TRAINING

AI協働開発ハンズオン講座

情報系トラック Day3

AI駆動テスト設計（単体・E2E・セキュリティ）と認定

Givery

本資料の二次転載禁止について

本資料の二次転載を禁止します。

本資料は株式会社ギブリーが作成した著作物です。受講者ご本人の学習および業務での参照に限りご利用いただけます。社外への配布、SNS等への掲載、第三者への共有はご遠慮ください。

研修中の画面撮影・録画も、資料が写り込む範囲ではお控えください。

Day2からの接続

Day2で作った観点表を、今日は実際に走るテストに変えます

ここまでにやったこと

Day1では仕様5行から動くWebアプリを作りました。Day2では既存の作業記録アプリに機能を足し、テスト観点を表に起こしました。表を作ったところで止まっています。

これからやること

午前は、その観点表を受入基準まで書き直し、ブラウザだけで走る単体テストに変えます。あわせてセキュリティの観点を2つ、自分のコードに当てて確認します。

そのあとどこで効くか

午後の実技課題では、別のデータに対して同じ手順を一人で回します。午前がその手順の練習にあたります。手順は同じで、題材だけが変わります。

01

今日の位置づけ

3日間のどこにいるか、今日1日がどう進むかを確認します

今日1日の流れ

午前は演習3本、午後はデモと実技課題です

成果物はすべてデスクトップの tpro-work フォルダの中に置きます。午後も同じ場所を使います。

Day2成果物の使い道

Day2のファイルは4つとも今日そのまま使います

観点表	Day2午後に作った観点の表です。今日は受入基準に書き直してから使います。行数が少なくても構いません
copilot-instructions.md	Day2で置いた指示ファイルは今日もそのまま効きます。テストの書き方の指示を1行足します
作業記録アプリ	日付・作業種別・内容・工数(時間) の4欄を持つ index.html です。今日は工数(時間)の入力チェックだけを扱います
tpro-work フォルダ	今日作るファイルは全部この中です。デスクトップに無い人は今この場で作ってください

テスト設計に入る前

この60分は手を動かしません。何を書くかを決める時間です

ここまでにやったこと

今日の並びと、Day2の成果物のどれを使うかを確認しました。ファイルは全員の手元で開いた状態です。

これからやること

60分かけて、AIに渡してよい仕事と人が決めるしかない仕事の線を引きます。持ち帰ってもらうのは受入基準の書き方と、観点の型5つです。

そのあとどこで効くか

休憩明けの100分で、ここで書いた受入基準をそのままCopilotへ渡してケース表を作ります。午後の実技課題でも同じ型を使います。

02

テスト設計とAIの分担

合格の線を引くのは人、候補を並べるのはAI。この分け方を使えるようにします

通過と充足の違い

テストが全件通っても、仕様を満たした証拠にはなりません

テストが通った

実装の振る舞いと、書いた期待結果が一致した状態です。期待結果の出どころは問われていません。実装を読んで期待結果を書けば、必ず全件通ります。件数を増やしても、通る件数が増えるだけです。

仕様を満たした

仕様に書かれた合格条件と、実装の振る舞いが一致した状態です。判定の元は仕様書と受入基準で、実装は根拠に使いません。1件でも合わなければ、実装かテストのどちらかが仕様から外れています。

