first light

prof. dr. nicolas meseth

1. your device
2. your assistant
3. getting ready
4. your first program
5. how our workshops work

part 1

your device



image: ai-generated from photos of the real device (gpt-image-2.5-sunburst)

the eye

a colour sensor, on the left



the eye

a colour sensor, on the left

the lamp

an RGB LED, on the right



the eye

a colour sensor, on the left

the lamp

an RGB LED, on the right

the wall

so the device does not see its own lamp



the eye

a colour sensor, on the left

the lamp

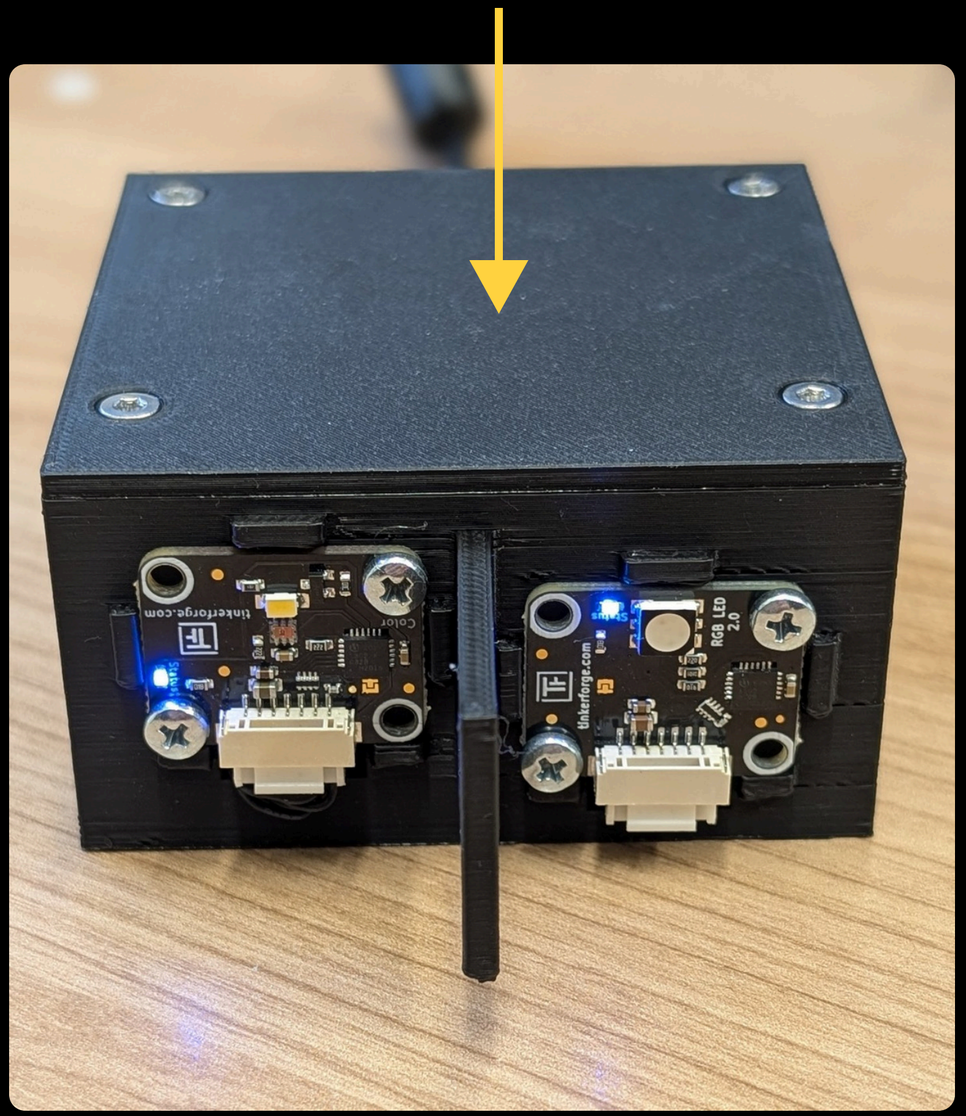
an RGB LED, on the right

the wall

so the device does not see its own lamp

inside

a Master Brick, connected to your laptop by USB



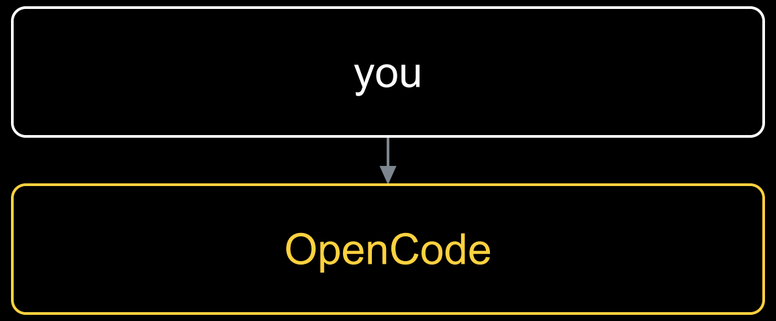
from your idea to the light

you

decide what should happen, and write code

your device

from your idea to the light

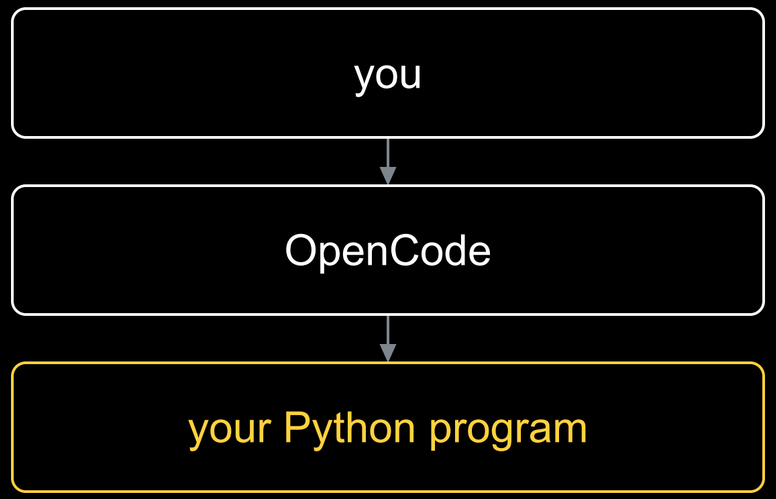


decide what should happen, and write code

your assistant helps you write and understand it

your device

from your idea to the light

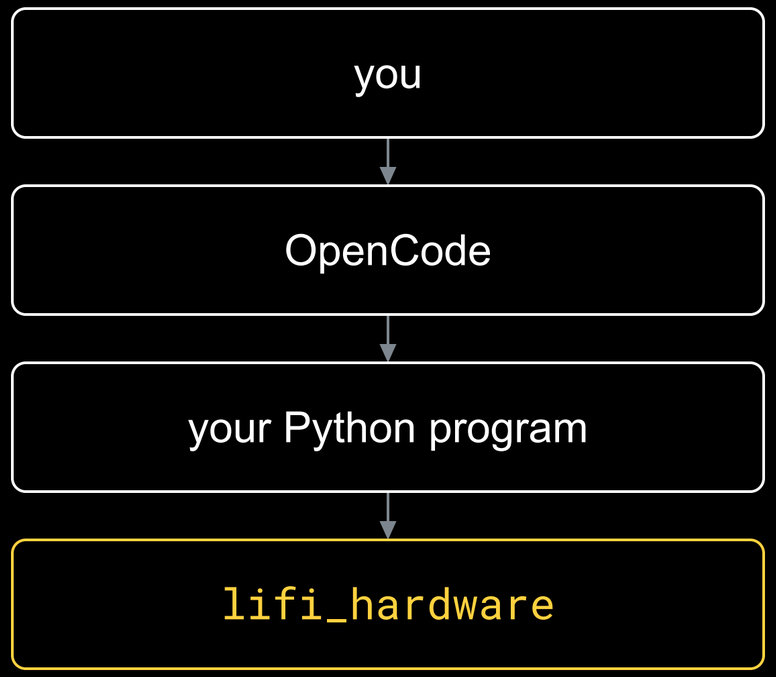


decide what should happen, and write code

your assistant helps you write and understand it

the code you write together, in `my-code`

from your idea to the light



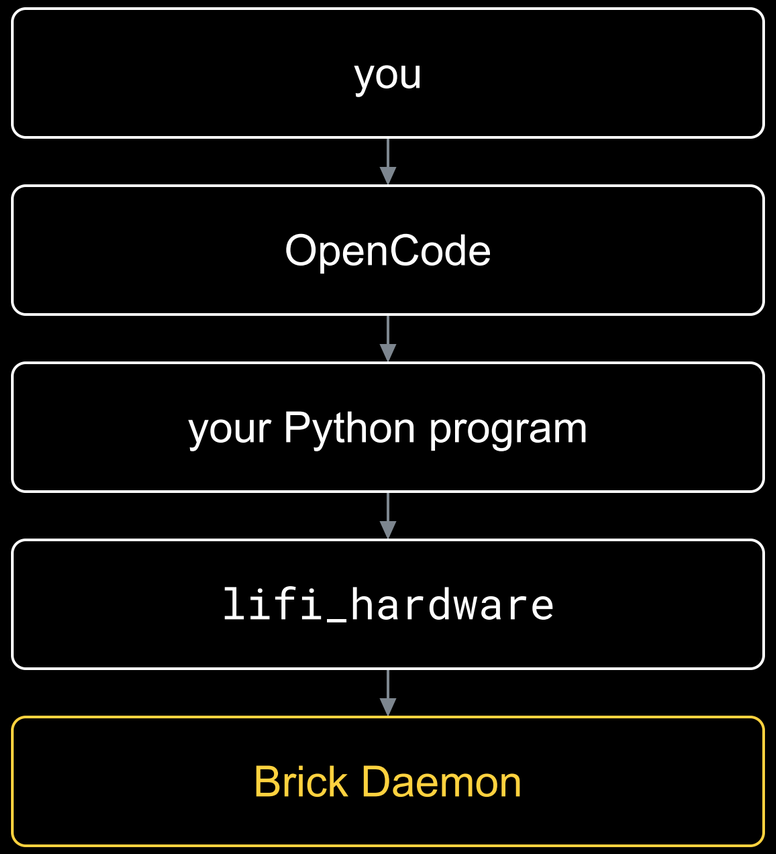
decide what should happen, and write code

your assistant helps you write and understand it

the code you write together, in `my-code`

the module that speaks the device's language

from your idea to the light



decide what should happen, and write code

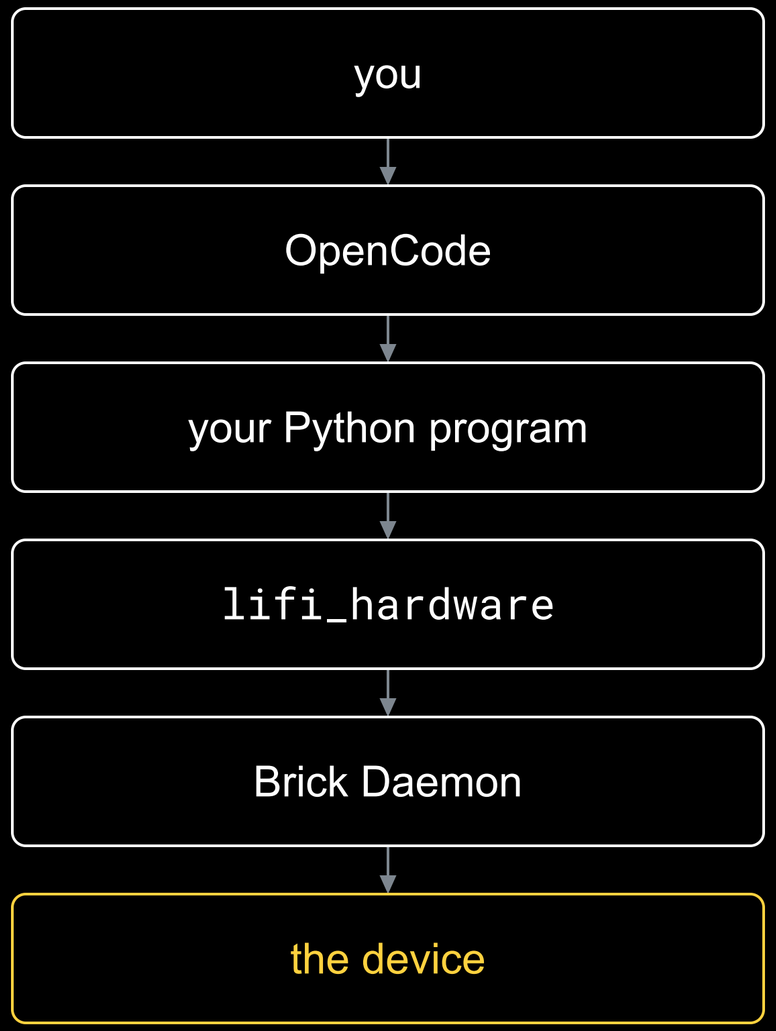
your assistant helps you write and understand it

the code you write together, in `my-code`

the module that speaks the device's language

keeps the USB connection open

from your idea to the light



decide what should happen, and write code

your assistant helps you write and understand it

the code you write together, in `my-code`

the module that speaks the device's language

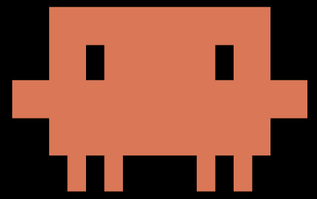
keeps the USB connection open

