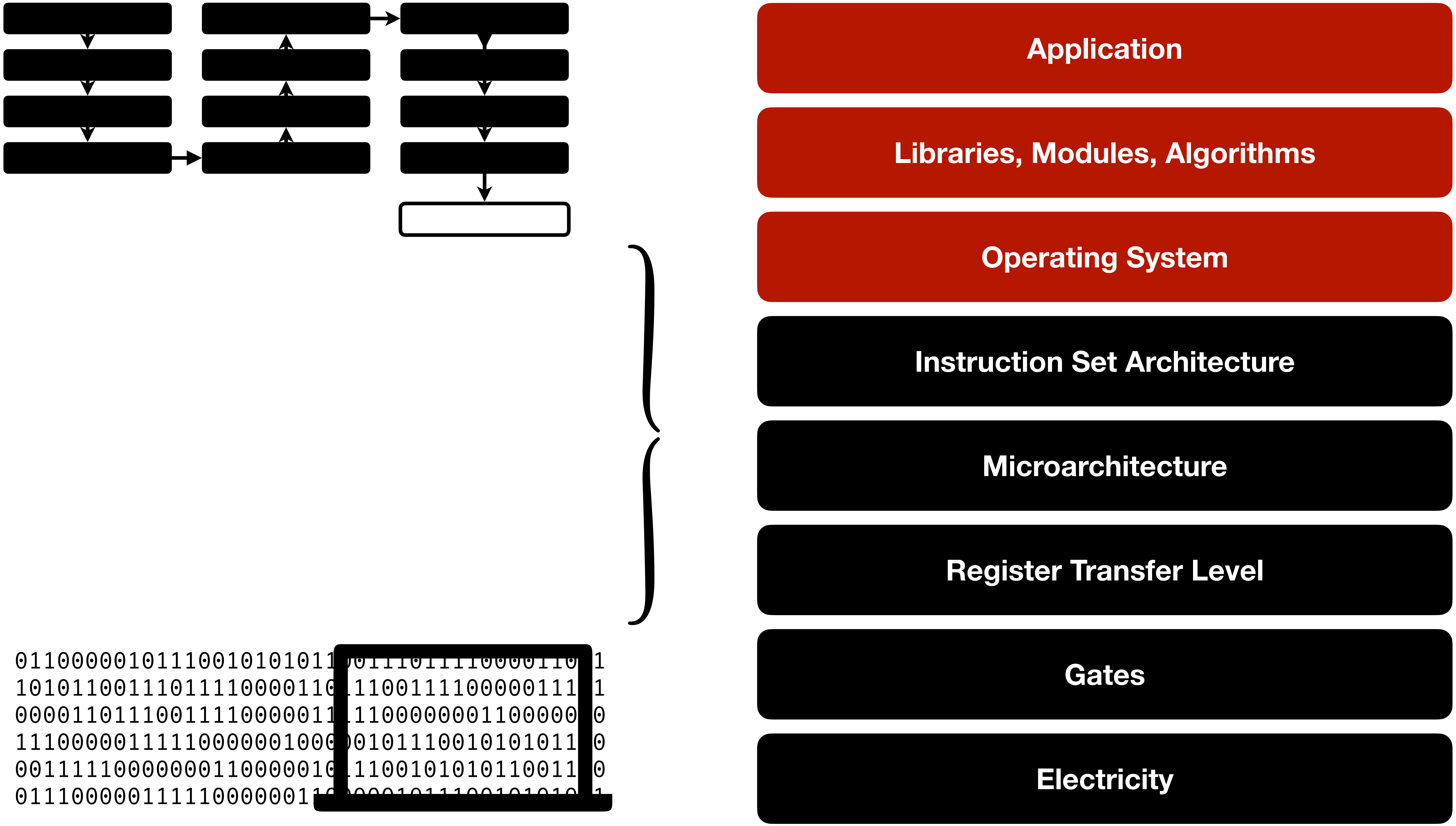
Instruction Set Architecture

Microarchitecture

Register Transfer Level

Gates

Electricity



Application

Libraries, Modules, Algorithms

Operating System

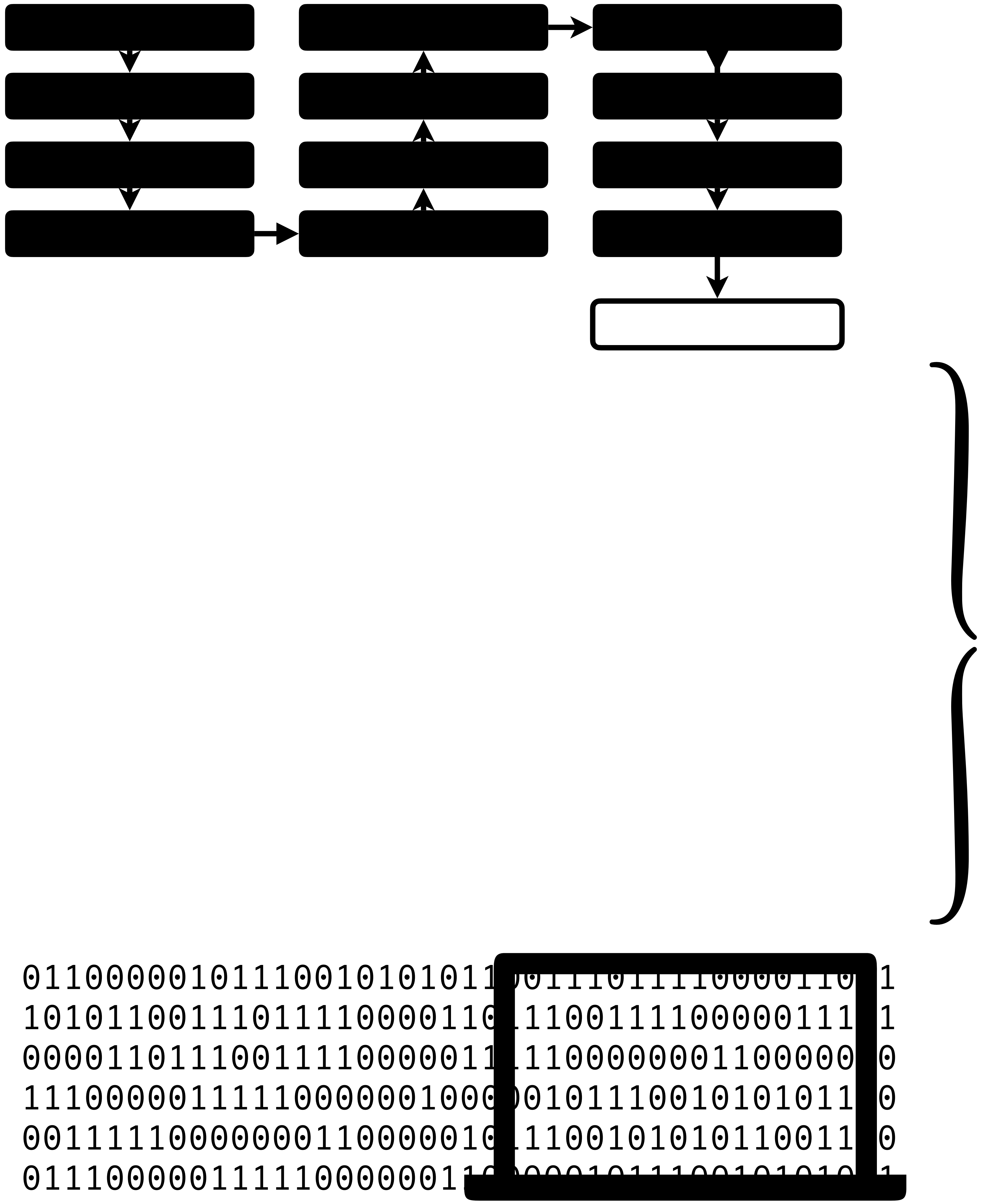
Instruction Set Architecture

Microarchitecture

Register Transfer Level

Gates

Electricity



Application

Libraries, Modules, Algorithms

Operating System

Instruction Set Architecture

Microarchitecture

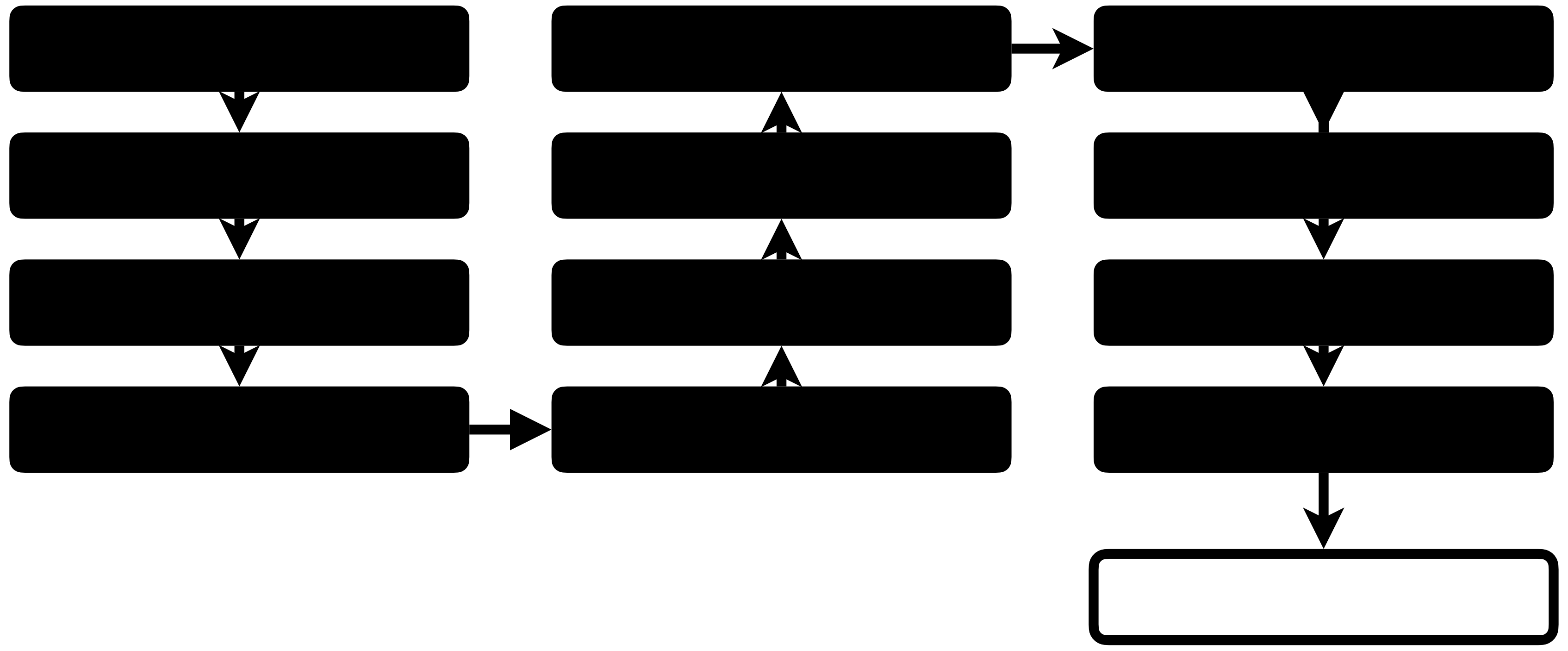
Register Transfer Level

Gates

Electricity

143

144

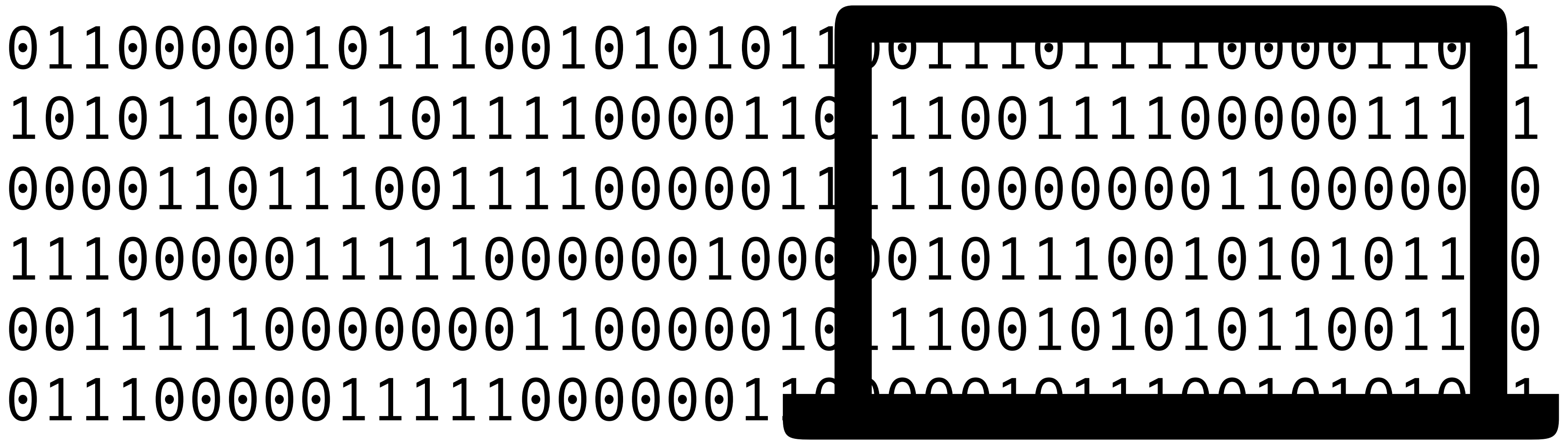


- Application
- Libraries, Modules, Algorithms
- Operating System
- Instruction Set Architecture
- Microarchitecture
- Register Transfer Level
- Gates
- Electricity

143

144

EE



Administrivia

Grading

Homework	60%
Quiz	15%
Final	25%

Administrivia

Homework

- Weekly assignments, starting today
- Due every **Monday 11:59:59pm** (generally)
- Late policy:
  - 4,320 minutes of late time
  - every minute past, 0.003% penalty
  - emergency, contact your advisor

Administrivia

Quiz and Exam

- Quiz: Monday, July 10, 6:00pm-7:20pm.
- Exam: Thursday, August 3, 6:00pm-7:20pm.

Administrivia

HELP!

- Resource page on course website
- Ed
  - Details: don’t just say “X doesn’t work”
  - No screenshots or giant code block
- Office hours:
  - TBD, do the survey
- Email me

Administrivia

Advice

- Practice, practice, practice…
- Start early
  - coding is fun but fighting for hours is not
- Write a little, test a little
  - you will make mistakes, make them easy to find
- Let me know your feedback; I’m still experimenting

Administrivia

Academic Dishonesty

- Do not copy code …it’s very obvious
- Do not show your solution
  - … online
  - … to others
  - use private Ed post if you’re unsure
- Discuss concept ok, code no
- Document your collaboration

Administrivia

Accessibility

- Contact SDS soon
- Víctor

A Whirlwind Tour of C

Why C?

- C is the *lingua franca* of computer programming
  - unix is written in C
  - many, many languages have C-like syntax
- C helps you understand how computers work
  - ~~to use C, you *have to* understand how computers work~~
- C is very fast, good for serious applications

The Anatomy of C

```c
#include <stdio.h>

void say_hello(void);

int main(void)
{
        say_hello();
        return 0;
}

void say_hello(void)
{
        printf("Hello, world!\n");
}
```

The Anatomy of C

```
#include <stdio.h>
```

<— Directives

```
void say_hello(void);
```

<— Declarations

```
int main(void)
{
        say_hello();
        return 0;
}
```

<— Declarations

```
void say_hello(void)
{
        printf("Hello, world!\n");
}
```

<— Declarations

The Anatomy of C

- A C program is a list of *declarations* and *directives*.
- Declarations tell us how to interpret *names*.
  - `say_hello` and `main` are functions.
- Directives (beginning with `#`) tell compiler to do stuff.
  - `#include <stdio.h>` tells compiler to import the standard I/O library.*

The Anatomy of C

```c
#include <stdio.h>

void say_hello(void);

int main(void)
{
        say_hello();
        return 0;
}

void say_hello(void)
{
        printf("Hello, world!\n");
}
```

- A special declaration is called `main`
- No top-level code — all code is in some functions, which are called by `main`, directly or indirectly
- Functions can call everything declared *above*, including itself

The Anatomy of C

```c
#include <stdio.h>

void say_hello(void);

int main(void)
{
        say_hello();
        return 0;
}

void say_hello(void)
{
        printf("Hello, world!\n");
}
```

- A function *signature* specifies its argument types and return types — write `void` if none
- A function is *declared* if the signature is followed by `;`
- A function is *defined* if it is followed by a block `{ .. }`

The Anatomy of C

```
#include <stdio.h>

int factorial(int x);  <— Argument type: int
^^^——— Return type: int
int main(void)
{
        int a;
        a = 20;
        int fact_a = factorial(a);
        printf("factorial(%d) = %d\n", a, fact_a);

        return 0;
}

int factorial(int x)
{
        if (x == 0) {
                return 1;
        }

        return x * factorial(x - 1);
}
```

The Anatomy of C

```
int main(void)
{
        int a;                             <-- tell compiler variable a of type int exists
        a = 20;
        int fact_a = factorial(a);
        printf("factorial(%d) = %d\n", a, fact_a);

        return 0;
}
```

- A *block* `{  ..  }` consists of a list of *statements.* Each statement ends with `;`
- A statement can *declare* a variable

The Anatomy of C

```
int main(void)
{
        int a;
        a = 20;                         <-- write 20 to a
        int fact_a = factorial(a);
        printf("factorial(%d) = %d\n", a, fact_a);

        return 0;
}
```

- A *block* `{ .. }` consists of a list of *statements.* Each statement ends with `;`
- A statement can *declare* a variable
- *assign* a variable

The Anatomy of C

```
int main(void)
{
    int a;
    a = 20;
    int fact_a = factorial(a);     <-- fact_a exists, call function, write result
    printf("factorial(%d) = %d\n", a, fact_a);

    return 0;
}
```

- A *block* `{ .. }` consists of a list of *statements.* Each statement ends with `;`
- A statement can *declare* a variable
- *assign* a variable

The Anatomy of C

```
int main(void)
{
        int a;
        a = 20;
        int fact_a = factorial(a);
        printf("factorial(%d) = %d\n", a, fact_a);
                                            ^-- call a function to print
        return 0;
}
```

- A *block* `{ .. }` consists of a list of *statements.* Each statement ends with `;`
- A statement can *declare* a variable
- *assign* a variable
- *call* a function

The Anatomy of C

```
int main(void)
{
        int a;
        a = 20;
        int fact_a = factorial(a);
        printf("factorial(%d) = %d\n", a, fact_a);

        return 0;                                    <-- exit main
}
```

- A *block* `{ .. }` consists of a list of *statements.* Each statement ends with `;`
- A statement can *declare* a variable
- *assign* a variable
- *call* a function
- …

Control-flow Compared

If

```c
if (x == 0) {
        do_stuff();
} else if (x == 1) {
        do_stuff()
} else {
        do_something_else();
}
```

C

```python
if x == 0:
    do_stuff()
elif x == 1:
    do_stuff()
else:
    do_something_else()
```

Python

Control-flow Compared

While

```
while (x != 0) {
        do_stuff();
}
```

C

```
while x == 0:
    do_stuff()
```

Python

Control-flow Compared

For

```
for (int i = 0; i < 200; i += 1) {
        do_stuff(i);
}
```

↕ Equivalent

```
int i = 0;
while (i < 200) {
        do_stuff(i);
        i += 1;
}
```

C

```
for x in iterator:
    do_stuff(x)
```

↕ Equivalent

```
x = the first element
while x.has_more():
    do_stuff(x)
    x = next(x)
```

Python

Control-flow Compared

Return, Continue, Break

```
while (x != 0) {
        return x;
        continue;
        break;
}
```

```
while x != 0:
    return x
    continue
    break
```

Boolean Compared

- C doesn't have Boolean (!)
  - any non-zero value is considered `true`, and zero is `false`
  - e.g. `if (42) { .. } –> if (true) { .. }`