解説

全件PASSが起きる仕組み

実装を読んで期待結果を書くと、食い違いは一度も表に出ません

工数の上限は0.5以上24以下という仕様です。実装は240で判定しています。同じ1件のケースを、実装から書いた場合と受入基準から書いた場合で並べます。

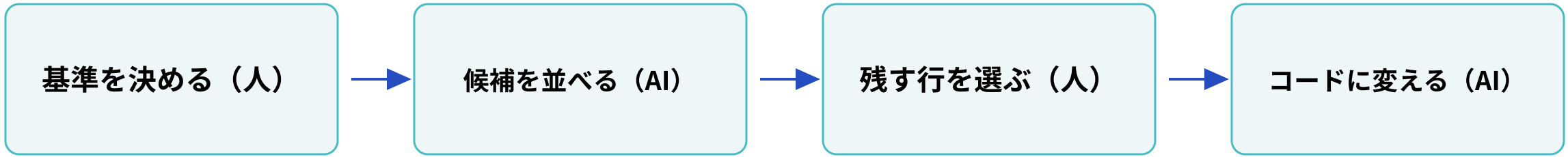
```
// validate.js（仕様は 0.5以上24以下）
function validateHours(v) {
const n = Number(v);
return n >= 0.5 && n <= 240; // 24 のはずが 240
}

// 実装を読んで書いた期待結果 → PASSになる
check('24.5は合格', validateHours('24.5'), true);

// 受入基準から書いた期待結果 → ここで初めてFAILになる
check('24.5は不合格', validateHours('24.5'), false);
```

人とAIの分担

線を引くのは人、並べるのはAI。この順番は入れ替えません



合格と不合格の境目を数字で書きます。仕様書を読んだ人にしか決められません

基準と観点の型を渡すと、20件でも30件でも数十秒で返ります

業務上あり得ない入力を消し、期待結果を基準と突き合わせて直します

選び終わった表を渡せば、テストコードは書かせて構いません

受入基準の書き方

数えられる形になるまで書き直します

1. **対象を1つに絞る**
   画面ではなく、入力欄1つか関数1つまで絞ります。今日は工数(時間)の欄だけです。範囲が広いと基準が文章になりません
2. **合格を数字で書く**
   0.5以上24以下、小数点は1つまで。適切な値、妥当な範囲、正しい入力は基準になりません
3. **不合格を列挙する**
   0、24.5、全角の８、空文字、abc、1..5。落ちてほしい入力を先に書き出します。ここがケース表の材料になります
4. **どうなるかを書く**
   不合格のとき何が起きるか。false を返す、一覧に行が追加されない。判定できる形で1つ決めます

書き直しの一例

同じことを言っているつもりで、期待結果が書けるかが変わります

仕様書によくある書き方

工数には適切な値を入力できること。不正な値はエラーとすること。 → 適切と不正の境目が読む人によって変わります。この2行からはケースを1件も起こせません

書き直したあと

工数(時間)は半角数字で0.5以上24以下、小数点は1つまで。範囲外・全角数字・空文字・数字以外は false を返し、一覧に行を追加しない。 → 境界が4つ、異常が4つ、そのまま数えられます

観点の型5つ

この5つは演習1でそのままCopilotに貼ります

型	狙い	工数(時間)欄での例
同値	同じ扱いになる範囲から代表を1つ選ぶ	8 を入れる（0.5から24の代表として1件）
境界	範囲の端と、その1つ外側を狙う	0.5 / 24 / 0.4 / 24.5
異常	型や書式が違う入力を渡す	全角の８ / 空文字 / abc / 1..5
状態	操作の順番や回数で変わる挙動を見る	空のまま追加ボタンを2回続けて押す
表示	画面に出た結果の見え方を見る	0.5 が 0.5 と出るか、追加した行が一覧の末尾に付くか

> 型ごとに最低1件を作れば抜けの大半は防げます。それ以上は工数と相談です。

実演

ケース候補を出す依頼文

最後の1文が無いと、実装をなぞった期待結果が返ります

この文をそのままCopilot Chatに貼ります。受入基準の3行は、自分が書いたものに差し替えます。

```
次の受入基準に対するテストケース候補を表で出してください。
列は 観点 / 入力 / 期待結果 の3列です。
観点は 同値・境界・異常・状態・表示 の5種類から選んでください。

【受入基準】
工数(時間)は半角数字で0.5以上24以下、小数点は1つまで。
範囲外・全角数字・空文字・数字以外は false を返す。
不合格のとき一覧に行は追加されない。

実装は見ずに、この基準だけから作ってください。
```