lamp and eye, on your table

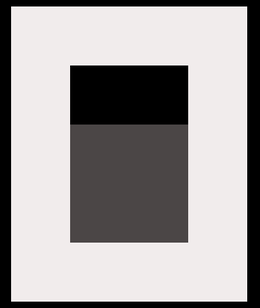
part 2

your assistant

do you know who these are?

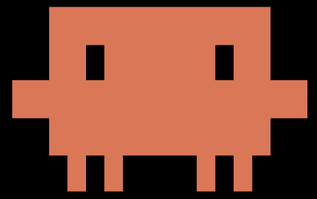






logos are trademarks of their owners.

do you know who these are?



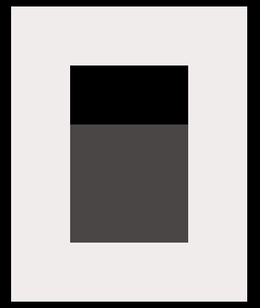
Claude Code

Anthropic



Codex

OpenAI



OpenCode

open source

logos are trademarks of their owners.

you know this already

a chat, like ChatGPT

one question, one answer

an agent, like OpenCode

one task, until it is done

you know this already

a chat, like ChatGPT	an agent, like OpenCode
one question, one answer	one task, until it is done
you paste in what it needs	it reads your files itself

you know this already

a chat, like ChatGPT	an agent, like OpenCode
one question, one answer	one task, until it is done
you paste in what it needs	it reads your files itself
you copy its code into a file	it writes the file itself

you know this already

a chat, like ChatGPT	an agent, like OpenCode
one question, one answer	one task, until it is done
you paste in what it needs	it reads your files itself
you copy its code into a file	it writes the file itself
you run it and report back	it runs it and reads the result

you know this already

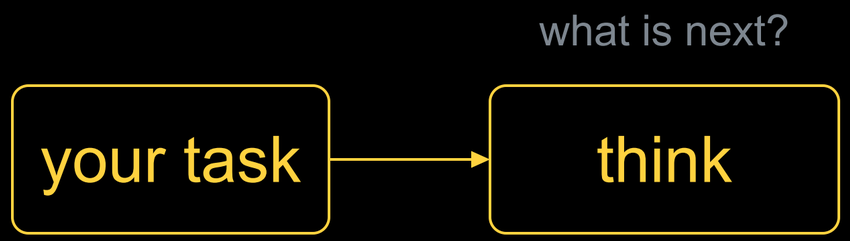
a chat, like ChatGPT	an agent, like OpenCode
one question, one answer	one task, until it is done
you paste in what it needs	it reads your files itself
you copy its code into a file	it writes the file itself
you run it and report back	it runs it and reads the result
you decide the next step	it tries again, until it works

you know this already

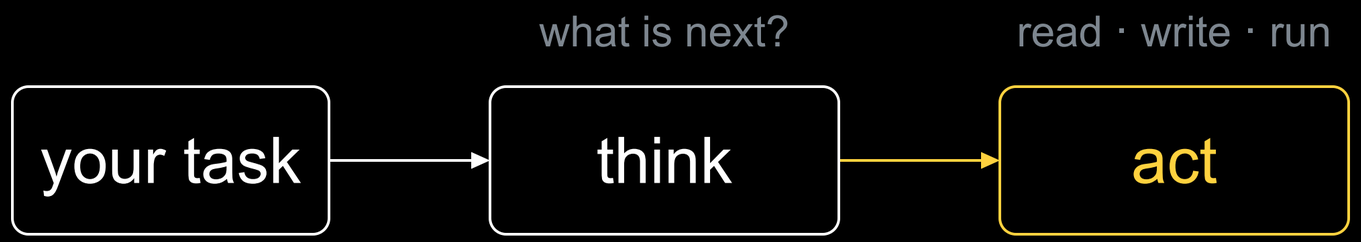
a chat, like ChatGPT	an agent, like OpenCode
one question, one answer	one task, until it is done
you paste in what it needs	it reads your files itself
you copy its code into a file	it writes the file itself
you run it and report back	it runs it and reads the result
you decide the next step	it tries again, until it works

the difference: it works in a loop, right where the work is.