C	Python
`x && y`	`x and y`
`x || y`	`x or y`
`!x`	`not x`

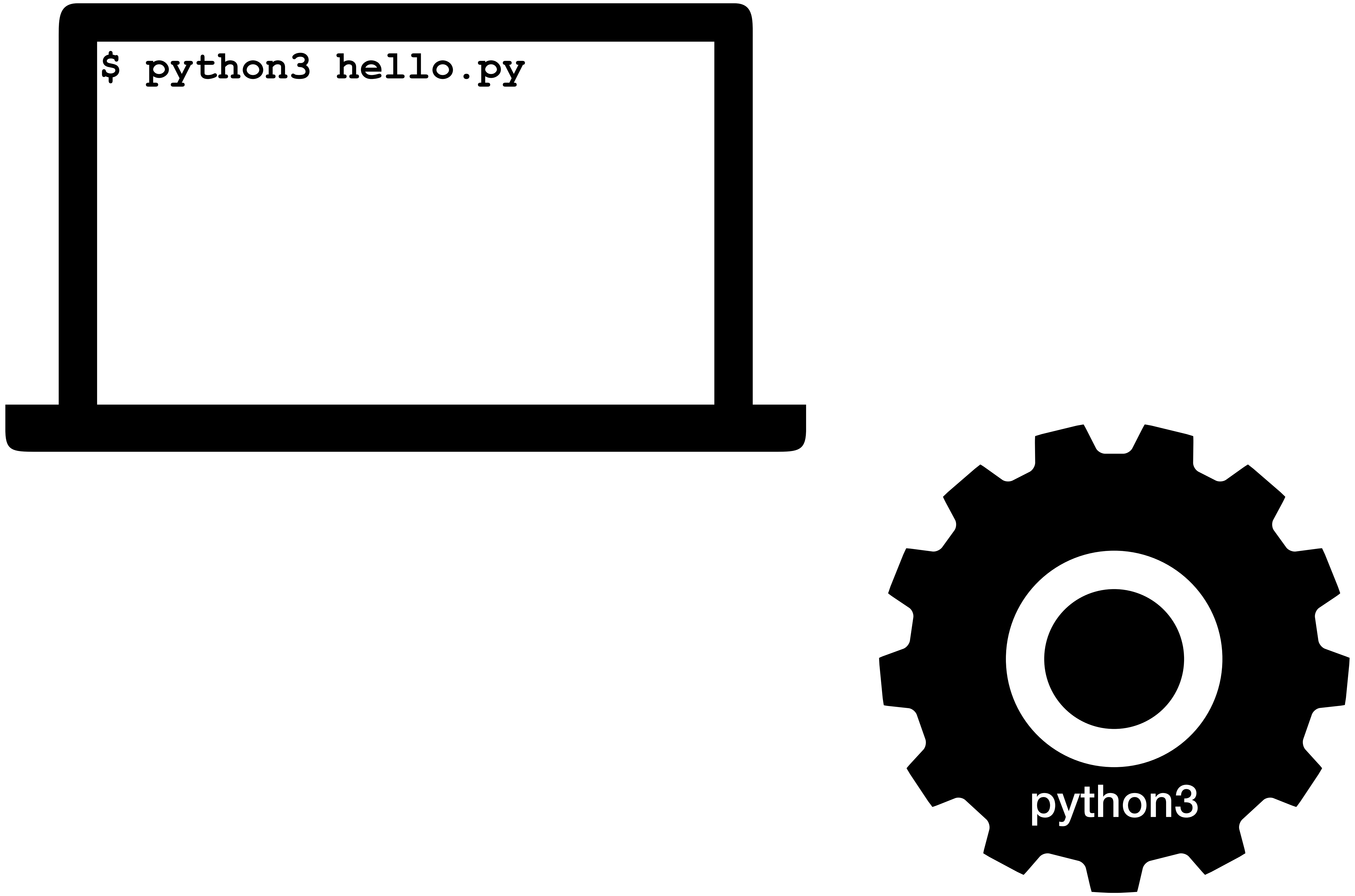
How to Run C

Review: how does Python work?

```
$ python3 hello.py
```

How to Run C

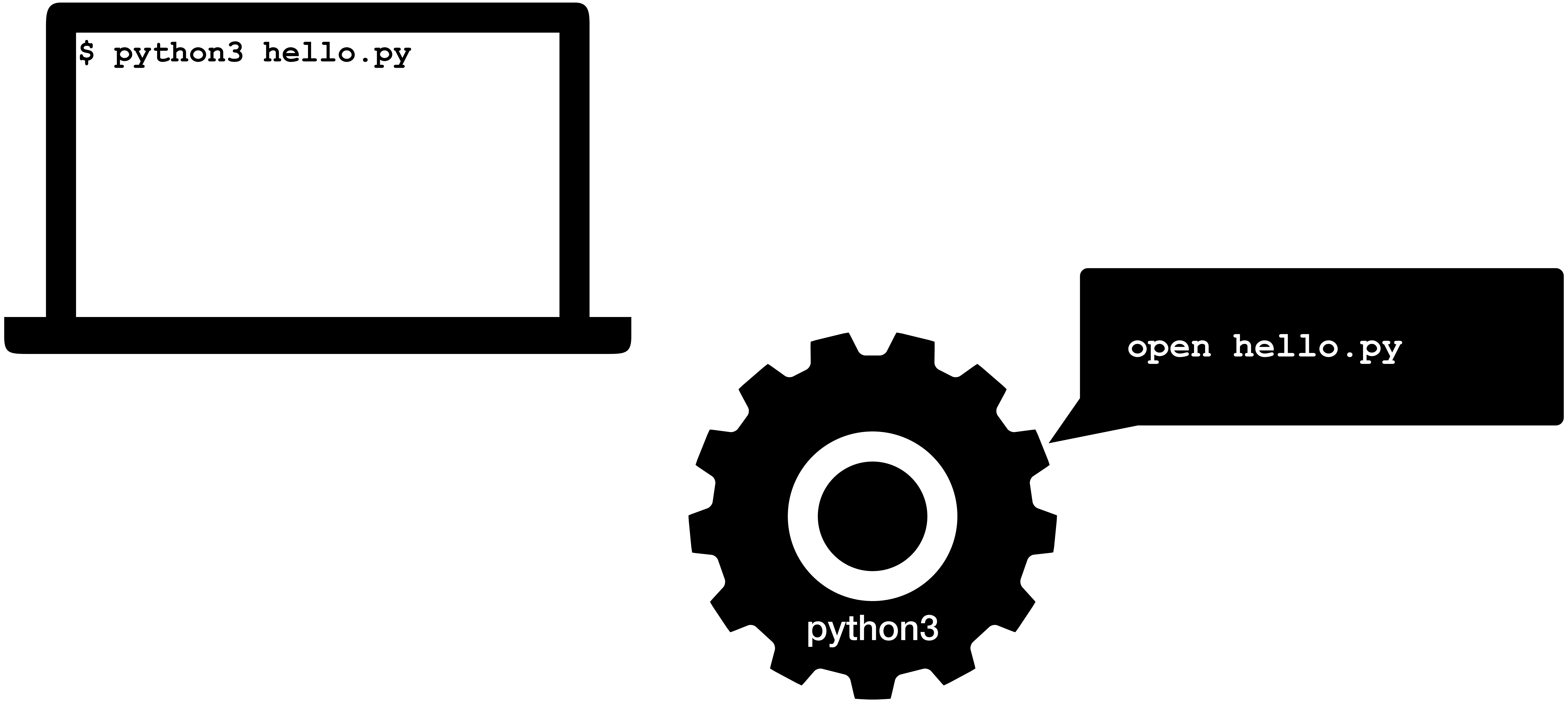
Review: how does Python work?



```
$ python3 hello.py
```

How to Run C

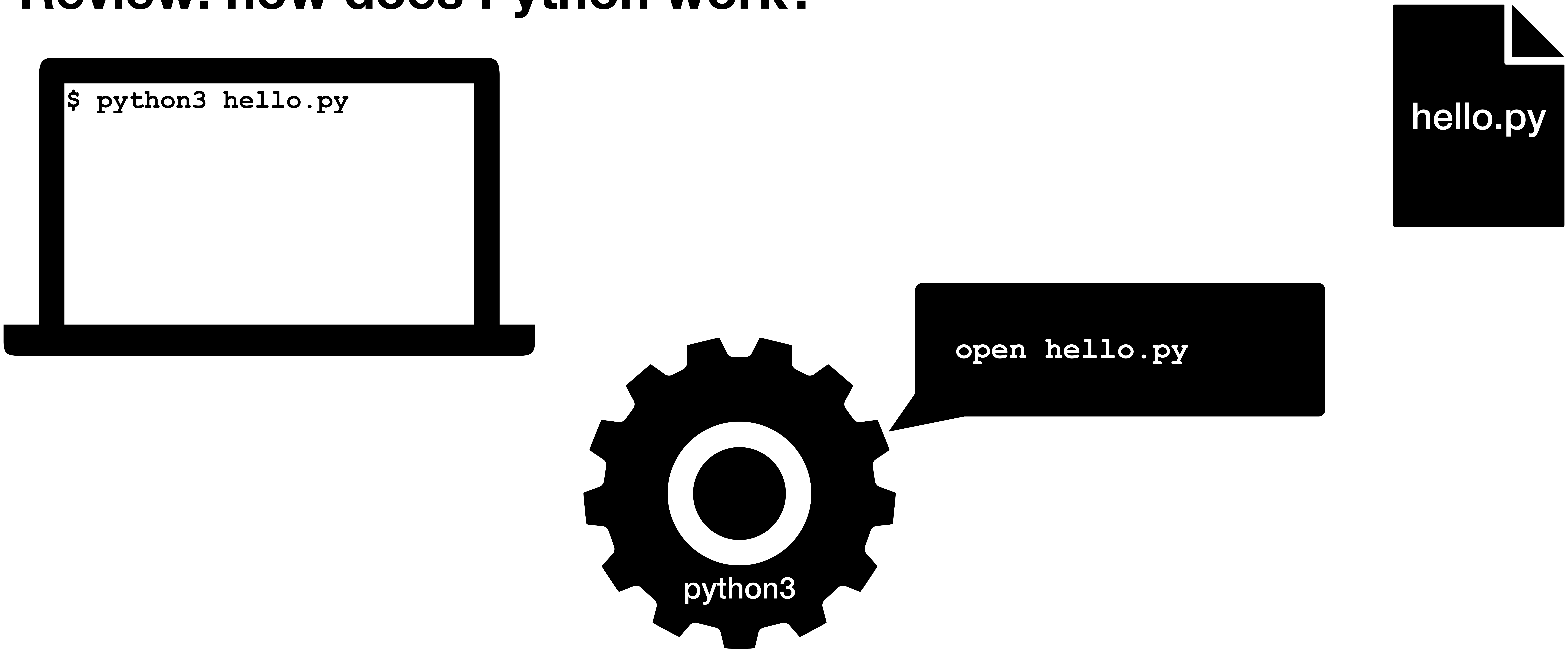
Review: how does Python work?



```
$ python3 hello.py
```

How to Run C

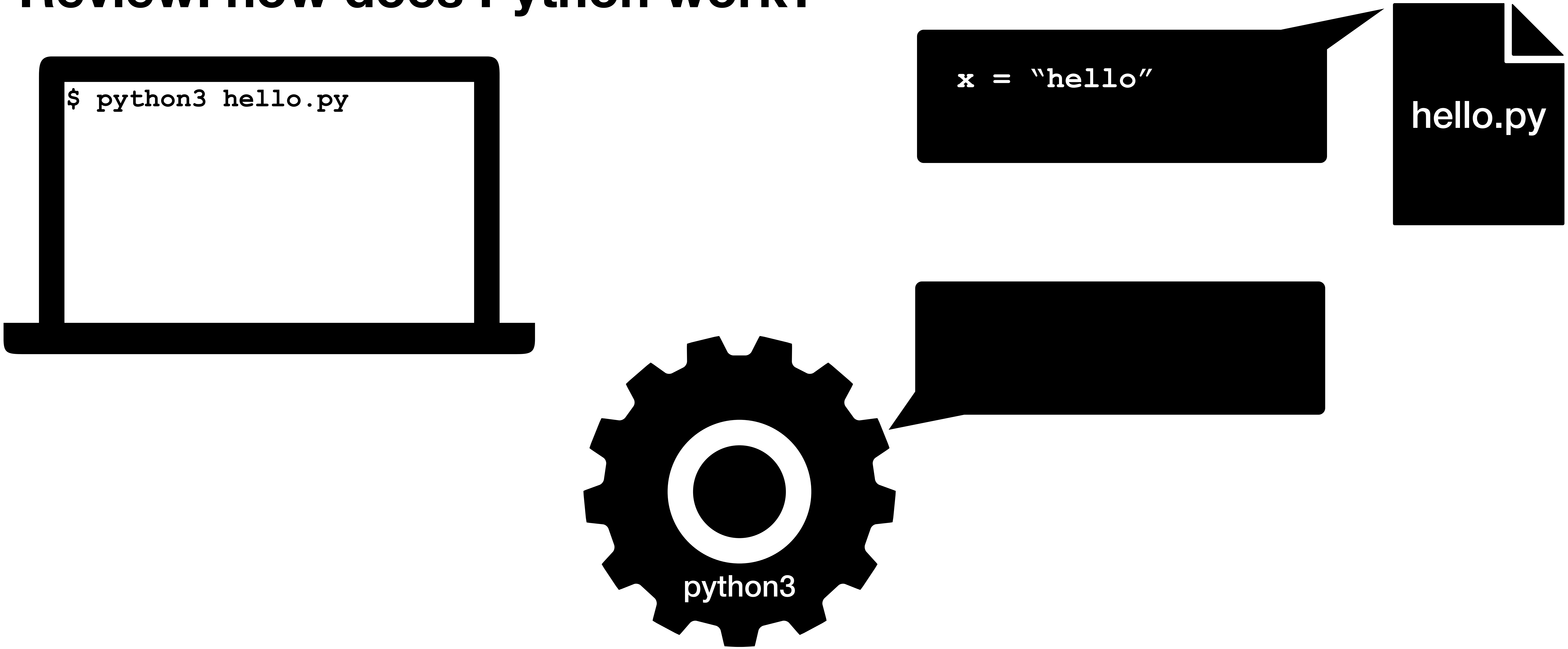
Review: how does Python work?



```
$ python3 hello.py
```

How to Run C

Review: how does Python work?

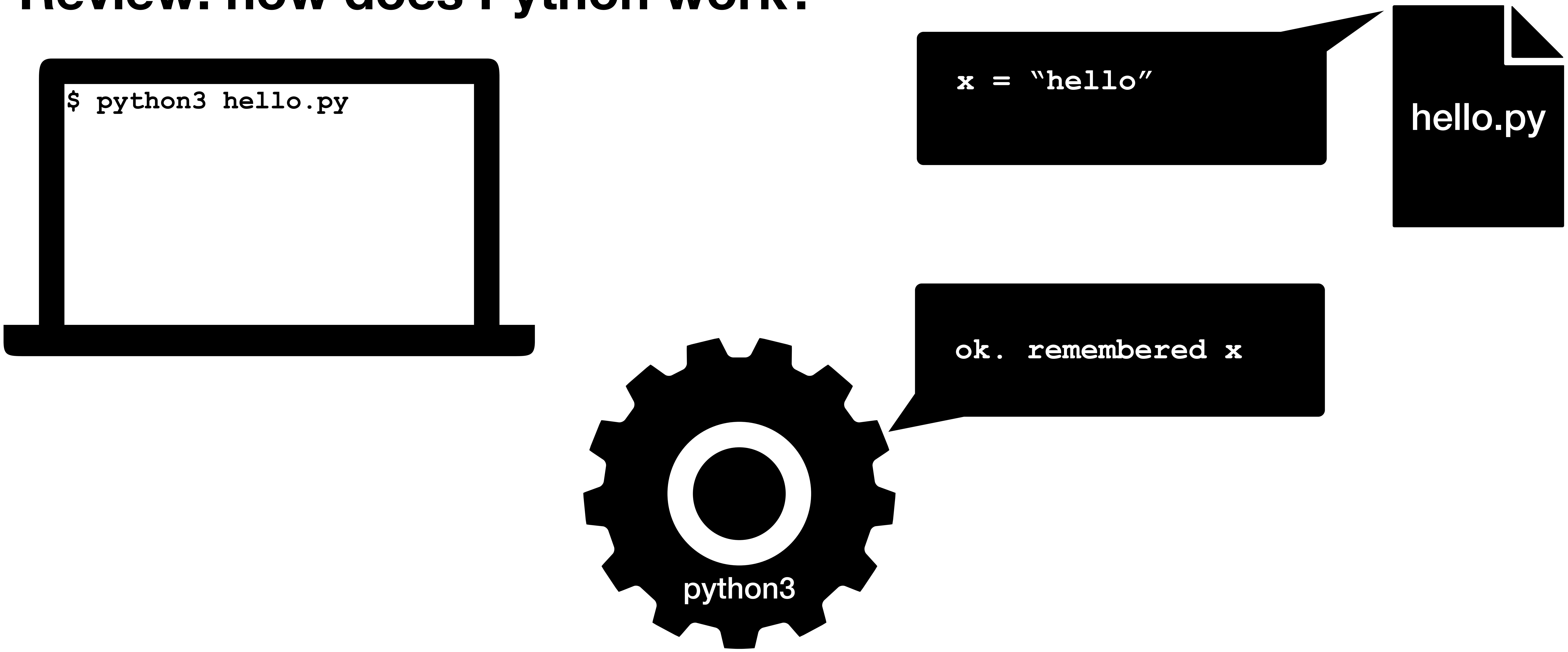


```
$ python3 hello.py
```

```
x = “hello”
```

How to Run C

Review: how does Python work?



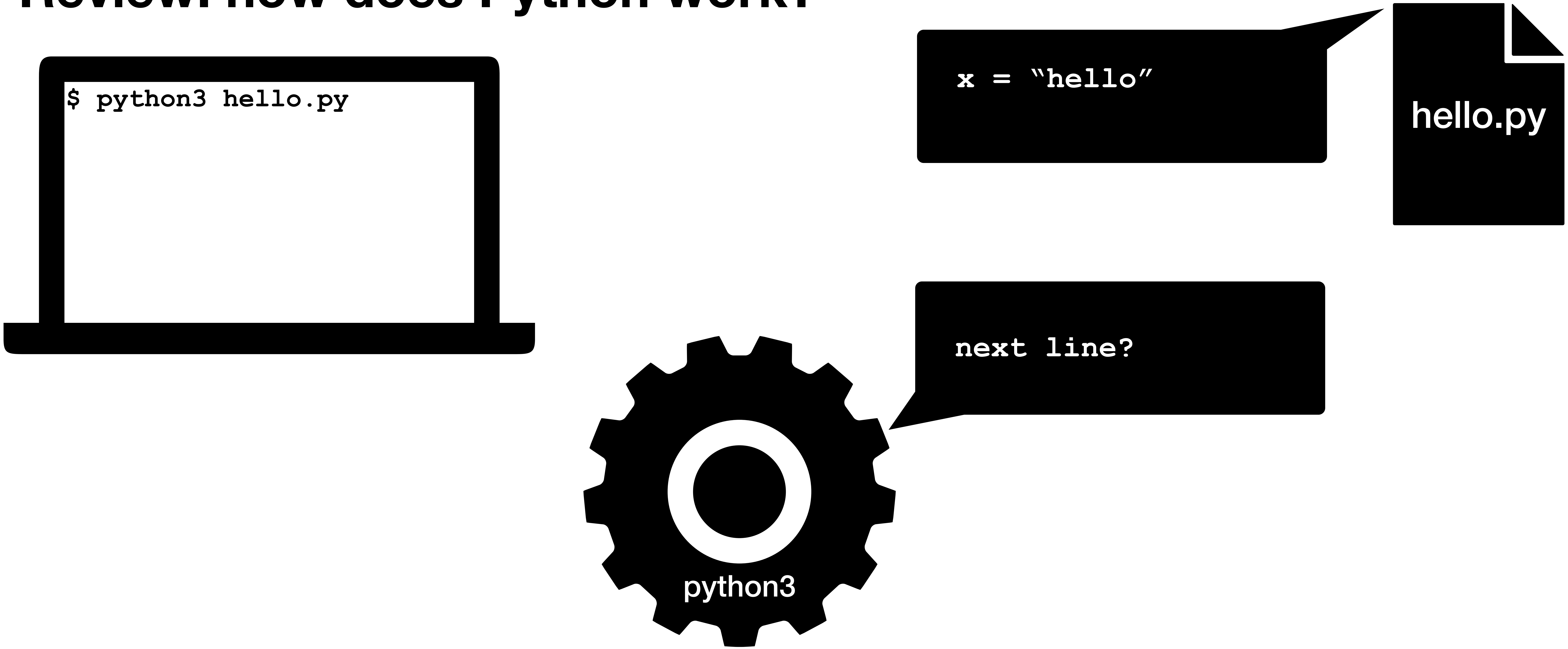
```
$ python3 hello.py
```

```
x = “hello”
```

```
ok. remembered x
```

How to Run C

Review: how does Python work?

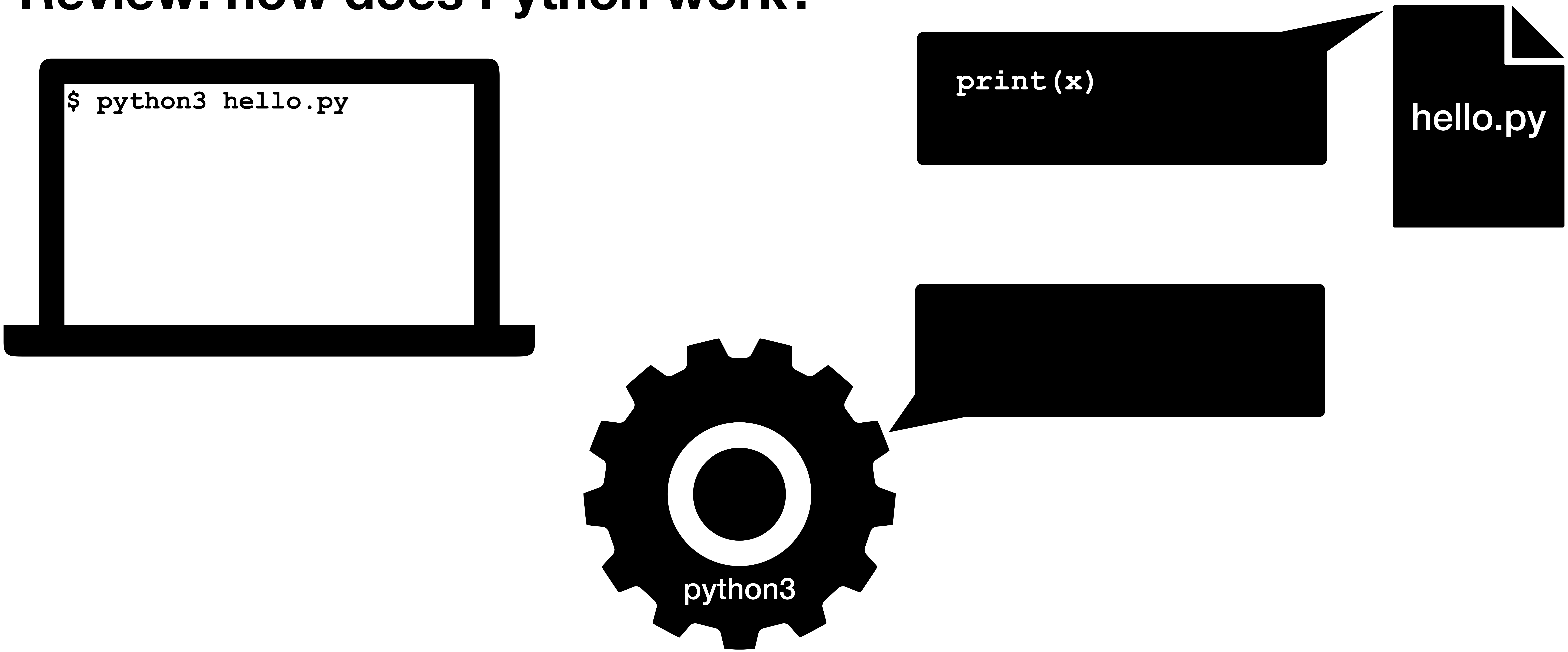


```
$ python3 hello.py
```

```
x = “hello”
```

How to Run C

Review: how does Python work?