クイズ

理解度チェック 1

Q. AIが出したテストケース表が返ってきました。最初にやることはどれですか。

A　件数が少ないので、追加の候補を出させる

B　期待結果の列を、受入基準と1件ずつ突き合わせる

C　テストコードに変えて、まず全件走らせてみる

D　カバレッジを測って、80%に届いているか確認する

正解は次のページで確認します。まず自分で考えてみてください。

解説

解説

正解はB。期待結果の出どころを、コードにする前に確かめます

B **期待結果の列を、受入基準と1件ずつ突き合わせる**

期待結果が実装から写されていると、全件PASSでも仕様との差は一度も表に出ません。表の状態で突き合わせるのが一番安く済みます。コードにしたあとで気づくと、書き直しの手間が増えます。

A 件数が少ないので追加の候補を出させる

件数は品質の指標になりません。5件でも境界を押さえていれば足りますし、30件あっても期待結果が実装の写しなら1件も仕様を確かめていません。

C テストコードに変えて全件走らせてみる

走らせれば結果は出ますが、期待結果が間違っていれば結果も間違います。順番が逆です。突き合わせは表のうちに済ませます。

D カバレッジを測って80%を確認する

通った行の割合は上がりますが、期待結果の正しさは測れません。今日の題材の validateHours は3行なので、1件走らせただけで100%になります。

突き合わせは全件やらなくて構いません。境界と異常の型に付いた4件だけで、写しかどうかは分かります。

ハンズオンに入る前

100分で演習を3本回します。成果物は tpro-work フォルダの中です

ここまでにやったこと

受入基準の書き方、観点の型5つ、期待結果を先に突き合わせる理由を扱いました。手はまだ動かしていません。

これからやること

100分で演習を3本回します。受入基準からケース表を作る、ケース表をブラウザで走るテストに変える、自分のコードにセキュリティレビューをかける。成果物はすべてデスクトップの tpro-work フォルダに置きます。

そのあとどこで効くか

午後の実技課題では、CSVを読み込むアプリに対して同じ3本を一人で回します。ここで手が動くようになっていれば、午後は題材が変わるだけです。

03

仕様書からテストまで

受入基準、ケース表、走るテストの3つを自分の手で作ります

演習の題材と仕様

題材は作業記録アプリの工数(時間)欄、1か所だけです

項目	内容
対象	Day2から使っている作業記録アプリ index.html の、工数(時間) 欄の入力チェック。判定は validate.js の validateHours(value) が持ちます
合格条件	半角数字で0.5以上24以下。小数点は1つまで。合格なら true を返します
不合格の扱い	false を返し、一覧に行を追加しません。エラー文の文言は今日は問いません
置き場所	デスクトップの tpro-work フォルダ。index.html と validate.js が同じ階層にあります
やらないこと	画面の見た目、日付欄と内容欄と作業種別欄のチェック、サーバー側の処理。今日は工数欄だけです

3段の受け渡し

段ごとに、次へ渡すファイルの名前が決まっています

仕様 → 受入基準

人が書きます。次へ渡すのは3行のテキスト。演習1の前半です

受入基準 → ケース表

人とAIで作ります。次へ渡すのは cases.md。演習1の後半です

ケース表 → 走るテスト

AIに書かせます。できるのは tests.html と tests.js。演習2です

3つのファイルはすべてデスクトップの tpro-work フォルダの中、index.html と同じ階層に作ります。

ワーク

演習1 ケース表の作成

客先から届く仕様は文章です。そこからケース表を起こすところが、今日いちばん現場で使う手順です

どんな場面か

客先から渡された仕様書に「工数には適切な値を入力できること」と1行だけ書いてあります。この1行では期待結果が書けないので、まず数えられる形に直してからケース候補を出します。派遣先で最初に頼まれるのは、たいていこの書き直しの部分です。

