

CODE IS MORE THAN THE FINAL ANSWER.

Make code make sense.

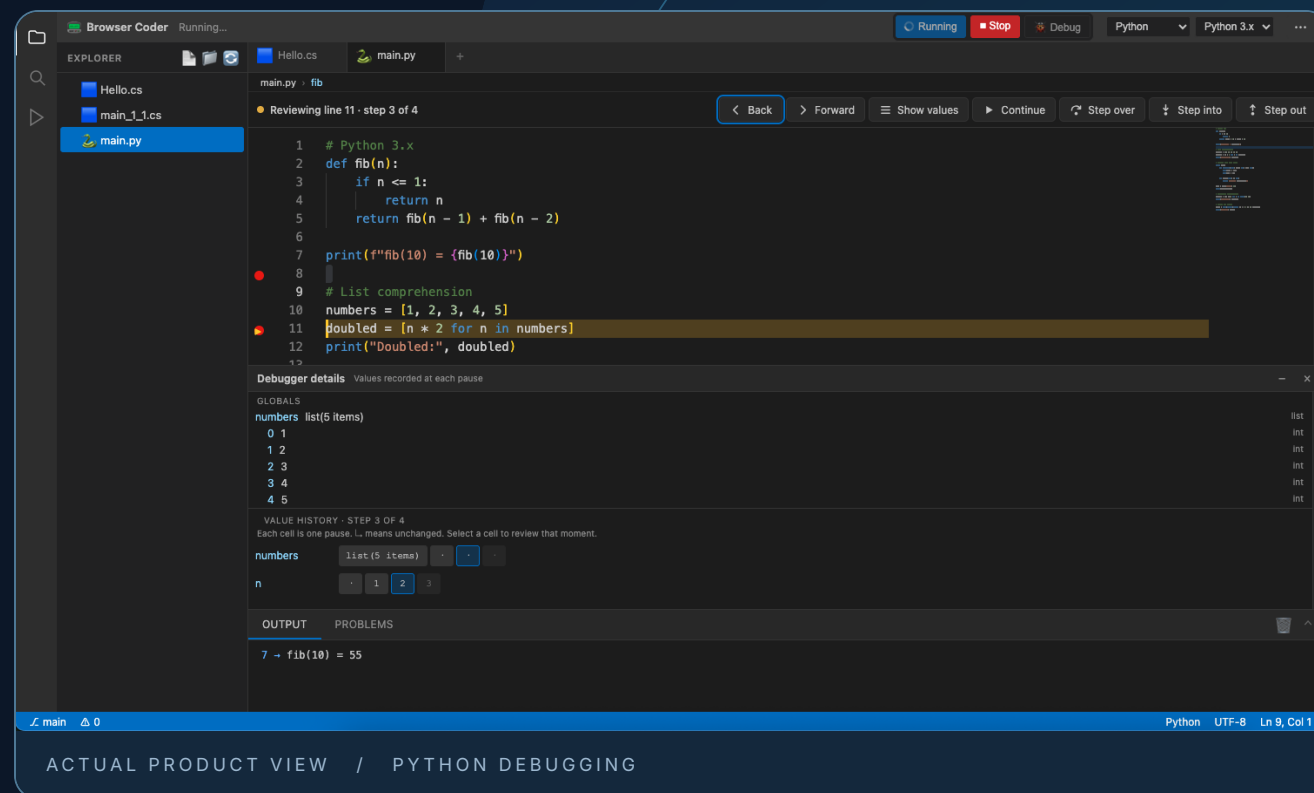
Browser Coder

A visual introduction to learning,
building and understanding code.

WRITE

RUN

EXPLORE



Revisit the moment.
Understand the change.

01 / MEET BROWSER CODER

A coding workspace built around understanding.

Learners can write a program, run it, and investigate how it reached its result—all in a browser-based workspace.



Build something real

Work with code files, interactive programs, drawings and web pages.



Make the process visible

Connect an answer to the source line, changing values and earlier pauses behind it.



Bring it into a lesson

Use a standalone workspace or the coding area inside an Arc Academy Step-Up activity.

The teaching question changes from

“Did it work?” to “Why did it work?”