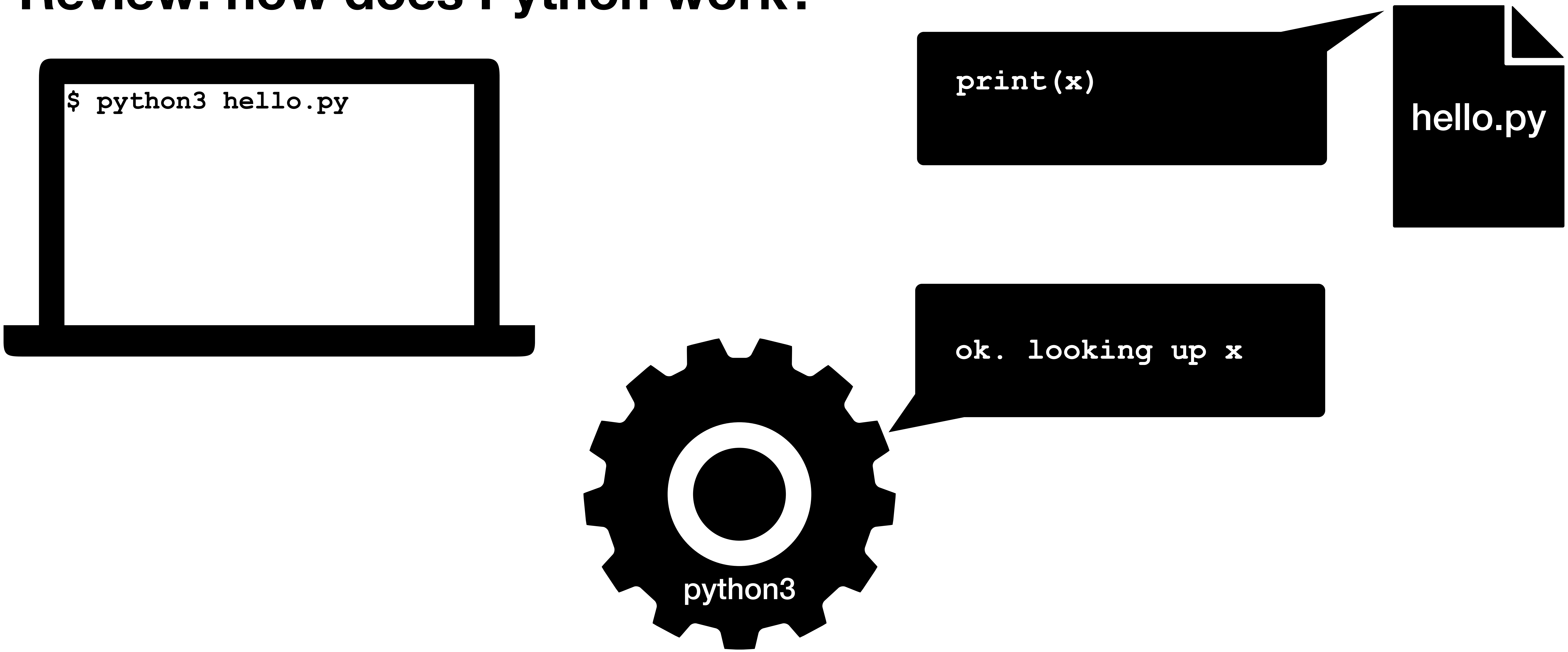


```
$ python3 hello.py
```

```
print(x)
```

How to Run C

Review: how does Python work?



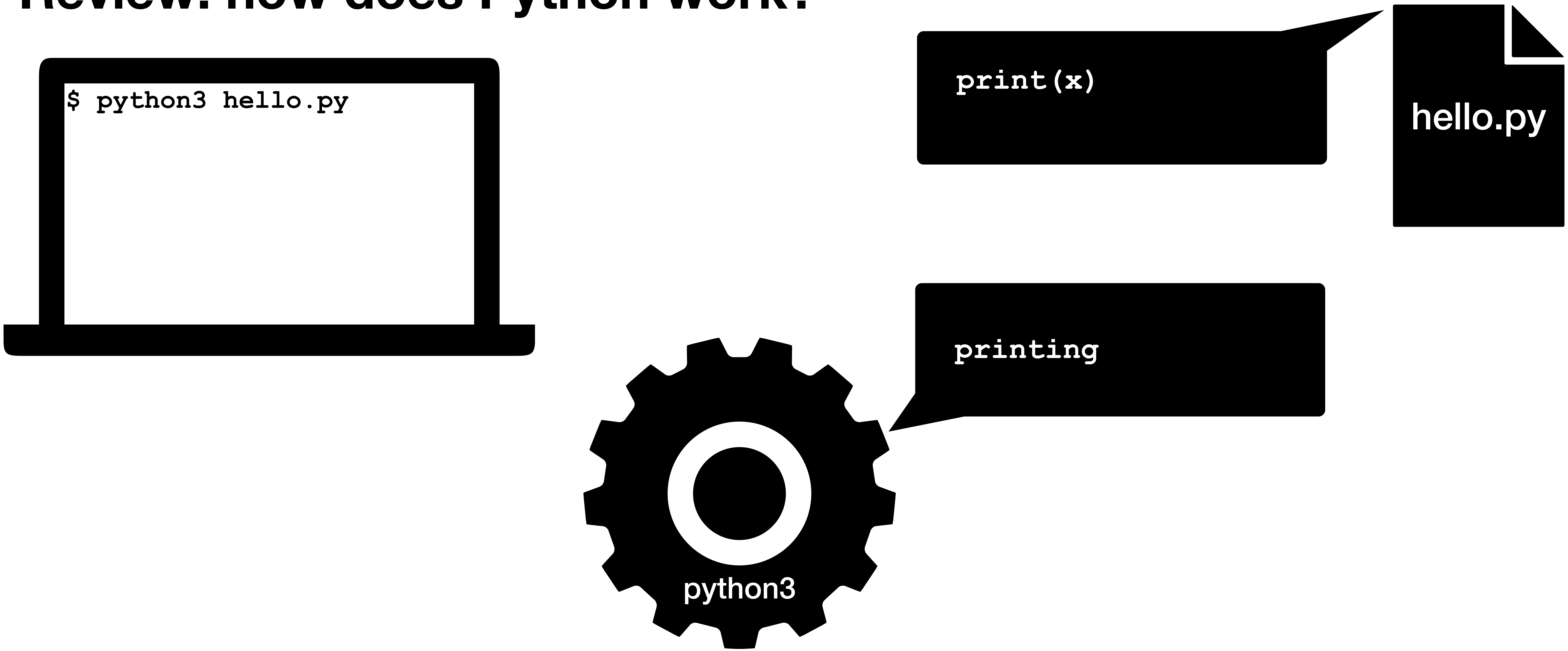
```
$ python3 hello.py
```

```
print(x)
```

```
ok. looking up x
```

How to Run C

Review: how does Python work?

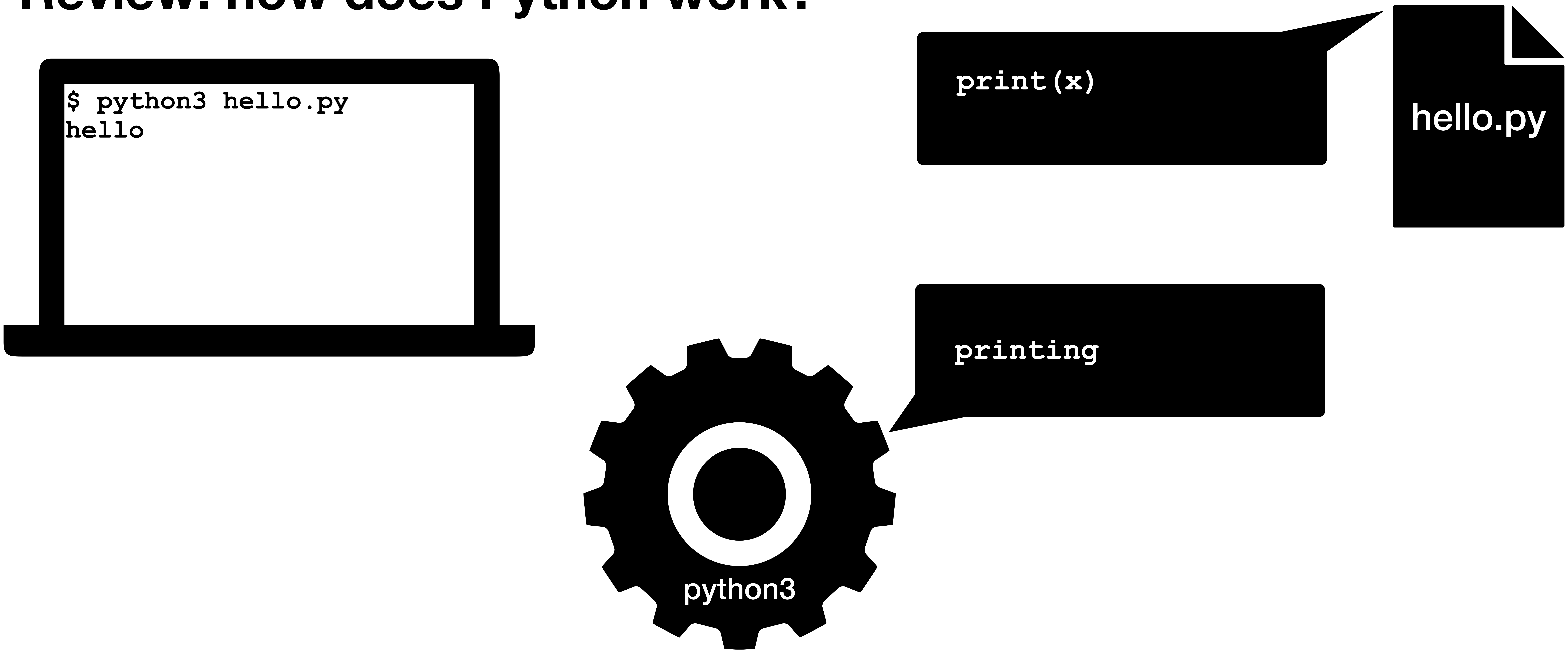


```
$ python3 hello.py
```

```
print(x)
```

How to Run C

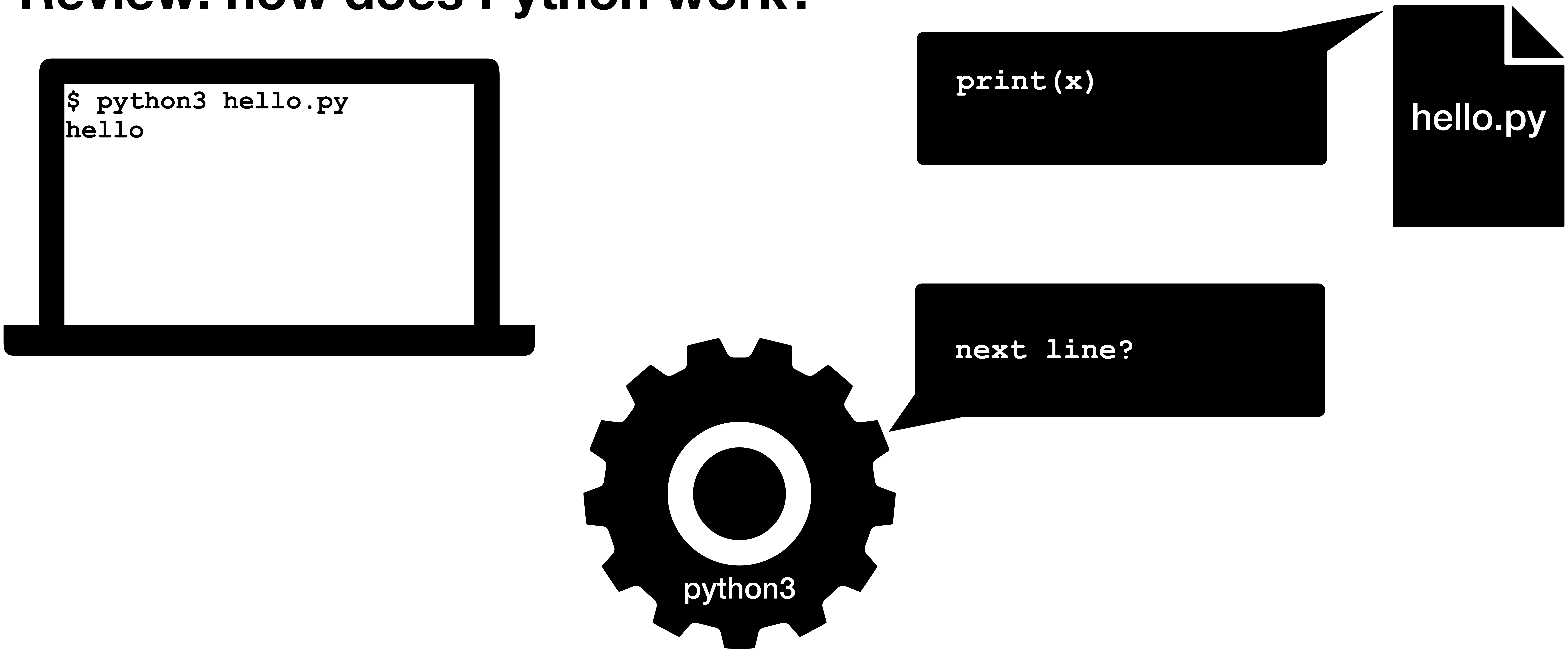
Review: how does Python work?



```
$ python3 hello.py
hello
```

How to Run C

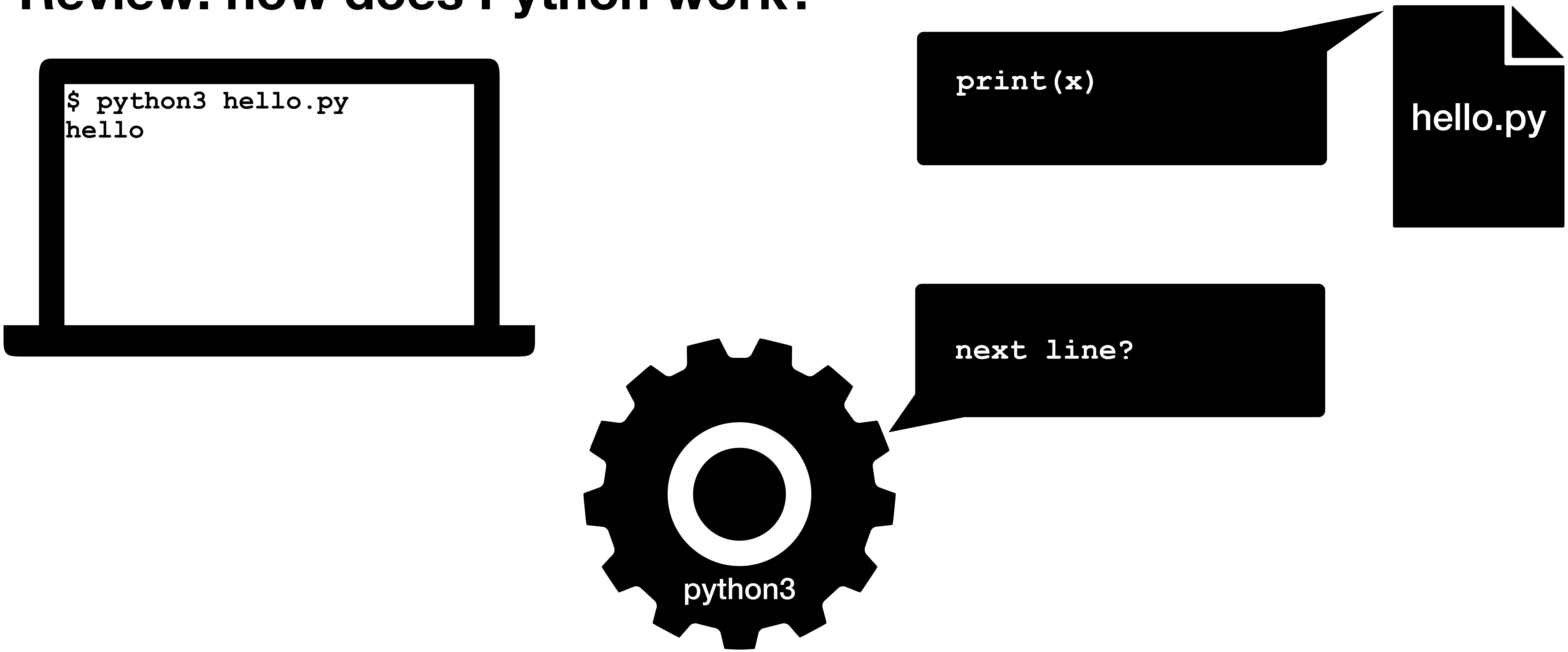
Review: how does Python work?



```
$ python3 hello.py
hello
```

How to Run C

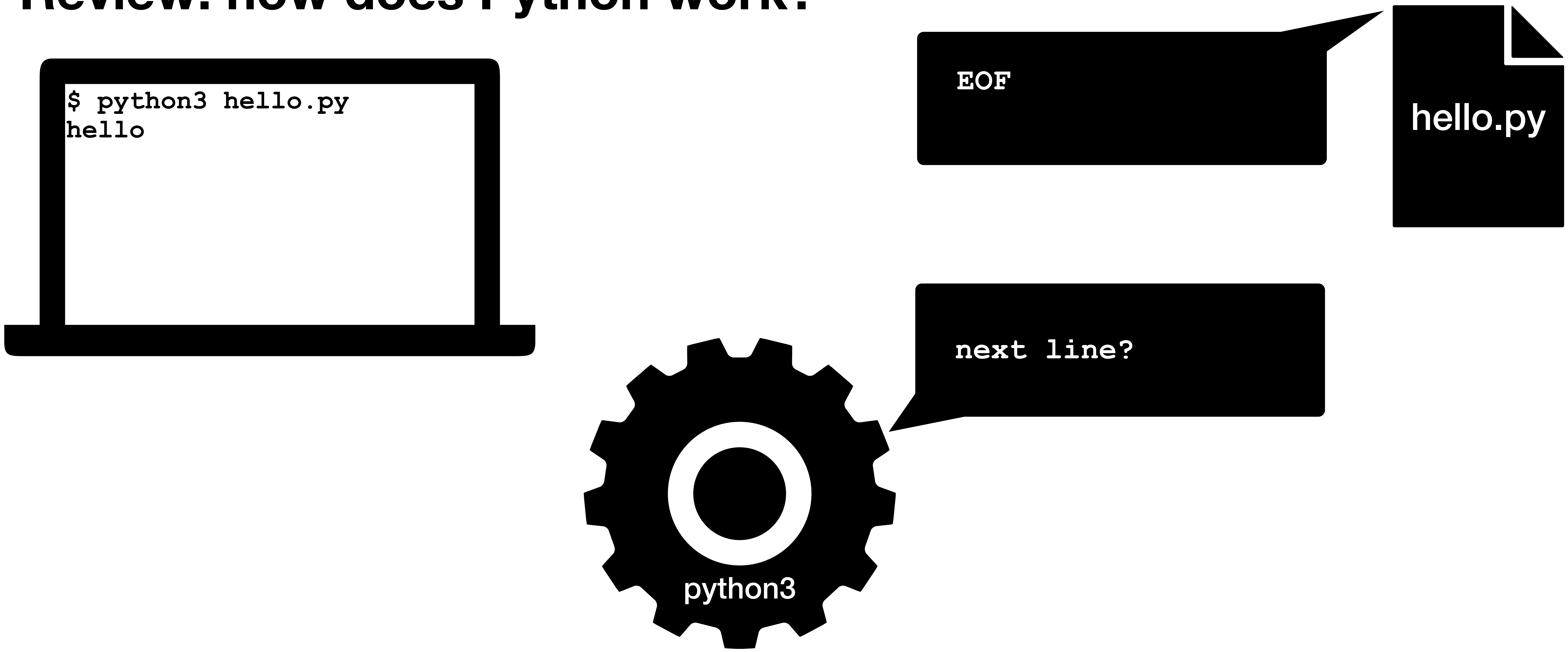
Review: how does Python work?



```
$ python3 hello.py
hello
```

How to Run C

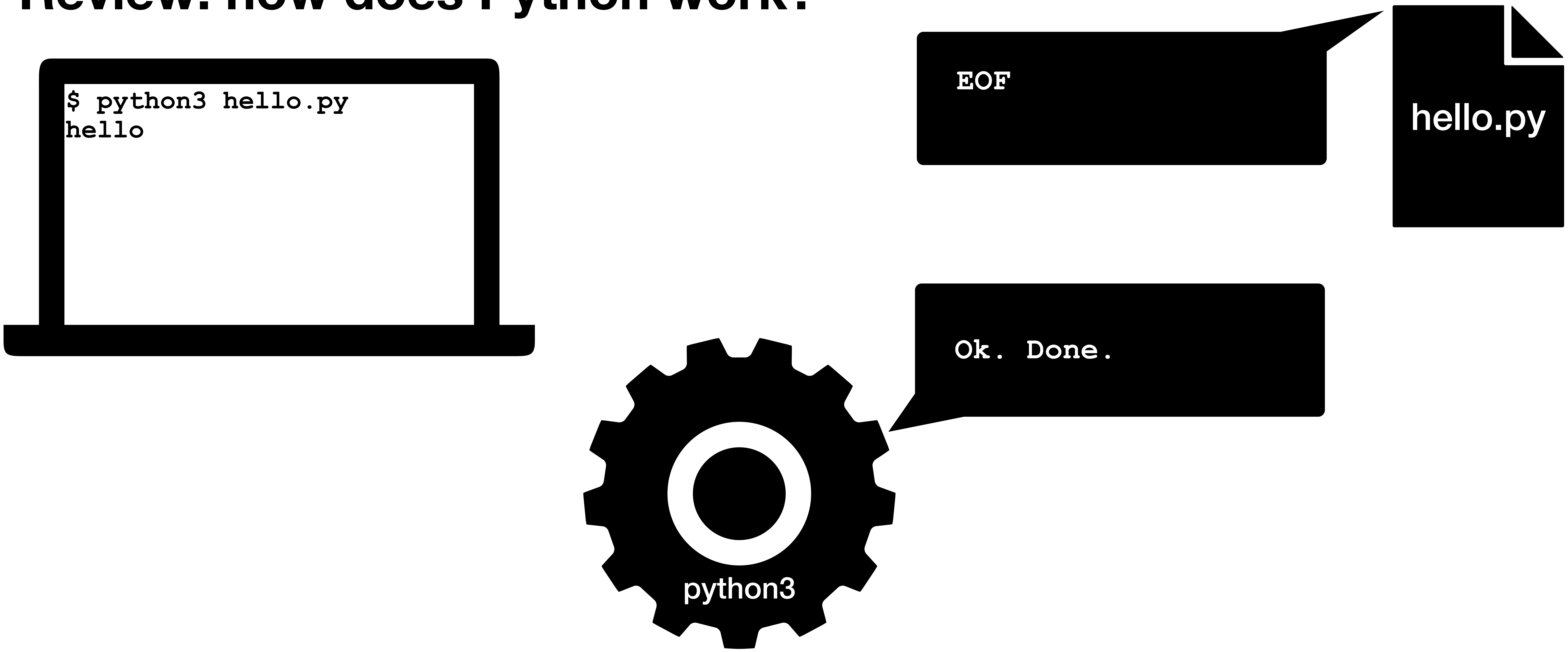
Review: how does Python work?



```
$ python3 hello.py
hello
```

How to Run C

Review: how does Python work?



```
$ python3 hello.py
hello
```

How to Run C

Review: how does Python work?

```
$ python3 hello.py
hello
$
```

How to Run C

Review: how does Python work?