やること

1. tpro-work フォルダに cases.md を新規作成し、先頭に受入基準を3行で書きます。半角数字だけ、0.5以上24以下、小数点は1つまで、の3点を数字を入れて書きます
2. Copilot Chat を Ctrl+Alt+I で開き、ケース候補を出す依頼文のページにある文をそのまま貼ります。【受入基準】の下は、自分が書いた3行に差し替えます
3. 返ってきた表を cases.md に貼り付け、業務上あり得ない行と、同じ扱いになる重複行を消して8件から12件に絞ります

ワーク

演習1 ケース表の作成（続き）

やること

1. 境界と異常の型に付いた4件だけ、期待結果を受入基準と突き合わせて、違っていれば自分で直します
2. 消した行を1つ選び、なぜ消したかを cases.md の末尾に1行書きます

できあがるもの

デスクトップの tpro-work フォルダに cases.md。受入基準3行、ケース表8件から12件、末尾に消した理由が1行入った状態です。

演習1の到達点

件数ではなく、中身の4点で見ます

達成の目安

- 観点の列に、5種類のうち3種類以上が入っている
- 境界の型に 0.4 / 0.5 / 24 / 24.5 の4件が入っている
- 期待結果の列に「適切」「妥当」「正しく」という語が1つも無い
- cases.md の末尾に、消した行の理由が1行ある

つまずいたら

- 8件に届かない。異常の型が抜けています。全角の８・空文字・abc・1..5 の4件を出させてください。状態の型は今日の題材では作りにくいので勧めません
- 20件以上返って絞れない。「同じ扱いになる入力は1件に寄せてください」と追加で頼ませます
- 期待結果が全部「エラー」で埋まっている。false を返すのか行が追加されないのか、どちらを見るのかを1件ずつ決めさせます
- Copilot が実装のコードを読んで期待結果を書いている。依頼文の最後の1文が抜けています。貼り直させてください

演習1の到達点（続き）

時間が余ったら

- 同じ受入基準を英語で貼り直して、返ってくる候補の粒度が変わるか比べます。cases.md には反映しません
- 日付欄の受入基準を3行だけ書いてみます。ケース表は作りません

実演

判定関数の中身

10行書けば、テストランナーは入れなくて済みます

比較して、結果を1行ずつ画面に足すだけの関数です。写経は要りません。演習2ではCopilotに書かせます。

```
const RESULT = document.getElementById('result');

function check(name, actual, expected) {
const ok = actual === expected;
const li = document.createElement('li');
li.textContent = (ok ? 'PASS' : 'FAIL') + ' ' + name;
li.style.fontWeight = ok ? 'normal' : 'bold';
RESULT.appendChild(li);
}

check('8は合格', validateHours('8'), true);
```

補足

tests.html の骨格

id は result で固定します

6行です。デスクトップの tpro-work フォルダに、index.html と validate.js と同じ階層で作ります。

```
<!doctype html>
<meta charset="utf-8">
<h1>テスト結果</h1>
<ul id="result"></ul>
<script src="validate.js"></script>
<script src="tests.js"></script>
```

> 結果が出ないときは F12 でコンソールを開き、赤い行を読みます。ul の id と getElementById の文字列が違う、が一番多い原因です。

解説

書く場所と動かす場所

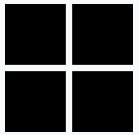
書くのはVSCode、動かすのはブラウザ。この2つを行き来します

VSCode

tests.html と tests.js を書いて Ctrl+S で保存する場所です

GitHub Copilot

Ctrl+Alt+I で開き、ケース表をテストコードに変える依頼を出します



Edge

tests.html を右クリックから開き、PASS と FAIL の並びを見ます

ワーク

演習2 ブラウザ実行

テスト結果の画面を客先に見せる場面を想定します。ツールを入れられない現場でも、この形なら出せます

どんな場面か

客先から「テストをやった証拠を見せてほしい」と言われる場面を想定します。テストランナーを入れられない現場でも、ブラウザで開いてPASSとFAILが並んだ画面をそのまま見てもらえます。今日はその画面を自分の手で作ります。

