

A Toy Robot and a Functional Pattern

関数型プログラミングにインスピレーションを
得た模様は、命令と解釈を分離します。

A Functional Programming-inspired pattern to
separate instructions and interpretations.



Rob Howard
@damncabbage
<http://robhoward.id.au>



Australia



Ambiata



JavaScript



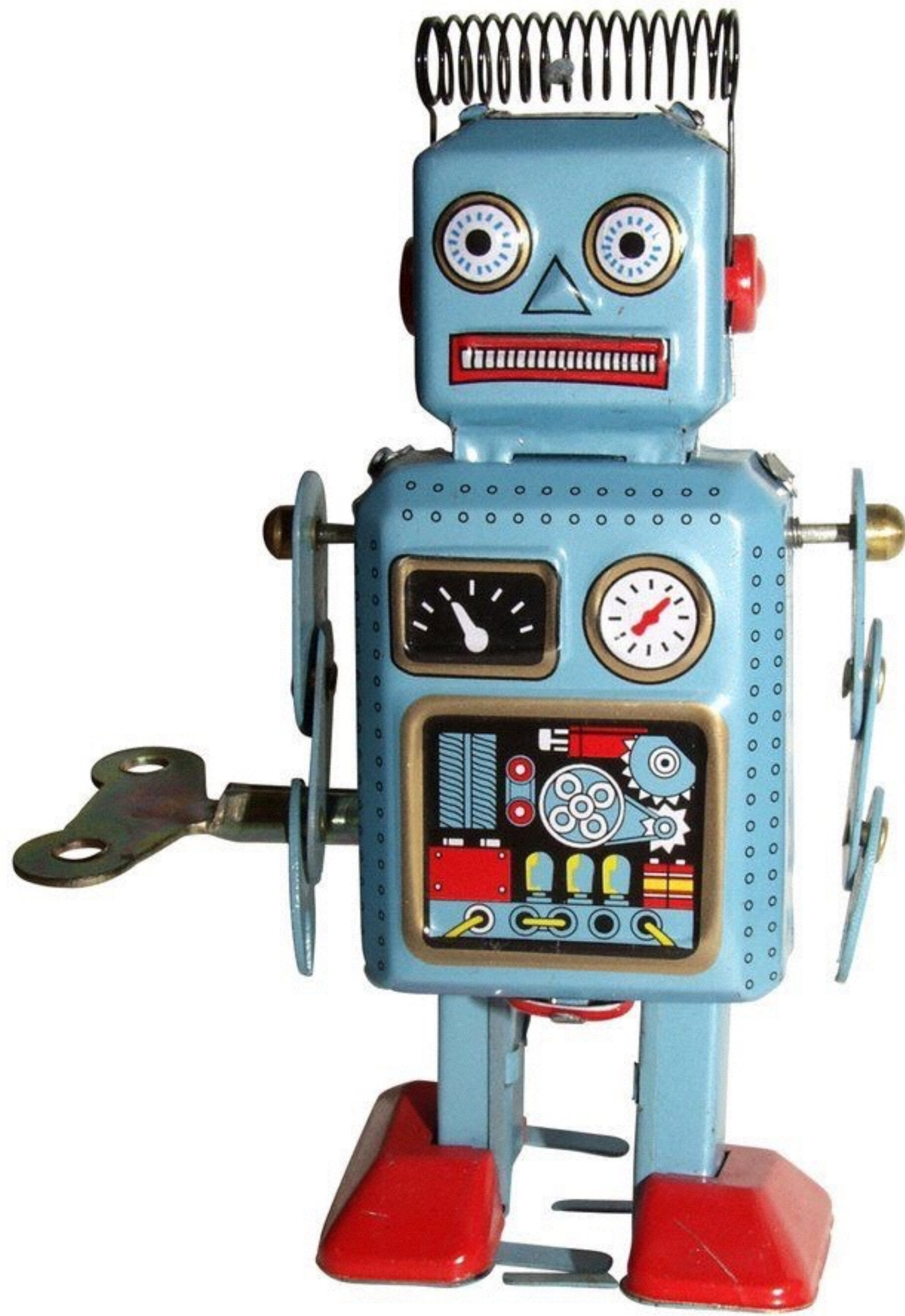
PureScript



Haskell

プレゼンに関して

- ▶ プレゼンテーションは英語で行われます。
- ▶ 各セクションの後に概要が表示されます。
- ▶ 翻訳はあまり良くないかもしれません。😓



Toy Robot Simulator

Description

- The application is a simulation of a toy robot moving on a square tabletop, of dimensions 5 units x 5 units.
- There are no other obstructions on the table surface.
- The robot is free to roam around the surface of the table, but must be prevented from falling to destruction. Any movement that would result in the robot falling from the table must be prevented, however further valid movement commands must still be allowed.

Create an application that can read in commands of the following form:

```
PLACE X,Y,F  
MOVE  
LEFT  
RIGHT  
REPORT
```

- PLACE will put the toy robot on the table in position X,Y and facing NORTH, SOUTH, EAST or WEST.
- The origin (0,0) can be considered to be the SOUTH WEST most corner.