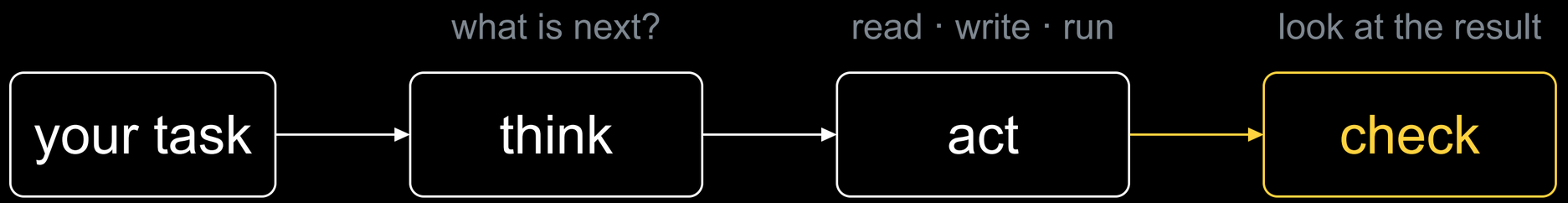
the agent loop



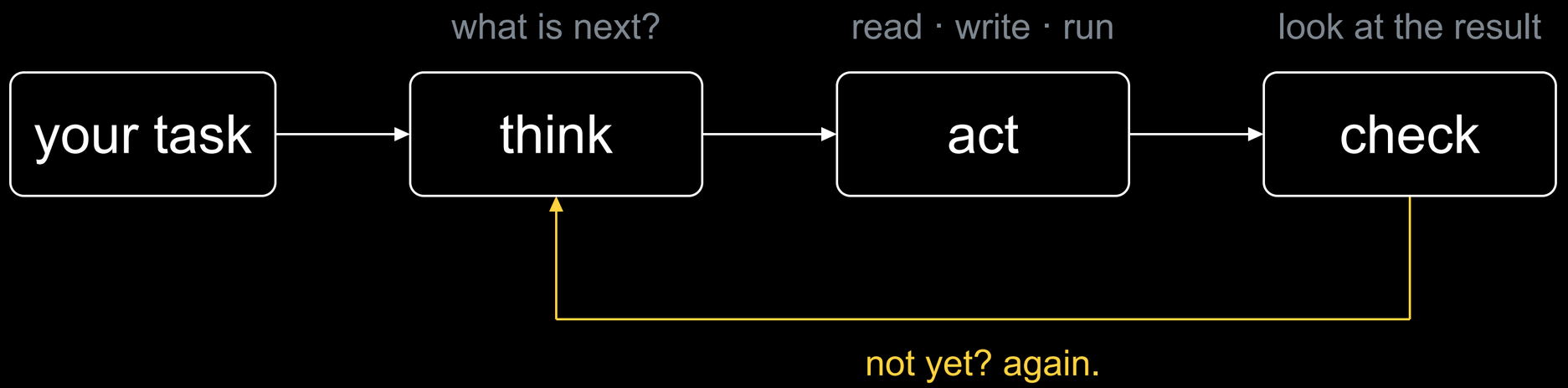
the agent loop



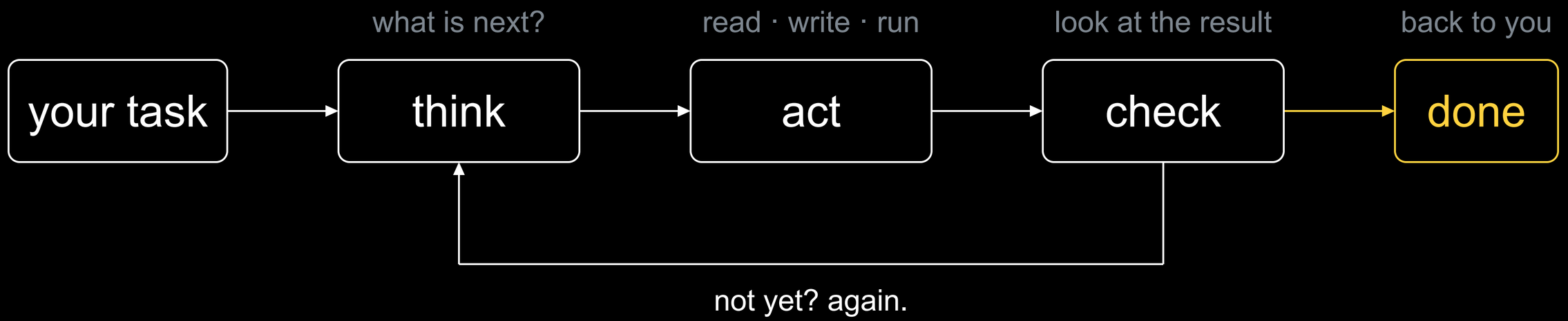
the agent loop

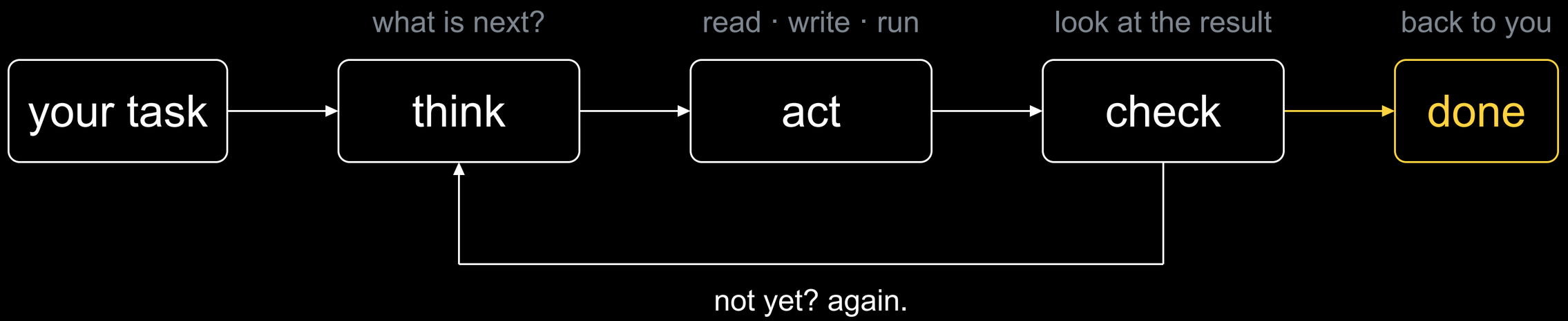


the agent loop



the agent loop





you give the task, and you check the result.

opencode

your assistant, set up for this course.

works on your laptop

in your course folder, with your files.

uses tools

reads and writes files, runs programs and commands.

knows the course

the website, the lecture notes, the module `lifi_hardware`.

it cannot see your device

the lamp has the last word.