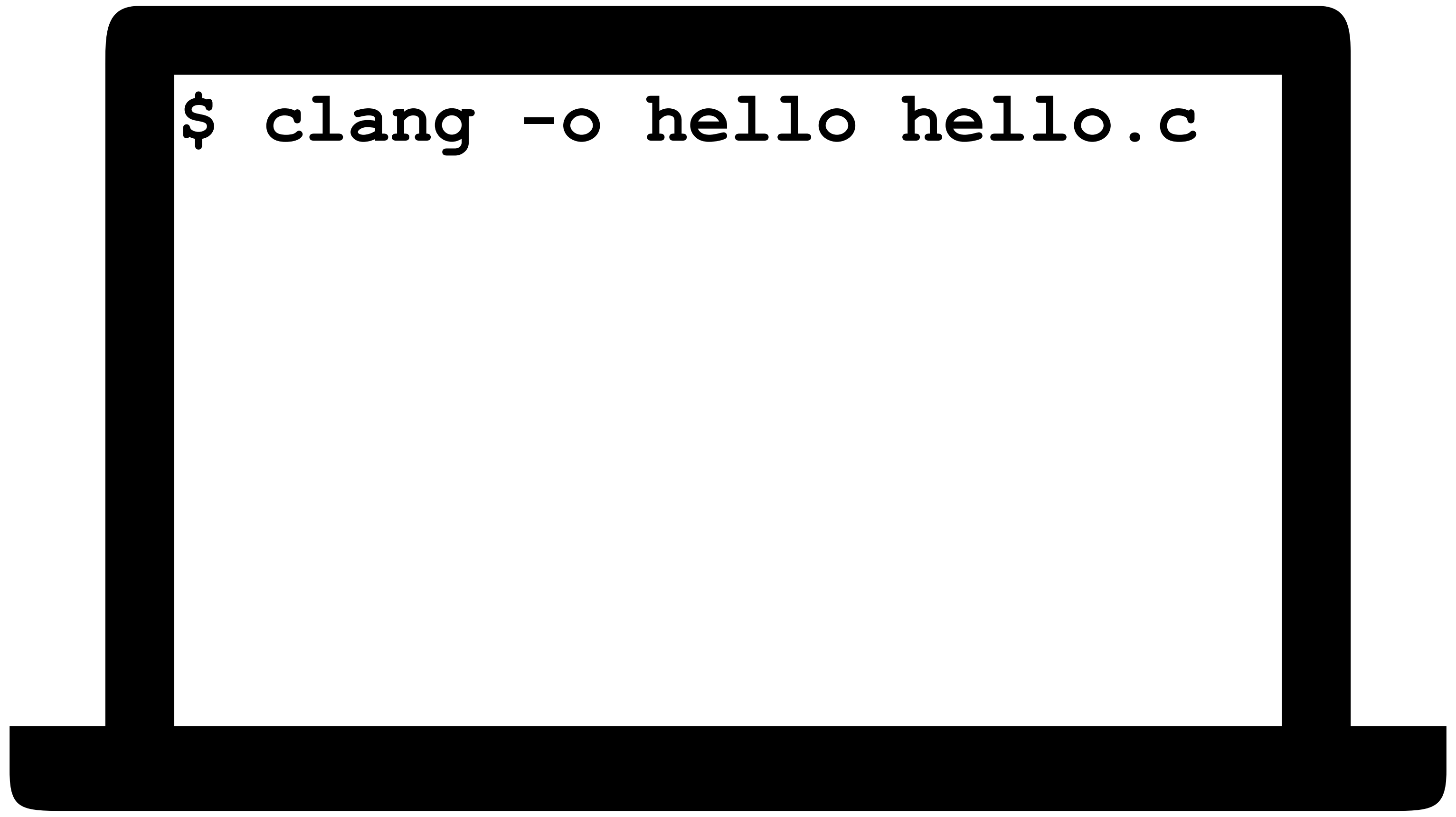
- There is a program that reads your Python script, and executes line by line
- This program is called Python interpreter

How to Run C

How about C?

How to Run C

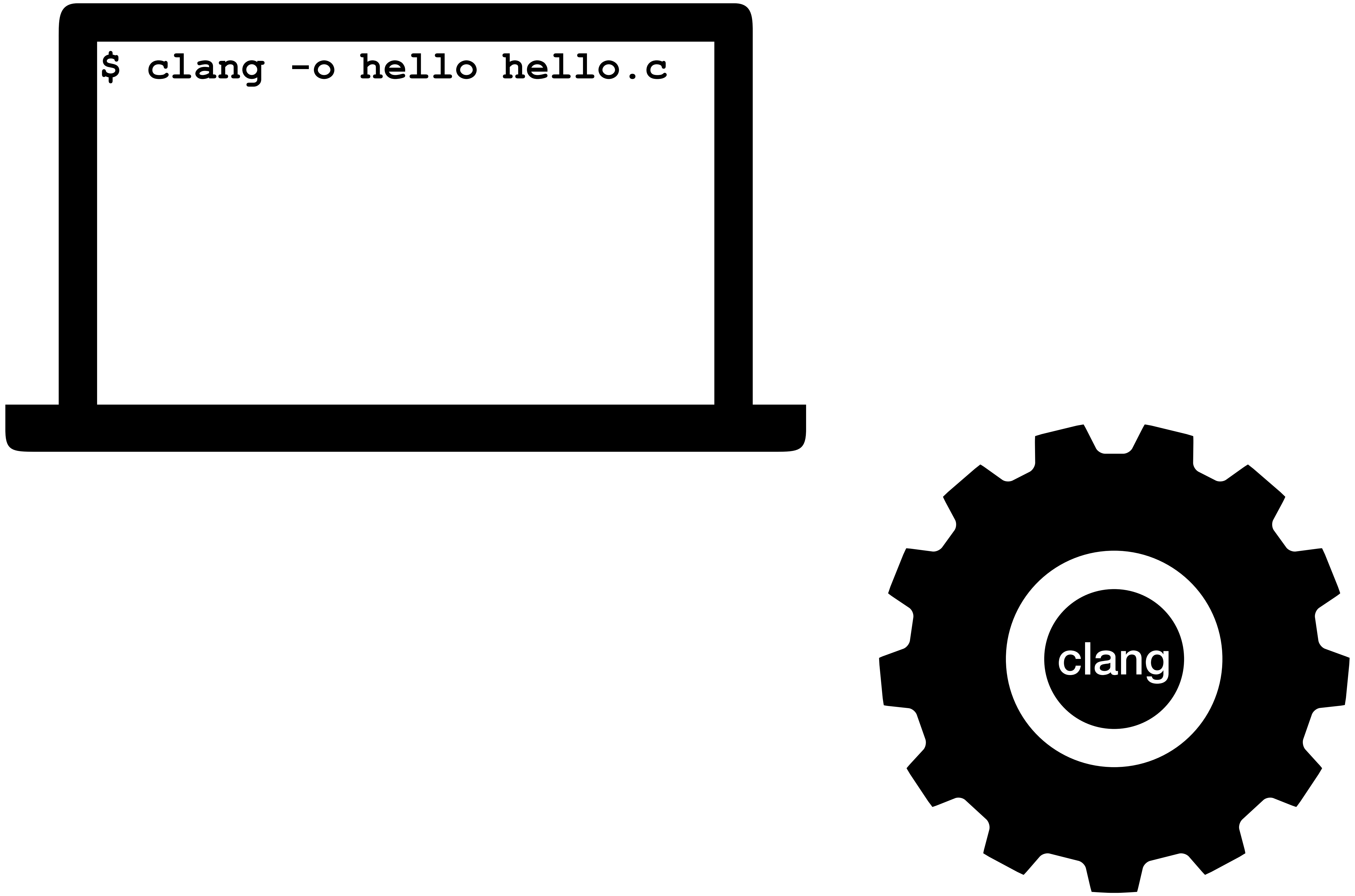
How about C?



```
$ clang -o hello hello.c
```

How to Run C

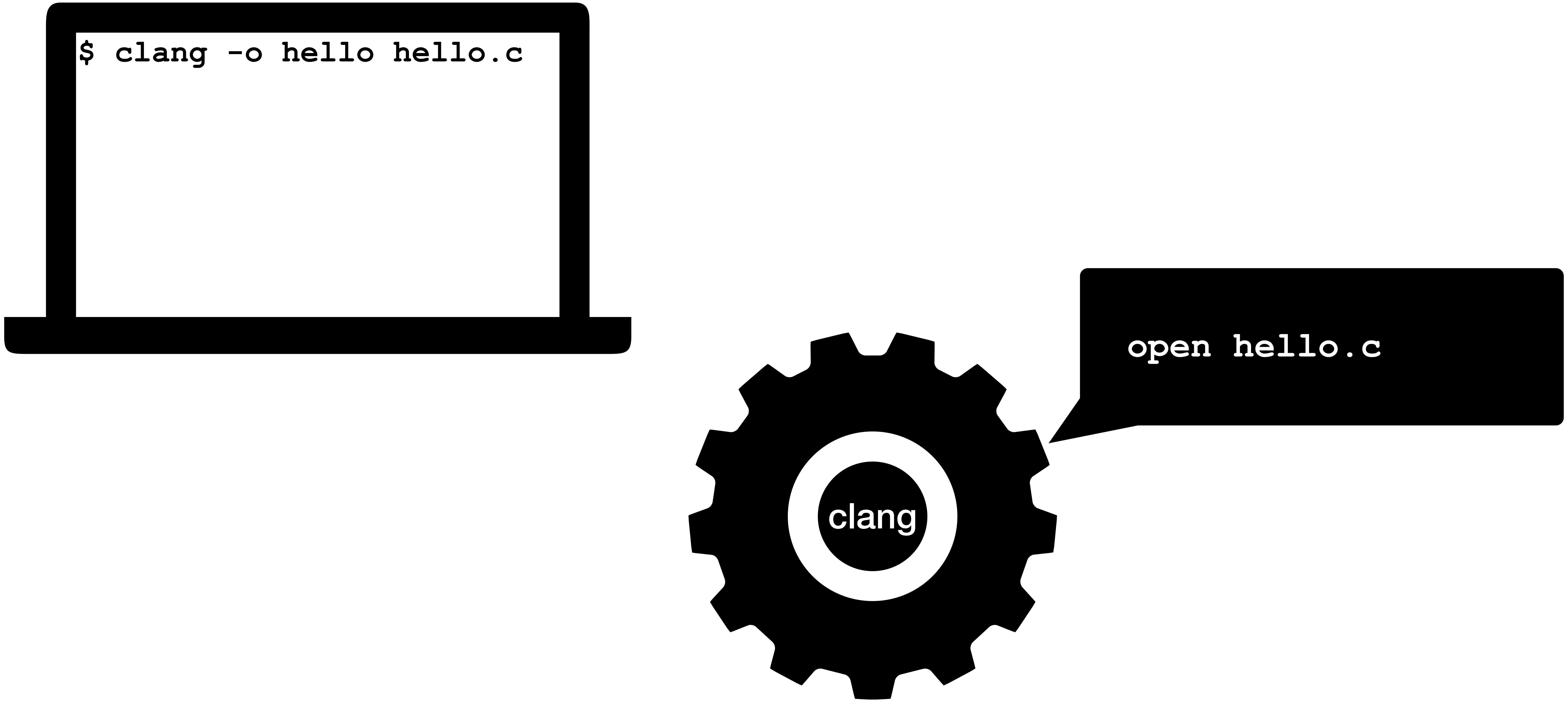
How about C?



```
$ clang -o hello hello.c
```

How to Run C

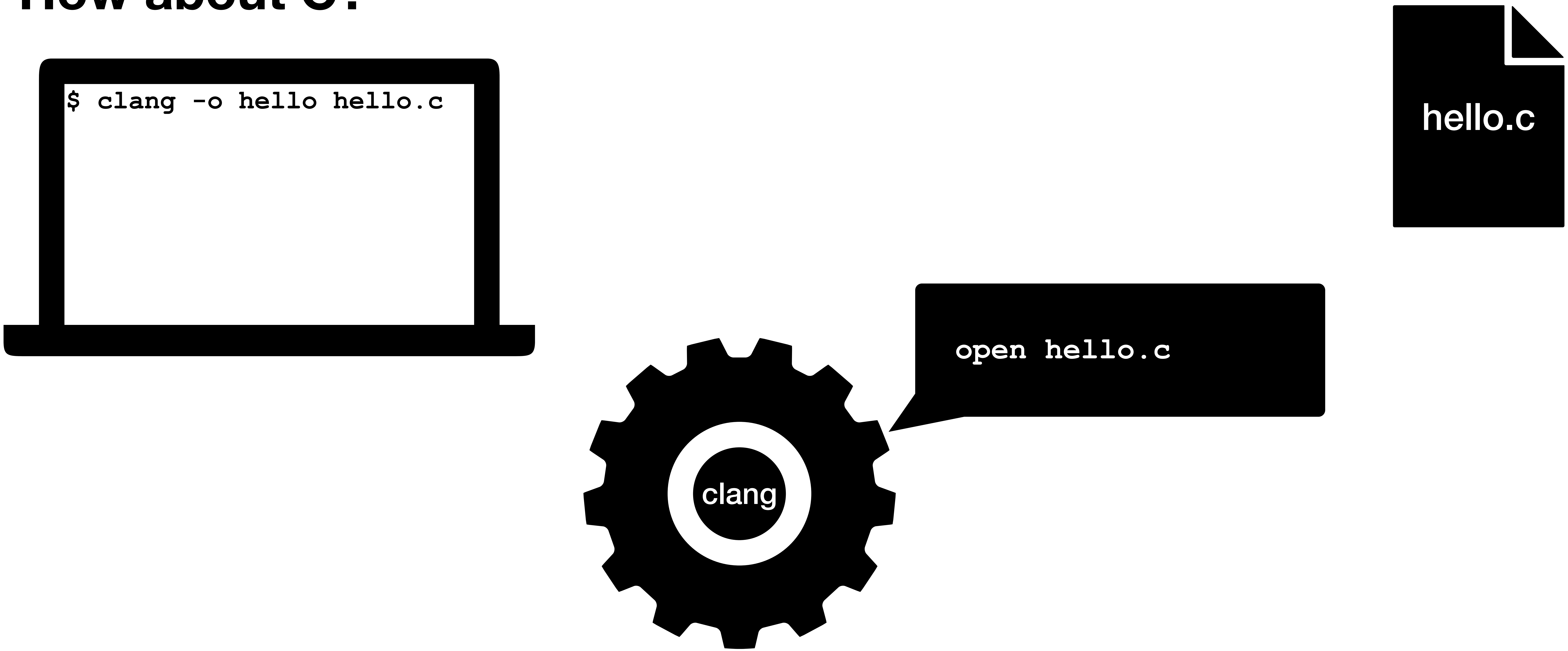
How about C?



```
$ clang -o hello hello.c
```

How to Run C

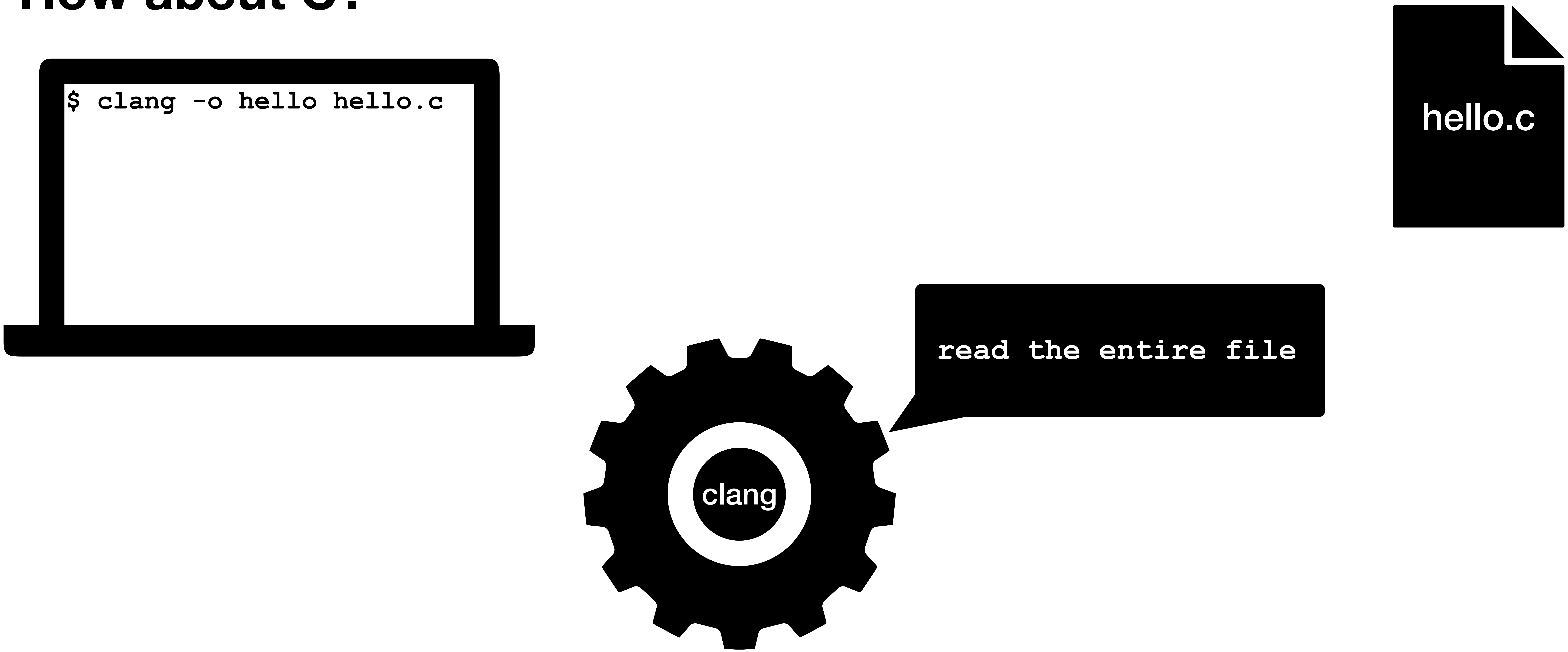
How about C?



```
$ clang -o hello hello.c
```

How to Run C

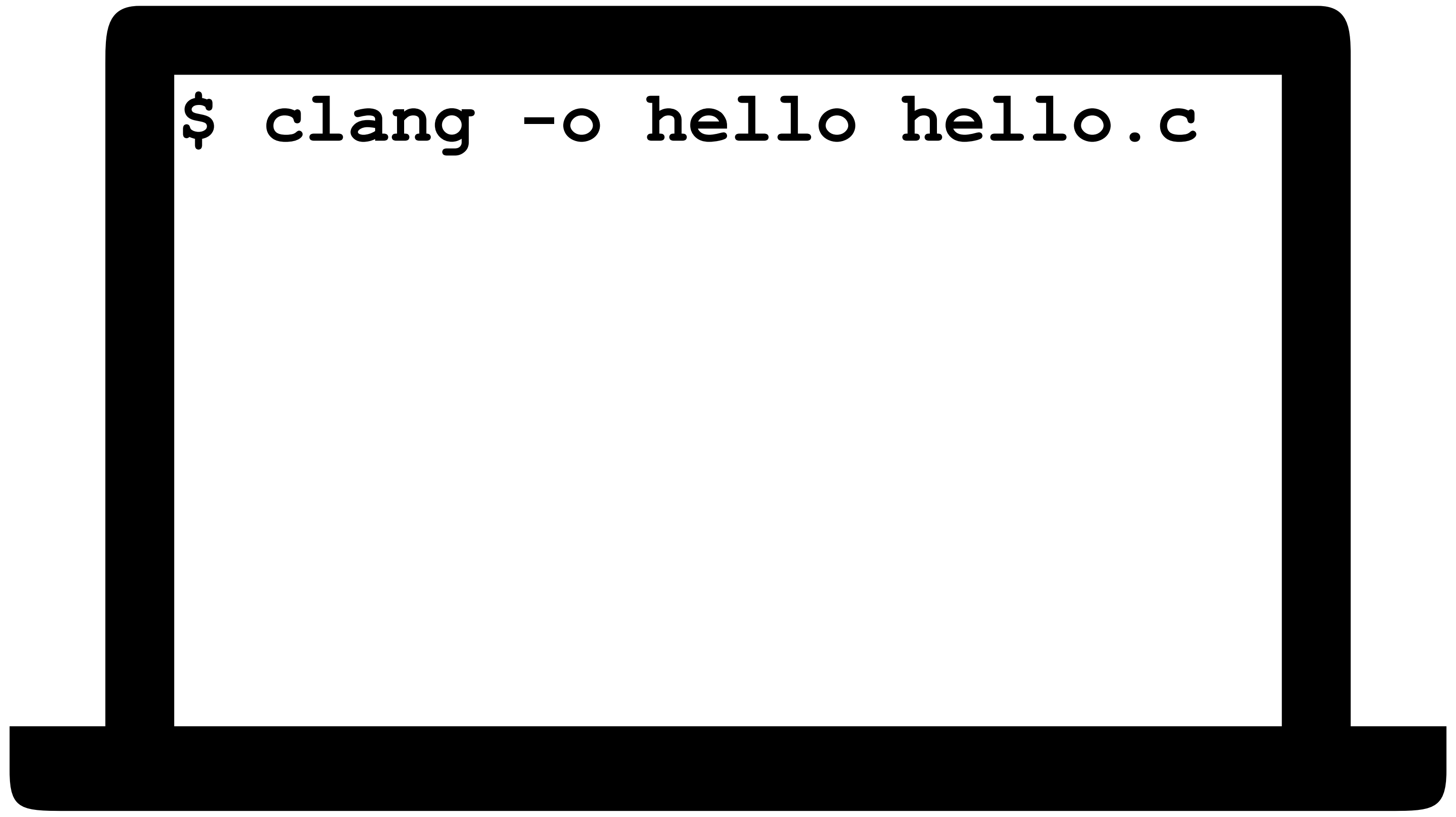
How about C?