> PLACE 1,1,NORTH

LEFT

MOVE

RIGHT

MOVE

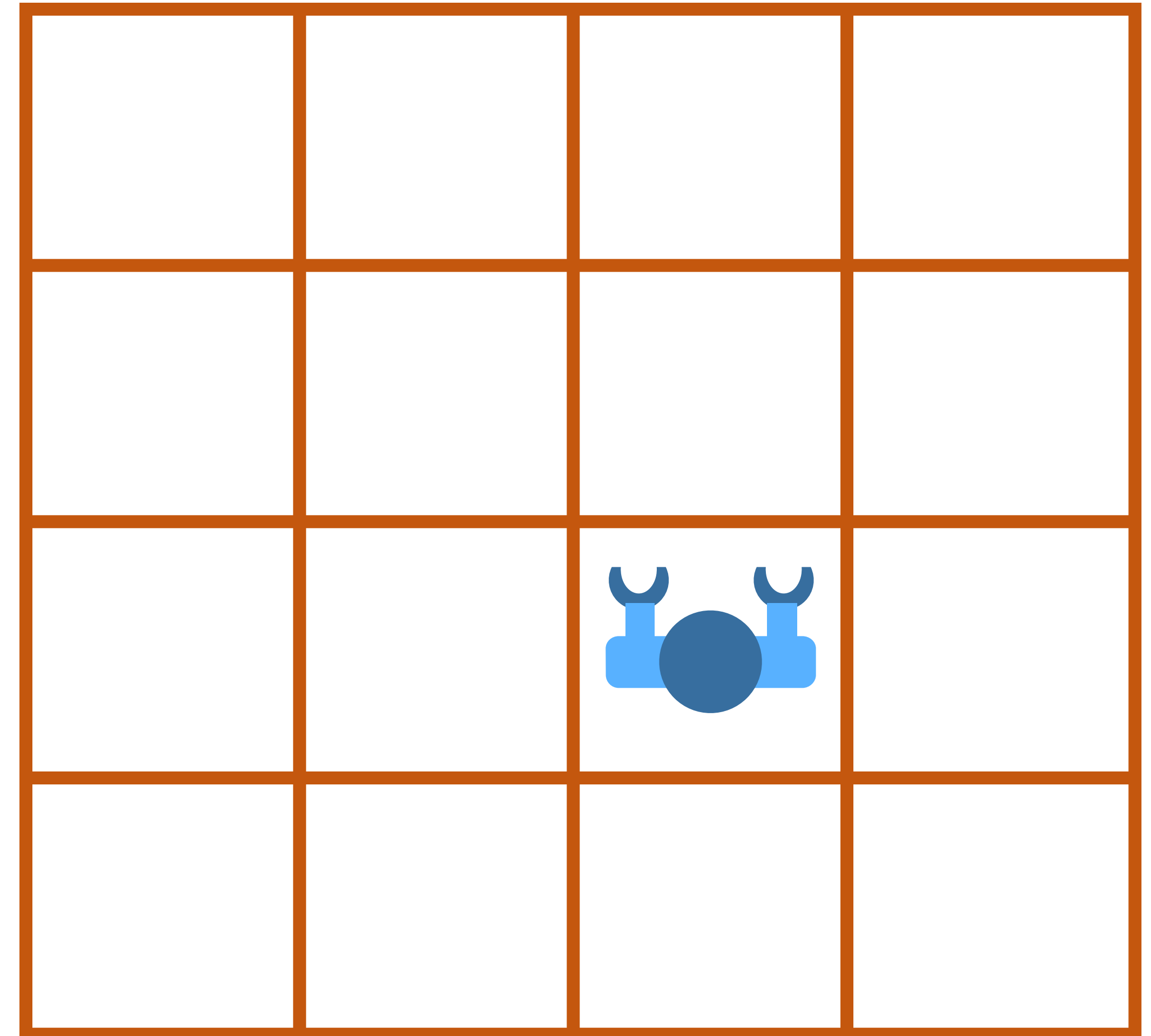
RIGHT

RIGHT

RIGHT

LEFT

REPORT



PLACE 1,1,NORTH

LEFT

MOVE

RIGHT

MOVE

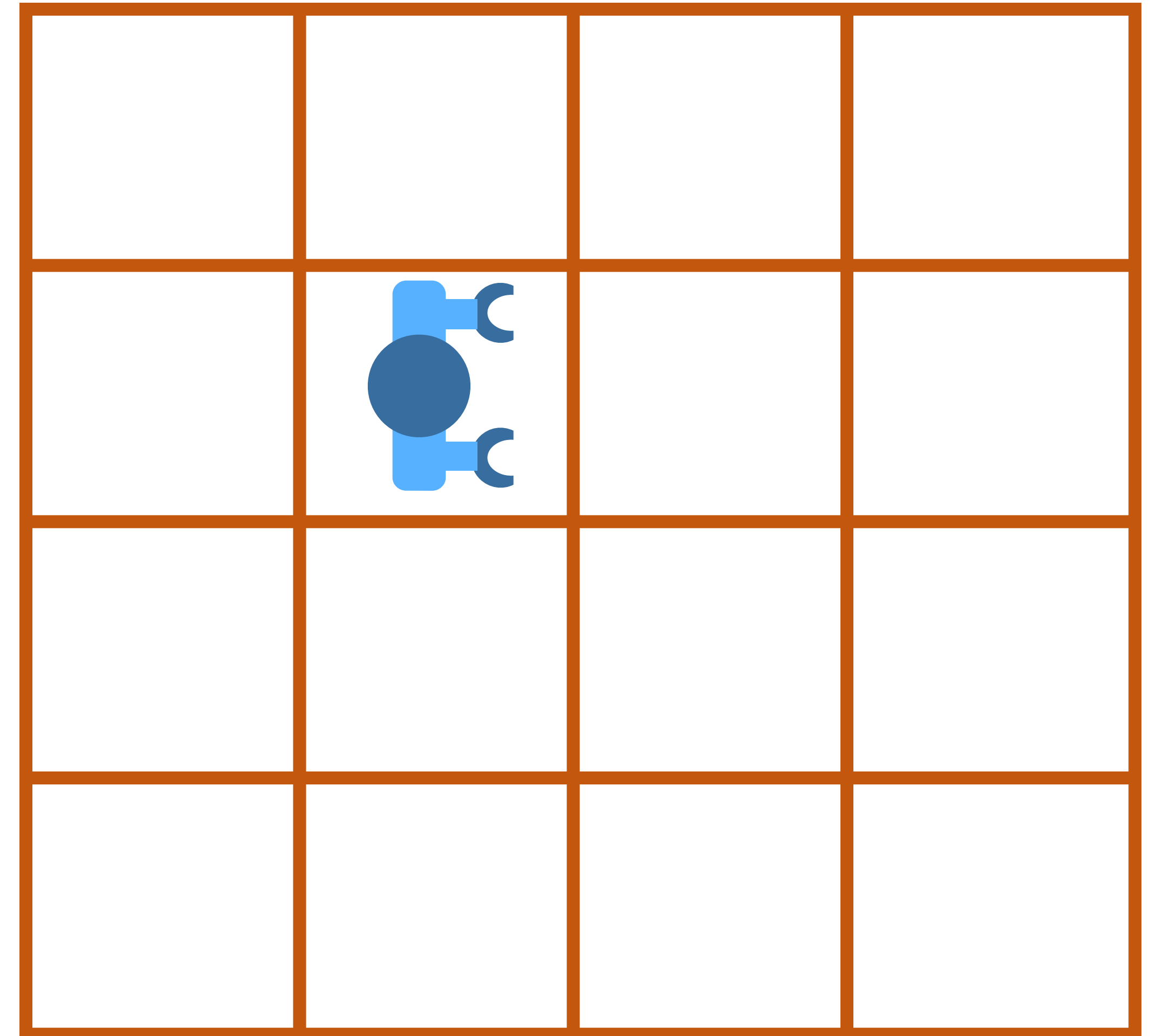
> **RIGHT**

RIGHT

RIGHT

LEFT

REPORT



PLACE 1,1,NORTH

LEFT

MOVE

RIGHT

MOVE

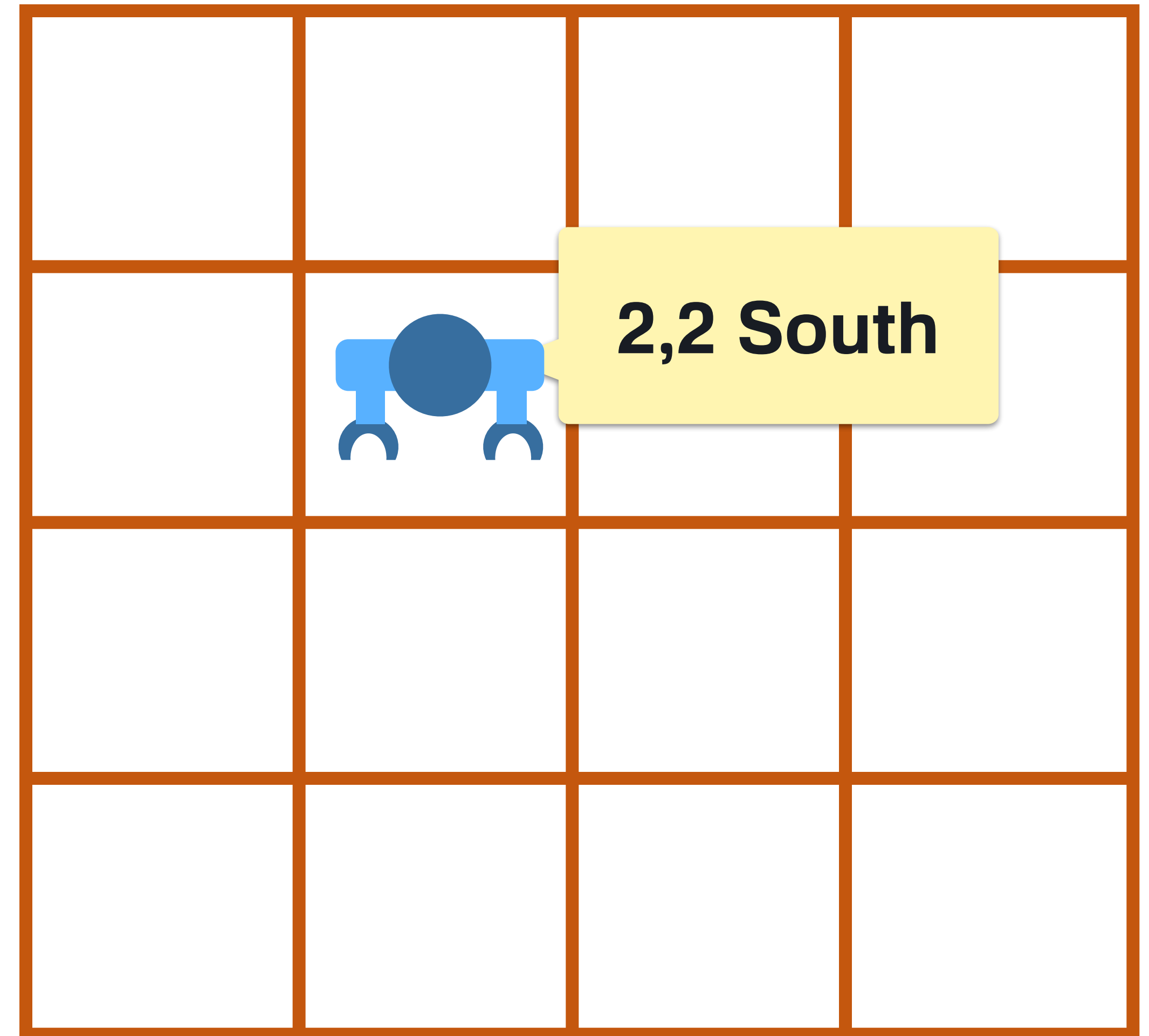
RIGHT

RIGHT

RIGHT

LEFT

> REPORT





```
$ cat instructions.txt
```

```
PLACE 1,1,NORTH
```

```
LEFT
```

```
MOVE
```

```
RIGHT
```

```
MOVE
```

```
RIGHT
```

```
RIGHT
```

```
RIGHT
```

```
LEFT
```

```
REPORT
```

```
$ cat instructions.txt | node robot.js  
2,2,SOUTH
```

「トイ・ロボット」に関して

- ▶ 「トイ・ロボット・シミュレーター」というコード演習は面接試験としてオーストラリアのルビー言語コミュニティでよく使われています。
- ▶ プログラムは幾つかのコマンドを読み込みます。
(例えばロボットを《テーブルに置く》や《向きを変える》、又は《前に進む》など)
- ▶ それに基づいてロボットを行動させ、最終的なロボットの位置を確認します。
- ▶ これは通常、単純なコンソールアプリケーションです。


```
function runRobot(width, height, input) {  
  let robot;  
  input.split("\n").forEach((line) => {  
    let matches;  
    if (  
      matches = line.match(  
        /PLACE ([\d]+),([\d]+), (NORTH|SOUTH|  
        EAST|WEST)/,  
      matches  
    ) {  
      let dir;  
      switch (matches[3]) {  
        case "NORTH": dir = 0; break;
```

```
function runRobot(width, height, input) {  
  let robot;  
  input.split("\n").forEach((line) => {  
    let matches;  
    if (  
      matches = line.match(  
        /PLACE ([\d]+),([\d]+), (NORTH|SOUTH|  
        EAST|WEST)/,  
      matches  
    ) {  
      let dir;  
      switch (matches[3]) {  
        case "NORTH": dir = 0; break;
```

```
function runRobot(width, height, input) {  
  let robot;  
  input.split("\\n").forEach((line) => {  
    let matches;  
    if (  
      matches = line.match(  
        /PLACE ([\\d]+), ([\\d]+), (NORTH|SOUTH|  
        EAST|WEST)/,  
      matches  
    ) {  
      let dir;  
      switch (matches[3]) {  
        case "NORTH": dir = 0; break;
```



```
function runRobot(width, height, input) {  
  let robot;  
  input.split("\n").forEach((line) => {  
    let matches;  
    if (  
      matches = line.match(  
        /PLACE ([\d]+),([\d]+), (NORTH|SOUTH|  
        EAST|WEST)/,  
      matches  
    ) {  
      let dir;  
      switch (matches[3]) {  
        case "NORTH": dir = 0; break;
```

```
function runRobot(width, height, input) {  
  let robot;  
  input.split("\n").forEach((line) => {  
    let matches;  
    if (  
      matches = line.match(  
        /PLACE ([\d]+),([\d]+), (NORTH|SOUTH|  
        EAST|WEST)/),  
      matches  
    ) {  
      let dir;  
      switch (matches[3]) {  
        case "NORTH": dir = 0; break;
```

```
function runRobot(width, height, input) {  
  let robot;  
  input.split("\n").forEach((line) => {  
    let matches;  
    if (  
      matches = line.match(  
        /PLACE ([\d]+),([\d]+), (NORTH|SOUTH|  
        EAST|WEST)/,  
      matches  
    ) {  
      let dir;  
      switch (matches[3]) {  
        case "NORTH": dir = 0; break;
```

```
    let dir;  
    switch (matches[3]) {  
        case "NORTH": dir = 0; break;  
        case "EAST":  dir = 1; break;  
        case "SOUTH": dir = 2; break;  
        case "WEST":  dir = 3; break;  
    }  
    const x = parseInt(matches[1], 10);  
    const y = parseInt(matches[2], 10);  
    dir && x >= 0 && x < width  
        && y >= 0 && y < height  
        && (robot = {x, y, dir});  
} else if (line === "LEFT") {  
    robot && (robot.dir =
```

```
    && (robot = {x, y, dir}),  
} else if (line === "LEFT") {  
    robot && (robot.dir =  
        (robot.dir === 0 ? 3 :  
            robot.dir - 1));  
} else if (line === "RIGHT") {  
    robot && (robot.dir = (  
        robot.dir + 1) % 4);  
} else if (line === "MOVE") {  
    if (robot) {  
        let newX = robot.x;  
        let newY = robot.y;  
        switch (robot.dir) {  
            case 0: newY = robot.y + 1; break;  
            case 1: newX = robot.x + 1; break;  
            case 2: newY = robot.y - 1; break;  
            case 3: newX = robot.x - 1; break;  
        }  
        robot = {x: newX, y: newY, dir: robot.dir};  
    }  
}
```



```
    newY = robot.y;  
    switch (robot.dir) {  
        case 0: newY = robot.y + 1; break;  
        case 1: newX = robot.x - 1; break;  
        case 2: newY = robot.y - 1; break;  
        case 3: newX = robot.x + 1; break;  
    }  
    newX >= 0 && newX < width  
        && newY >= 0 && newY < height  
        && (  
            robot.x = newX,  
            robot.y = newY  
        );  
}  
else if (line == "REPORT") {
```

```
    },  
    }  
} else if (line === "REPORT") {  
    const dirStr = x => {  
        switch(x) {  
            case 0: return "NORTH";  
            case 1: return "EAST";  
            case 2: return "SOUTH";  
            case 3: return "WEST";  
        }  
    }  
}
```

```
robot && console.log(robot.x, robot.y,  
    dirStr(robot.dir));
```

```
    }  
    )  
};
```

```
        case 3: return WEST;
    }
}
robot && console.log(robot.x, robot.y,
    dirStr(robot.dir));
}
});
}
```

```
runRobot(
    5, 5
    "PLACE 1,1,EAST\nMOVE\nMOVE\nLEFT\nMOVE\nRIGHT\nRIGHT\nREPORT",
); // "0 2 SOUTH" (0,2 South)
```

String

String → [Things Happen]


```
function runRobot(width, height, input) {  
    // ...  
}
```

```
const w = 3, h = 3; // width and height
it("tests runRobot is correct", function() {

});
```

```
function runRobot(width, height, input) {  
    // ...  
}
```

```
const w = 3, h = 3;  
it("tests runRobot is correct", function() {  
    expect(  
        runRobot(w, h, "PLACE 1,1,EAST\nREPORT")  
    ).toBe(  
  
    );  
});
```

```
function runRobot(width, height, input) {  
    // ...  
}
```

```
const w = 3, h = 3;  
it("tests runRobot is correct", function() {  
    expect(  
        runRobot(w, h, "PLACE 1,1,EAST\nREPORT")  
    ).toBe(  
        // 何...?  
    );  
});
```

```
function runRobot(width, height, input) {  
    // ...  
}  
  
const w = 3, h = 3;  
it("tests runRobot is correct", function() {  
    expect(  
        runRobot(w, h, "PLACE 1,1,EAST\nREPORT")  
    ).toBe(  
        undefined // 素晴らしい事だな  
    );  
});
```

```
function runRobot(width, height, input) {  
    // ...  
}
```

```
const w = 3, h = 3;  
it("tests runRobot is correct", function() {  
    spyOn(console, "log");  
    runRobot(w, h, "PLACE 1,1,EAST\nREPORT");  
    expect(console.log).toHaveBeenCalledWith(  
        "1,1,EAST"  
    );  
});
```



```
const w = 3, h = 3;
it("tests runRobot is correct", function() {
  spyOn(console, "log");
  runRobot(w, h, [
    "PLACE 1,1,EAST",
    "REPORT",
  ].join("\n"))
});
expect(console.log).toHaveBeenCalledWith(
  "1,1,EAST"
);
});
```

```
const w = 3, h = 3;
it("tests runRobot is correct", function() {
  spyOn(console, "log");
  runRobot(w, h, [
    "PLACE 1,1,EAST", "LEFT", "FORWARD",
    "REPORT",
  ].join("\n")
);
expect(console.log).toHaveBeenCalledWith(
  "1,2,NORTH"
);
});
```

```
const w = 3, h = 3;
it("tests runRobot is correct", function() {
  spyOn(console, "log");
  runRobot(w, h, [
    "PLACE 1,1,EAST", "FORWARD", "RIGHT",
    "REPORT",
  ].join("\n")
);
expect(console.log).toHaveBeenCalledWith(
  "0,1,SOUTH"
);
});
```

```
const w = 3, h = 3;
it("tests runRobot is correct", function() {
  spyOn(console, "log");
  runRobot(w, h, [
    "PLACE 1,1,EAST", "裏庭には二羽庭には二羽鶏がいる",
    "REPORT",
  ].join("\n")
);
expect(console.log).toHaveBeenCalledWith(
  "0,1,SOUTH"
);
});
```

```
const w = 3, h = 3;
it("tests runRobot is correct", function() {
  spyOn(console, "log");
  runRobot(w, h, [
    "PLACE 1,1,EAST", "LEFT ",
    "REPORT", // "LEFT " ignored; has spaces
  ].join("\n")
);
expect(console.log).toHaveBeenCalledWith(
  "1,1,EAST"
);
});
```

```
const w = 3, h = 3;
it("tests runRobot is correct", function() {
  spyOn(console, "log");
  runRobot(w, h, [
    "PLACE 9,9,EAST", // Out of bounds,
    "REPORT",         // so PLACE is ignored.
  ].join("\n")
);
expect(console.log).toNotHaveBeenCalled();

});
```

