L02

AI Game Development & Computer Science Summer Program

Lecture 2: Expressions, Values, and Types

From abstraction to expressions

Last time we hid details. Today we make tiny ideas precise.

Abstraction

- Choose what matters.
- Name the idea.
- Hide the rest.

Expression

- A piece of code that produces a value.
- Example: 3 + 4.
- Result: 7.

Type

- The kind of value.
- Example: number, string, boolean.
- Types catch bad assumptions.

Question: what value does this expression produce, and what kind of value is it?

Expressions evaluate to values

Evaluate means: reduce the expression until one value remains.

```
3 + 4 * 2
3 + 8
11
```

Trace rule

- Do one operation at a time.
- Keep the value after each step.
- Say the type out loud.

Why it matters

- Computers do exactly what the expression says.
- Tracing helps you find wrong assumptions before they become bugs.

Values have types

A type tells us what operations make sense.

number

- Example: 42
- Useful for: add, compare, round

string

- Example: "score"
- Useful for: join, slice, measure

boolean

- Example: true
- Useful for: choose, guard, test

A type is a promise about what the value can do.

Names make values reusable

A name lets you reuse a value without repeating the whole expression.

```
const baseScore = 40;
const bonus = 8;
const total = baseScore + bonus;
```

`const` first

- Use `const` when a value should not change.
- This makes your program easier to trust.

Trace it

- baseScore is 40.
- bonus is 8.
- total is 48.

TypeScript can infer types

Inference means TypeScript can often figure out the type from the value.

```
const points = 42;      // number
const name = "Ada";    // string
const ready = true;    // boolean
```

Use inference when it is obvious

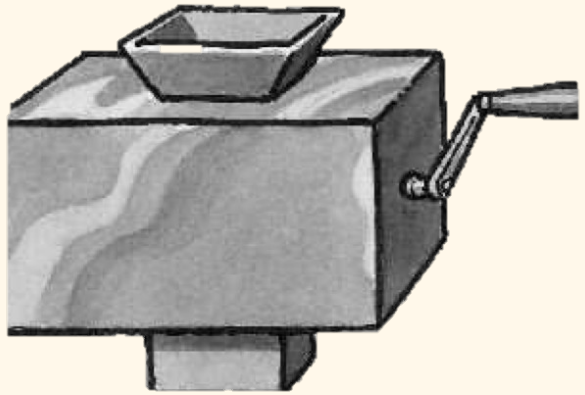
- You write less noise.
- The type checker still protects you.

Add a type when it clarifies intent

- Function inputs.
- Public return values.
- Values with a small set of choices.

Domain and range

A function has allowed inputs and possible outputs.



A function machine: input goes in, output comes out

Domain

- Allowed inputs.
- Example: a score from 0 to 100.

Range

- Possible outputs.
- Example: pass, review, or retry.

Types are one way to make domain and range visible in code.

Same value, different representations

A value is the idea. A representation is how we write it.

One value

- The idea: four.
- It can be written many ways.

Many representations

- 4
- "four"
- 0b100
- IV

Programming habit

- Know which representation you have.
- Convert only when the program needs it.

This is why types matter: the same-looking symbol can mean different things in different contexts.

Type errors are design feedback

A type error is not a personal failure. It is a message about a mismatch.

```
const total = 40 + "8";
// What did we mean?
// 48 or "408"?
```

Ask three questions

- What value do I have?
- What type is it?
- What type do I need?

Fix the idea first

- Do not silence the error.
- Clarify the intended value.
- Then change the code.

AI can help you practice tracing

Use AI to make examples. Do not outsource the thinking.

Good prompt

- Give me three TypeScript expressions.
- Show each evaluation step.
- Include the type of each final value.

Your job

- Check every step.
- Run the code yourself.
- Explain why the type makes sense.

Generated code is a draft until you can trace it.

Studio build: score summary

Build one tiny program that turns raw values into a useful result.

```
type Result = "pass" | "review";

const summarize = (score: number): Result
=>
  score >= 70 ? "pass" : "review";
```