EXPLORE THE GUIDE

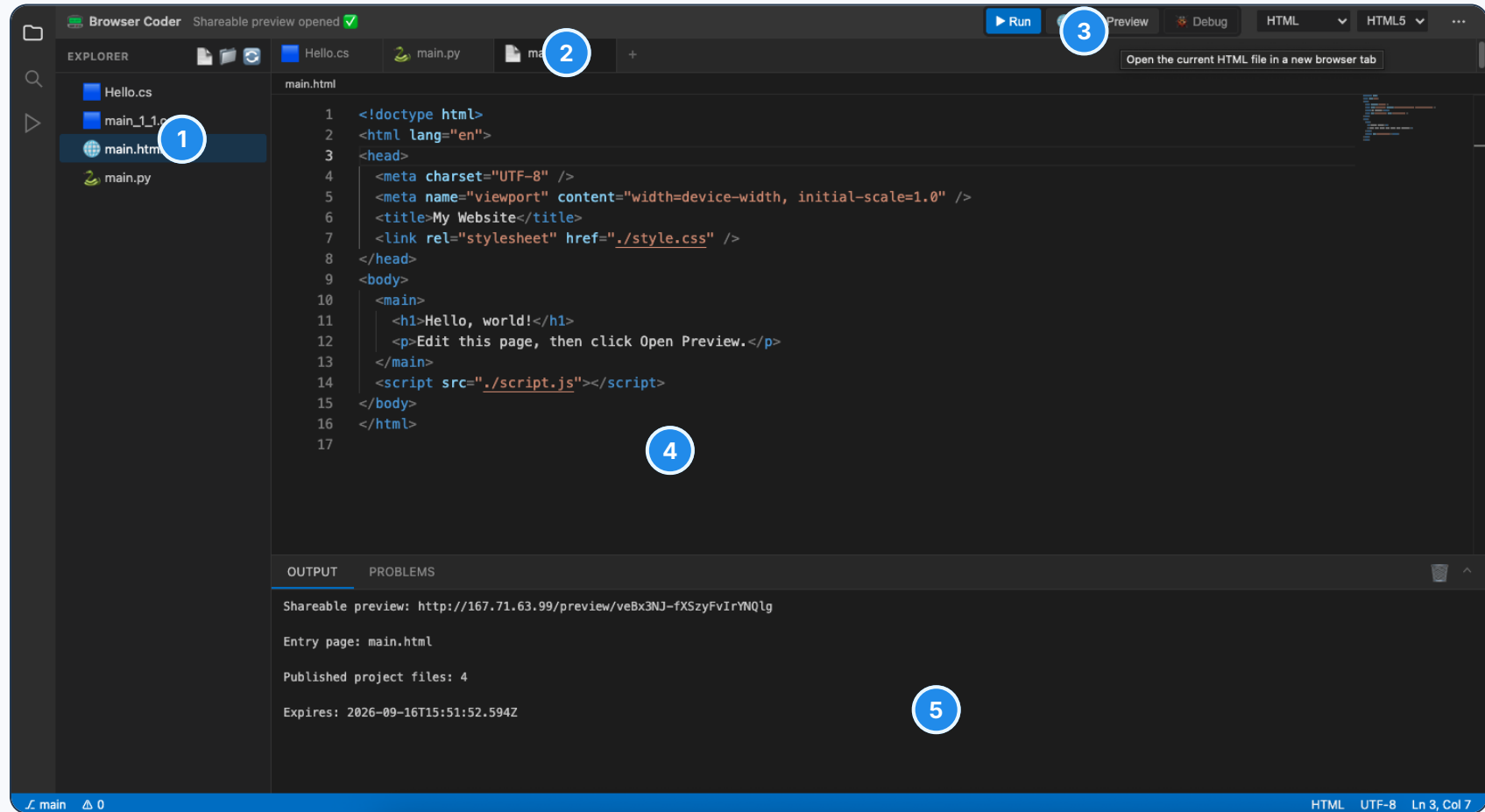
01	Meet the workspace	03–05
02	Learn while you work	06–07
03	See inside a program	08–10
04	Create and share	11–13
05	Design the learning experience	14–18
06	See the complete demo	19–21
07	Documentation and sources	22

Real screenshots, plain-language explanations and a ready-to-present demo sequence.

02 / THE WORKSPACE

One place to write, run and review.

The main areas stay together, so learners can connect what they change with what happens next.