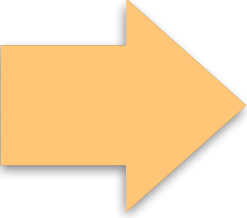
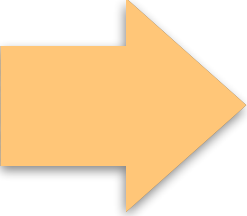
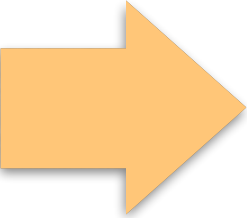
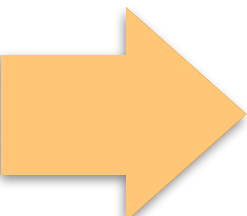
「トイ・ロボット」の例に関して

- ▶ 「トイ・ロボット」の例があります。例は「runRobot」と呼ばれる関数あります。この関数はストリングを受け入れ、それを使って多くのアクションを実行します。
- ▶ この関数はあまりにも多くのことを行うので、テストするのは難しいです。ストリングを解析し、ロボットの動きを計算し、ロボットの位置を報告するためのテストが必要です。
- ▶ 現在、すべてを一度にテストする必要があります。


```
function runRobot(width, height, input) {
  let robot;
  input.split("\n").forEach((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|SOUTH|
        EAST|WEST)/,
      matches
    ) {
      let dir;
      switch (matches[3]) {
        case "NORTH": dir = 0; break;
        case "EAST":  dir = 1; break;
        case "SOUTH": dir = 2; break;
        case "WEST":  dir = 3; break;
      }
      const x = parseInt(matches[1],10);
      const y = parseInt(matches[2],10);
      dir && x >= 0 && x < width
        && y >= 0 && y < height
        && (robot = {x, y, dir});
    } else if (line === "LEFT") {
      robot && (robot.dir =
        (robot.dir === 0 ? 3 :
        robot.dir - 1));
    }
  });
}
```

```
function runRobot(width, height, input) {
  let robot;
  input.split("\n").forEach((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|SOUTH|
        EAST|WEST)/,
      matches
    ) {
      let dir;
      switch (matches[3]) {
        case "NORTH": dir = 0; break;
        case "EAST":  dir = 1; break;
        case "SOUTH": dir = 2; break;
        case "WEST":  dir = 3; break;
      }
      // ...
    } else if (line === "LEFT")    { // ...
    } else if (line === "RIGHT")  { // ...
    } else if (line === "MOVE")   { // ...
    } else if (line === "REPORT") { // ...
    }
  });
}
```

```
function runRobot(width, height, input) {
  let robot;
  input.split("\n").forEach((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|SOUTH|
        EAST|WEST)/,
      matches
    ) {
      let dir;
      switch (matches[3]) {
        case "NORTH": dir = 0; break;
        case "EAST":  dir = 1; break;
        case "SOUTH": dir = 2; break;
        case "WEST":  dir = 3; break;
      }
      // ...
    } else if (line === "LEFT")    { // ...
    } else if (line === "RIGHT")  { // ...
    } else if (line === "MOVE")   { // ...
    } else if (line === "REPORT") { // ...
    }
  });
}
```



```
function runRobot(width, height, input) {
  let robot;
  input.split("\n").forEach((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|SOUTH|
        EAST|WEST)/,
      matches
    ) {
      let dir;
      switch (matches[3]) {
        case "NORTH": dir = 0; break;
        case "EAST":  dir = 1; break;
        case "SOUTH": dir = 2; break;
        case "WEST":  dir = 3; break;
      }
      // ...
    } else if (line === "LEFT") { // ...
    } else if (line === "RIGHT") { // ...
    } else if (line === "MOVE") { // ...
    } else if (line === "REPORT") { // ...
    }
  });
}
```

String → [Things Happen]



手戻り

String →

String → Commands

String → Commands

Commands →

String → Commands

Commands → [Things Happen]

```

function runRobot(width, height, input) {
  let robot;
  input.split("\n").forEach((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|SOUTH|
        EAST|WEST)/,
      matches
    ) {
      let dir;
      switch (matches[3]) {
        case "NORTH": dir = 0; break;
        case "EAST":  dir = 1; break;
        case "SOUTH": dir = 2; break;
        case "WEST":  dir = 3; break;
      }
      // ...
    } else if (line === "LEFT")    { // ...
    } else if (line === "RIGHT")  { // ...
    } else if (line === "MOVE")   { // ...
    } else if (line === "REPORT") { // ...
    }
  });
}

```



手戻り

```
function runRobot(width, height, input) {
  let robot;
  input.split("\n").forEach((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|SOUTH|
        EAST|WEST)/,
      matches
    ) {
      let dir;
      switch (matches[3]) {
        case "NORTH": dir = 0; break;
        case "EAST":  dir = 1; break;
        case "SOUTH": dir = 2; break;
        case "WEST":  dir = 3; break;
      }
      // ...
    } else if (line === "LEFT")    { // ...
    } else if (line === "RIGHT")  { // ...
    } else if (line === "MOVE")   { // ...
    } else if (line === "REPORT") { // ...
    }
  });
}
```

```

function stringToCommands(input) {
  return input.split("\n").map((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|
        SOUTH|EAST|WEST)/, matches
    ) {
      switch (matches[3]) { /* ... */ }
      const x = parseInt(matches[1], 10);
      // ...
      return { type: "PLACE", x, y, dir }

    } else if (line === "LEFT") {
      return { type: "LEFT" }
    } else if (line === "RIGHT") {
      return { type: "RIGHT" }
    } else if (line === "MOVE") {
      return { type: "MOVE" }
    } else if (line === "REPORT") {
      return { type: "REPORT" }
    }
  }).filter(x => x !== undefined);
}

```

```

function runRobot(width, height, input) {
  let commands = stringToCommands(input);
  executeCommands(commands);
}

function executeCommands(w, h, commands) {
  // ...
}

```

```

function stringToCommands(input) {
  return input.split("\n").map((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|
        SOUTH|EAST|WEST)/), matches
    ) {
      switch (matches[3]) { /* ... */ }
      const x = parseInt(matches[1], 10);
      // ...
      return { type: "PLACE", x, y, dir }

    } else if (line === "LEFT") {
      return { type: "LEFT" }
    } else if (line === "RIGHT") {
      return { type: "RIGHT" }
    } else if (line === "MOVE") {
      return { type: "MOVE" }
    } else if (line === "REPORT") {
      return { type: "REPORT" }
    }
  }).filter(x => x !== undefined);
}

```

```

function runRobot(width, height, input) {
  let commands = stringToCommands(input);
  executeCommands(commands);
}

function executeCommands(w, h, commands) {
  // ...
}

```

```

function stringToCommands(input) {
  return input.split("\n").map((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|
        SOUTH|EAST|WEST)/, matches
    ) {
      switch (matches[3]) { /* ... */ }
      const x = parseInt(matches[1], 10);
      // ...
      return { type: "PLACE", x, y, dir }

    } else if (line === "LEFT") {
      return { type: "LEFT" }
    } else if (line === "RIGHT") {
      return { type: "RIGHT" }
    } else if (line === "MOVE") {
      return { type: "MOVE" }
    } else if (line === "REPORT") {
      return { type: "REPORT" }
    }
  }).filter(x => x !== undefined);
}

```

```

function runRobot(width, height, input) {
  let commands = stringToCommands(input);
  executeCommands(commands);
}

function executeCommands(w, h, commands) {
  let robot;
  commands.forEach((command) => {
    switch (command.type) {
      case "PLACE":
        // ...
      case "LEFT":
        // ...
      case "RIGHT":
        // ...
      case "MOVE":
        // ...
      case "REPORT":
        // ...
    }
  });
}

```

```

function stringToCommands(input) {
  return input.split("\n").map((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|
        SOUTH|EAST|WEST)/), matches
    ) {
      switch (matches[3]) { /* ... */ }
      const x = parseInt(matches[1], 10);
      // ...
      return { type: "PLACE", x, y, dir }
    } else if (line === "LEFT") {
      return { type: "LEFT" }
    } else if (line === "RIGHT") {
      return { type: "RIGHT" }
    } else if (line === "MOVE") {
      return { type: "MOVE" }
    } else if (line === "REPORT") {
      return { type: "REPORT" }
    }
  }).filter(x => x !== undefined);
}

```

```

function runRobot(width, height, input) {
  let commands = stringToCommands(input);
  executeCommands(commands);
}

function executeCommands(w, h, commands) {
  let robot;
  commands.forEach((command) => {
    switch (command.type) {
      case "PLACE":
        dir && cmd.x >= 0 && cmd.x < width
        && cmd.y >= 0 && cmd.y < height
        && (robot = {
          x: cmd.x, y: cmd.y, dir: cmd
        });
      case "LEFT":
        // ...
      case "RIGHT":
        // ...
      case "MOVE":
        // ...
      case "REPORT":
        // ...
    }
  });
}

```



```

function stringToCommands(input) {
  return input.split("\n").map((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|
        SOUTH|EAST|WEST)/), matches
    ) {
      switch (matches[3]) { /* ... */ }
      const x = parseInt(matches[1], 10);
      // ...
      return { type: "PLACE", x, y, dir }
    } else if (line === "LEFT") {
      return { type: "LEFT" }
    } else if (line === "RIGHT") {
      return { type: "RIGHT" }
    } else if (line === "MOVE") {
      return { type: "MOVE" }
    } else if (line === "REPORT") {
      return { type: "REPORT" }
    }
  }).filter(x => x !== undefined);
}

```

```

function runRobot(width, height, input) {
  let commands = stringToCommands(input);
  executeCommands(commands);
}

function executeCommands(w, h, commands) {
  let robot;
  commands.forEach((command) => {
    switch (command.type) {
      case "PLACE":
        dir && cmd.x >= 0 && cmd.x < w && cmd.y >= 0 && cmd.y < h && (robot = {
          type: "PLACE", x: cmd.x, y: cmd.y, dir: cmd.dir
        });
        break;
      case "LEFT":
        robot.dir = (robot.dir + 3) % 4;
        break;
      case "RIGHT":
        robot.dir = (robot.dir - 3) % 4;
        break;
      case "MOVE":
        let dx = robot.dir === 0 ? 1 : robot.dir === 1 ? 0 : robot.dir === 2 ? -1 : 0;
        let dy = robot.dir === 0 ? 0 : robot.dir === 1 ? 1 : robot.dir === 2 ? 0 : -1;
        let nx = robot.x + dx;
        let ny = robot.y + dy;
        if (nx < 0 || nx >= w || ny < 0 || ny >= h) break;
        if (robot.x === nx && robot.y === ny) break;
        robot.x = nx;
        robot.y = ny;
        break;
      case "REPORT":
        console.log(`Robot at (${robot.x}, ${robot.y}) facing ${robot.dir}`);
        break;
    }
  });
}

```



```

function stringToCommands(input) {
  return input.split("\n").map((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|
        SOUTH|EAST|WEST)/, matches
    ) {
      switch (matches[3]) { /* ... */ }
      const x = parseInt(matches[1], 10);
      // ...
      return { type: "PLACE", x, y, dir }
    } else if (line === "LEFT") {
      return { type: "LEFT" }
    } else if (line === "RIGHT") {
      return { type: "RIGHT" }
    } else if (line === "MOVE") {
      return { type: "MOVE" }
    } else if (line === "REPORT") {
      return { type: "REPORT" }
    }
  }).filter(x => x !== undefined);
}

```

Commands are a list of...
Command objects

Command is one of...

```

{ type: "PLACE",
  x: number,
  y: number,
  dir: Direction
}
| { type: "LEFT" }
| { type: "RIGHT" }
| { type: "MOVE" }
| { type: "REPORT" }

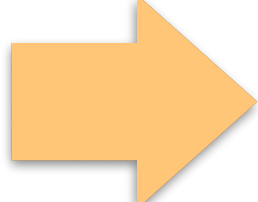
```

Direction is one of...

```

"NORTH"
| "EAST"
| "SOUTH"
| "WEST"

```



```
function stringToCommands(input) {
  return input.split("\n").map((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|
        SOUTH|EAST|WEST)/, matches
    ) {
      switch (matches[3]) { /* ... */ }
      const x = parseInt(matches[1], 10);
      // ...
      return { type: "PLACE", x, y, dir }
    } else if (line === "LEFT") {
      return { type: "LEFT" }
    } else if (line === "RIGHT") {
      return { type: "RIGHT" }
    } else if (line === "MOVE") {
      return { type: "MOVE" }
    } else if (line === "REPORT") {
      return { type: "REPORT" }
    }
  }).filter(x => x !== undefined);
}
```

Commands are a list of...
Command objects

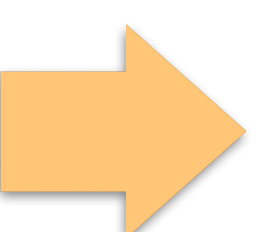
Command is one of...

```
{ type: "PLACE",
  x: number,
  y: number,
  dir: Direction
}
```

```
| { type: "LEFT" }
| { type: "RIGHT" }
| { type: "MOVE" }
| { type: "REPORT" }
```