やること

1. tpro-work フォルダに tests.html を、前のページの6行で作ります。id は result のままにします
2. cases.md を開いた状態で Copilot Chat に頼みます。「この表の各行を check(name, actual, expected) の呼び出しに変えて tests.js を作ってください。check の定義は書かないでください」

ワーク

演習2 ブラウザ実行（続き）

やること

1. 返ってきたコードを tests.js として tpro-work フォルダに保存し、tests.html をエクスプローラーから右クリックしてブラウザで開きます
2. FAILが出た行を1件選び、直すのは実装かテストかを cases.md の末尾に理由付きで1行書きます

できあがるもの

tpro-work フォルダに tests.html と tests.js。ブラウザに PASS と FAIL が並んだ一覧が出ていれば達成です。

演習2の到達点

FAILが1件出ていること。全件PASSは未達です

達成の目安

- 太字のFAILが1件出ている。24.5 を不合格としたケースが落ちます
- PASSの件数が、cases.md のケース件数から1を引いた数と一致している
- cases.md の末尾に、実装かテストかどちらを直すかの判断が1行ある

つまずいたら

- 画面が真っ白。tests.html と validate.js が同じ階層に無い場合です。script の src とフォルダの中身を見比べます
- 全件PASSで終わった。期待結果が実装から写されています。24.5 の行の期待結果を false に直して走らせ直させます
- 一覧が出ない。F12 のコンソールに getElementById が null と出ていれば、ul の id が result になっていません
- validateHours が未定義と出る。script の順番が逆です。validate.js を tests.js より先に置きます

演習2の到達点（続き）

時間が余ったら

- 件数を画面の上に「12件中11件PASS」の形で出す行を足します。tests.html だけで完結し、後続には影響しません
- validate.js の上限を仕様どおり24に直し、FAILが消えることを確かめます。確かめたら240に戻しておきます

FAILの1行

太字のFAIL 1件
テストが仕事をした証拠です

全件PASSは、テストが何も見ていない可能性を含みます。落ちるはずの入力で落ちるかどうかを、必ず1件は確かめます

入力を画面に出す2つの書き方

ここまでは仕様どおり動くかを見ました。次は、想定外の入力を渡されても壊れないかを見ます

innerHTML に入れる

el.innerHTML = '結果: ' + input; 入力に <img src=x onerror=alert(1)> が入っていると、タグとして解釈されて実行されます。作業記録アプリの内容欄をこの書き方で表示していれば、同じことが起きます。

textContent に入れる

el.textContent = '結果: ' + input; タグは文字として表示されるだけです。配布アプリの render() は td.textContent を使っているので、すでにこの形になっています。

解説

SQL文の組み立て

危ういのは、文字列を足して命令を作る書き方そのものです

今日の題材にDBはありません。ここは概念だけ扱います。客先のコードで最初に見る場所です。上が文字列連結、下が値を分けて渡す書き方で、行数は変わりません。

```
// 危うい書き方（入力が命令の一部になる）
const sql = "SELECT * FROM logs WHERE user = '" + name + "'";
// name に ' OR '1'='1 が入ると WHERE の条件が消えます

// 値を分けて渡す書き方
const sql = "SELECT * FROM logs WHERE user = ?";
db.query(sql, [name]);
```

> 攻撃文字列を自分で作って試すことはしません。仕組みを知りたい場合は OWASP Top 10 のページを見てください。

ワーク

演習3 レビュー依頼

客先のコードレビューで最初に見る2か所です。全部読まなくても機械的に確認できます

どんな場面か

客先から他人が書いたファイルを渡されて、目を通しておいてほしいと言われる場面を想定します。全部読む時間が無いときほど、観点を2つに絞って機械的に確認する頼み方が効きます。今日はその頼み方を1回やってみます。