- 1 Explorer**
Your project's files and folders.
- 2 Open files**
Switch between files without leaving the workspace.
- 3 Action bar**
Run, preview or debug; choose a language.
- 4 Code editor**
Read and change the program.
- 5 Output & Problems**
See results and find reported issues.

Actual product screenshot S1 · HTML workspace and preview publishing.

03 / WRITE & RUN

The first useful moment is a small change.

- 1 Start with a file**
Choose a language starter or open the lesson's prepared project.
- 2 Change something simple**
Edit a greeting, a number or a question. Make a prediction before running it.
- 3 Run and respond**
Select **Run**. When the program asks for input, type an answer and press Enter.
- 4 Compare the result**
Read Output, change one thing and try again. **Stop** ends an active run.

ILLUSTRATIVE PYTHON TASK

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

THE CONVERSATION WITH THE PROGRAM

What is your name?

Alex|

Hello, Alex!

A simple example of an interactive program, not a product screenshot.

A useful classroom habit

Ask learners to predict the next output before they press Run.

04 / ORGANIZE & KEEP YOUR WORK

A project can be more than one file.

ILLUSTRATIVE PROJECT

My first website

my-website		
HTML	index.html	The page
CSS	style.css	The appearance
JS	script.js	The interaction
IMG	images/	The artwork

The files belong together.
The workspace keeps them together.



Bring existing work in

Import supported files, folders or a ZIP project. Organize them in the Explorer.



Find what matters

Search across files, replace text and jump to a file by name.



Save locally, take a copy

The standalone workspace autosaves in the browser. Download files or a project ZIP for a separate copy.



Move and rename carefully

Supported references can update with a file move or rename. Warnings flag cases that need a manual check.

Keep an independent copy

Browser autosave is not the same as a cloud backup or automatic synchronization between devices.

05 / UNDERSTAND THE CODE

Help at the exact point of confusion.

An unfamiliar word should not have to interrupt the whole lesson.



Point to a taught keyword

Hover over supported code words or operators to see a curated explanation and an example.



Ask for an explanation

The code context menu also offers **Explain this keyword**.



Keep the explanation readable

Teaching content is available in English and Hebrew, while the actual code keeps its correct form.

ILLUSTRATED TEACHING EXAMPLE

```
1 for number in range(3):  
2     print(number)
```

for



A loop repeats a set of instructions, once for each item in a sequence.

In this example, the program prints
0, then 1, then 2.

Example wording shown here is explanatory artwork, not a captured hover card.

Explain the idea. Keep the learner in the code.

06 / FEEDBACK THAT TEACHES

An error becomes a next step.

Instead of treating an error as a dead end, show what it means and where to look.

REAL MESSAGE + PLAIN-LANGUAGE HELP

ILLUSTRATIVE PYTHON EXAMPLE

```
prnt("Hello")
```

NameError

The name prnt is not defined.

A possible intended correction:

```
print("Hello")
```

01 Locate the issue

Editor markers and the Problems panel point to reported errors in the project.

02 Understand the message

Supported errors add a meaning, a likely cause and an example without hiding the original message.