Direction is one of...

```
"NORTH"
| "EAST"
| "SOUTH"
| "WEST"
```



```
function stringToCommands(input) {
  return input.split("\n").map((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|
        SOUTH|EAST|WEST)/, matches
    ) {
      switch (matches[3]) { /* ... */ }
      const x = parseInt(matches[1], 10);
      // ...
      return { type: "PLACE", x, y, dir }
    } else if (line === "LEFT") {
      return { type: "LEFT" }
    } else if (line === "RIGHT") {
      return { type: "RIGHT" }
    } else if (line === "MOVE") {
      return { type: "MOVE" }
    } else if (line === "REPORT") {
      return { type: "REPORT" }
    }
  }).filter(x => x !== undefined);
}
```

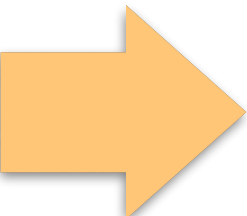
Commands are a list of...
Command objects

Command is one of...

```
{ type: "PLACE",
  x: number,
  y: number,
  dir: Direction
}
| { type: "LEFT" }
| { type: "RIGHT" }
| { type: "MOVE" }
| { type: "REPORT" }
```

Direction is one of...

```
"NORTH"
| "EAST"
| "SOUTH"
| "WEST"
```



```
function stringToCommands(input) {
  return input.split("\n").map((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|
        SOUTH|EAST|WEST)/, matches
    ) {
      switch (matches[3]) { /* ... */ }
      const x = parseInt(matches[1], 10);
      // ...
      return { type: "PLACE", x, y, dir }
    } else if (line === "LEFT") {
      return { type: "LEFT" }
    } else if (line === "RIGHT") {
      return { type: "RIGHT" }
    } else if (line === "MOVE") {
      return { type: "MOVE" }
    } else if (line === "REPORT") {
      return { type: "REPORT" }
    }
  }).filter(x => x !== undefined);
}
```

Commands are a list of...
Command objects

Command is one of...

```
{ type: "PLACE",
  x: number,
  y: number,
  dir: Direction
}
| { type: "LEFT" }
| { type: "RIGHT" }
| { type: "MOVE" }
| { type: "REPORT" }
```

Direction is one of...

```
"NORTH"
| "EAST"
| "SOUTH"
| "WEST"
```

```

function stringToCommands(input) {
  return input.split("\n").map((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|
        SOUTH|EAST|WEST)/, matches
    ) {
      switch (matches[3]) { /* ... */ }
      const x = parseInt(matches[1], 10);
      // ...
      return { type: "PLACE", x, y, dir }
    } else if (line === "LEFT") {
      return { type: "LEFT" }
    } else if (line === "RIGHT") {
      return { type: "RIGHT" }
    } else if (line === "MOVE") {
      return { type: "MOVE" }
    } else if (line === "REPORT") {
      return { type: "REPORT" }
    }
  }).filter(x => x !== undefined);
}

```

Commands are a list of...
Command objects

Command is one of...

```

{ type: "PLACE",
  x: number,
  y: number,
  dir: Direction
}
| { type: "LEFT" }
| { type: "RIGHT" }
| { type: "MOVE" }
| { type: "REPORT" }

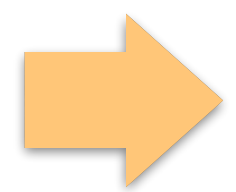
```

Direction is one of...

```

"NORTH"
| "EAST"
| "SOUTH"
| "WEST"

```

```
function stringToCommands(i: string): Commands {  
  return i.split("\n").map((line) => {  
    let matches;  
    if (  
      matches = line.match(  
        /PLACE ([\d]+),([\d]+),(NORTH|  
        SOUTH|EAST|WEST)/), matches  
    ) {  
      switch (matches[3]) { /* ... */ }  
      const x = parseInt(matches[1], 10);  
      // ...  
      return { type: "PLACE", x, y, dir }  
    } else if (line === "LEFT") {  
      return { type: "LEFT" }  
    } else if (line === "RIGHT") {  
      return { type: "RIGHT" }  
    } else if (line === "MOVE") {  
      return { type: "MOVE" }  
    } else if (line === "REPORT") {  
      return { type: "REPORT" }  
    }  
  }).filter(x => x !== undefined);  
}
```

```
type Commands =  
  Array<Command>
```

```
type Command =  
  { type: "PLACE",  
    x: number,  
    y: number,  
    dir: Direction  
  }  
| { type: "LEFT" }  
| { type: "RIGHT" }  
| { type: "MOVE" }  
| { type: "REPORT" }
```

```
type Direction =  
  "NORTH"  
| "EAST"  
| "SOUTH"  
| "WEST"
```

```

function stringToCommands(input) {
  return input.split("\n").map((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|
        SOUTH|EAST|WEST)/, matches
    ) {
      switch (matches[3]) { /* ... */ }
      const x = parseInt(matches[1], 10);
      // ...
      return { type: "PLACE", x, y, dir }
    } else if (line === "LEFT") {
      return { type: "LEFT" }
    } else if (line === "RIGHT") {
      return { type: "RIGHT" }
    } else if (line === "MOVE") {
      return { type: "MOVE" }
    } else if (line === "REPORT") {
      return { type: "REPORT" }
    }
  }).filter(x => x !== undefined);
}

```

Commands are a list of...
Command objects

Command is one of...

```

{ type: "PLACE",
  x: number,
  y: number,
  dir: Direction
}
| { type: "LEFT" }
| { type: "RIGHT" }
| { type: "MOVE" }
| { type: "REPORT" }

```

Direction is one of...

```

"NORTH"
| "EAST"
| "SOUTH"
| "WEST"

```


3つの関数に関して

- ▶ 「runRobot」関数は、stringToCommands、executeCommands、runRobotの3つの関数に分割できます。
- ▶ 最初はストリングを受け取り、いくつかのコマンドを返します（"Command"）。各コマンドはオブジェクトであり、アクションを表します。
- ▶ 2番目の関数はこれらのコマンドを受け取り、いくつかのアクションを実行します。
- ▶ 3番目の関数は2番目の関数を使用します。

```
function stringToCommands(input) {
  return input.split("\n").map((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|
        SOUTH|EAST|WEST)/), matches
    ) {
      switch (matches[3]) { /* ... */ }
      const x = parseInt(matches[1], 10);
      // ...
      return { type: "PLACE", x, y, dir }

    } else if (line === "LEFT") {
      return { type: "LEFT" }
    } else if (line === "RIGHT") {
      return { type: "RIGHT" }
    } else if (line === "MOVE") {
      return { type: "MOVE" }
    } else if (line === "REPORT") {
      return { type: "REPORT" }
    }
  }).filter(x => x === undefined);
}
```

```
it("tests stringToCommands", function() {  
    expect(  
        stringtoCommands([  
            "PLACE 1,1,EAST",  
            "REPORT",  
        ]).join("\n")  
    ).toEqual([  
        { type: "PLACE", x:1, y:1, dir: "EAST" },  
        { type: "REPORT" },  
    ])  
  
});
```

```
it("tests stringToCommands", function() {  
    expect(  
        stringtoCommands([  
            "PLACE 9,9,EAST",  
            "REPORT",  
        ]).join("\n")  
    ).toEqual([  
        { type: "PLACE", x:9, y:9, dir: "EAST" },  
        { type: "REPORT" },  
    ])  
  
});
```

```
it("tests stringToCommands", function() {  
    expect(  
        stringtoCommands([  
            "LEFT",  
            "REPORT",  
        ]).join("\n")  
    ).toEqual([  
        { type: "LEFT" },  
        { type: "REPORT" },  
    ])  
  
});
```

```
it("tests stringToCommands", function() {  
    expect(  
        stringtoCommands([  
            "裏庭には二羽庭には二羽鶏がいる",  
            "REPORT",  
        ]).join("\n")  
    ).toEqual([  
        // ... ignoring that first "command"  
        { type: "REPORT" },  
    ])  
  
});
```

```
it("tests executeCommands", function() {
```

```
// ...
```

```
});
```


分離に関して

- ▶ `stringToCommands`は小さく、定義された入力と出力を持っているので、テストする方が簡単です。
- ▶ `executeCommands`は、チェックする戻り値がないため、まだテストするのが難しいです。しかし、もはや文字列入力を解析しないので、入力の順列を少なくしてテストを実行できます。

"PLACE 1,2,EAST
LEFT
RIGHT
..."



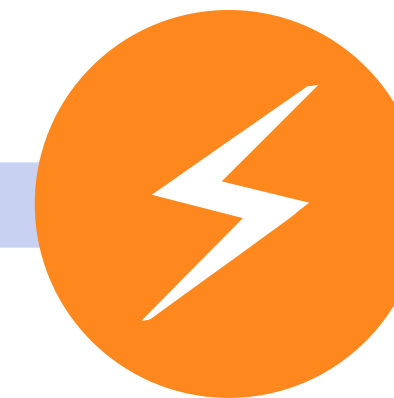
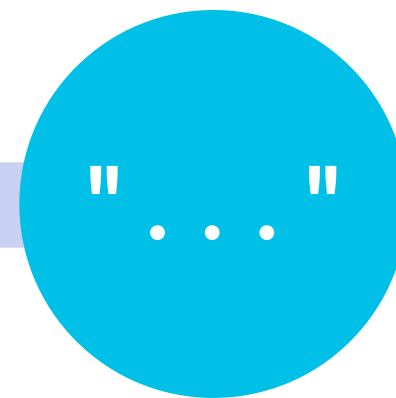
console.log(
"2,2,EAST"
)

```
[ {type: "PLACE",  
  x:1, y:2,  
  dir:"EAST"},  
  {type: "LEFT"},  
  {type: "RIGHT"},  
  ...  
]
```

Instructions

命令

"PLACE 1,2,EAST
LEFT
RIGHT
..."



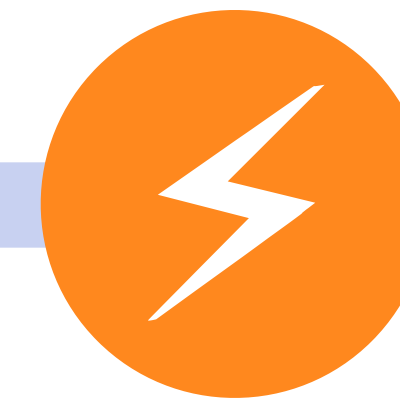
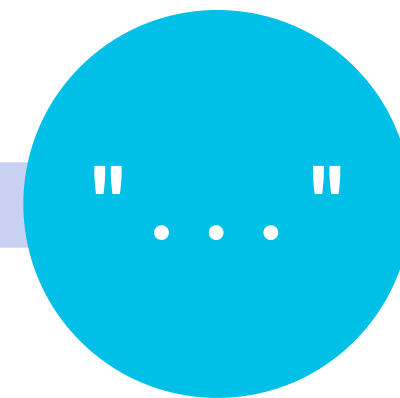
console.log(
"2,2,EAST"
)

```
[ {type: "PLACE",  
  x:1, y:2,  
  dir:"EAST"},  
  {type: "LEFT"},  
  {type: "RIGHT"},  
  ...  
]
```

Instructions

命令

"PLACE 1,2,EAST
LEFT
RIGHT
..."



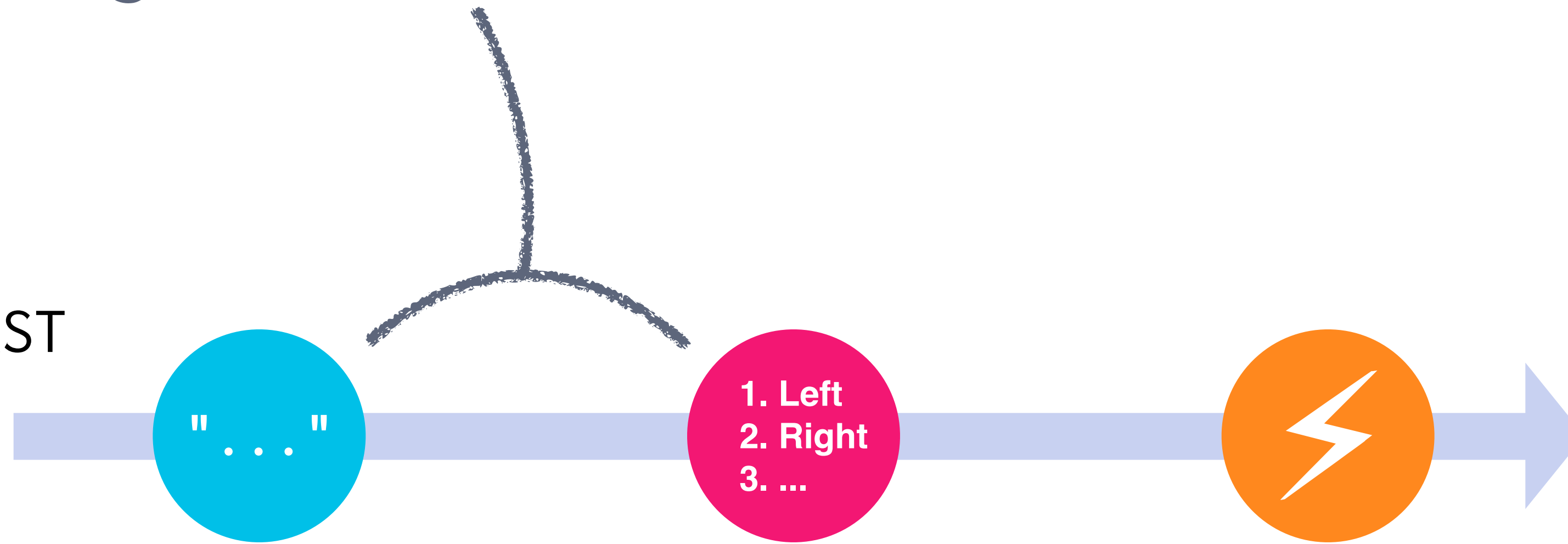
console.log(
 "2,2,EAST"
)

```
[ {type: "PLACE",  
  x:1, y:2,  
  dir:"EAST"},  
  {type: "LEFT"},  
  {type: "RIGHT"},  
  ...  
]
```

Interpreter
解釈

stringToCommands

"PLACE 1,2,EAST
LEFT
RIGHT
..."