やること

1. tpro-work フォルダの review_target.js を開きます。このファイルには外部入力を扱う箇所が2件あります
2. Copilot Chat に貼ります。「このファイルを2つの観点だけで見てください。観点1、外部入力の文字列を innerHTML に渡している箇所。観点2、文字列連結でSQL文を組み立てている箇所。該当箇所を行番号付きで挙げ、直し方を1行で示してください」

ワーク

演習3 レビュー依頼（続き）

やること

1. 返ってきた指摘の該当行を必ず開いて目で見ます。実際に外部入力が入る経路になっているものと、そうでないものを分けます
2. 指摘の一覧と自分の判断を security_review.md に書いて tpro-work フォルダに保存します

できあがるもの

デスクトップの tpro-work フォルダに security_review.md。指摘された行番号と、外部入力の経路かどうかの自分の判断が並んだ状態です。

演習3の到達点

2件が行番号付きで挙がり、経路かどうかの判断が書けていること

達成の目安

- innerHTML に渡している箇所と、SQL文を文字列連結している箇所の2件が、行番号付きで挙がっている
- security_review.md に、2件それぞれ外部入力の経路かどうかの判断が1行ずつある

つまずいたら

- 「該当なし」が返る。対象ファイルが review_target.js になっているか確認し、観点1と観点2を別々に投げ直させます
- 指摘が10件返って読めない。行番号順に上から3件だけ見る、と絞らせます
- 提案をそのまま適用してファイルが壊れた。Ctrl+Z で戻し、適用前に差分を見る手順をその場で見せます

時間が余ったら

- AIが出した書き換えを1件だけ実際に適用し、ブラウザで開いて表示が壊れていないか確かめます
- 同じ依頼文から「行番号付きで」を外して投げ、指摘の使いにくさがどう変わるか比べます。ファイルには反映しません

午前の到達点

cases.md、走るテスト、レビュー結果。この3つが tpro-work に残っていれば午前は達成です

残るもの
cases.md、tests.html、tests.js、security_review.md。すべてデスクトップの tpro-work フォルダの中です。
午後の実技課題で開き直すので消さないでください

持ち帰る考え方
期待結果の出どころは仕様であって実装ではない。全件PASSを見たら、落ちるはずの入力で落ちるかを1件確かめる

午後の予定
昼休憩のあとはE2Eのデモから始まります。手は動かしません。VSCodeとブラウザ、cases.md と tests.html は開いたままにしてください

午前からの接続

午前に作った tests.html を、そのまま午後の材料に使います

ここまでにやったこと

午前は受入基準を3行書き、cases.md にケース表を起こして、tests.html をブラウザで開きました。PASS が並ぶ中に FAIL が1件出て、実装と仕様の食い違いを見つけたところで止まっています。セキュリティ観点も2つ、自分のコードにかけました。

これからやること

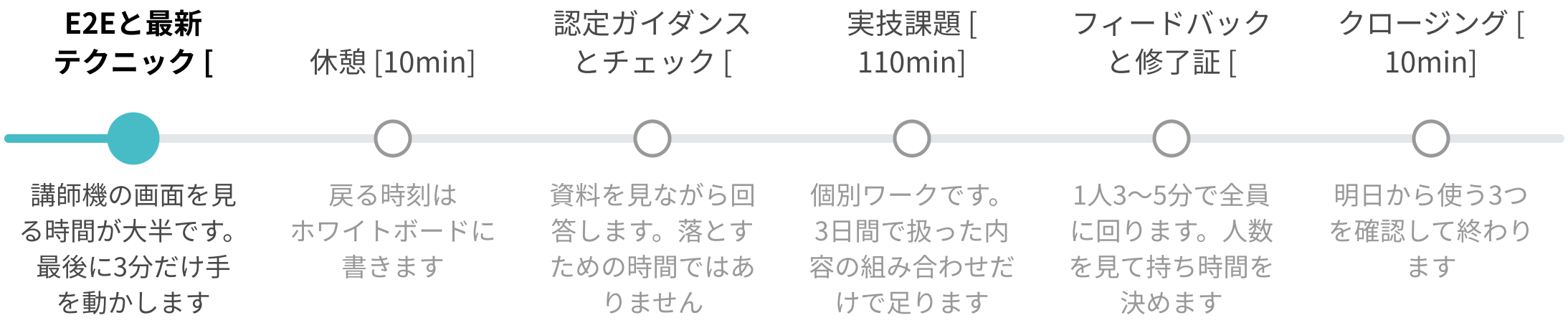
ここからは、同じアプリを人が操作する順番どおりに動かして確かめる方法を見ます。ツールを貸与PCに入れられないので講師機のデモになりますが、落ちた出力の読み方と貼り方だけは、午前の FAIL を使って自分の手で1回試します。