03 Make a deliberate correction

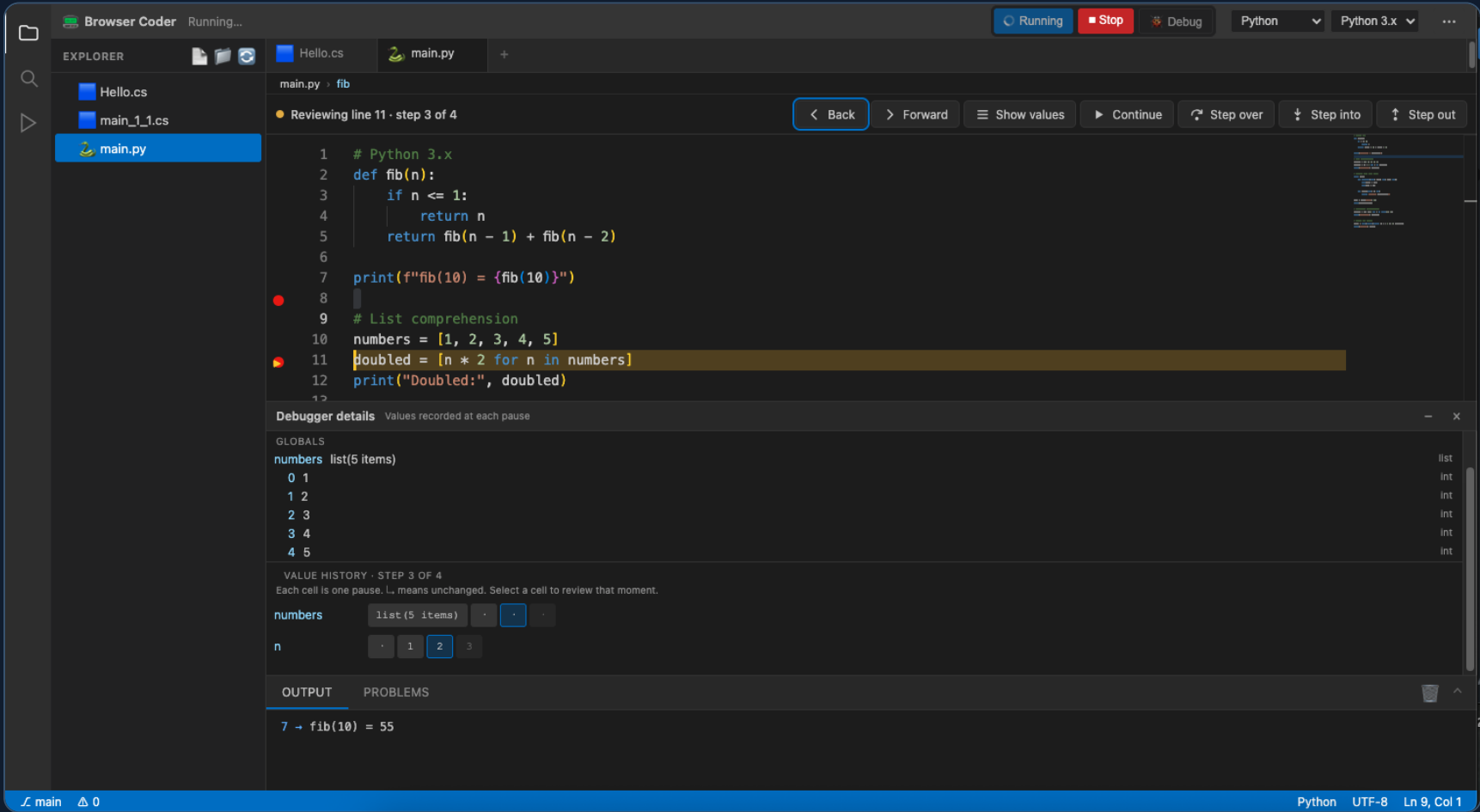
A small set of clear mistakes offers an optional visible fix, such as changing prnt to print.

Support, not an automatic solution

The help is curated. It does not diagnose every mistake or rewrite an entire assignment.

Pause the program. See what it knows.

The code, the selected moment and the recorded values appear together.



Debug

Investigate a program while it runs.

A breakpoint

A line where you ask the program to pause.

A variable

A named value the program is using.

A step

A controlled move through the program.

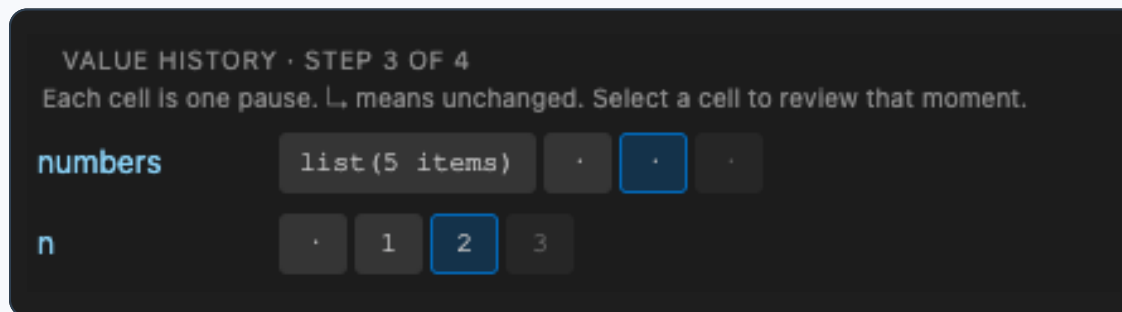
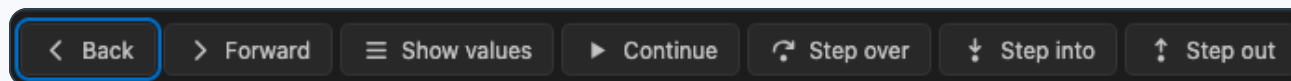
“What do you expect this value to be?”