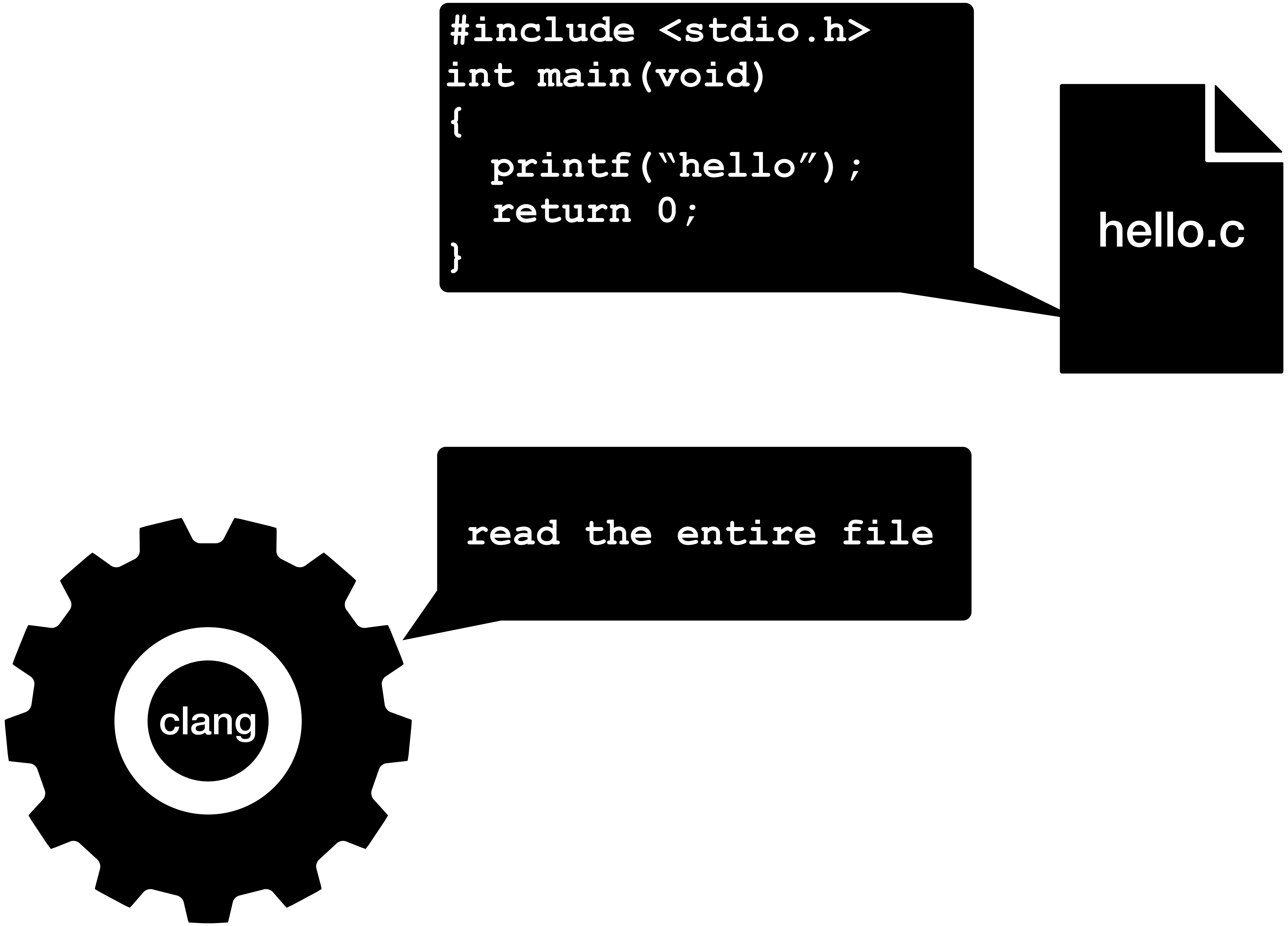
```
$ clang -o hello hello.c
```

How to Run C

How about C?



```
$ clang -o hello hello.c
```

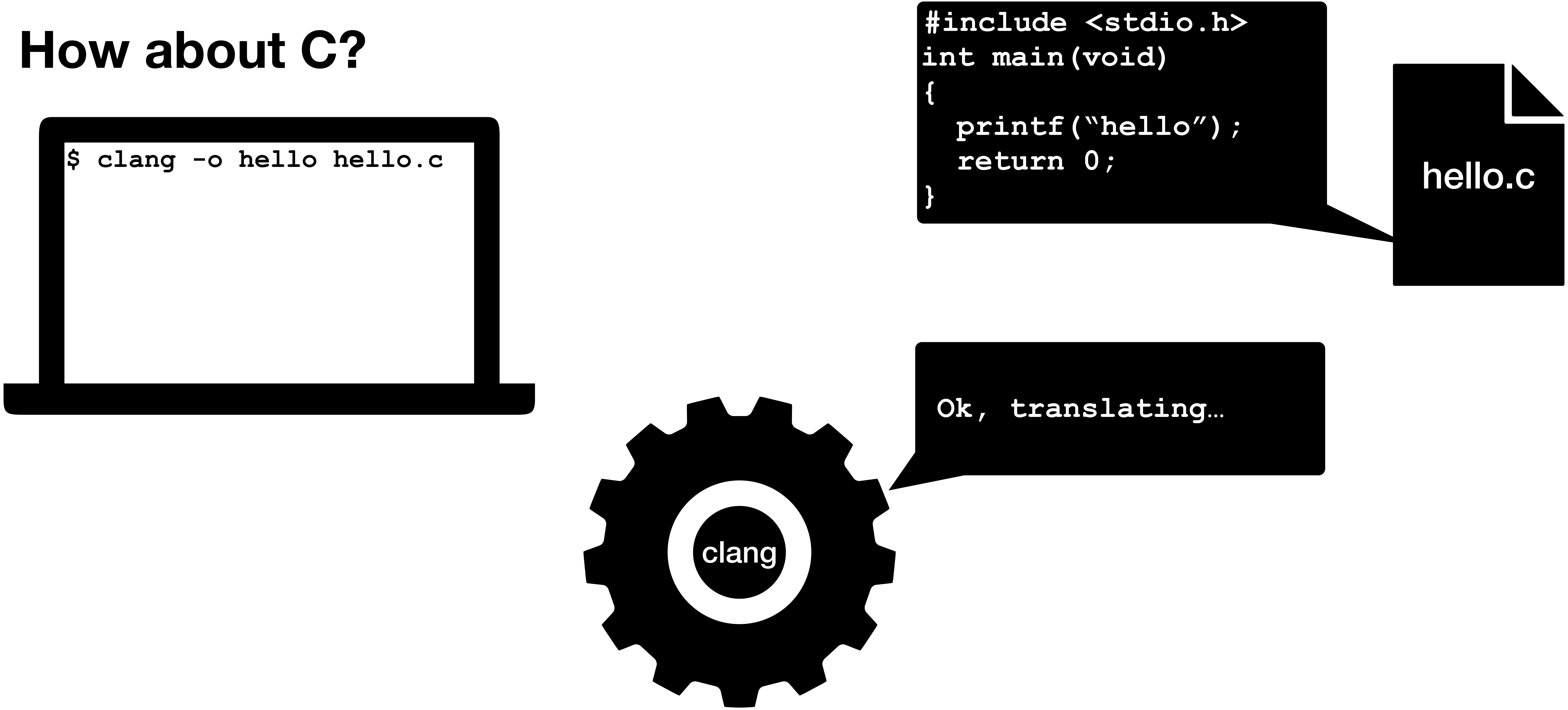


```
#include <stdio.h>
int main(void)
{
  printf(“hello”);
  return 0;
}
```

```
read the entire file
```

How to Run C

How about C?

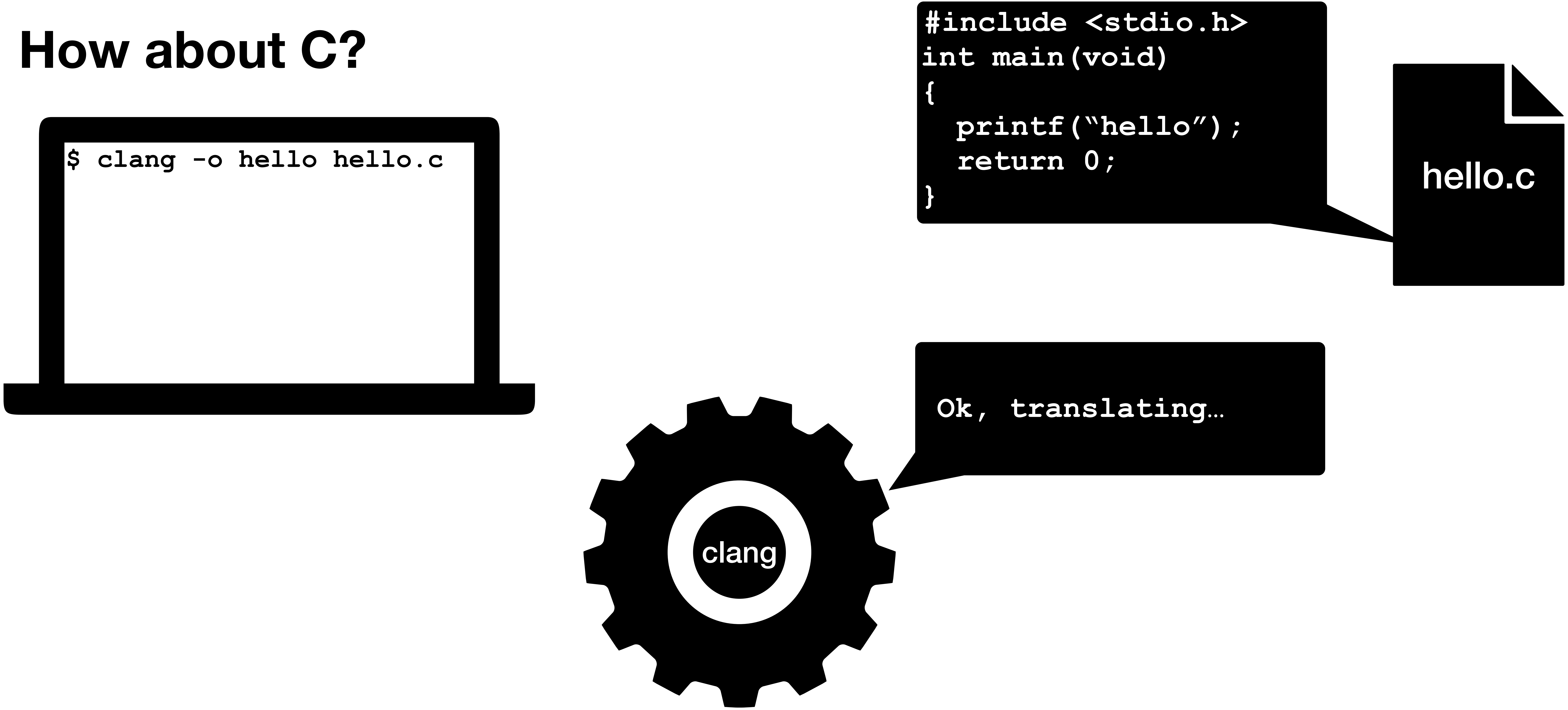


```
$ clang -o hello hello.c
```

```
#include <stdio.h>
int main(void)
{
  printf("hello");
  return 0;
}
```

How to Run C

How about C?

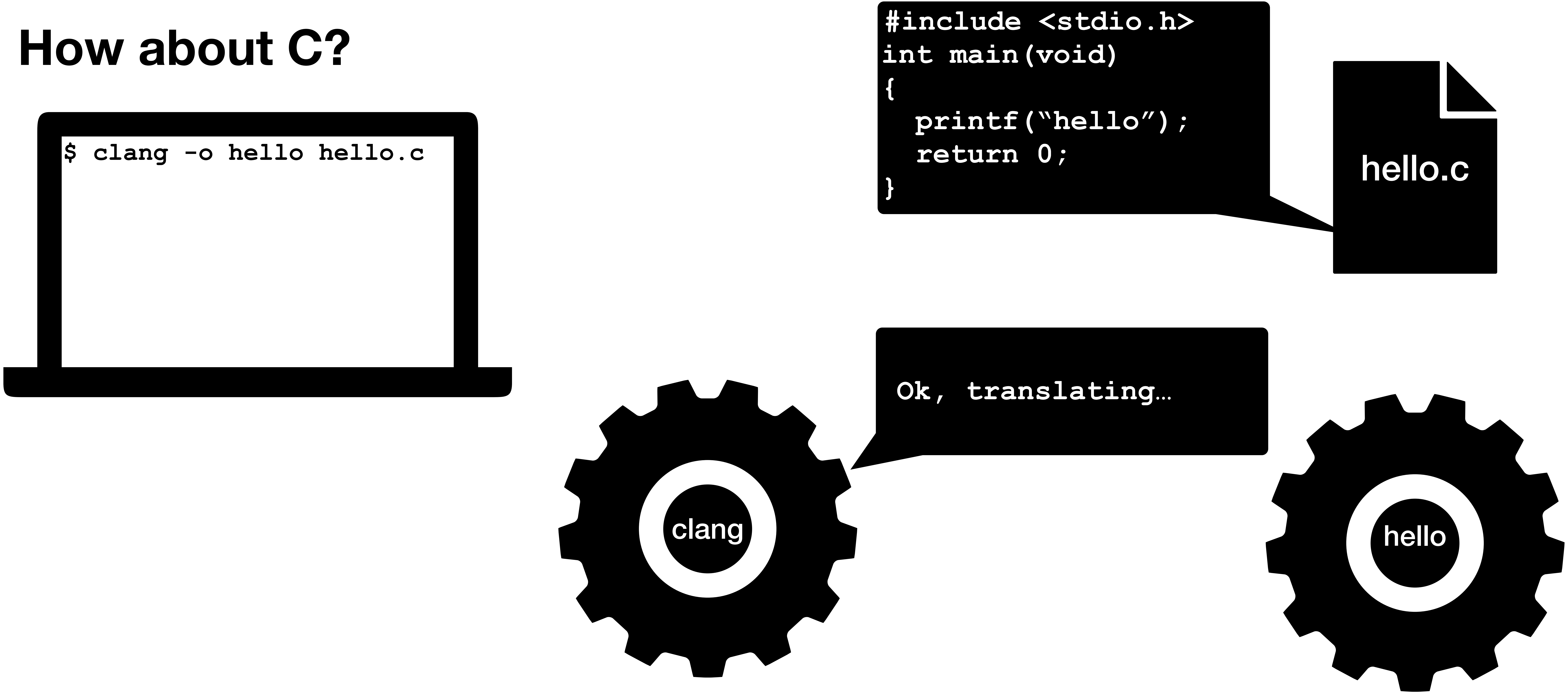


```
$ clang -o hello hello.c
```

```
#include <stdio.h>
int main(void)
{
  printf("hello");
  return 0;
}
```

How to Run C

How about C?



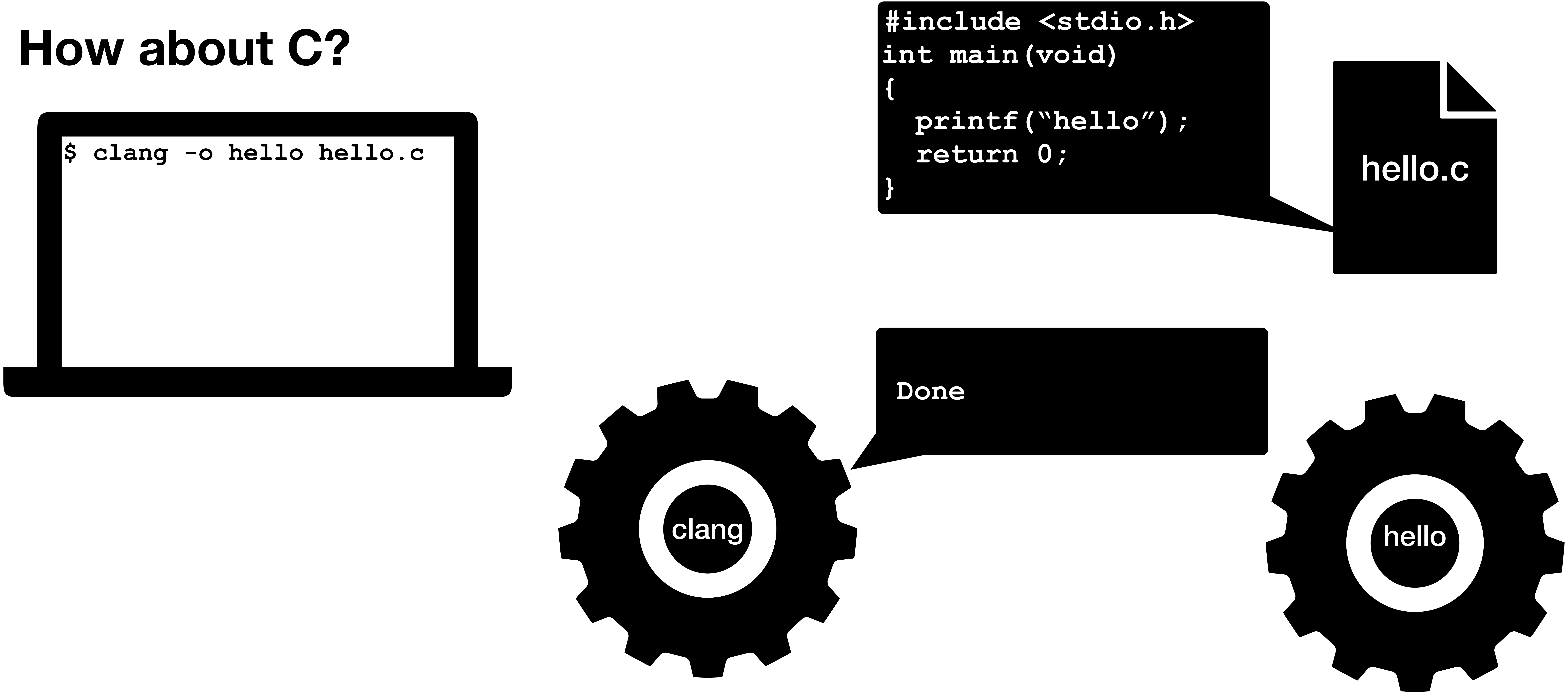
```
$ clang -o hello hello.c
```

```
#include <stdio.h>
int main(void)
{
  printf(“hello”);
  return 0;
}
```

Ok, translating…

How to Run C

How about C?