Build steps

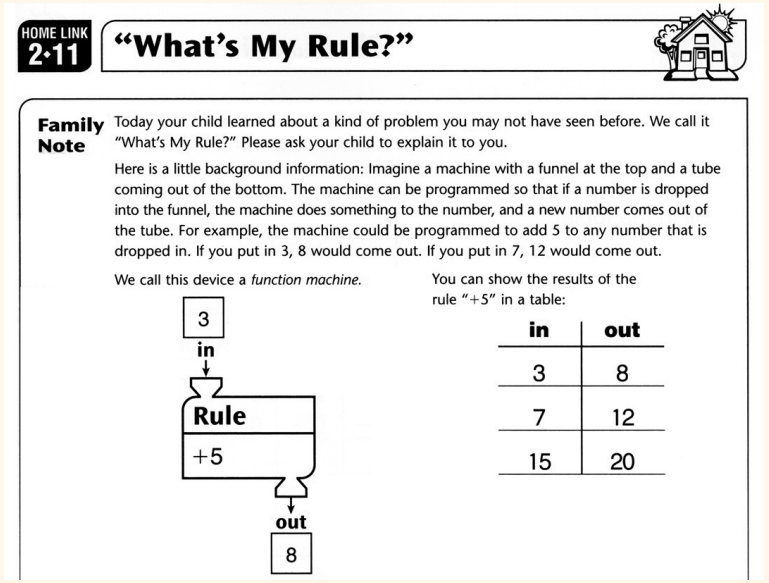
- Trace scores 90, 70, and 69.
- If setup is green, add code and test.
- If yellow/red, save the value table and exact error.

Discuss

- What is the input type?
- What is the output type?
- What assumption did the code make?

Activity: find the rule

Look at input-output pairs and infer the hidden expression.



HOME LINK 2·11

"What's My Rule?"

Family Note Today your child learned about a kind of problem you may not have seen before. We call it "What's My Rule?" Please ask your child to explain it to you.

Here is a little background information: Imagine a machine with a funnel at the top and a tube coming out of the bottom. The machine can be programmed so that if a number is dropped into the funnel, the machine does something to the number, and a new number comes out of the tube. For example, the machine could be programmed to add 5 to any number that is dropped in. If you put in 3, 8 would come out. If you put in 7, 12 would come out.

We call this device a *function machine*.

You can show the results of the rule "+5" in a table:

in	out
3	8
7	12
15	20

Source visual: infer the rule from examples

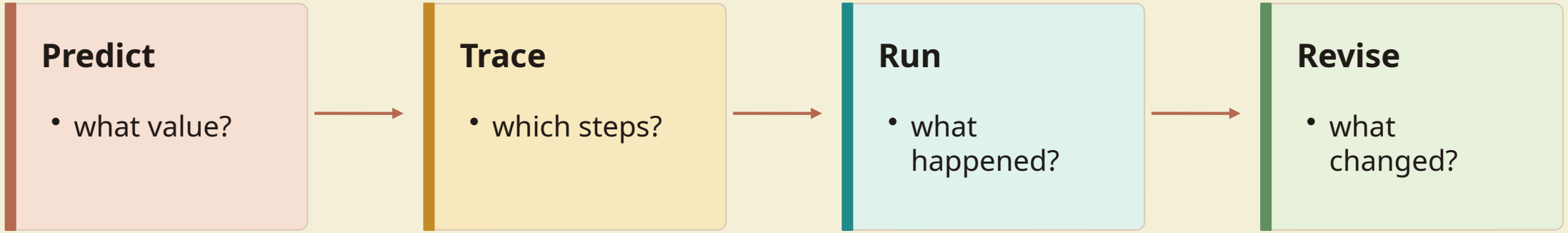
Your turn

- Pick three input-output pairs.
- Guess the expression.
- Test a new input.

This is the same habit you use when reading unfamiliar code: infer the rule, then test it.

Programming is revision

A good trace makes the next edit obvious.



What good programmers do

- Work in small changes.
- Use errors as clues.
- Explain the value and type.

Lab plan: practice, build, apply

Use the main lab block to turn the concept into working evidence.

Guided practice

- Trace `baseScore + bonus * 2` when `baseScore = 10` and `bonus = 3`.
- Write the type of `score`, `"score"`, `score >= 10`, and `String(score)`.
- Rewrite `level = "3"` as the value and type the program actually needs.

Build target

- Copy `labs/snippets/L02-l02-types-starter.ts.txt` to `starter/src/l02-types.ts`.
- Run `npm run build`, read the type errors, then fix the constants.
- Use at least one union type such as `"easy" | "normal" | "hard"`.

Project time

- Define the first P1 values: score, lives, level, or turn.
- Write the expected type for each value.

P1: Game Rules Milestone

L03-L15

Bonus activities

Choose one if the main lab is solid and tested.

Stretch options

- Make a smaller domain with a union type.
- Find a case where string plus number gives the wrong result.
- Ask AI for five typed values and reject the broad or invalid ones.

Done when

- The main lab still works.
- You can explain the tradeoff or edge case.
- You saved one proof object: test, trace, screenshot, or note.

P1: Game Rules Milestone

L03-L15

Checkpoint

Move on when you can explain the value, type, and assumption.

Exit ticket

- One expression I can trace.
- One value and its type.
- One type error I can explain.
- One AI answer I would verify.

Next class preview

- Functions as abstractions.
- Parameters and return values.
- Contracts we can test.