Actual product screenshot S2 · A Python program under review.

08 / REVIEW THE JOURNEY

Missed the interesting moment? Go back.

ACTUAL DETAIL FROM SCREENSHOT S2



SELECT A RECORDED PAUSE

Compare values without starting over.

Review earlier pauses

Back and **Forward** move through recorded moments. Selecting a value-history cell opens that moment directly.

Follow what changed

The selected line and recorded values move together, making it easier to discuss a loop or a changing counter.

Then continue the live program

Continue and the step controls operate from the program's actual paused position.

An important distinction

This reviews recorded snapshots. It does not reverse the real program or undo its effects. The recording is cleared when the debug session ends.

09 / FOLLOW OUTPUT BACK TO CODE

Connect the answer to the line behind it.

ACTUAL DETAILS FROM SCREENSHOT S2

```
7 print(f"fib(10) = {fib(10)}")
```



THE SOURCE LINE AND ITS OUTPUT

OUTPUT	PROBLEMS
7 → fib(10) = 55	

7

A visible source reference

Mapped output can take the learner back to its matching statement.

ANOTHER WAY TO INVESTIGATE

Observe a value without pausing.

A **logpoint** prints a chosen value when its line runs, then lets the program continue.

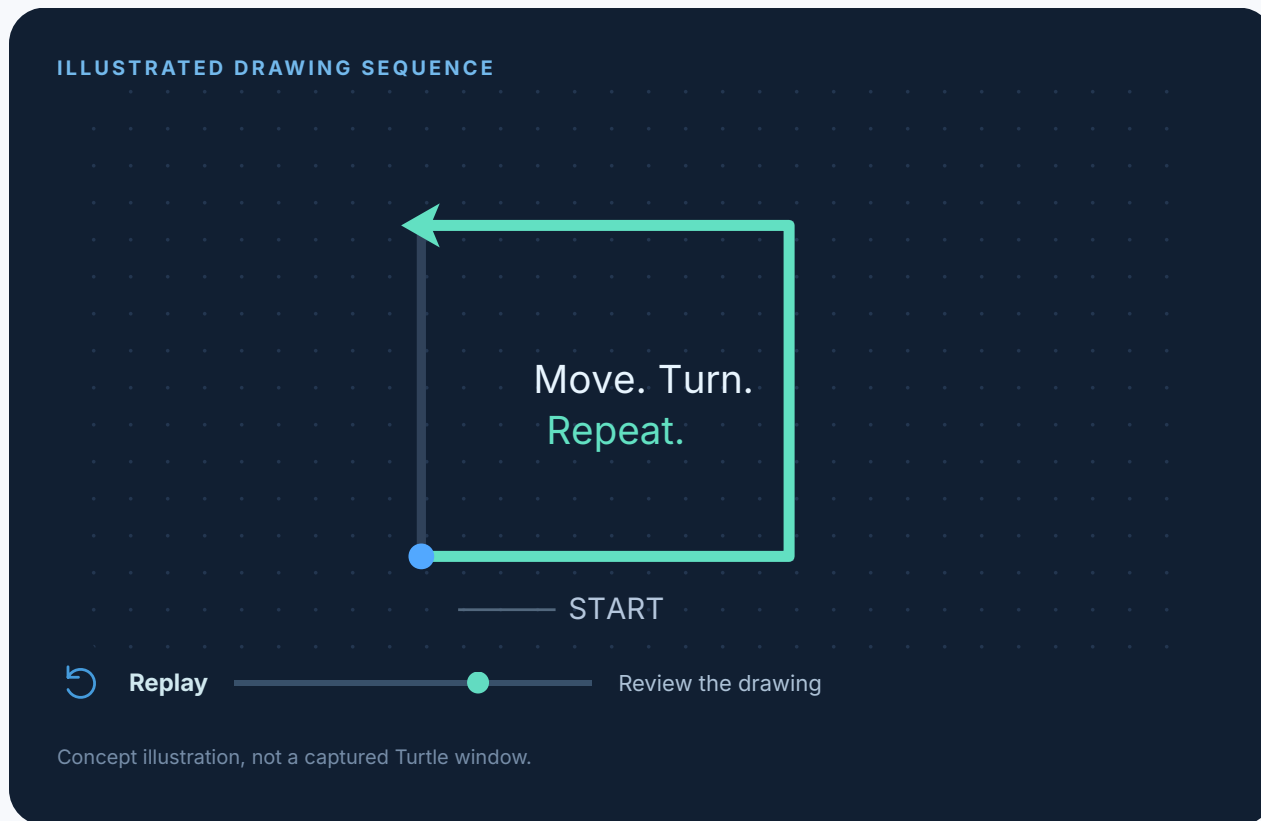
Illustrative loop output

```
score = 1  
score = 2  
score = 3
```