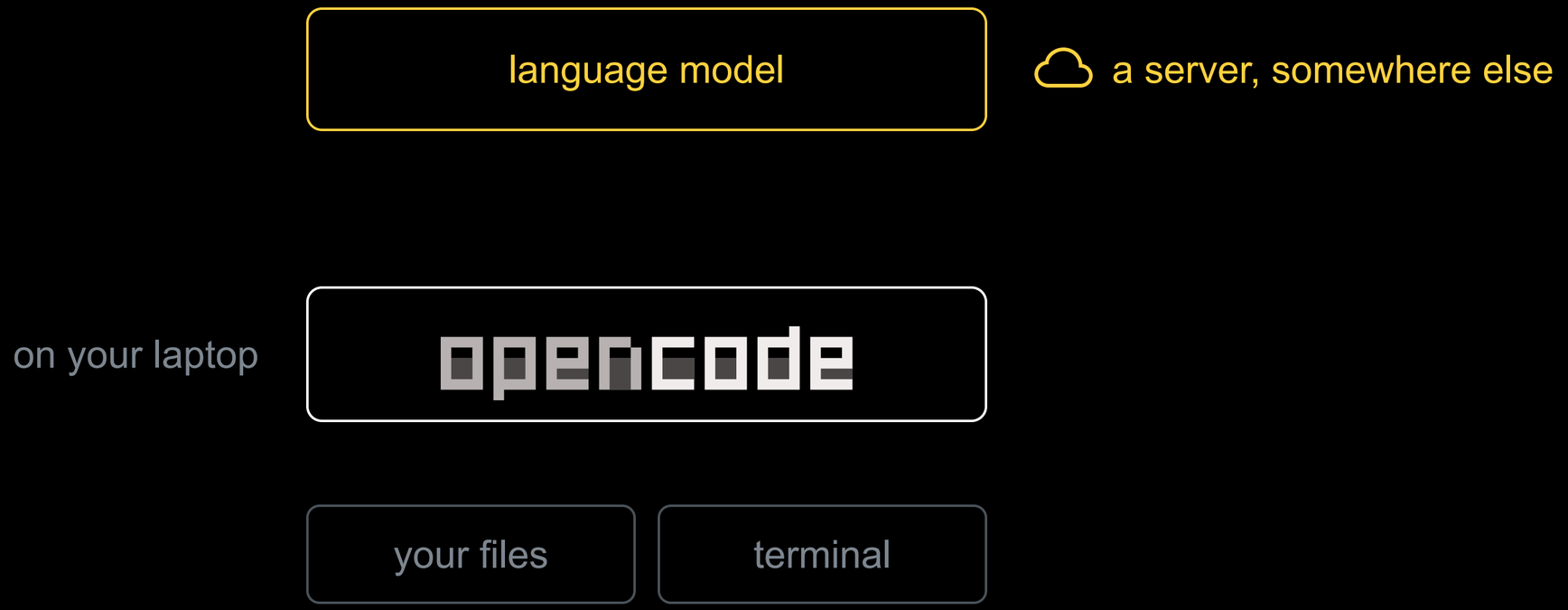
on your laptop



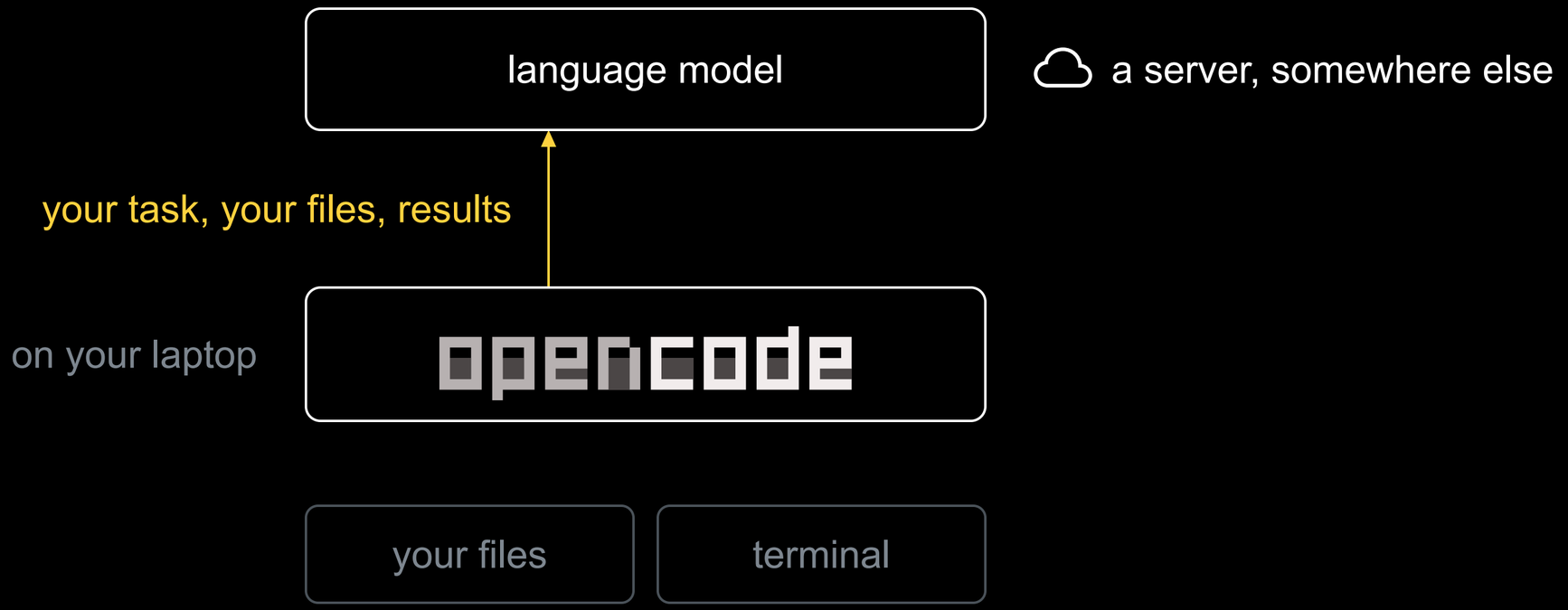
your files

terminal

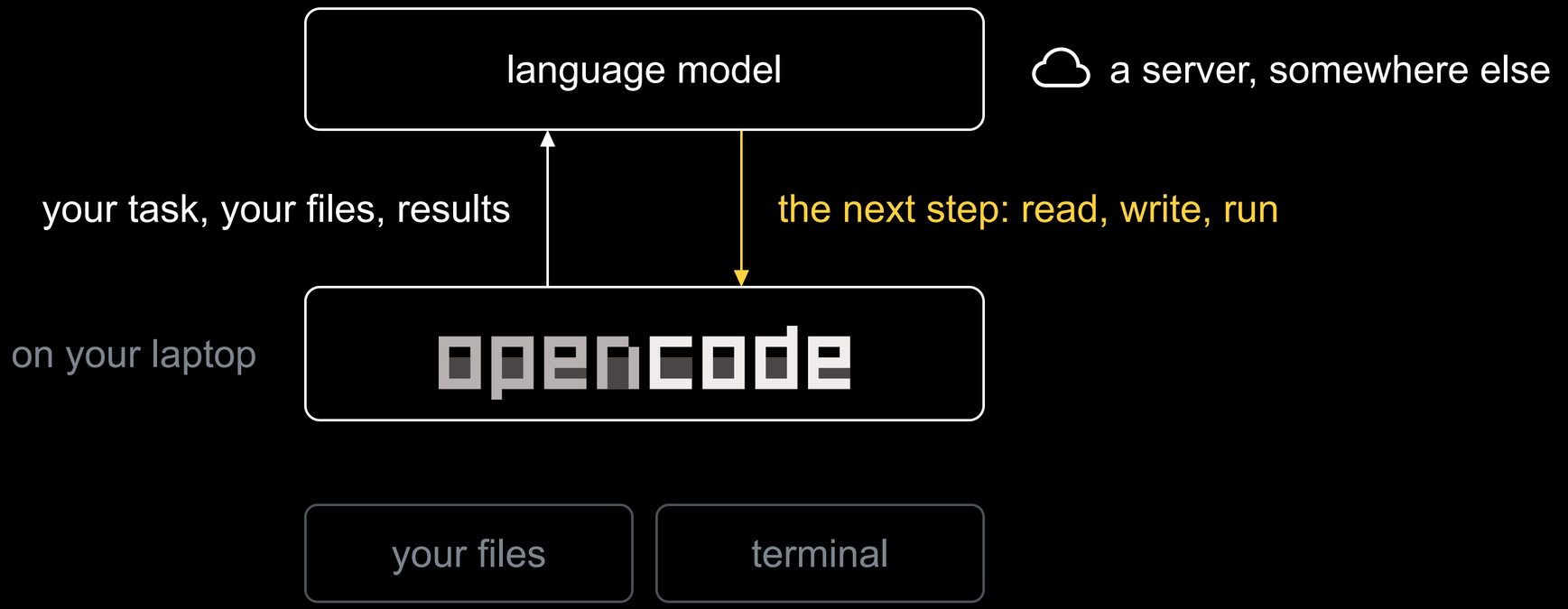
the program and the model



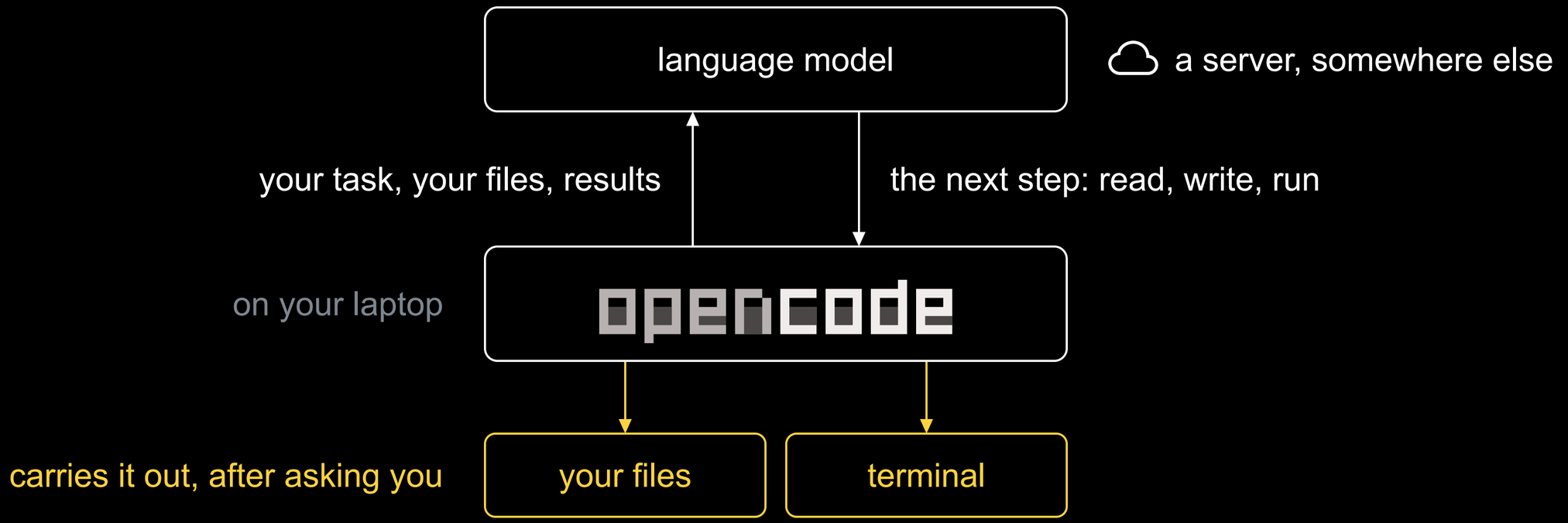
the program and the model



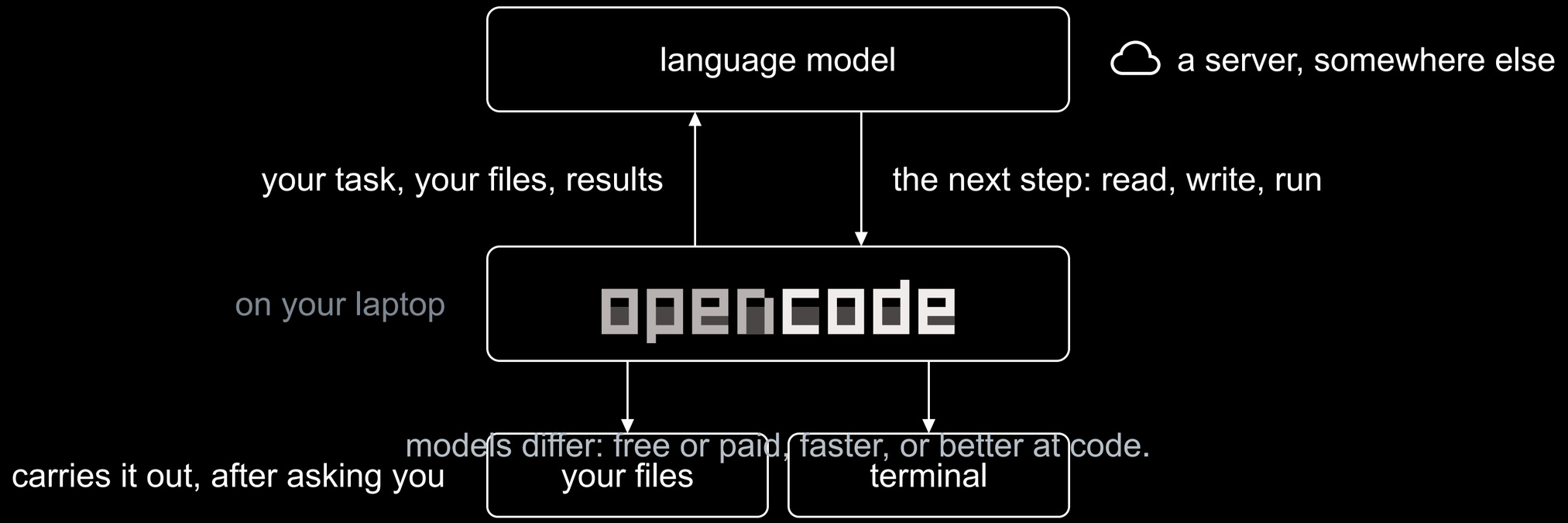
the program and the model



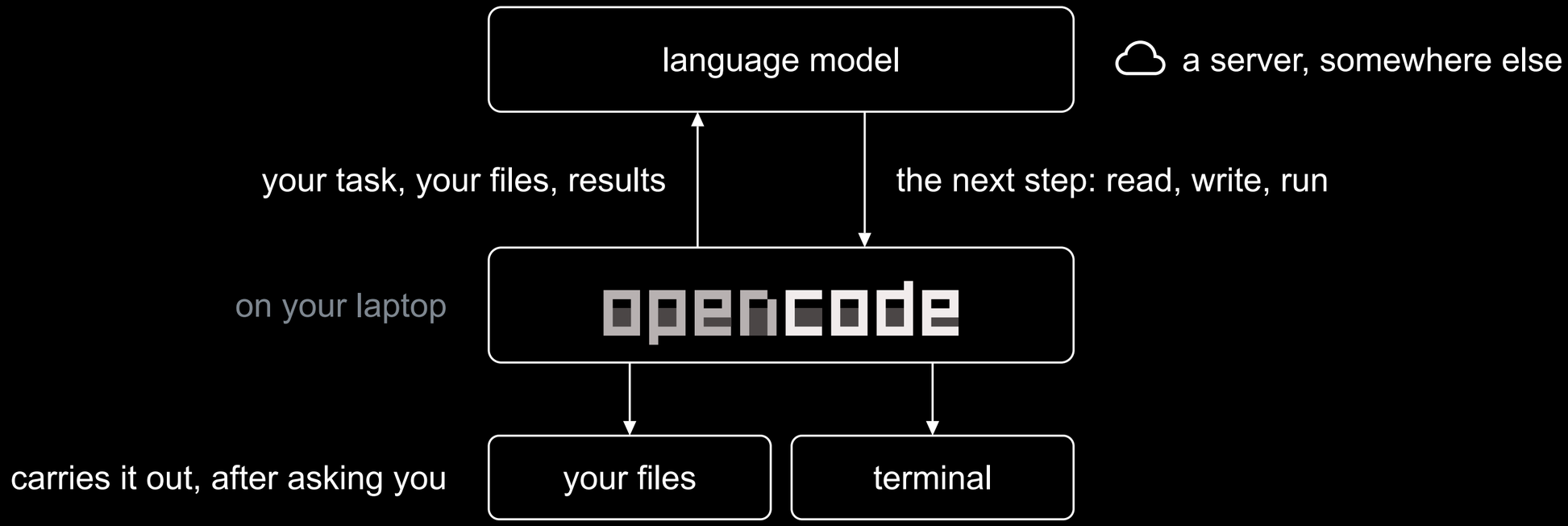
the program and the model



the program and the model



the program and the model