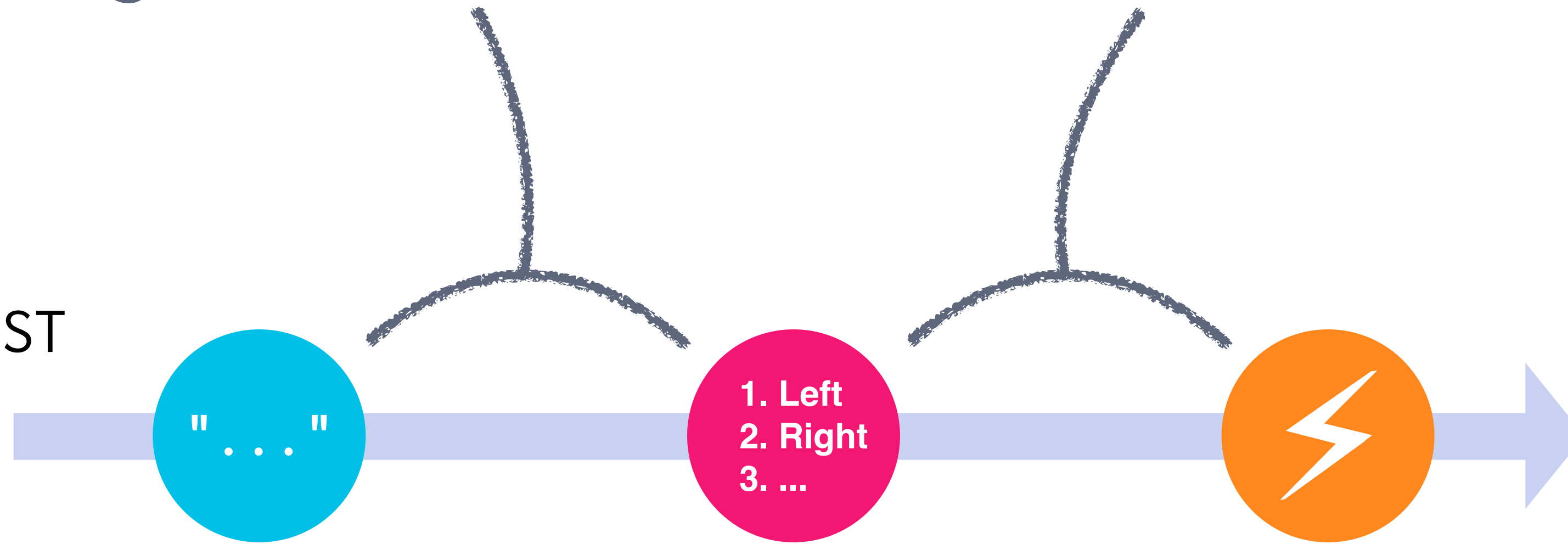


`console.log(
 "2,2,EAST"
)`

```
[ {type: "PLACE",  
  x:1, y:2,  
  dir:"EAST"},  
  {type: "LEFT"},  
  {type: "RIGHT"},  
  ...  
]
```

stringToCommands executeCommands

"PLACE 1,2,EAST
LEFT
RIGHT
..."



console.log(
 "2,2,EAST"
)

```
[ {type: "PLACE",  
  x:1, y:2,  
  dir:"EAST"},  
  {type: "LEFT"},  
  {type: "RIGHT"},  
  ...  
]
```


"PLACE 1,2,EAST
LEFT
RIGHT
..."



console.log(
"2,2,EAST"
)

```
[ {type: "PLACE",  
  x:1, y:2,  
  dir:"EAST"},  
  {type: "LEFT"},  
  {type: "RIGHT"},  
  ...  
]
```

```
[{"t": "PLACE",  
  "x": 1, "y": 2,  
  "f": "EAST"},  
{"t": "LEFT"},  
{"t": "RIGHT"},  
...]
```

```
"PLACE 1,2,EAST  
LEFT  
RIGHT  
..."
```

```
const instructions = [  
  place(1,2,east),  
  left,  
  right,  
  ...  
];
```

"{...}"

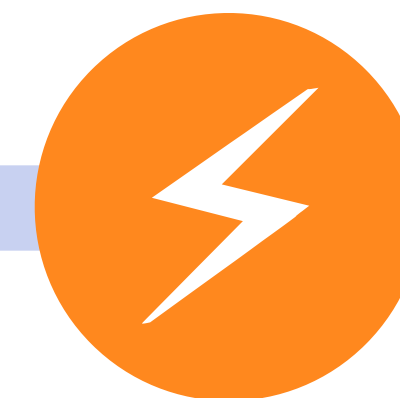
"..."

[...]

1. Left
2. Right
3. ...



```
console.log(...)  
console.log(...)  
console.log(...)
```



```
robot()  
  .move(instructions)  
  .withConnection(serial)  
  .asFutureChain()  
  .toPromise()
```



```
it('moves around',()=>{  
  let r = robot();  
  let instructions =  
    [ ... ];  
});
```

分離に関して

- ▶ 命令と解釈を分けているので、もっと多くのことができます。
- ▶ 複数の方法で指示を出すことができます。
- ▶ これらの命令については、複数の解釈が可能です。

```

function stringToCommands(input) {
  return input.split("\n").map((line) => {
    let matches;
    if (
      matches = line.match(
        /PLACE ([\d]+),([\d]+),(NORTH|
        SOUTH|EAST|WEST)/, matches
    ) {
      switch (matches[3]) { /* ... */ }
      const x = parseInt(matches[1], 10);
      // ...
      return { type: "PLACE", x, y, dir }
    } else if (line === "LEFT") {
      return { type: "LEFT" }
    } else if (line === "RIGHT") {
      return { type: "RIGHT" }
    } else if (line === "MOVE") {
      return { type: "MOVE" }
    } else if (line === "REPORT") {
      return { type: "REPORT" }
    }
  }).filter(x => x !== undefined);
}

```

Commands are a list of...
Command objects

Command is one of...

```

{ type: "PLACE",
  x: number,
  y: number,
  dir: Direction
}
| { type: "LEFT" }
| { type: "RIGHT" }
| { type: "MOVE" }
| { type: "REPORT" }

```

Direction is one of...

```

"NORTH"
| "EAST"
| "SOUTH"
| "WEST"

```

```
function
  return input.split(

  matches = line.match(
    SOUTH|EAST|WEST)
)
const
  // ...
  return { type: "PLACE", x, y, dir }


  return { type: "LEFT" }

  return { type: "RIGHT" }

  return { type: "MOVE"

  return { type: "REPORT" }
}
}
```

Remember these?



Commands are a list of...
Command objects

Command is one of...

```
{ type: "PLACE",
  x: number,
  y: number,
  dir: Direction
}
| { type: "LEFT" }
| { type: "RIGHT" }
| { type: "MOVE" }
| { type: "REPORT" }
```

Direction is one of...

```
"NORTH"
| "EAST"
| "SOUTH"
| "WEST"
```

**We've made a
little language.**

(限られた言語)

Commands are a list of...
Command objects

Command is one of...

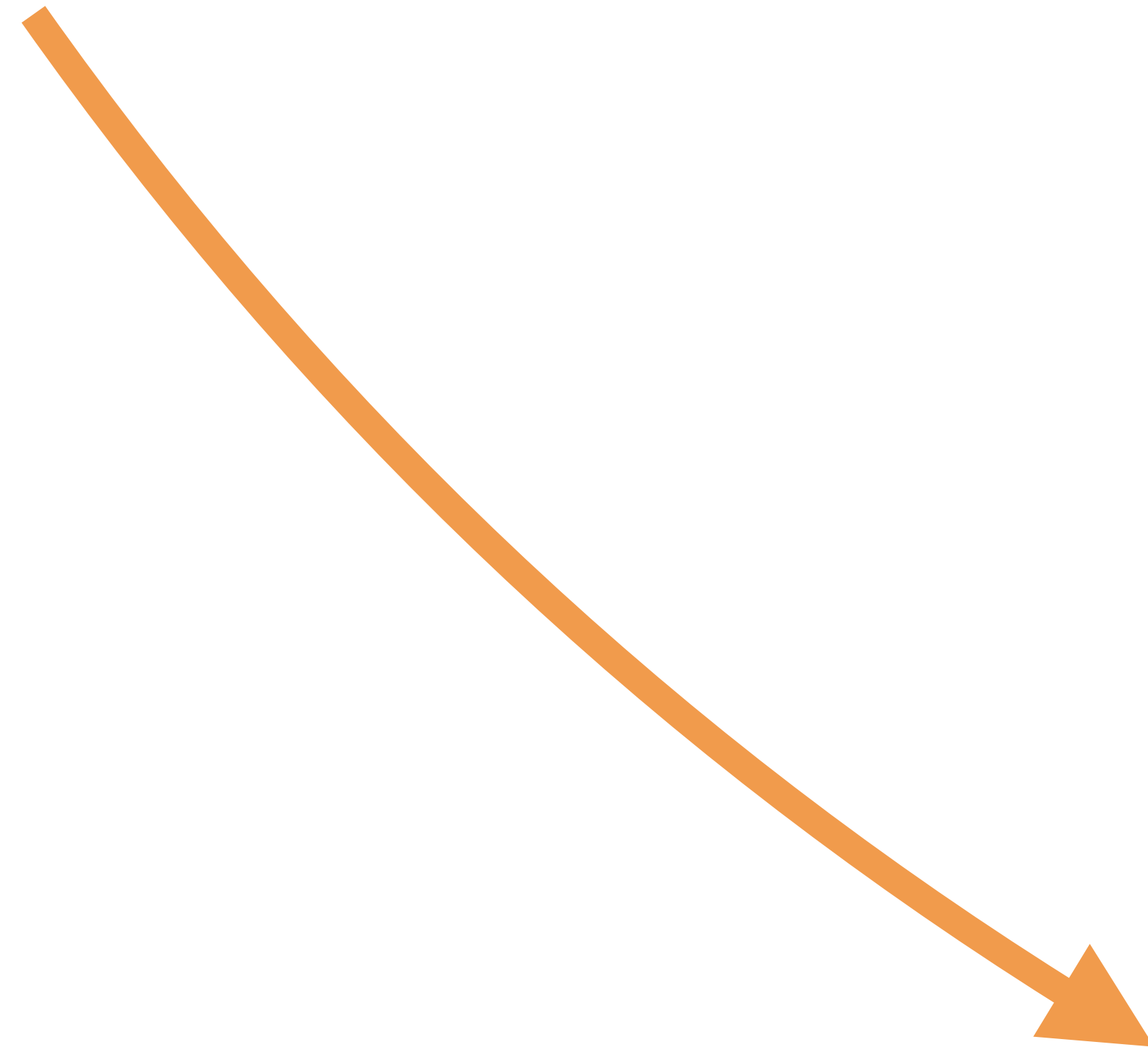
```
{ type: "PLACE",  
  x: number,  
  y: number,  
  dir: Direction  
}  
| { type: "LEFT" }  
| { type: "RIGHT" }  
| { type: "MOVE" }  
| { type: "REPORT" }
```

Direction is one of...

```
"NORTH"  
| "EAST"  
| "SOUTH"  
| "WEST"
```

Words

単語



Commands are a list of...
Command objects

Command is one of...

```
{ type: "PLACE",  
  x: number,  
  y: number,  
  dir: Direction  
}  
| { type: "LEFT" }  
| { type: "RIGHT" }  
| { type: "MOVE" }  
| { type: "REPORT" }
```

Direction is one of...

```
"NORTH"  
| "EAST"  
| "SOUTH"  
| "WEST"
```

Words

単語

Commands are a list of...
Command objects

Command is one of...

```
{ type: "PLACE",  
  x: number, // 1,2,3,...  
  y: number, // 1,2,3,...  
  dir: Direction  
}
```

```
| { type: "LEFT" }  
| { type: "RIGHT" }  
| { type: "MOVE" }  
| { type: "REPORT" }
```

Direction is one of...

```
"NORTH"  
| "EAST"  
| "SOUTH"  
| "WEST"
```

Sentences

文

Commands are a list of...
Command objects

Command is one of...

```
{ type: "PLACE",  
  x: number,  
  y: number,  
  dir: Direction  
}
```

```
| { type: "LEFT" }  
| { type: "RIGHT" }  
| { type: "MOVE" }  
| { type: "REPORT" }
```

Direction is one of...

```
"NORTH"  
| "EAST"  
| "SOUTH"  
| "WEST"
```

Sentences

文

Commands are a list of...
Command objects

Command is one of...

```
{ type: "PLACE",  
  x: number,  
  y: number,  
  dir: Direction  
}
```

```
| { type: "LEFT" }  
| { type: "RIGHT" }  
| { type: "MOVE" }  
| { type: "REPORT" }
```

Direction is one of...

```
"NORTH"  
| "EAST"  
| "SOUTH"  
| "WEST"
```

Sentences

文

Commands are a list of...
Command objects

Command is one of...

```
{ type: "PLACE",  
  x: number,  
  y: number,  
  dir: Direction  
}
```

```
| { type: "LEFT" }  
| { type: "RIGHT" }  
| { type: "MOVE" }  
| { type: "REPORT" }
```

Direction is one of...

```
"NORTH"
```

```
| "EAST"
```

```
| "SOUTH"
```

```
| "WEST"
```

Document

書面



Commands are a list of...
Command objects

Command is one of...

```
{ type: "PLACE",  
  x: number,  
  y: number,  
  dir: Direction  
}  
| { type: "LEFT" }  
| { type: "RIGHT" }  
| { type: "MOVE" }  
| { type: "REPORT" }
```

Direction is one of...

```
"NORTH"  
| "EAST"  
| "SOUTH"  
| "WEST"
```


例えば

Place the robot at 1 across (from the right),
1 upwards (from the bottom), and have it face North.