Useful when a learner wants to watch a repeated change without stopping on every repetition.

Less guessing about where an answer came from.

Turn instructions into something you can see.



Turtle is a drawing cursor that follows Python instructions.

See the instructions take shape

Explore lines, turns, shapes and patterns in a separate graphics window.

Revisit how the drawing was made

Replay or move through recorded drawing commands while the matching Python line is highlighted.

Make projects more expressive

Use supported local images, including SVG backgrounds, in visual activities.

Suggested activity

Predict a shape, run the program, then replay the drawing to explain each change of direction.

Build a web page. Open it beyond the editor.

A web project brings together a page, its styling, its interaction and its local images.

1 Prepare the page

Keep the HTML and the files it uses in the project.

2 Select Open Preview

The page opens in a separate browser tab.

3 Send the preview link

A recipient can view the published result without opening the editor.

ACTUAL DETAILS FROM SCREENSHOT S1

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8" />
5   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6   <title>My Website</title>
7   <link rel="stylesheet" href="./style.css" />
8 </head>
9 <body>
10  <main>
11    <h1>Hello, world!</h1>
12    <p>Edit this page, then click Open Preview.</p>
13  </main>
14  <script src="./script.js"></script>
15 </body>
16 </html>
```

THE PUBLISHING CONFIRMATION