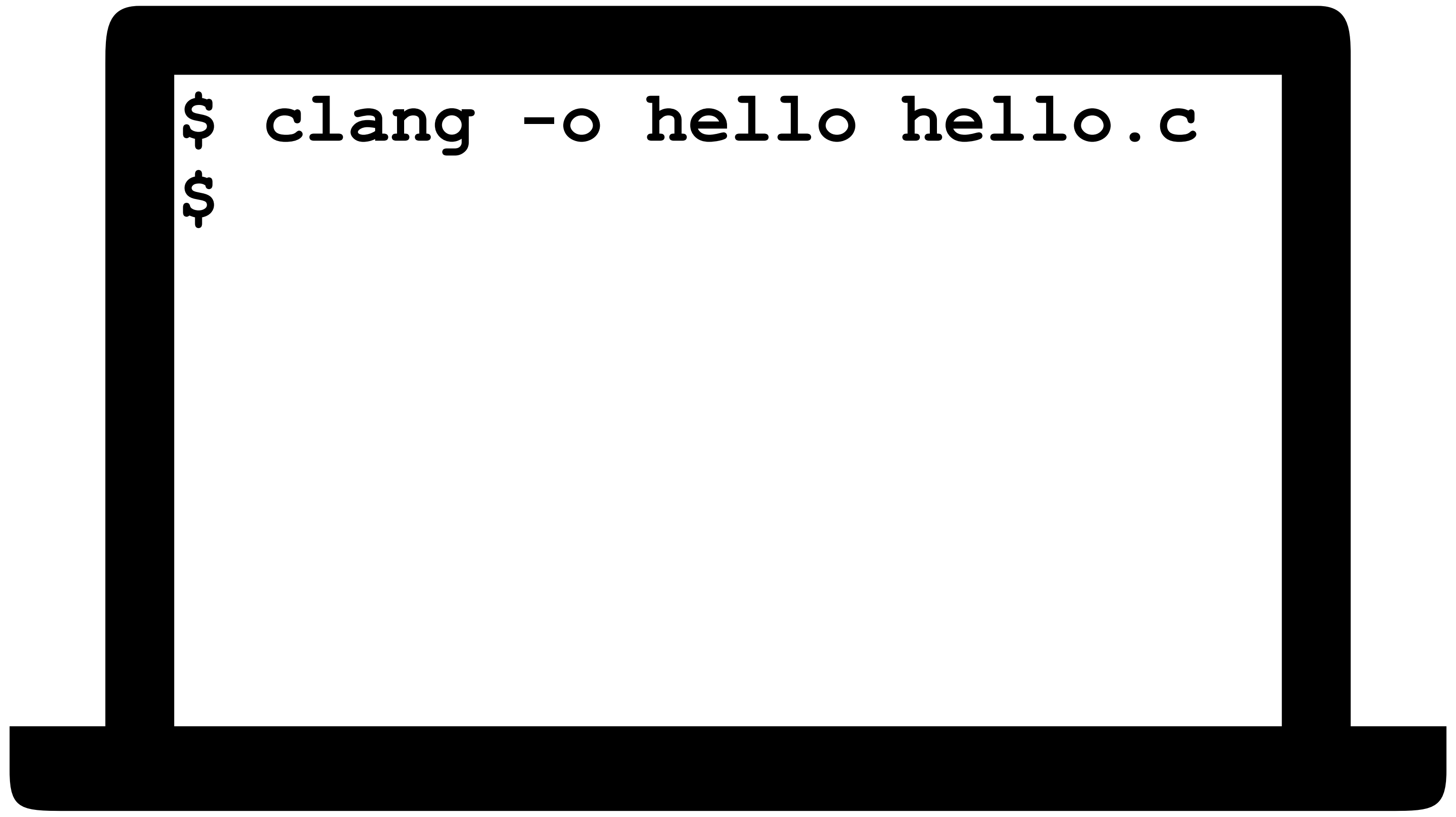
```
$ clang -o hello hello.c
```

```
#include <stdio.h>
int main(void)
{
  printf(“hello”);
  return 0;
}
```

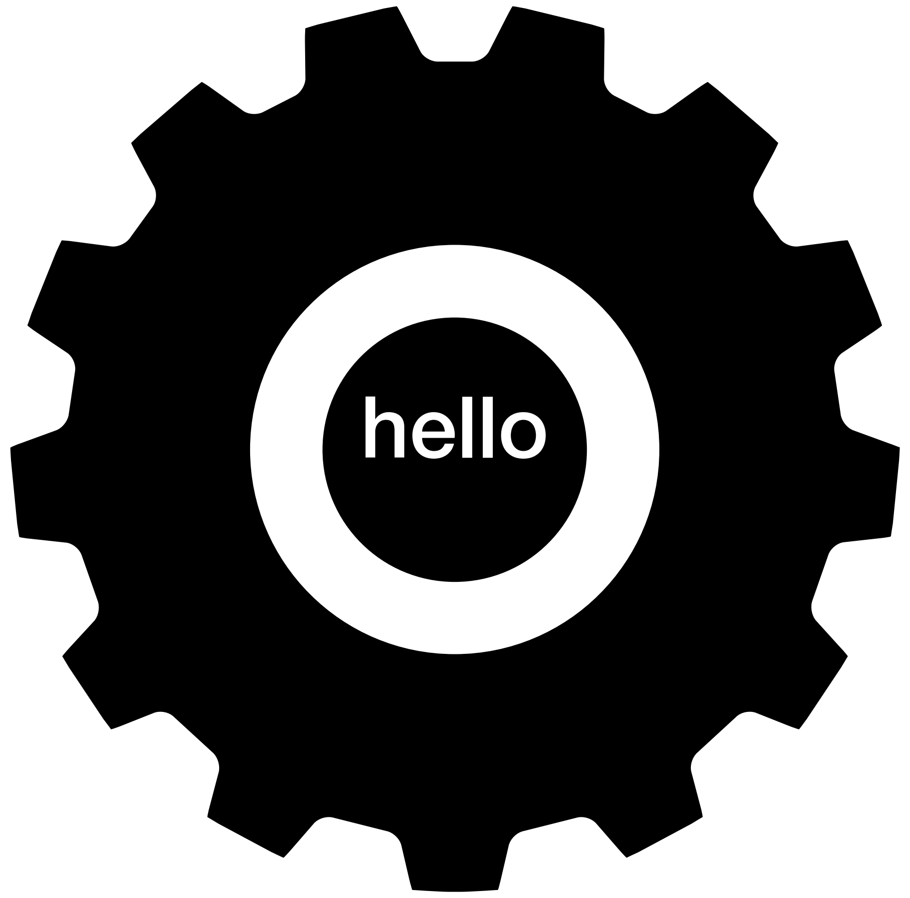
```
Done
```

How to Run C

How about C?

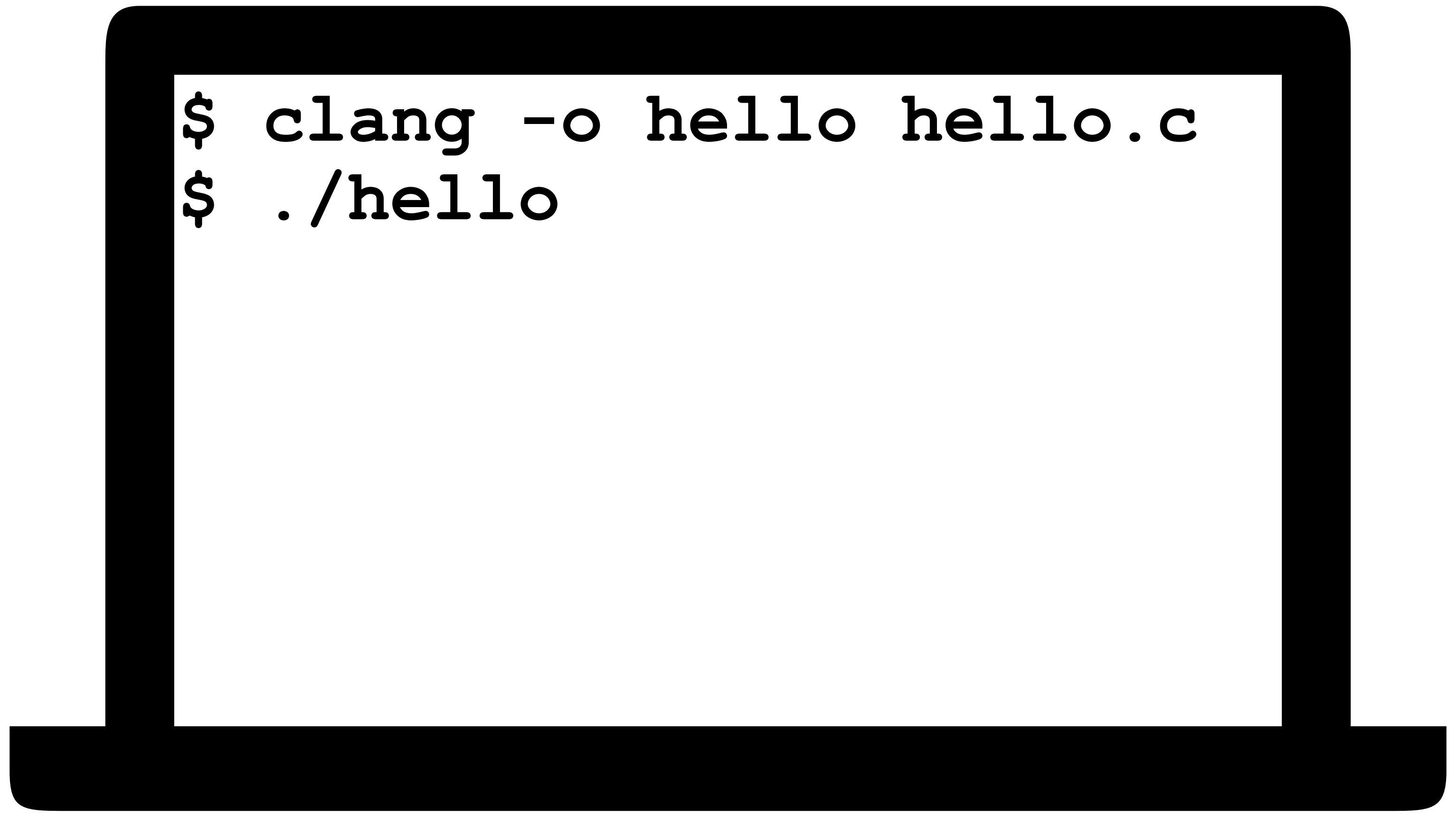


```
$ clang -o hello hello.c
$
```

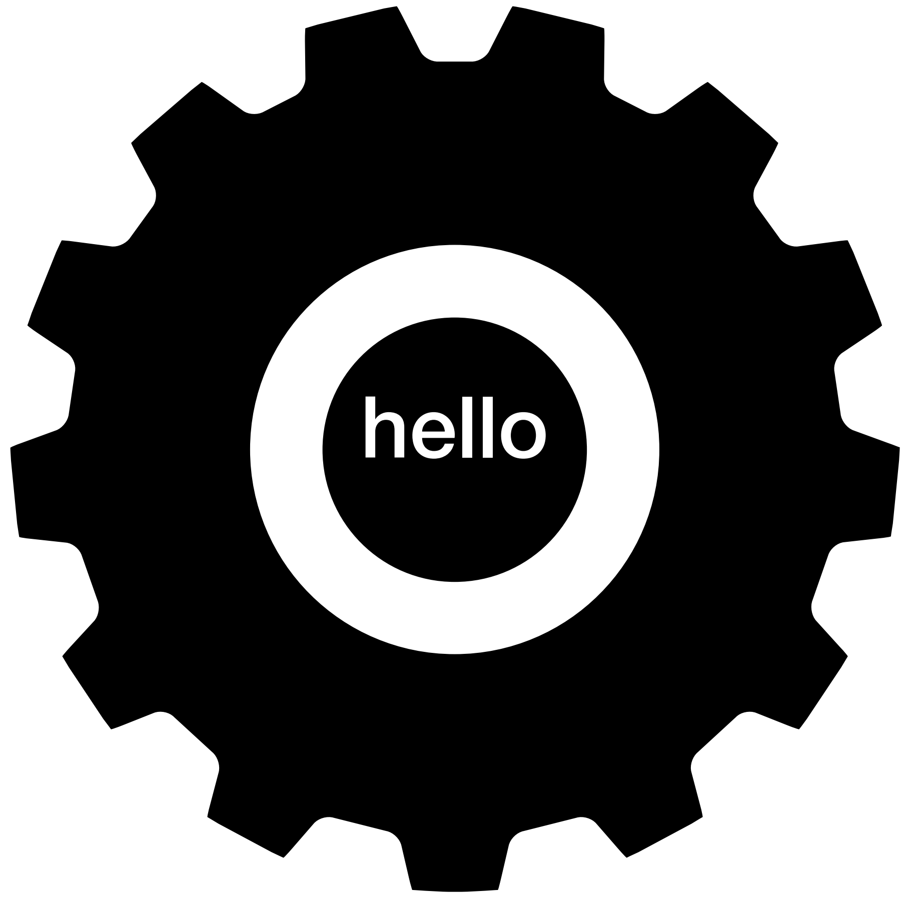


How to Run C

How about C?

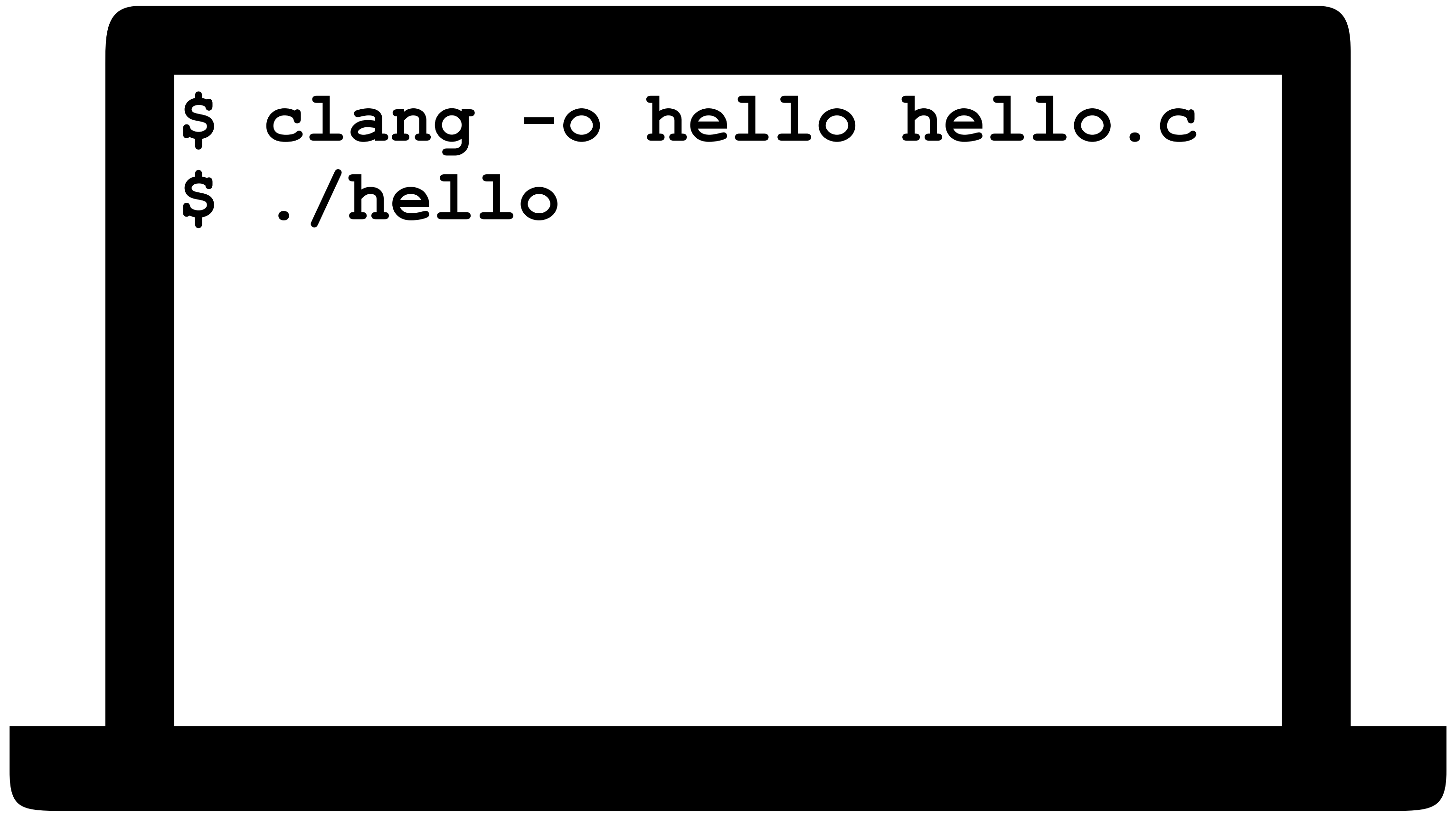


```
$ clang -o hello hello.c
$ ./hello
```

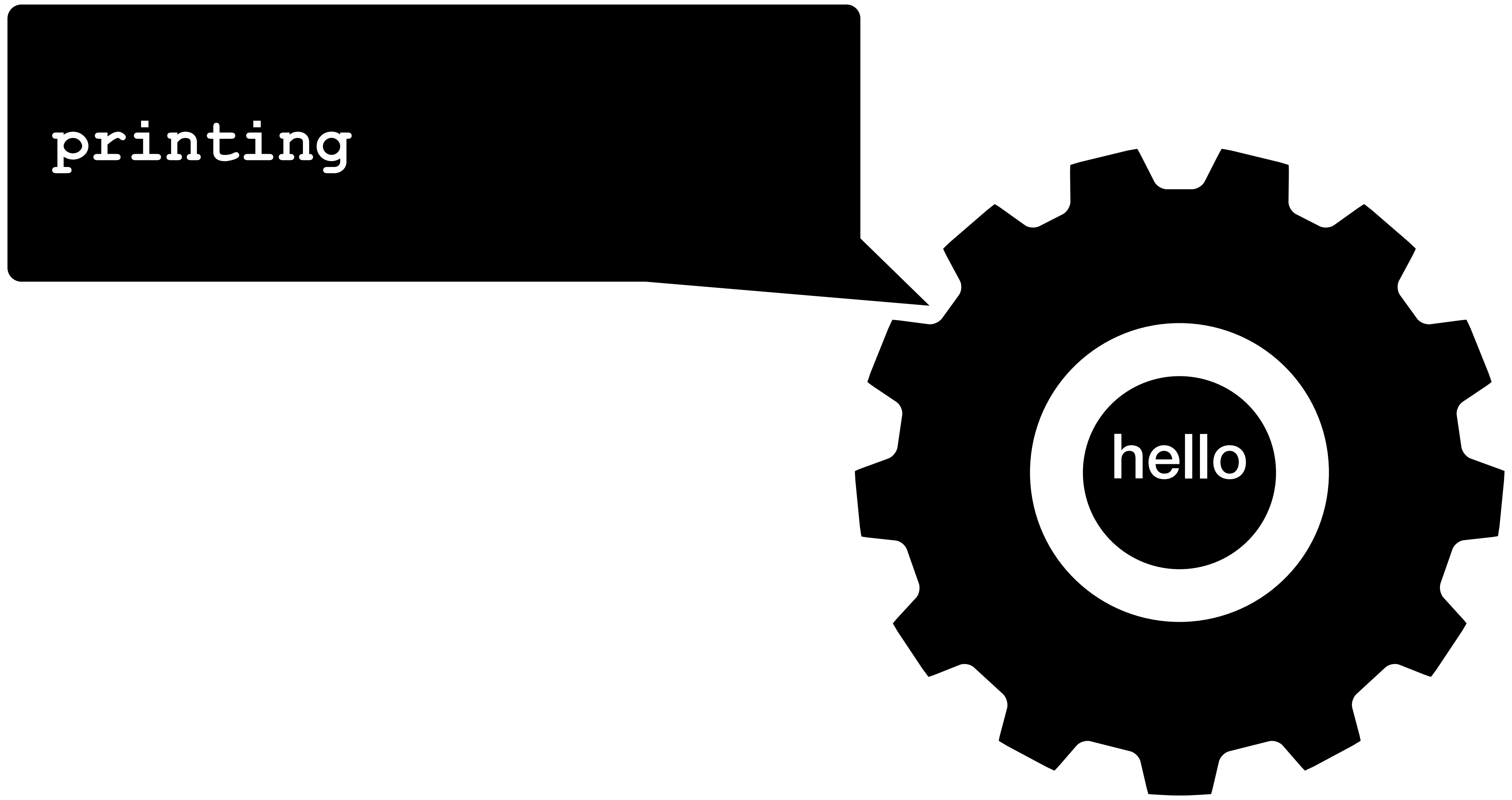


How to Run C

How about C?