models differ: free or paid, faster, or better at code.

what you type travels to that server.



the course model

your course folder selects it. it needs the course key, which your course gives you.



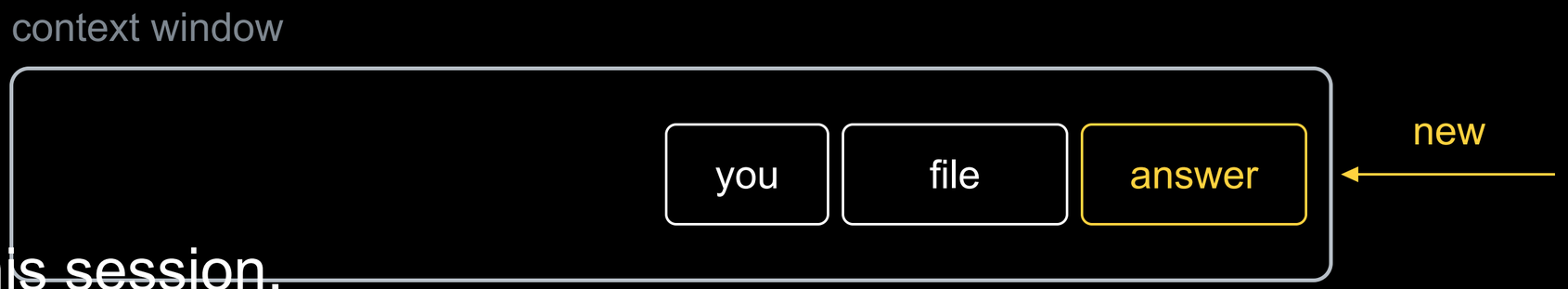
free models as a fallback

OpenCode comes with free models. they work without a key.



a key is only for OpenCode

never into a chat, a file or a screenshot. whoever has a key works at its owner's expense.