Turn left.

Go forward one space.

Turn right.

Go forward one space.

Turn right.

Turn right.

Report the robot's position.

```
[  
  { type: "PLACE", x: 1, y: 1, dir: "NORTH" },  
  
  { type: "LEFT" },  
  { type: "FORWARD" },  
  { type: "RIGHT" },  
  { type: "FORWARD" },  
  { type: "RIGHT" },  
  { type: "RIGHT" },  
  { type: "REPORT" },  
]
```

DSL

(Domain-Specific Language)

ドメイン固有言語

INTERNAL DSL

(Domain-Specific Language)

言語内ドメイン固有言語

限られた言語に関して

- ▶ 指示は限られた言語で表現することができます。

**This idea also
shows up in...**

Validation

Validate.js

```
const constraints = {
  username: {
    presence: true,
    exclusion: {
      within: ["bert"],
      message: "'%{value}' is not allowed",
    }
  },
  password: {
    presence: true,
    length: {
      minimum: 6,
      message: "must be at least 6 characters",
    }
  },
};
```

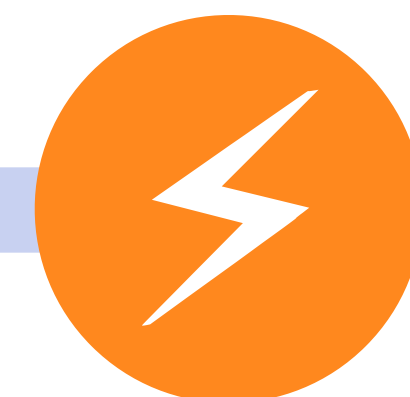
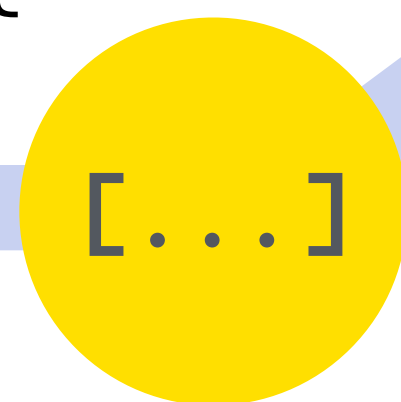
```
const constraints = {  
  username: { ... },  
  password: { ... },  
};
```

```
const constraints = {  
  username: { ... },  
  password: { ... },  
};
```

```
validate(  
  { username: "rob", password: "acceptable" },  
  constraints  
);  
// undefined (no errors)
```

```
validate(  
  { username: "rob", password: "short" },  
  constraints  
);  
// {"password": ["Password must be at least 6 characters"]}
```

```
const constraints = {  
  username: { ... }  
  password: ...  
  ...  
};
```



console.log(...)
console.log(...)
console.log(...)

validate()

User Input
入力

+

io-ts


```
const Person = t.interface({  
  name: t.string,  
  age: t.number  
})  
  
t.validate(  
  JSON.parse('{ "name": "Giulio Canti", "age": 43 }'),  
  Person  
);  
// => Right({name: "Giulio Canti", age: 43})
```

検証に関して

- ▶ フォーム検証ライブラリでは、このパターンもよく使用されます。
- ▶ 検証ルールは指示です。

Redux-Saga

```
function* fetchUserFromServer(action) {  
  const user = yield call(  
    Api.fetchUser,  
    action.payload.userId  
  );  
  yield put({  
    type: "USER_FETCH_SUCCEEDED",  
    user: user  
  });  
}
```

```
function* fetchUserFromServer(action) {  
  const user = yield call(  
    Api.fetchUser,  
    action.payload.userId  
  );  
  yield put({  
    type: "USER_FETCH_SUCCEEDED",  
    user: user  
  });  
}
```

```
function* fetchUserFromServer(action) {  
  const user = yield {..., CALL:{..., action:{  
    fn:    Api.fetchUser,  
    args:  [ action.payload.userId ],  
  }}};  
  yield put({  
    type: "USER_FETCH_SUCCEEDED",  
    user: user  
  });  
}
```

```
function* fetchUserFromServer(action) {  
  const user = yield {..., CALL:{..., action:{  
    fn:    Api.fetchUser,  
    args: [ action.payload.userId ],  
  }}};  
  yield put({  
    type: "USER_FETCH_SUCCEEDED",  
    user: user  
  });  
}
```



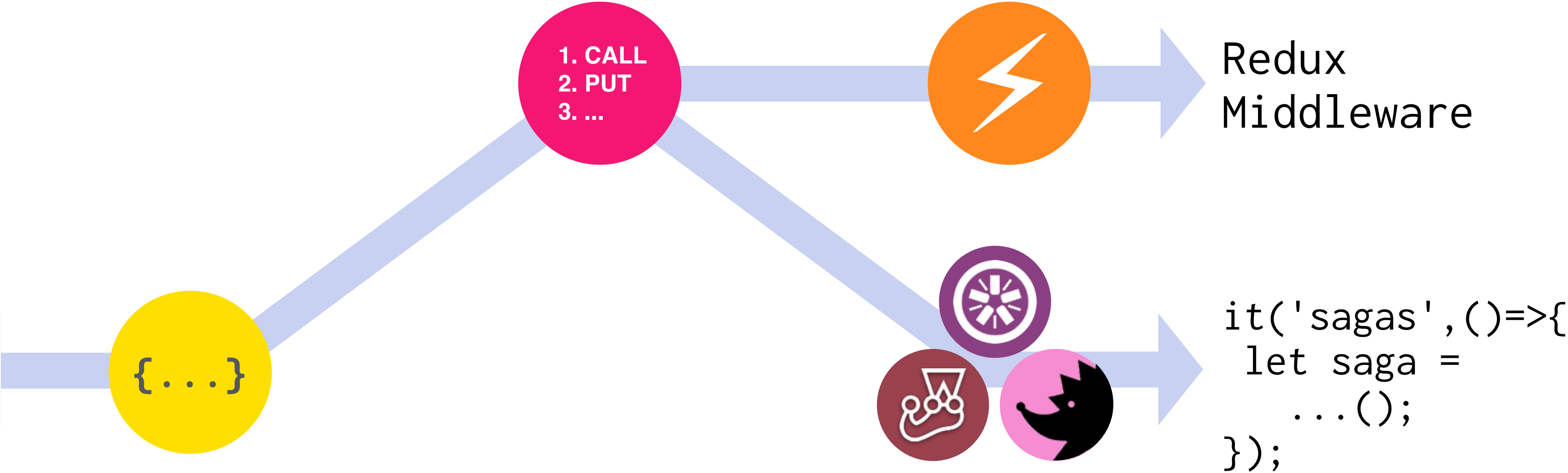
```
function* fetchUserFromServer(action) {  
  const user = yield {..., CALL:{..., action:{  
    fn:    Api.fetchUser,  
    args: [ action.payload.userId ],  
  }}};  
  yield put({  
    type: "USER_FETCH_SUCCEEDED",  
    user: user  
  });  
}
```

```
function* fetchUserFromServer(action) {  
  const user = yield {..., CALL:{..., action:{  
    fn:    Api.fetchUser,  
    args: [ action.payload.userId ]  
  }}};  
  yield {..., PUT: {..., action:{  
    type: "USER_FETCH_SUCCEEDED",  
    user: user  
  }}};  
}
```

```
function* fetchUserFromServer(action) {  
  const user = yield {..., CALL:{..., action:{  
    fn:    Api.fetchUser,  
    args: [ action.payload.userId ]  
  }}};  
  yield {..., PUT: {..., action:{  
    type: "USER_FETCH_SUCCEEDED",  
    user: user  
  }}};  
}
```

**Where's the
Interpreter?**

Saga
Definition



Tests

```
const action = {
  payload: { userId: 123 }}
};
const gen = fetchUserFromServer(action);

assert.deepEqual(
  gen.next().value,
  call(Api.FetchUser, action.payload.userId)
);

assert.deepEqual(
  gen.next().value,
  put({
    type: "USER_FETCH_SUCCEEDED",
    user: makeFakeUser(action.payload.userId),
  })
);
```

Redux-Saga's Redux Middleware


```
// ...
```

```
// declarative effects
```

: is.array(effect)	? runParallelEffect(effect, effectId,
currCb)	
: (data = isEffect.take(effect))	? runTakeEffect(data, currCb)
: (data = isEffect.put(effect))	? runPutEffect(data, currCb)
: (data = isEffect.all(effect))	? runAllEffect(data, effectId, currCb)
: (data = isEffect.race(effect))	? runRaceEffect(data, effectId, currCb)
: (data = isEffect.call(effect))	? runCallEffect(data, effectId, currCb)
: (data = isEffect.cps(effect))	? runCPSEffect(data, currCb)
: (data = isEffect.fork(effect))	? runForkEffect(data, effectId, currCb)
: (data = isEffect.join(effect))	? runJoinEffect(data, currCb)
: (data = isEffect.cancel(effect))	? runCancelEffect(data, currCb)
: (data = isEffect.select(effect))	? runSelectEffect(data, currCb)

```
// ...
```

```
// See:
```

```
// https://github.com/redux-saga/redux-saga/blob/master/src/internal/proc.js
```

```
switch(effect.type) {  
  case "take":    return runTakeEffect(...);  
  case "put":     return runPutEffect(...);  
  case "all":     return runAllEffect(...);  
  case "race":    return runRaceEffect(...);  
  case "call":    return runCallEffect(...);  
  case "cps":     return runCPSEffect(...);  
  case "fork":    return runForkEffect(...);  
  case "join":    return runJoinEffect(...);  
  case "cancel":  return runCancelEffect(...);  
  case "select":  return runSelectEffect(...);  
}
```

```
switch(effect.type) {  
  case "take":    return runTakeEffect(...);  
  case "put":     return runPutEffect(...);  
  case "all":     return runAllEffect(...);  
  case "race":    return runRaceEffect(...);  
  case "call":    return runCallEffect(...);  
  case "cps":     return runCPSEffect(...);  
  case "fork":    return runForkEffect(...);  
  case "join":    return runJoinEffect(...);  
  case "cancel":  return runCancelEffect(...);  
  case "select":  return runSelectEffect(...);  
}
```

```
switch(effect.type) {  
  case "take":      return runTakeEffect(...);  
  case "put":      return runPutEffect(...);  
  case "all":      return runAllEffect(...);  
  case "race":      return runRaceEffect(...);  
  case "call":      return runCallEffect(...);  
  case "cps":       return runCPSEffect(...);  
  case "fork":      return runForkEffect(...);  
  case "join":      return runJoinEffect(...);  
  case "cancel":    return runCancelEffect(...);  
  case "select":    return runSelectEffect(...);  
}
```

Redux-Sagaに関して

- ▶ 「Saga」関数は命令です。
- ▶ インタプリタはアプリケーションのテストであり、Redux-Sagaミドルウェアです。

**What's so "Functional"
about all this?**

**That
"Pure Functions"
Thing.**

```
function addOne(x) {  
    return x + 1;  
}
```

```
var r = addOne(5);  
    // r is 6
```

出力が入力によって完全に決定される場合、
関数は「pure」です。


```
function addOne(x) {  
    return x + 1;  
}
```

```
var r = addOne(1) + addOne(3);  
// r is 6
```

出力が入力によって完全に決定される場合、
関数は「pure」です。

```
function addOne(x) {  
  console.log("Hi!");  
  return x + 1;  
}  
var r = addOne(5);  
  // Logged: "Hi!"  
  // r is 6
```

実行を観察できる場合、関数は「impure」です。

```
function addOne(x) {  
  console.log("Hi!");  
  return x + 1;  
}  
var r = addOne(1) + addOne(3);  
// Logged: "Hi!Hi!"  
// r is 6
```

実行を観察できる場合、関数は「impure」です。

```
function addOne(x) {  
    console.log("Hi!");  
    return x + 1;  
}  
var r = addOne(1) + addOne(3);  
    // Logged: "Hi!Hi!"  
    // r is 6
```