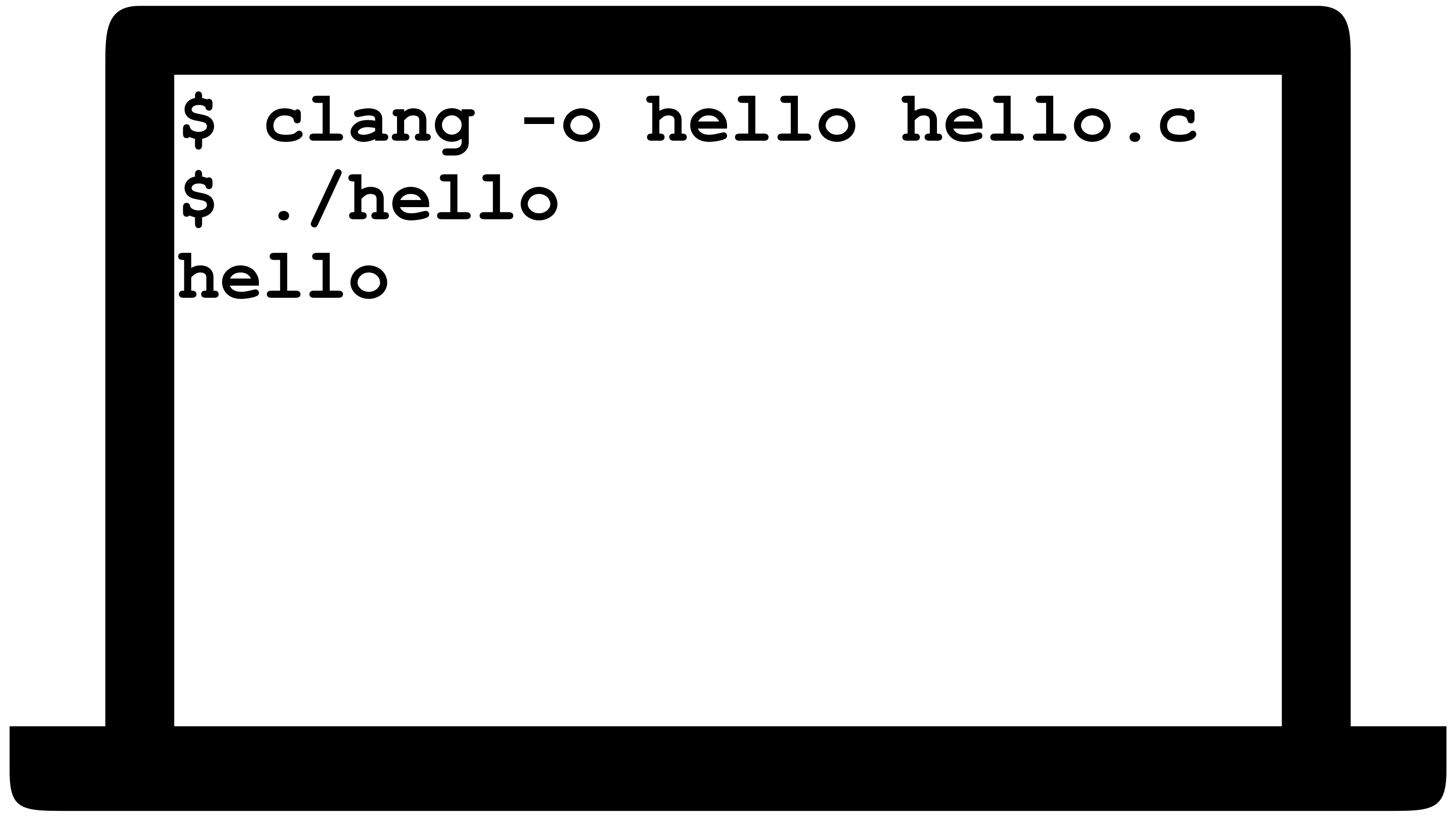


```
$ clang -o hello hello.c
$ ./hello
```

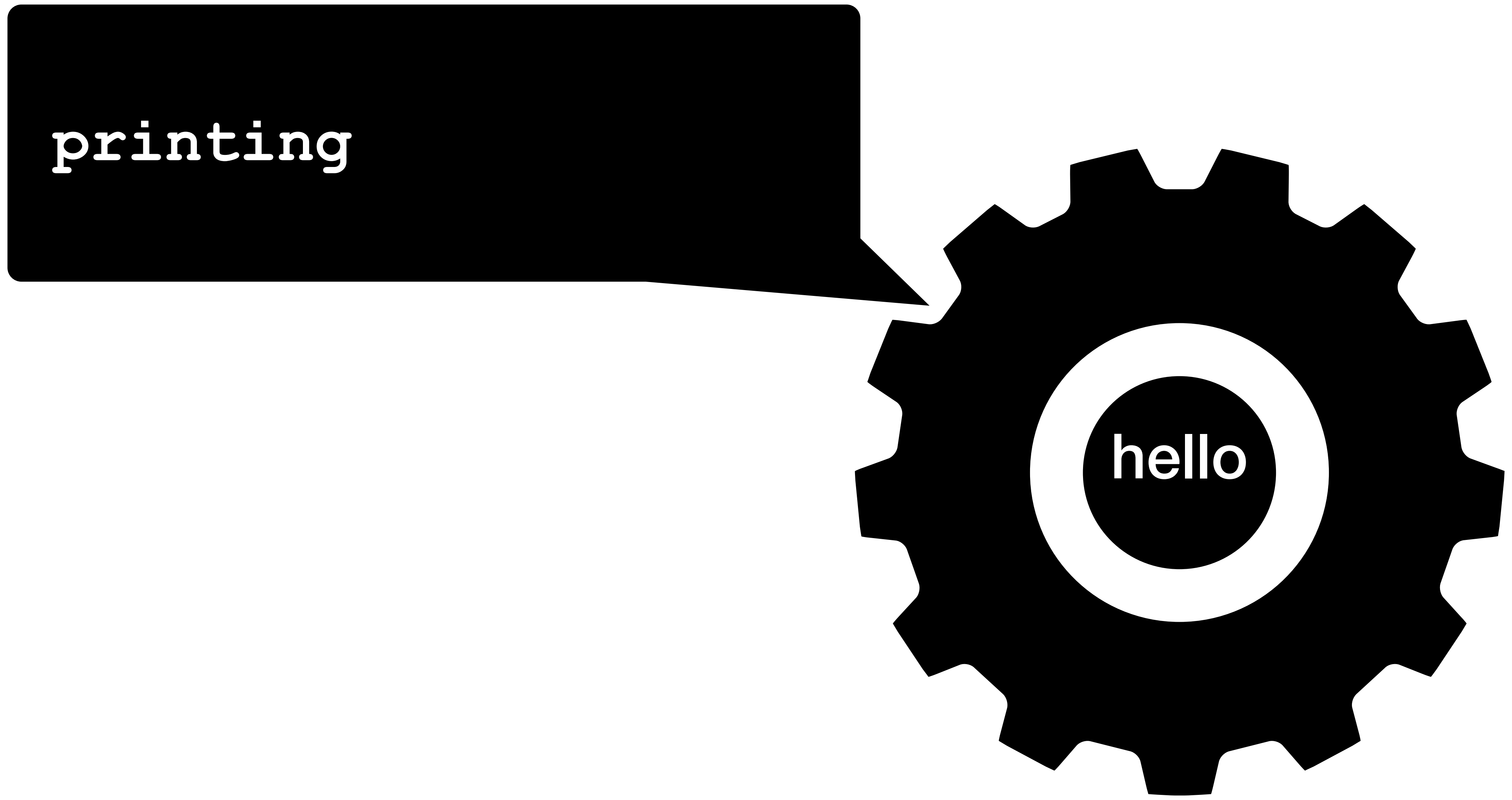


How to Run C

How about C?

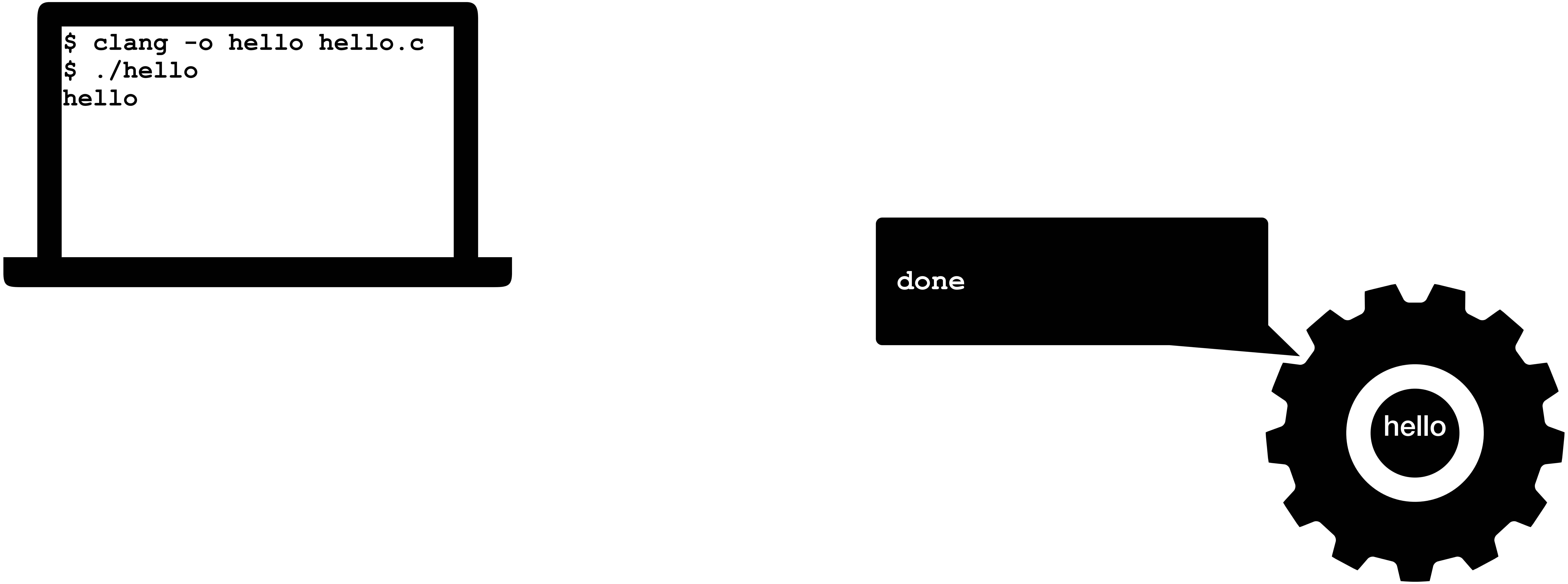


```
$ clang -o hello hello.c
$ ./hello
hello
```



How to Run C

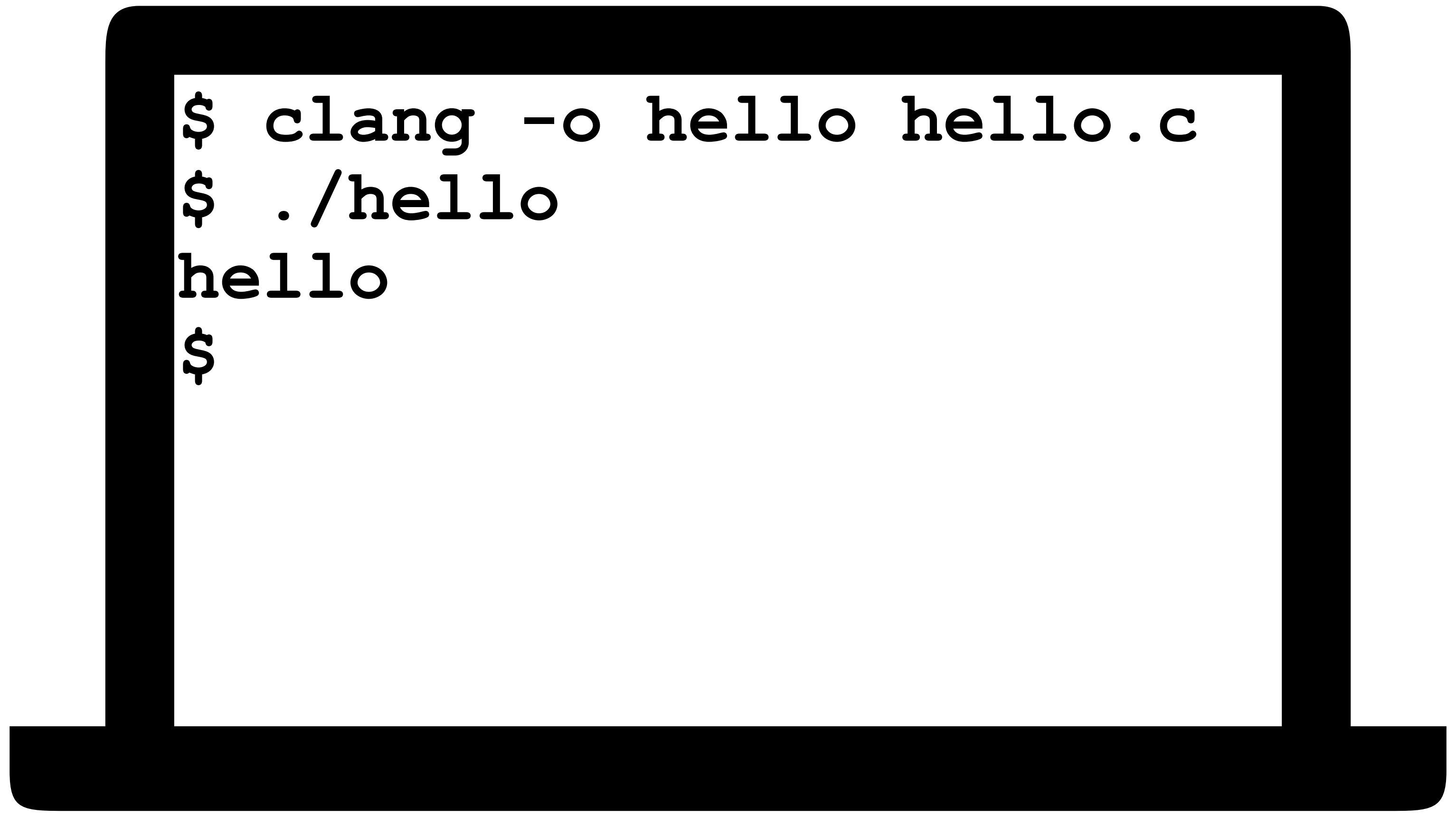
How about C?



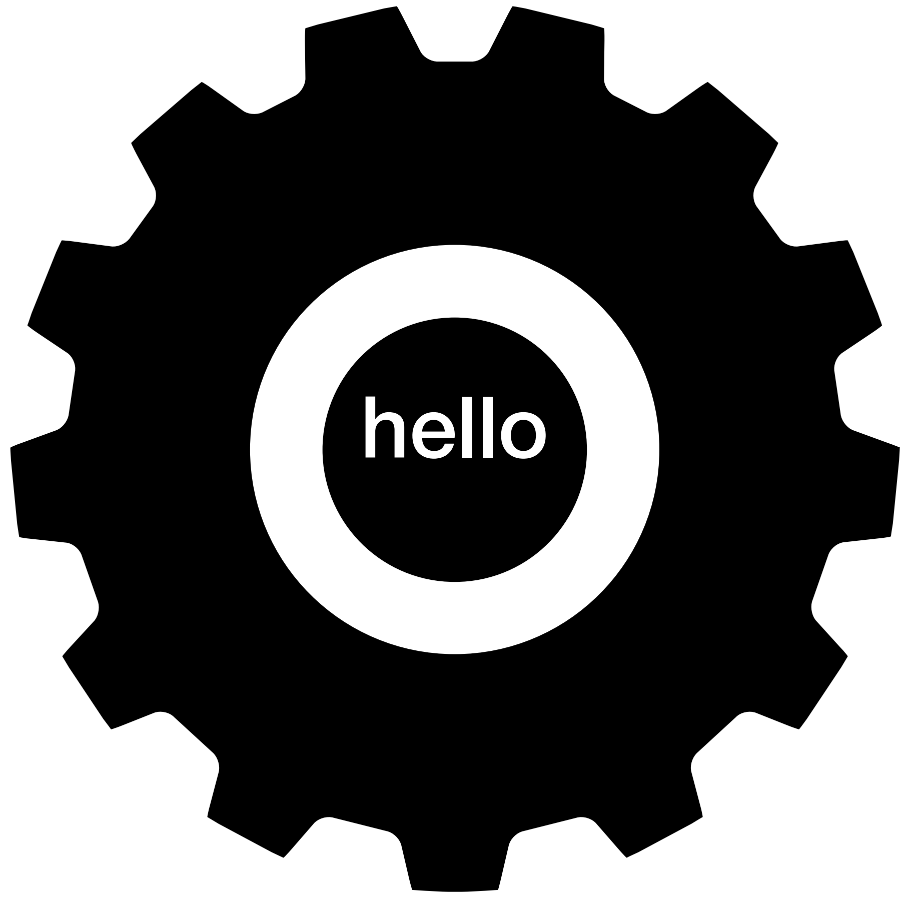
```
$ clang -o hello hello.c
$ ./hello
hello
```

How to Run C

How about C?

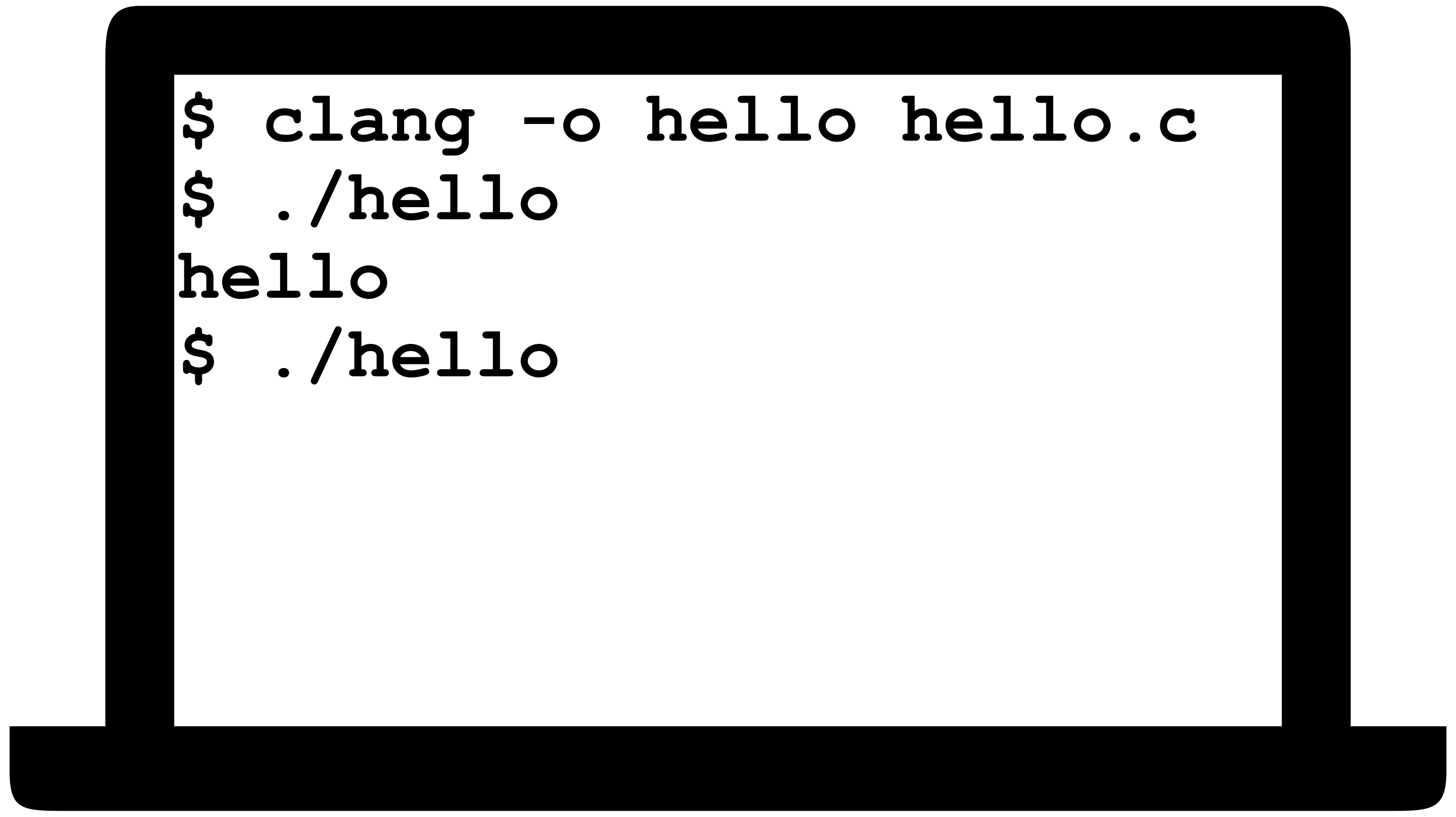


```
$ clang -o hello hello.c
$ ./hello
hello
$
```

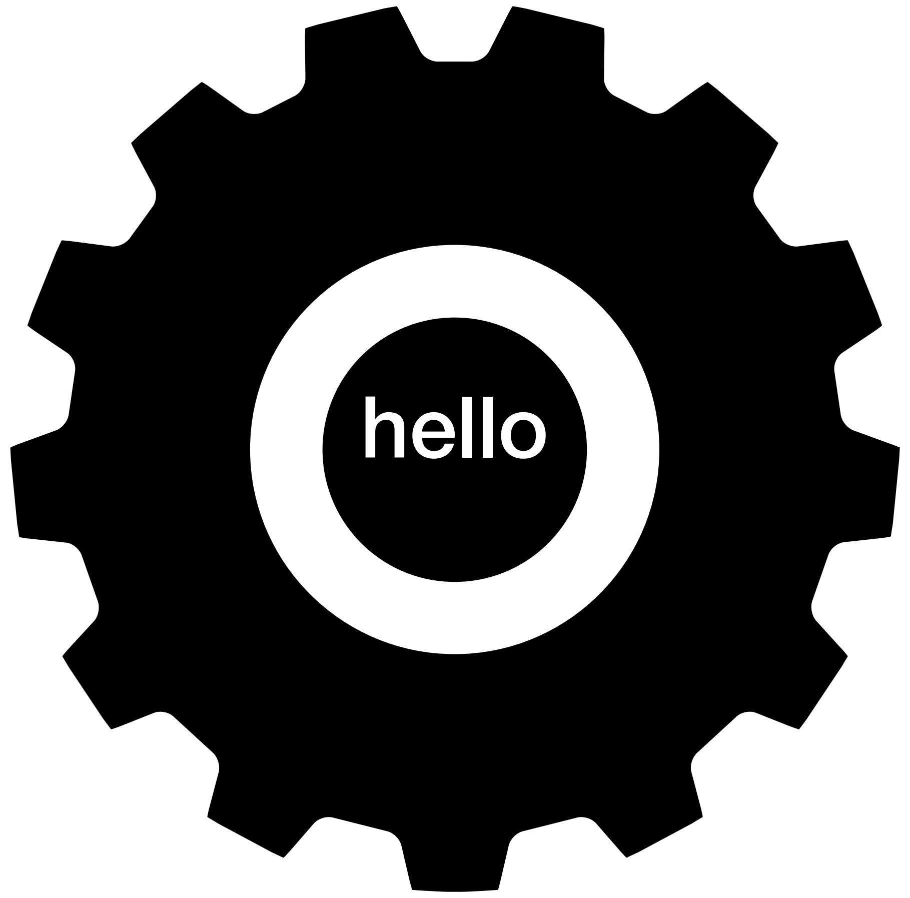


How to Run C

How about C?