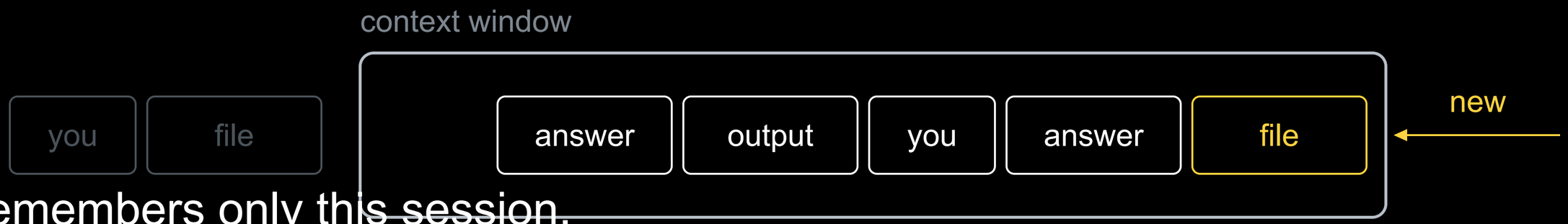


it remembers only this session.



it remembers only this session.

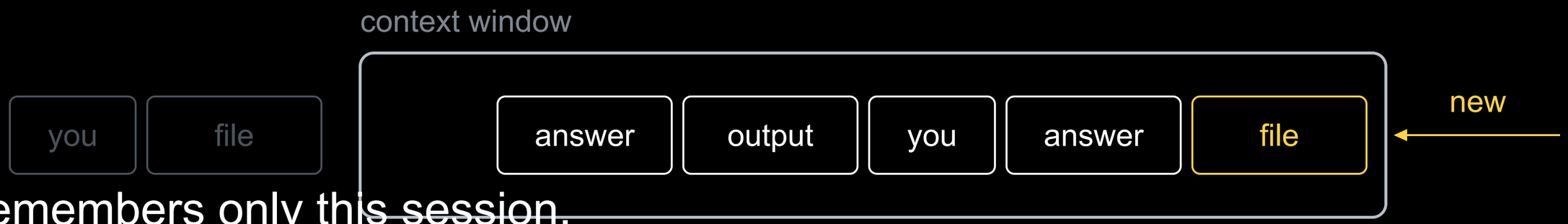
the window has a size, counted in tokens.



it remembers only this session.

the window has a size, counted in tokens.

when it is full, it compacts: old parts shrink to a summary.



it remembers only this session.

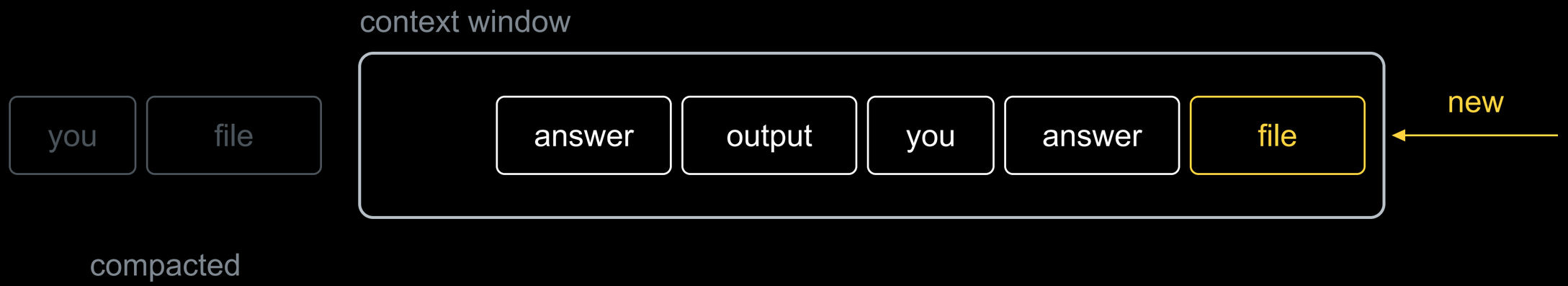
~~compacted~~

the window has a size, counted in tokens.

when it is full, it compacts: old parts shrink to a summary.

start a new session for every new task.

its memory: the context window



it remembers only this session.
the window has a size, counted in tokens.
when it is full, it compacts: old parts shrink to a summary.

start a new session for every new task.

what should last goes into a file it reads at every start, like its course instructions.

it asks first

changes and commands wait for your ok. read, then allow.

it asks first

changes and commands wait for your ok. read, then allow.

it works in your course folder

keep private files out of it.

it asks first

changes and commands wait for your ok. read, then allow.

it works in your course folder

keep private files out of it.

it can be wrong

and it sounds just as sure. run it, measure it, check it.

it asks first

changes and commands wait for your ok. read, then allow.

it works in your course folder

keep private files out of it.

it can be wrong

and it sounds just as sure. run it, measure it, check it.

it can make mistakes

one ok too quick, and a file is gone. keep a backup of what matters.

it asks first

changes and commands wait for your ok. read, then allow.

it works in your course folder

keep private files out of it.

it can be wrong

and it sounds just as sure. run it, measure it, check it.

it can make mistakes

one ok too quick, and a file is gone. keep a backup of what matters.

you can stop it

any time, with the stop button.

it asks first

changes and commands wait for your ok. read, then allow.

it works in your course folder

keep private files out of it.

it can be wrong

and it sounds just as sure. run it, measure it, check it.

it can make mistakes

one ok too quick, and a file is gone. keep a backup of what matters.

you can stop it

any time, with the stop button.

your chat leaves your laptop

no passwords, no keys, no personal data in the chat.

it asks first

changes and commands wait for your ok. read, then allow.

it works in your course folder

keep private files out of it.

it can be wrong

and it sounds just as sure. run it, measure it, check it.

it can make mistakes

one ok too quick, and a file is gone. keep a backup of what matters.

you can stop it

any time, with the stop button.

your chat leaves your laptop

no passwords, no keys, no personal data in the chat.

you decide what it may do.

part 3

getting ready

1 install OpenCode Desktop.

step by step: docs.lifi-project.de/software

1. install OpenCode Desktop.
2. download the course folder as a zip and unpack it.

step by step: docs.lifi-project.de/software

1. install OpenCode Desktop.
2. download the course folder as a zip and unpack it.
3. open the folder in OpenCode and start a session.

step by step: docs.lifi-project.de/software

/onboarding



image: ai-generated (gpt-image-2.5-flare)

/onboarding

it asks you three questions.



image: ai-generated (gpt-image-2.5-flare)

/onboarding

it asks you three questions.

it installs Python, `lifi_hardware`, VS Code, Brick Daemon and Brick Viewer.



image: ai-generated (gpt-image-2.5-flare)

/onboarding

it asks you three questions.

it installs Python, `lifi_hardware`, VS Code, Brick Daemon and Brick Viewer.

it checks every step before the next one.



image: ai-generated (gpt-image-2.5-flare)

/onboarding

it asks you three questions.

it installs Python, `lifi_hardware`, VS Code, Brick Daemon and Brick Viewer.

it checks every step before the next one.

no device yet? it skips the device test.



image: ai-generated (gpt-image-2.5-flare)

your laptop is ready for your device.