実行を観察できる場合、関数は「impure」です。

Pure things:

stringToCommands()

t.interface()

fetchUserFromServer()

**Representing impure actions as
instructions to be later interpreted
can make them pure.**

**impureなアクションを、
後に解釈される指示として表現することで、
それらをpureにすることができます。**

**Representing impure actions as
instructions to be later interpreted
can make them pure.**

impureなアクションを、
後に解釈される指示として表現することで、
それらをpureにすることができます。

A Quick Confession

**It's also known as the
"Interpreter Pattern"
(from "Design Patterns")**

**これはGoFの
「Interpreter パターン」 とも呼ばれます。**

**It just shows up in the
"FP" community a lot.**

**このような考え方は、Functional Programming
のコミュニティで人気があります。**

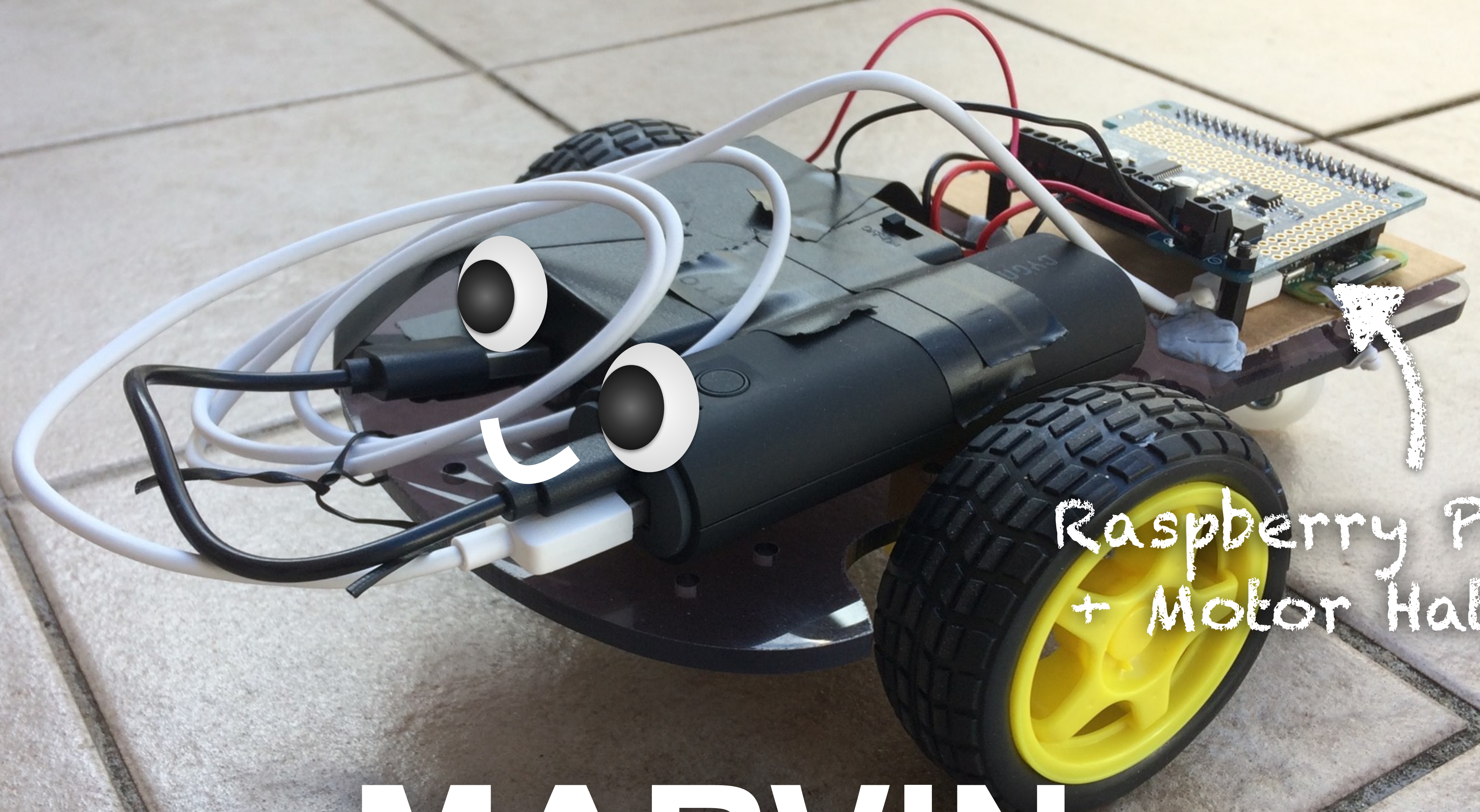
(Free Monads in Haskell and Scala, and
Haskell's entire way of doing side-effects
with its `IO` Type.)

**It just shows up in the
"FP" community a lot.**

**このような考え方は、Functional Programming
のコミュニティで人気があります。**

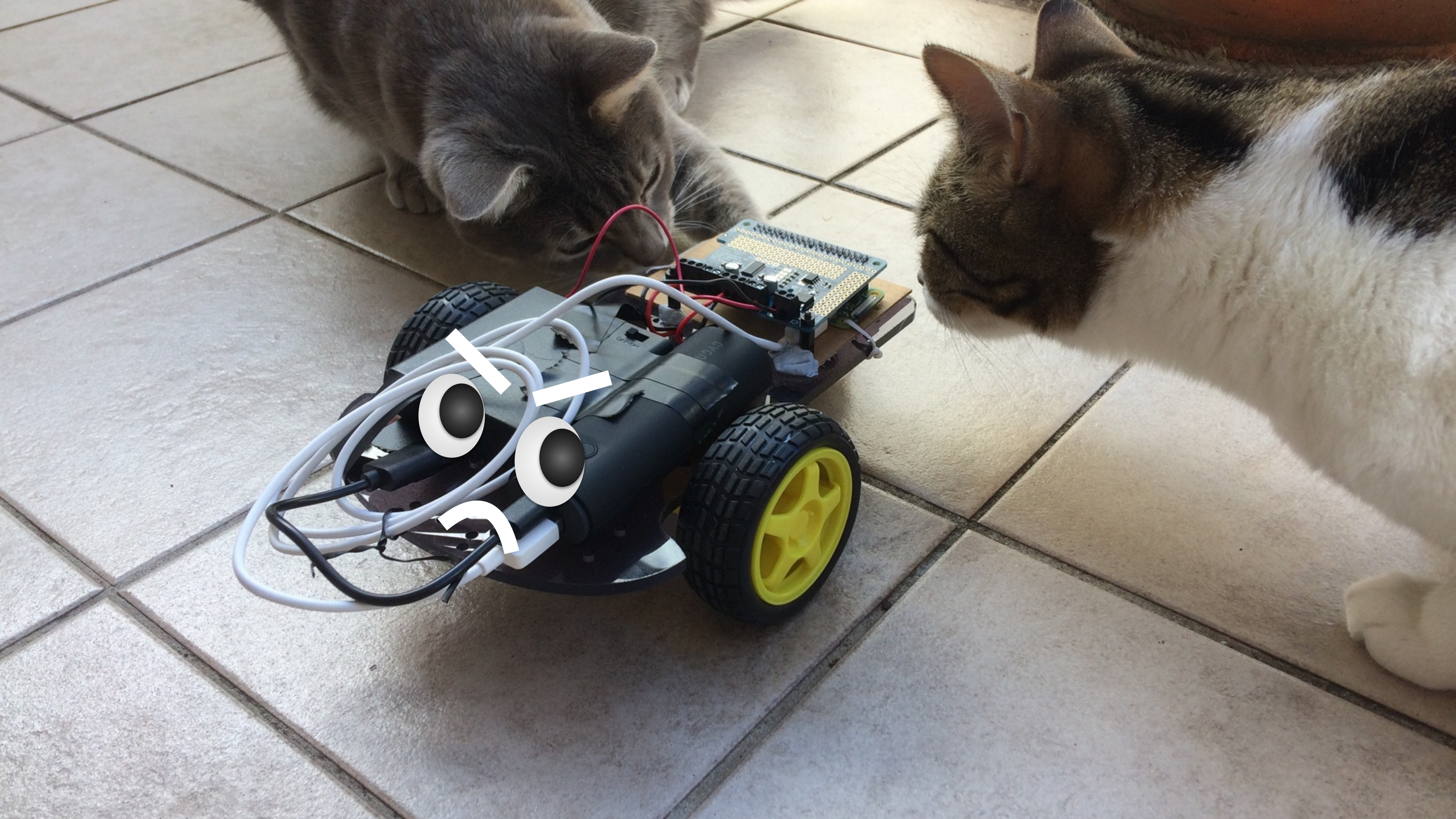
(たとえば、Free Monadsや HaskellのIO型)

Back to the Robot



Raspberry Pi
+ Motor Hat

MARVIN




```
const j5 = require("johnny-five");  
const rp = require("raspi-io");  
const hat = j5.Motor.SHIELD_CONFIGS.ADAFRUIT_V2;  
const board = new j5.Board({io: rp()});
```

```
const j5 = require("johnny-five");
const rp = require("raspi-io");
const hat = j5.Motor.SHIELD_CONFIGS.ADAFRUIT_V2;
const board = new j5.Board({io: rp()});
```

```
board.on("ready", function() {
  const motors = {
    left: new j5.Motor(hat.M1),
    right: new h5.Motor(hat.M2),
  };

```

```
    // ...
  });
```

```
const j5 = require("johnny-five");  
const rp = require("raspi-io");  
const hat = j5.Motor.SHIELD_CONFIGS.ADAFRUIT_V2;  
const board = new j5.Board({io: rp()});
```

```
board.on("ready", function() {  
  const motors = { left: ..., right: ... };
```

```
    // ...  
});
```

```
const j5 = require("johnny-five");
const rp = require("raspi-io");
const hat = j5.Motor.SHIELD_CONFIGS.ADAFRUIT_V2;
const board = new j5.Board({io: rp()});
const Marvin = require("./marvin");
board.on("ready", function() {
  const motors = { left: ..., right: ... };
  const r = new Marvin(motors);

  // ...
});
```

```
const j5 = require("johnny-five");
const rp = require("raspi-io");
const hat = j5.Motor.SHIELD_CONFIGS.ADAFRUIT_V2;
const board = new j5.Board({io: rp()});
const Marvin = require("./marvin");
board.on("ready", function() {
  const motors = { left: ..., right: ... };
  const r = new Marvin(motors);

  // ...
});
```

```
const logic = require("./logic");

function Marvin(motors) { ... }

const motorSpeed = 100; // 0 - 250
const turnTime = 700; // milliseconds
const forwardTime = 700; // milliseconds

Marvin.prototype.initialState = logic.initialState;

Marvin.prototype.turnLeft = function(state) {
  if (!logic.isPlaced(state)) return; // not placed yet? ignore

  const newState = logic.turnLeft(state)

  motors.left.reverse(motorSpeed);
  motors.right.forward(motorSpeed);
  return sleep(turnTime, (res) => {
    motors.left.stop()
    motors.right.stop()
    res(newState);
  });
});
```

```
const logic = require("../logic");

function Marvin(motors) { ... }

const motorSpeed = 100; // 0 - 250
const turnTime = 700; // milliseconds
const forwardTime = 700; // milliseconds

Marvin.prototype.initialState = logic.initialState;

Marvin.prototype.turnLeft = function(state) {
  if (!logic.isPlaced(state)) return; // not placed yet? ignore

  const newState = logic.turnLeft(state)

  motors.left.reverse(motorSpeed);
  motors.right.forward(motorSpeed);
  return sleep(turnTime, (res) => {
    motors.left.stop()
    motors.right.stop()
    res(newState);
  });
});
```



```
const logic = require("./logic");

function Marvin(motors) { ... }

const motorSpeed = 100; // 0 - 250
const turnTime = 700;    // milliseconds
const forwardTime = 700; // milliseconds

Marvin.prototype.initialState = logic.initialState;

Marvin.prototype.turnLeft = function(state) {
  if (!logic.isPlaced(state)) return; // not placed yet? ignore

  const newState = logic.turnLeft(state)

  motors.left.reverse(motorSpeed);
  motors.right.forward(motorSpeed);
  return sleep(turnTime, (res) => {
    motors.left.stop()
    motors.right.stop()
    res(newState);
  });
});
```

```
const j5 = require("johnny-five");
const rp = require("raspi-io");
const hat = j5.Motor.SHIELD_CONFIGS.ADAFRUIT_V2;
const board = new j5.Board({io: rp()});
const Marvin = require("./marvin");
board.on("ready", function() {
  const motors = { left: ..., right: ... };
  const r = new Marvin(motors);

  // ...
});
```

```
board.on("ready", function() {  
  const motors = { left: ..., right: ... };  
  const r = new Marvin(motors);
```

```
    // ...  
});
```

```
board.on("ready", function() {  
  const motors = { left: ..., right: ... };  
  const r = new Marvin(motors);  
  const input = [  
    "PLACE 1,1,NORTH",  
    "LEFT",  
    "FORWARD",  
    "RIGHT",  
    // ...  
  ];  
  
  // ...  
});
```

```
board.on("ready", function() {  
  const motors = { left: ..., right: ... };  
  const r = new Marvin(motors);  
  const input = ["PLACE 1,1,NORTH", "LEFT", ...];
```

```
    // ...  
});
```

```
board.on("ready", function() {  
  const motors = { left: ..., right: ... };  
  const r = new Marvin(motors);  
  const input = ["PLACE 1,1,NORTH", "LEFT", ...];  
  const commands = stringToCommands(input);
```

```
    // ...  
});
```

```
board.on("ready", function() {  
  const motors = { left: ..., right: ... };  
  const r = new Marvin(motors);  
  const input = ["PLACE 1,1,NORTH", "LEFT", ...];  
  const commands = stringToCommands(input);  
  commands.reduce((p, command) => {  
  
    }, Promise.resolve(r.initialState));  
});
```

```
board.on("ready", function() {
  const motors = { left: ..., right: ... };
  const r = new Marvin(motors);
  const input = ["PLACE 1,1,NORTH", "LEFT", ...];
  const commands = stringToCommands(input);
  commands.reduce((p, command) => {
    switch(command.type) {
      case "PLACE":
        return p.then(state => {
          log(`Place on ${x},${y} facing ${dir}`);
          return state;
        });
    }
  }, Promise.resolve(r.initialState));
});
```



```
board.on("ready", function() {
  const motors = { left: ..., right: ... };
  const r = new Marvin(motors);
  const input = ["PLACE 1,1,NORTH", "LEFT", ...];
  const commands = stringToCommands(input);
  commands.reduce((p, command) => {
    switch(command.type) {
      case "PLACE":
        return p.then(passThrough(() =>
          log(`Place on ${x},${y} facing ${dir}`)
        ));
    }
  }, Promise.resolve(r.initialState));
});
```

```
board.on("ready", function() {  
  const motors = { left: ..., right: ... };  
  const r = new Marvin(motors);  
  const input = ["PLACE 1,1,NORTH", "LEFT", ...];  
  const commands = stringToCommands(input);  
  commands.reduce((p, command) => {  
    switch(command.type) {  
      case "PLACE":  
        return p.then(passThrough(() => ...));  
  
        // ...  
    }  
  }, Promise.resolve(r.initialState));  
});
```

```
board.on("ready", function() {
  const motors = { left: ..., right: ... };
  const r = new Marvin(motors);
  const input = ["PLACE 1,1,NORTH", "LEFT", ...];
  const commands = stringToCommands(input);
  commands.reduce((p, command) => {
    switch(command.type) {
      case "PLACE":
        return p.then(passThrough(() => ...));
      case "LEFT":
        return p.then(r.turnLeft);
      // ...
    }
  }, Promise.resolve(r.initialState));
});
```