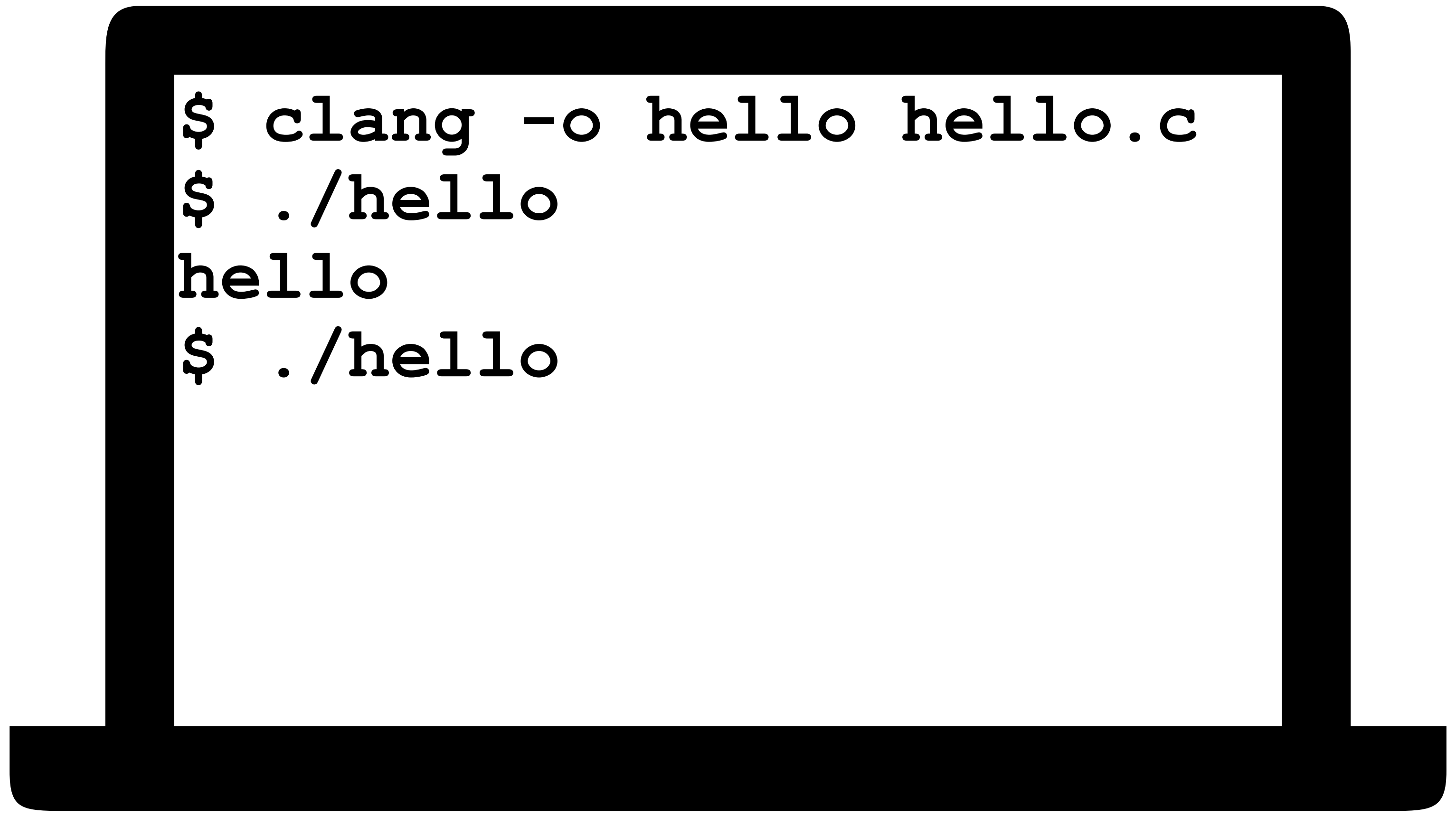


```
$ clang -o hello hello.c
$ ./hello
hello
$ ./hello
```

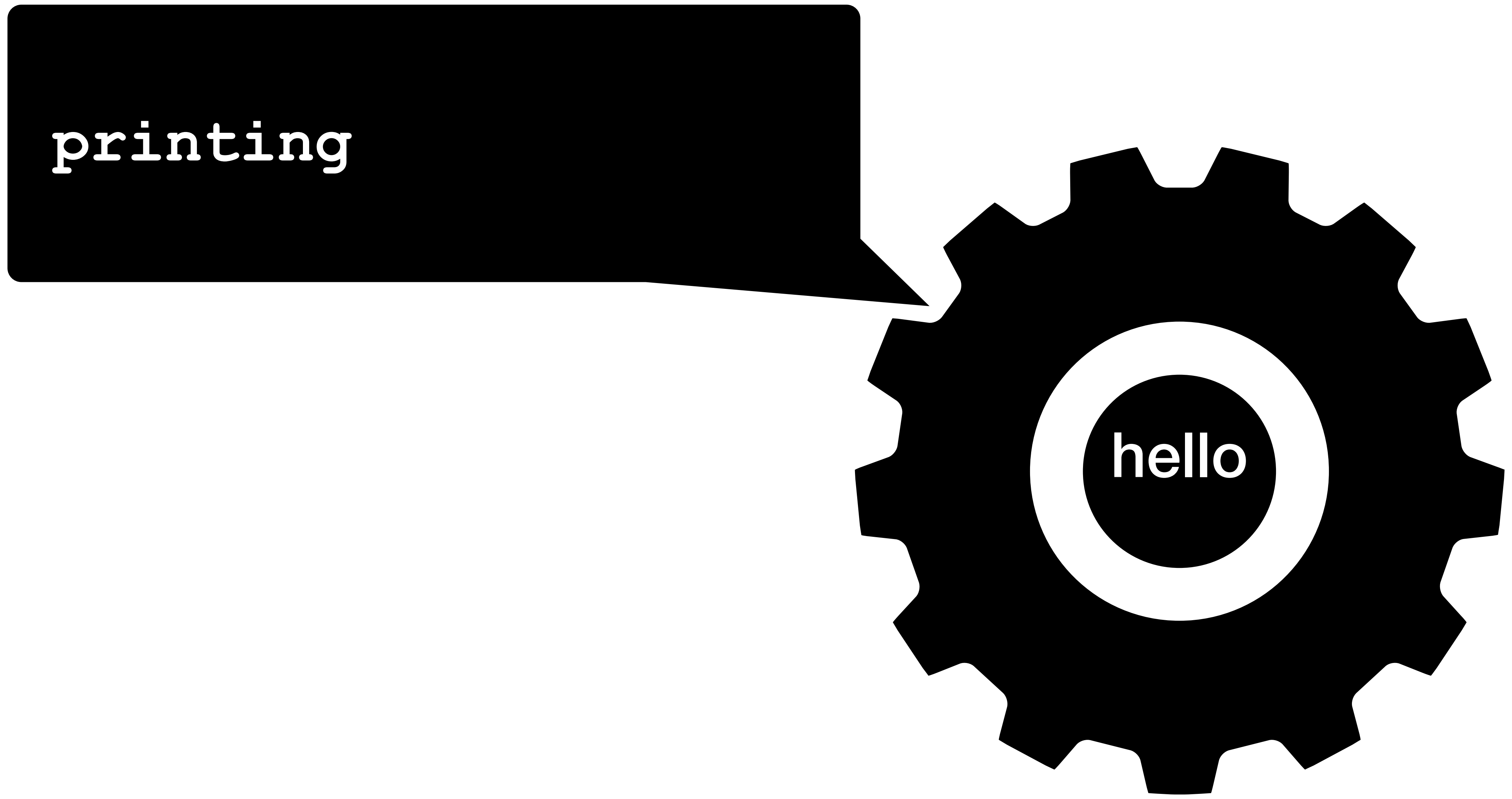


How to Run C

How about C?

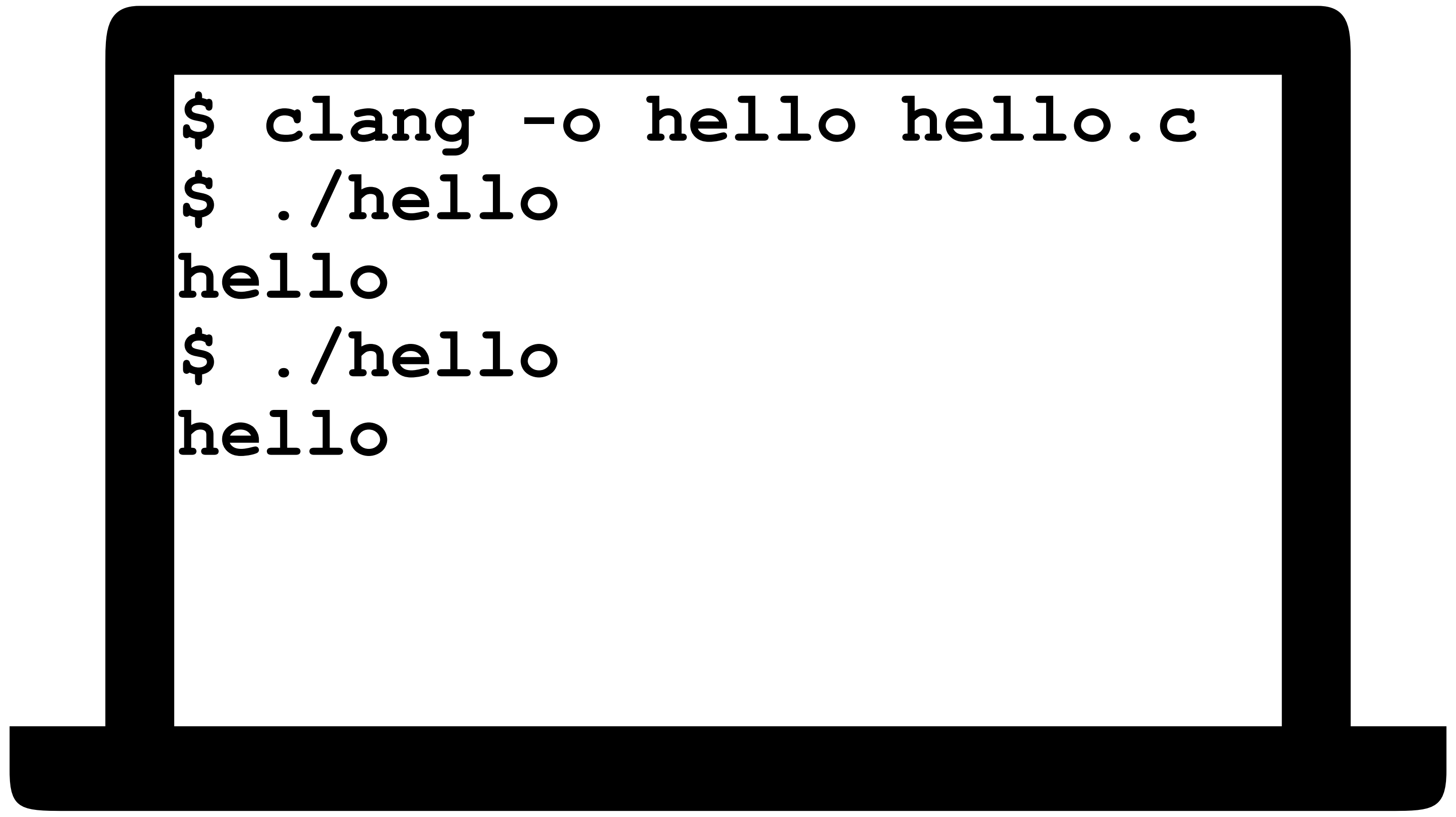


```
$ clang -o hello hello.c
$ ./hello
hello
$ ./hello
```



How to Run C

How about C?

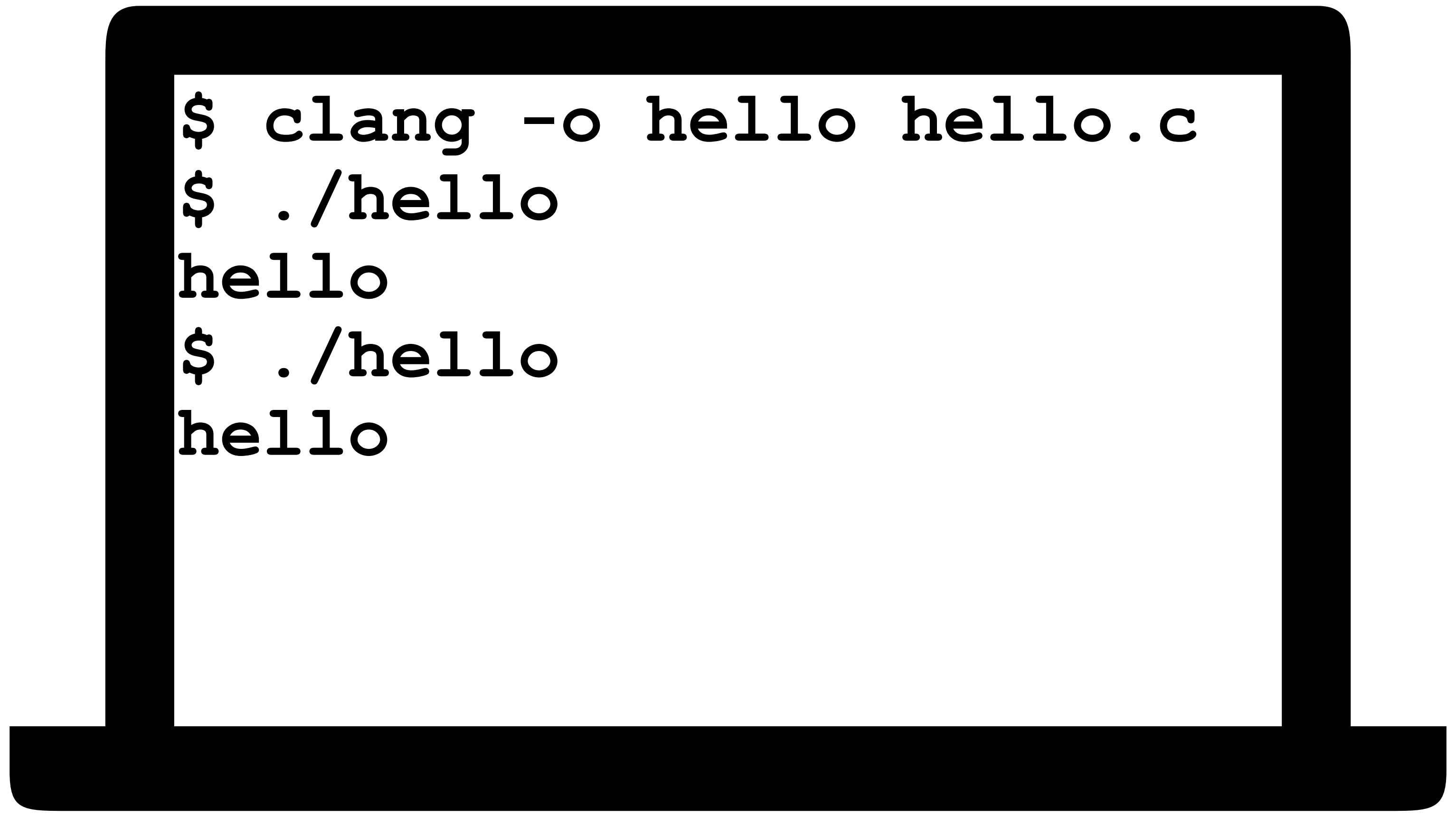


```
$ clang -o hello hello.c
$ ./hello
hello
$ ./hello
hello
```

printing
hello

How to Run C

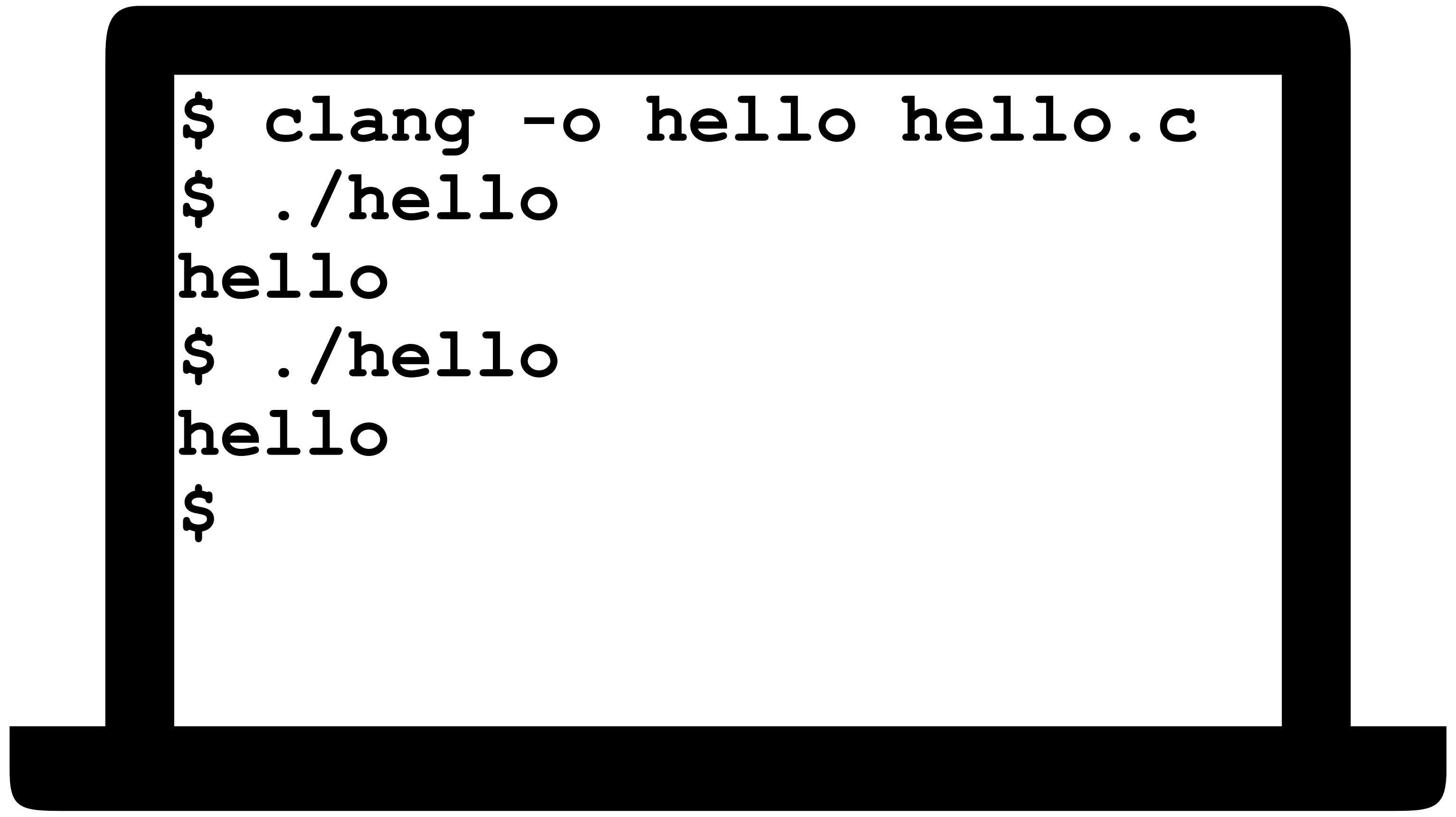
How about C?



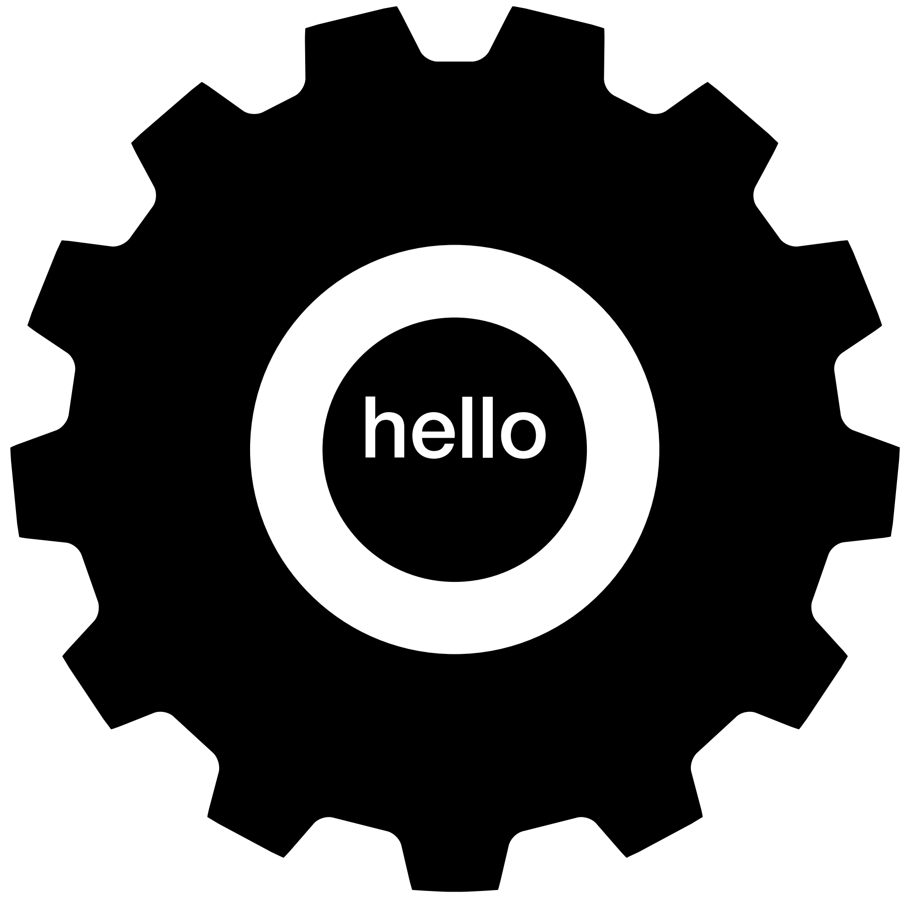
```
$ clang -o hello hello.c
$ ./hello
hello
$ ./hello
hello
```

How to Run C

How about C?



```
$ clang -o hello hello.c
$ ./hello
hello
$ ./hello
hello
$
```



How to Run C

How about C?

- `clang` translates your source code (text) into a file containing machine instructions
- to “compile the source code into an executable”
- you have a new executable; running that executable doesn’t involve clang anymore
- `clang` is a compiler

To-do

- Fill out the survey (if you haven’t already)
- Read the homepage of the course website
- Get familiar with the Resources page (also open to suggestions)
- HW0 is out, due next Tuesday