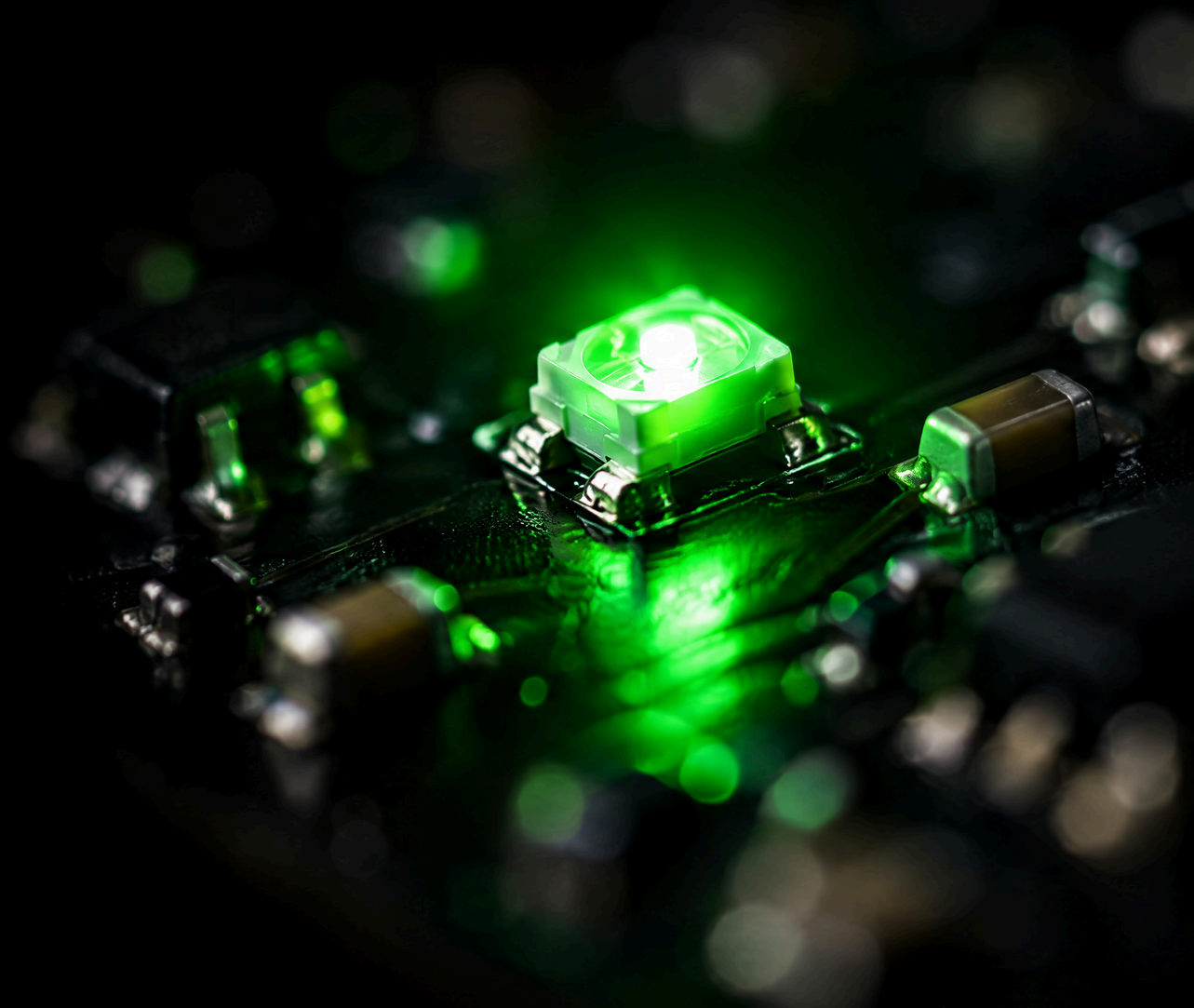


image: ai-generated (gpt-image-2.5-flare)

your laptop is ready for your device.

once you plug it in, this LED lights up green.

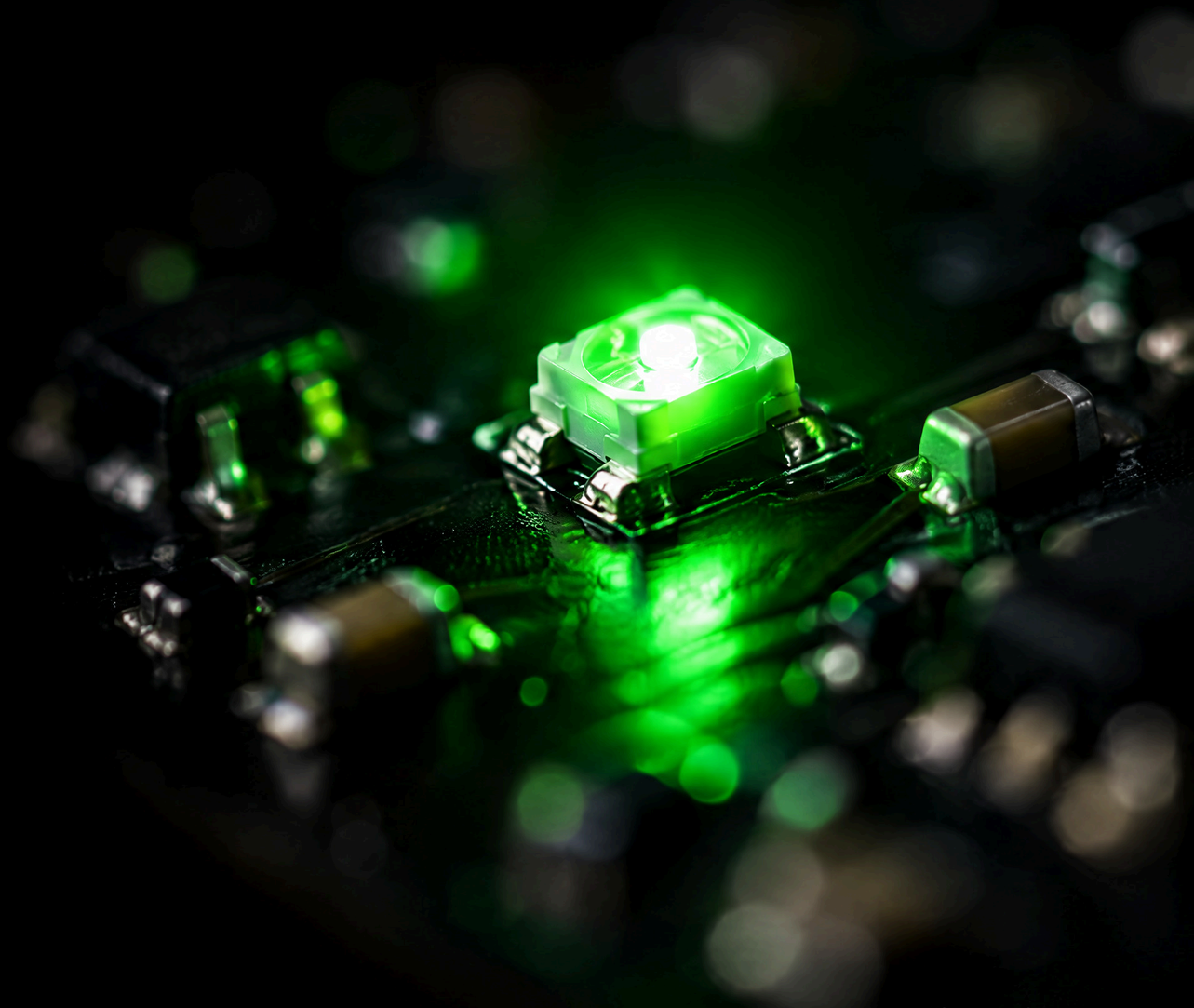


image: ai-generated (gpt-image-2.5-flare)

part 4

your first program

in VS Code, create a new file in `my-code`: `hello.py`

in VS Code, create a new file in `my-code`: `hello.py`
the ending `.py` says: this text is Python.

in VS Code, create a new file in `my-code`: `hello.py`
the ending `.py` says: this text is Python.

a program is plain text,
until Python runs it.

```
print("Hello, world!")
```

```
print("Hello, world!")
```

save it. then open a terminal (terminal › new terminal) and type:

```
cd my-code
python hello.py
```

on a mac: `python3` instead of `python`.

```
print("Hello, world!")
```

save it. then open a terminal (terminal › new terminal) and type:

```
cd my-code
python hello.py
```

```
Hello, world!
```

on a mac: `python3` instead of `python`.

```
name = input("What is your name? ")
print("Hello, " + name + "!")
```

```
name = input("What is your name? ")
print("Hello, " + name + "!")
```

```
What is your name? Ada
Hello, Ada!
```

```
name = input("What is your name? ")
print("Hello, " + name + "!")
```

```
What is your name? Ada
Hello, Ada!
```

`input()` waits until you press enter, and hands back what you typed.

```
name = input("What is your name? ")
print("Hello, " + name + "!")
```

```
What is your name? Ada
Hello, Ada!
```

`input()` waits until you press enter, and hands back what you typed.