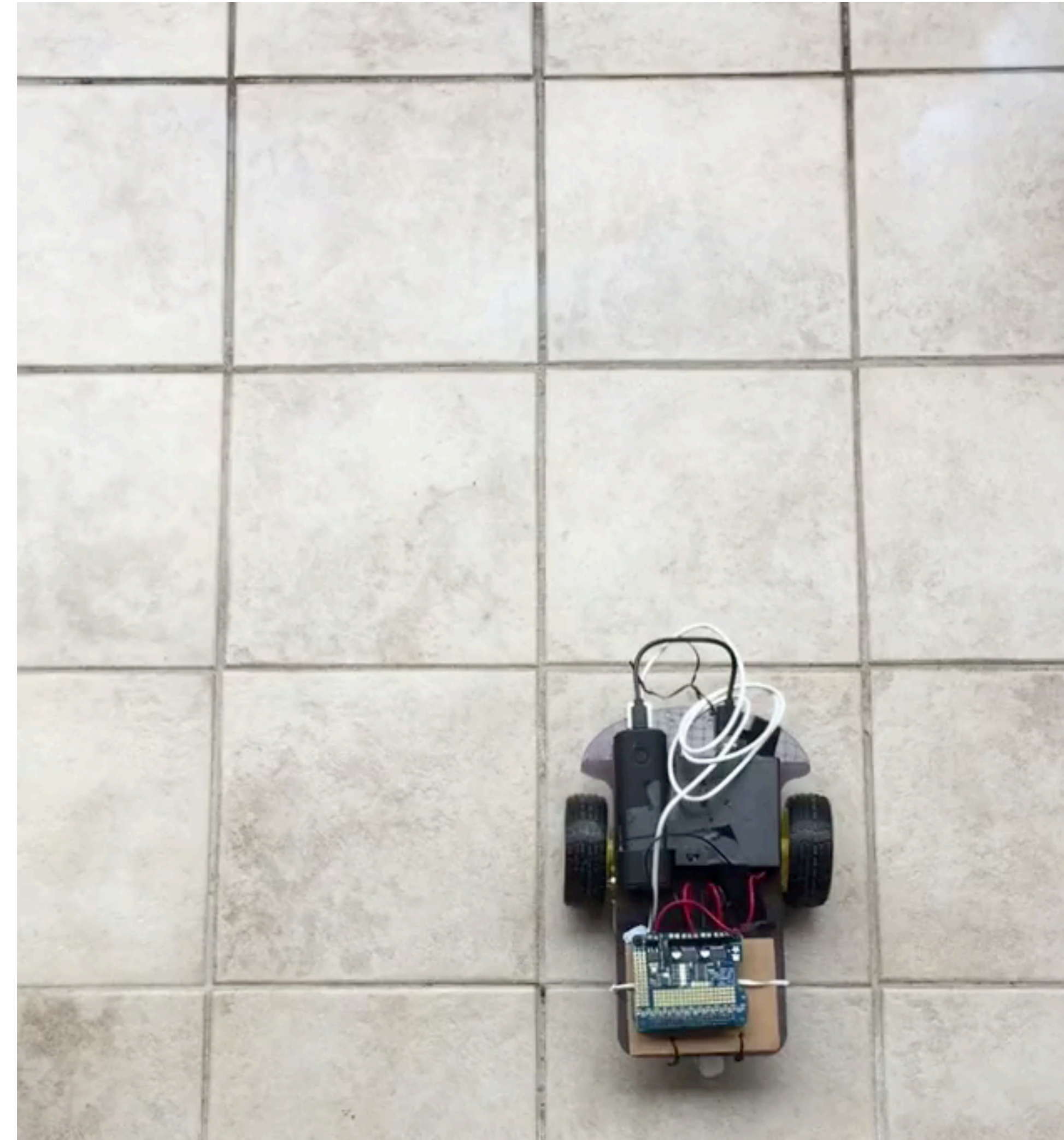
```
commands.reduce((p, command) => {  
  switch(command.type) {  
    case "PLACE":  
      return p.then(passThrough(() =>  
        log(`Place on ${x},${y} facing ${dir}`)  
      ));  
    case "LEFT":      return p.then(r.turnLeft);
```

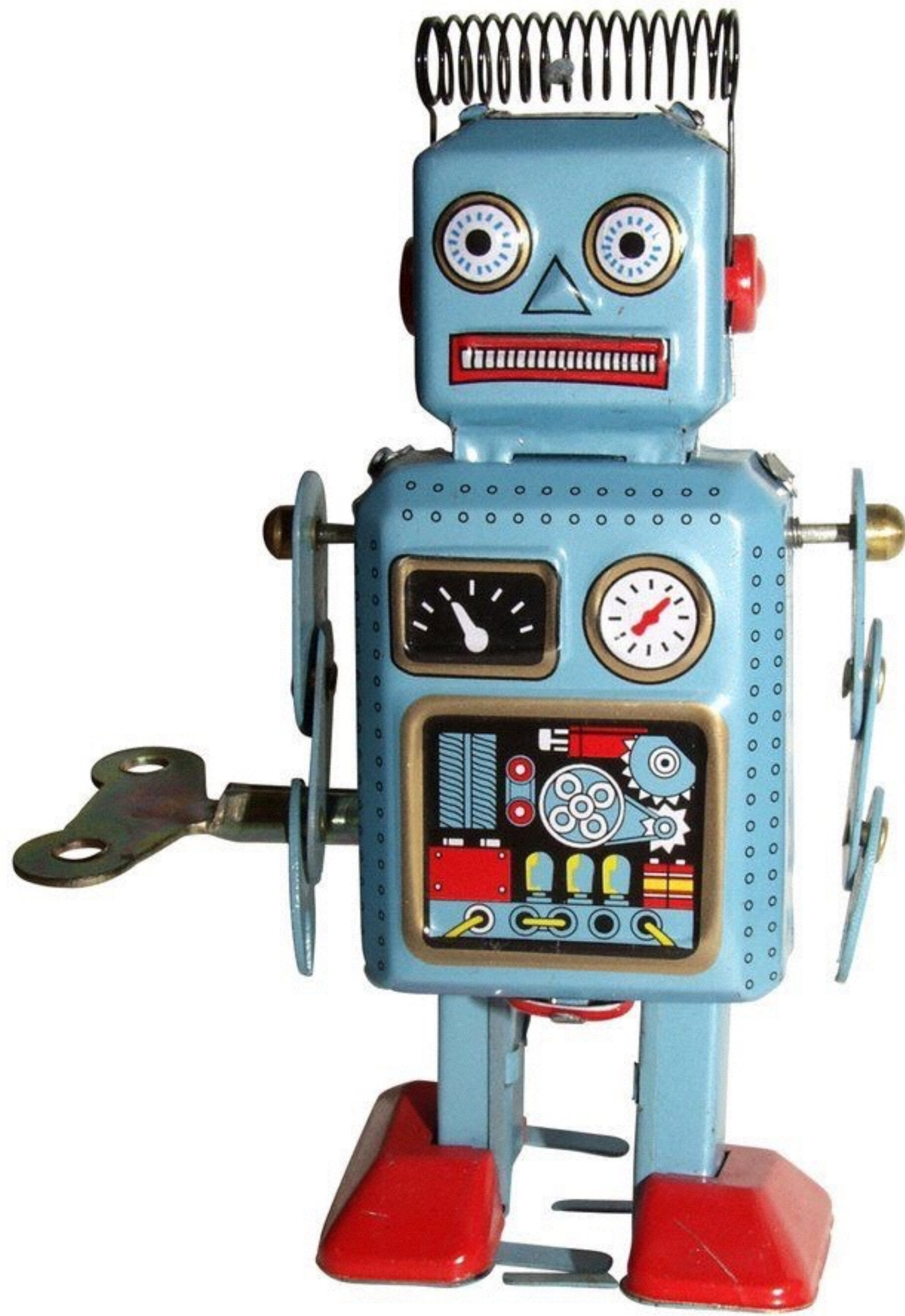
```
    // ...  
  }, Promise.resolve(r.initialState));  
});
```

```
commands.reduce((p, command) => {  
  switch(command.type) {  
    case "PLACE":  
      return p.then(passThrough(() =>  
        log(`Place on ${x},${y} facing ${dir}`)  
      ));  
    case "LEFT":      return p.then(r.turnLeft);  
    case "RIGHT":     return p.then(r.turnRight);  
    case "FORWARD":   return p.then(r.moveForward);  
    case "REPORT":  
      return p.then(passThrough((state) =>  
        log(`${state.x},${state.y},${dirStr(state.dir)}`)  
      ));  
  }, Promise.resolve(r.initialState));  
});
```



```
> .then(r.place(1,1,north))  
  .then(r.turnLeft)  
  .then(r.moveForward)  
  .then(r.turnRight)  
  .then(r.moveForward)  
  .then(r.turnRight)  
  .then(r.moveForward)  
  .then(r.turnRight)  
  .then(r.moveForward)  
  .then(r.turnRight)  
  .then(r.turnRight)
```





```
[{"t": "PLACE",  
  "x": 1, "y": 2,  
  "f": "EAST"},  
 {"t": "LEFT"},  
 {"t": "RIGHT"},  
 ...]
```

```
"PLACE 1,2,EAST  
LEFT  
RIGHT  
..."
```

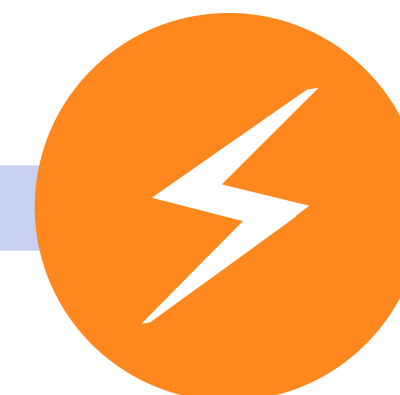
```
const instructions = [  
  place(1,2,east),  
  left,  
  right,  
  ...  
];
```

"{...}"

"..."

[...]

1. Left
2. Right
3. ...



```
console.log(...)  
console.log(...)  
console.log(...)
```

```
robot()  
  .move(instructions)  
  .withConnection(serial)  
  .asFutureChain()  
  .toPromise()
```

```
it('moves around',()=>{  
  let r = robot();  
  let instructions =  
    [ ... ];  
});
```


“

Functional Core,
Imperative [Effectful] Shell”

— Gary Bernhardt, "Boundaries"
<https://www.youtube.com/watch?v=yTkzNHF6rMs>

A Toy Robot and a Functional Pattern

関数型プログラミングにインスピレーションを
得た模様は、命令と解釈を分離します。

A Functional Programming-inspired pattern to
separate instructions and interpretations.



Rob Howard
@damncabbage
<http://robhoward.id.au>