```
Shareable preview: http://167.71.63.99/preview/veBx3NJ-fXSzyFvIrYNQlg
Entry page: main.html
Published project files: 4
Expires: 2026-09-16T15:51:52.594Z
```

The screenshot shows publishing, not the rendered page. Referenced files must also be present in the project.

A link to a fixed version

Publishing again creates a new preview. Existing links keep the earlier version until their configured expiry.

Share the result—or share the project.

“LOOK AT WHAT I MADE.”



A web preview

Share the finished browser-facing result of an HTML project.

What the recipient sees

The page as it was published, with the included supporting files.

A natural use

Show a class project, a small website or an interactive page.

[VIEW THE RESULT](#)

“HELP ME UNDERSTAND THIS.”



A project snapshot

Share a fixed snapshot of the project's files for someone to inspect.

What the recipient can do

Open their own copy and try changes without modifying your original.

A natural use

Send a teacher the exact code you are asking about.

[EXPLORE THE CODE](#)

Sharing is not live co-editing

Recipients do not share a cursor or edit one synchronized workspace. Links are temporary; keep a separate download for long-term storage.

13 / TEACHER-DEFINED PROJECT CHECKS

Give learners a useful “Check my work”.

When a task includes prepared checks, learners can test their work and see what needs attention.



See each result

Checks can pass, fail or be skipped, with a clear summary rather than an unexplained wall of output.



Understand the next change

A teacher-written explanation can show the expected result and what the learner’s code produced.



Improve and check again

Use the feedback to revise the program, then rerun the task’s checks.

ILLUSTRATIVE SCORE-CALCULATOR TASK

Check my work

2/3

checks passed

One case needs attention.

**The score starts at zero**

Passed

**Adding five gives five**

Passed

**Adding zero changes nothing**

Expected 5; received 6.

Illustrated feedback, not a screenshot or the result of a live test.

Prepared by the task author

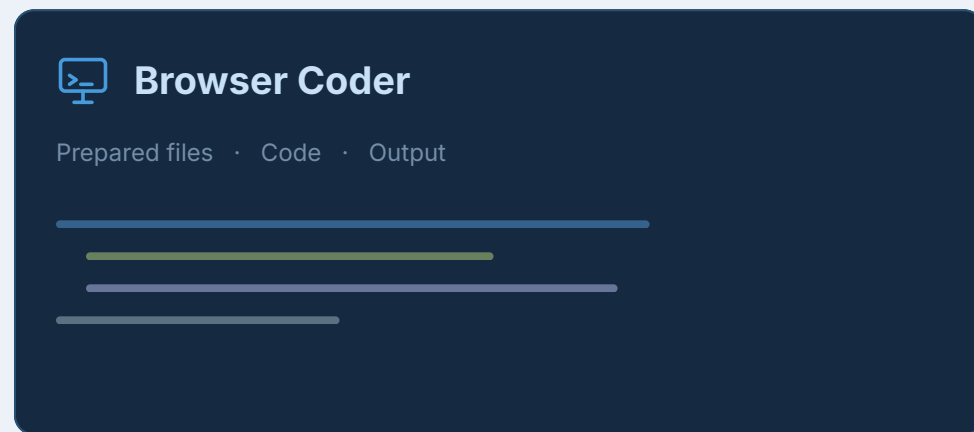
Checks must be supplied with the lesson. Browser Coder does not automatically invent the assessment criteria or provide a full gradebook.

Keep the coding activity inside the lesson.

ILLUSTRATED LESSON STRUCTURE

Lesson instructions

Explain the task and what the learner should create.



The surrounding platform manages the lesson context.

Begin with the right material

The host lesson can load a complete project, receive workspace changes and request a run.

Choose the level of freedom

Host settings can make work read-only, control running, or lock changes to the file structure.

KEEP SUPPORT FILES OUT OF THE WAY

X_HIDDEN_

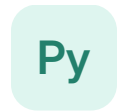
This filename prefix hides a file from the student-facing view. Applied to a folder, it hides its contents too.

Hidden files can still run or be imported by the program. They are excluded from normal student downloads.

Hidden files simplify the learning view; they are not a secure vault for passwords or secrets.

Different languages. A familiar workspace.

Six programming languages are documented for running and debugging. The same workspace also supports web and supporting project files.



Python

First programs, questions, logic and Turtle drawings.



JavaScript

Programs and the behavior of interactive web pages.



TypeScript

Explore JavaScript-style code with explicit type checks.



Java

Practice structured programs and working with classes.



C#

Build programs using C# language features.



PHP

Practice the PHP language through runnable programs.

HTML + CSS

Create and preview web pages.

Markdown

Preview formatted project notes.

Images + data

Keep supporting material with the code.

Examples above suggest learning uses, not bundled courses. Versions and available actions depend on the selected language and task settings.

A workspace that can meet learners differently.

TWO INTERFACE LANGUAGES

English

עברית

Interface labels and teaching explanations are available in English and Hebrew.

print("Hello")

The code keeps its correct direction and spelling.

Dark

print("Hello")

Light

print("Hello")

High contrast

print("Hello")

Illustrative theme samples,
 not captured settings screens.

- 

More than mouse-only interaction

Keyboard navigation, accessible control labels and announcements support different ways of working.



Choose a readable display

Dark, light and high-contrast themes offer different viewing options.



Fit the lesson space

The interface adapts when it appears inside an embedded learning activity.

The same tools support different teaching moments.



GUIDED INTRODUCTION

Predict. Run. Explain.

Use a short starter program. Ask learners to predict its answer, run it and explain the difference between their prediction and the result.



WORKING THROUGH A MISTAKE

Find the changing value.

Pause a program at a useful point. Compare recorded values and follow an output line back to the code before making a change.



PROJECT SHOWCASE

Build something shareable.

Create a small website or a visual Python activity. Share the web result, or send a project snapshot to discuss how it was made.

FOR LEARNERS