`name` is a variable: a name for that value, so you can use it again.

the task

write a timer for the terminal. it asks how many seconds, then counts down, one line per second.

a coloured lamp shows how it stands: green while it runs, yellow in the last five seconds, red when time is up.



image: ai-generated (gpt-image-2.5-flare)

1 get the number of seconds from the user

1. get the number of seconds from the user
2. check if the input is valid

1. get the number of seconds from the user
2. check if the input is valid
3. count down each second

1. get the number of seconds from the user
2. check if the input is valid
3. count down each second
4. show a coloured lamp

1. get the number of seconds from the user
2. check if the input is valid
3. count down each second
4. show a coloured lamp
5. decide which colour

1	get the number of seconds from the user	`input()`
2	check if the input is valid	`while`
3	count down each second	`for`
4	show a coloured lamp	`def`
5	decide which colour	`if / else`

cut it into steps

1	get the number of seconds from the user	`input()`
2	check if the input is valid	`while`
3	count down each second	`for`
4	show a coloured lamp	`def`
5	decide which colour	`if / else`

more on this: cutting problems

one step at a time, and run it after each one.

ask it like this:

“explain what `input()` hands back.”

ask it like this:

“explain what `input()` hands back.”
“read my `timer.py`. is step 2 right?”

ask it like this:

“explain what `input()` hands back.”
“read my `timer.py`. is step 2 right?”
“run it with five as input. what goes wrong?”

and your assistant?

ask it like this:

“explain what `input()` hands back.”
“read my `timer.py`. is step 2 right?”
“run it with five as input. what goes wrong?”
“give me a hint for step 3, not the code.”

ask it like this:

“explain what `input()` hands back.”
“read my `timer.py`. is step 2 right?”
“run it with five as input. what goes wrong?”
“give me a hint for step 3, not the code.”

let it explain, not only write.

1	get the number of seconds from the user	`input()`
2	check if the input is valid	`while`
3	count down each second	`for`
4	show a coloured lamp	`def`
5	decide which colour	`if / else`

```
text = input("How many seconds? ")
seconds = int(text)
print("Timer:", seconds, "seconds")
```

```
text = input("How many seconds? ")
seconds = int(text)
print("Timer:", seconds, "seconds")
```

`input()` always hands back text, even if you type 20.

```
text = input("How many seconds? ")
seconds = int(text)
print("Timer:", seconds, "seconds")
```

`input()` always hands back text, even if you type 20.

`int()` turns the text into a whole number, so you can count with it.

what if someone types five?

```
How many seconds? five

Traceback (most recent call last):
  File "…\my-code\timer.py", line 2, in <module>
    seconds = int(text)
ValueError: invalid literal for int() with base 10: 'five'
```

what if someone types five?

```
How many seconds? five

Traceback (most recent call last):
  File "…\my-code\timer.py", line 2, in <module>
    seconds = int(text)
ValueError: invalid literal for int() with base 10: 'five'
```

read it from the bottom: the last line says what went wrong.

```
How many seconds? five

Traceback (most recent call last):
  File "…\my-code\timer.py", line 2, in <module>
    seconds = int(text)
ValueError: invalid literal for int() with base 10: 'five'
```

read it from the bottom: the last line says what went wrong.
above it: where. the file, line 2, and the line itself.

what if someone types five?

```
How many seconds? five

Traceback (most recent call last):
  File "…\my-code\timer.py", line 2, in <module>
    seconds = int(text)
ValueError: invalid literal for int() with base 10: 'five'
```

read it from the bottom: the last line says what went wrong.
above it: where. the file, line 2, and the line itself.

stuck? ask your assistant.

it can run your program itself and read the error.

✓	get the number of seconds from the user	`input()`
2	check if the input is valid	`while`
3	count down each second	`for`
4	show a coloured lamp	`def`
5	decide which colour	`if / else`

```
text = input("How many seconds? ")
while not text.isdigit():
    text = input("Digits only, please: ")
seconds = int(text)
```

```
text = input("How many seconds? ")
while not text.isdigit():
    text = input("Digits only, please: ")
seconds = int(text)
```

`text.isdigit()` is true if the text has only digits.

```
text = input("How many seconds? ")
while not text.isdigit():
    text = input("Digits only, please: ")
seconds = int(text)
```

`text.isdigit()` is true if the text has only digits.

`while` asks again, as long as the answer is not a number.

✓	get the number of seconds from the user	`input()`
✓	check if the input is valid	`while`
3	count down each second	`for`
4	show a coloured lamp	`def`
5	decide which colour	`if / else`

```
import time

for remaining in range(seconds, 0, -1):
    print(remaining, "s")
    time.sleep(1)

print("time is up!")
```

```
import time

for remaining in range(seconds, 0, -1):
    print(remaining, "s")
    time.sleep(1)

print("time is up!")
```

`range(seconds, 0, -1)` counts down from `seconds` to 1.

```
import time

for remaining in range(seconds, 0, -1):
    print(remaining, "s")
    time.sleep(1)

print("time is up!")
```

`range(seconds, 0, -1)` counts down from `seconds` to 1.

the indented lines repeat, once for every number. `time.sleep(1)` waits one second.

✓	get the number of seconds from the user	`input()`
✓	check if the input is valid	`while`
✓	count down each second	`for`
4	show a coloured lamp	`def`
5	decide which colour	`if / else`

```
def lamp(r, g, b):
    on = f"\033[48;2;{r};{g};{b}m"
    off = "\033[0m"
    print(on + " " * 12 + off, end=" ")

lamp(0, 200, 0)
print("running")
```

```
def lamp(r, g, b):
    on = f"\033[48;2;{r};{g};{b}m"
    off = "\033[0m"
    print(on + " " * 12 + off, end=" ")

lamp(0, 200, 0)
print("running")
```

`def` gives a piece of code a name. `r`, g and b are its parameters.

```
def lamp(r, g, b):
    on = f"\033[48;2;{r};{g};{b}m"
    off = "\033[0m"
    print(on + " " * 12 + off, end=" ")

lamp(0, 200, 0)
print("running")
```

`def` gives a piece of code a name. `r`, g and b are its parameters.

you fill in the values when you call it: red, green, blue, each 0 to 255.

```
def lamp(r, g, b):
    on = f"\033[48;2;{r};{g};{b}m"
    off = "\033[0m"
    print(on + " " * 12 + off, end=" ")

lamp(0, 200, 0)
print("running")
```

`def` gives a piece of code a name. `r`, g and b are its parameters.

you fill in the values when you call it: red, green, blue, each 0 to 255.

the strange codes inside? you do not have to understand them.

a function hides the details.

a function hides the details.

you need to know what `lamp()` does, not how.

a function hides the details.

you need to know what `lamp()` does, not how.
`print()` and `input()` are functions too. did you miss their insides?

a function hides the details.

you need to know what `lamp()` does, not how.
`print()` and `input()` are functions too. did you miss their insides?
`led.set_color()` hides a whole device behind one line.

a function hides the details.

you need to know what `lamp()` does, not how.
`print()` and `input()` are functions too. did you miss their insides?
`led.set_color()` hides a whole device behind one line.

curious? open it up with your assistant.

✓	get the number of seconds from the user	`input()`
✓	check if the input is valid	`while`
✓	count down each second	`for`
✓	show a coloured lamp	`def`
5	decide which colour	`if / else`

```
for remaining in range(seconds, 0, -1):
    if remaining <= 5:
        lamp(255, 200, 0)   # yellow
    else:
        lamp(0, 200, 0)     # green
    print(remaining, "s")
    time.sleep(1)

lamp(255, 0, 0)             # red
print("time is up!")
```

```
for remaining in range(seconds, 0, -1):
    if remaining <= 5:
        lamp(255, 200, 0)   # yellow
    else:
        lamp(0, 200, 0)     # green
    print(remaining, "s")
    time.sleep(1)

lamp(255, 0, 0)             # red
print("time is up!")
```

`if` the condition is true, the first block runs. otherwise, the `else` block.

✓	get the number of seconds from the user	`input()`
✓	check if the input is valid	`while`
✓	count down each second	`for`
✓	show a coloured lamp	`def`
✓	decide which colour	`if / else`

```
import time

def lamp(r, g, b):
    on = f"\033[48;2;{r};{g};{b}m"
    off = "\033[0m"
    print(on + " " * 12 + off, end=" ")

text = input("How many seconds? ")
while not text.isdigit():
    text = input("Digits only, please: ")
```

the button copies all 23 lines. paste them into `timer.py` and read them there.

part 5

how our workshops work

1 your assistant

stuck? ask in this order

1. your assistant
2. your partner

stuck? ask in this order

1. your assistant
2. your partner
3. the team next to you

stuck? ask in this order

1. your assistant
2. your partner
3. the team next to you
4. your lecturer

say what you wanted to happen.
let it run the program, or send it a screenshot.
ask why, not only how.

- [ ] OpenCode installed, course folder open
- [ ] assistant answers
- [ ] `/onboarding` done
- [ ] `hello.py` runs
- [ ] timer counts down
- [ ] timer shows its colours

before challenge 0:

form a team of two.

finish the setup and your timer.

read “the project” and “challenge 0”.

with your device, `lamp()` becomes `led.set_color()`.



image: ai-generated (gpt-image-2.5-flare)