A visible path from idea to result.

FOR INSTRUCTORS

Concrete evidence to discuss.

FOR PROGRAM LEADERS

A coding space within the lesson.

Suggested teaching scenarios. These are not customer testimonials or measured learning-outcome claims.

A managed environment, not an unrestricted computer.



Room to experiment. Boundaries around execution.

The service is designed to run classroom code with restrictions on what programs can access and how much resource they can consume.

01 Control access

Language-aware checks and execution restrictions limit prohibited system and network activity.

02 Keep activity bounded

Working folders, output limits and limits on simultaneous runs help contain the workload.

03 End abandoned sessions

Stop ends a run. Leaving the page or removing the embedded workspace disconnects its live session.

04 Operate with evidence

Automated tests and service-health checks support deployment and maintenance decisions.

Capacity is deployment-specific The repository does not promise a fixed number of simultaneous students. Hosting, workload and measured resource needs determine capacity.

A complete demo in about ten minutes.

Prepare first A short Python input task, a breakpoint example, a simple mistake, a Turtle drawing, an HTML project with its dependencies, and a task with checks.		
MINUTES	SHOW THIS	THE POINT TO MAKE
00-01	Orient the audience Point to Explorer, the editor and Output.	"The files, instructions and result stay together."
01-03	Make one visible change Open a short Python task, edit it, Run and answer its input question.	"A small edit produces an observable result."
03-05	Inspect a pause Set a breakpoint on executable code. Debug, step through a few pauses, then use Back and value history.	"We can inspect how a value changed."
05-06	Explain a mistake Use a prepared simple error, show its explanation and apply an offered fix where available.	"Feedback gives the learner a next step."
06-08	Show a visual result Open a prepared Turtle example and replay it, then preview a complete HTML project.	"Code can become a drawing or a web page."
08-10	Check and share Run a task with prepared checks. Show the difference between a preview link and a project snapshot.	"The learner can improve the work and discuss it."

Suggested presentation sequence, not a record of a live session. Confirm the deployed version, permissions and browser pop-up settings before presenting.

20 / PRACTICAL QUESTIONS

Useful things to know before you start.

01 Do learners install programming tools?

They use the browser workspace. Code execution depends on the hosted Browser Coder service rather than a full programming setup on each learner's computer.

03 Can an earlier run be replayed forever?

No. Debug review belongs to the active session and is cleared when it ends. Turtle replay concerns the drawing commands currently available in the graphics view.

05 Does every task include automatic checks?

No. A task author must supply its checks. A task without prepared checks does not gain a meaningful assessment simply by opening it in Browser Coder.

02 Where is the work saved?

The standalone workspace saves in that browser. In an embedded lesson, the host platform controls persistence. Download important work; browser storage is not a cross-device backup.

04 Do shared links keep changing?

No. Previews and shared projects are snapshots, not live synchronized documents. A new publication creates a new link, and links expire according to deployment settings.

06 Is it a full school platform or an unrestricted development machine?

No. Browser Coder supplies the coding workspace. Course management belongs to the surrounding platform, and the execution environment limits system access and external packages.

About this guide.

Prepared from the repository’s current documentation and selected implementation files, reviewed on **9 September 2026**. Source titles below are clickable.

[01] Product overview

[README.md](#) · [Documentation map](#)

[03] Running programs

[Execution behavior](#) · [Interactive console](#)

[05] Debugger & recorded pauses

[History behavior](#) · [Debugger interface](#)

[07] Turtle graphics

[Drawing and replay behavior](#)

[09] Project sharing

[Snapshot sharing](#) · [Opening a recipient’s copy](#)

[11] Integration & interface

[ARCHITECTURE.md](#) · [English interface](#) · [Hebrew interface](#)

[02] Workspace & starters

[Workspace initialization](#) · [Frontend ownership](#)

[04] Teaching & error help

[Hover explanations](#) · [Error help](#) · [Optional small fixes](#)

[06] Output tracing & logpoints

[Output-to-source tracing](#) · [Editor actions](#)

[08] Web previews

[PREVIEWS.md](#) · [Preview assembly](#)

[10] Teaching checks & hidden files

[Writing task checks](#) · [Visibility rules](#)

[12] Operation & deployment

[DEPLOY.md](#) · [Security model](#)

SCREENSHOTS

S1: HTML preview publishing. S2: Python debugging. Both supplied by the project owner; enlarged details are crops of these screenshots.

SCOPE

Documentation-based overview, not a live production test or certification. Illustrations and activities are labelled. Deployed behavior depends on version and settings.

Write. Run. Understand.

[Explore the repository](#) →