そのあとどこで効くか

実技課題では、必須要件のひとつとしてテスト結果の表示を自分で作ります。今日見る失敗の読み方は、そこで画面が真っ白になったときの調べ方になります。派遣先で落ちたテストを渡されたときも同じ手順です。

午後の流れ

手を動かすのは、この章の短い演習1つと実技課題の110分です



昼休憩から戻ったところです。この先で手を止めてよい場面と、手を動かす場面を先に切り分けておきます。

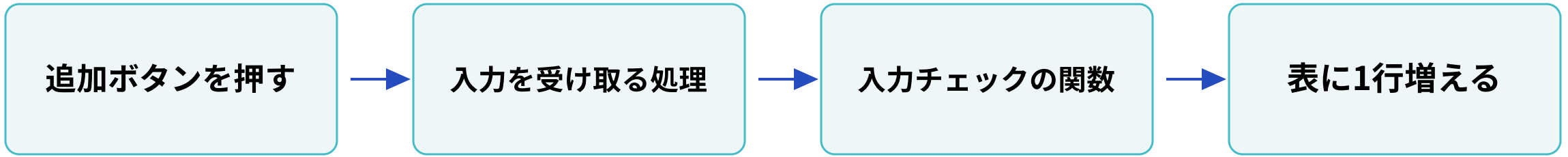
01

E2Eテストと最新テクニック

画面をつないで確かめる方法と、今日の環境で使える技術の線引きが分かる状態にします

落ちたときに疑う範囲

単体テストとE2Eは上下ではなく、疑う範囲の広さが違います



人が最初に触る入口です。E2Eはここから始めます

フォームの日付・作業種別・内容・工数の4項目を読み取って関数へ渡します

午前に tests.html で確かめたのは、この1点だけです

画面に出るところまで見て、はじめて業務が回ったと言えます

> 午前の単体テストが囲むのは3番目だけ。E2Eは1番から4番までをまとめて囲みます。落ちたときに調べる範囲が、そのまま囲みの幅です。

補足

テストの層と配分

E2Eは数を絞り、下の層で拾えることは下で拾います

層	確かめること	件数の考え方
単体テスト	関数に値を入れて戻り値を見る	一番多く置きます。午前に書いたのがここです
結合テスト	画面と処理のつなぎ目	今日の環境ではブラウザで開いて手で確認します
E2Eテスト	操作の流れ全体	止まると業務が回らない道筋の数だけに絞ります
探索的テスト	人が触って気づくこと	自動化しません。件数では数えません

E2Eは増やすほど壊れます。画面の文言を1つ変えただけで落ちるためです。修正が積もって誰も直さなくなった状態が、現場でよく見る落ちっぱなしのテストです。

補足

デモで使うツール

Playwright は画面操作を記録して、テストコードに書き出します

記録	ブラウザで操作すると、その手順がテストコードとして書き出されます
実行	書き出したコードを、何度でも同じ順番で流せます
失敗の証拠	落ちた時点のスクリーンショットと操作の記録がファイルに残ります
導入の条件	Node.js のインストールが要ります。貸与PCでは入れられないため講師機で見せます
持ち帰り先	https://playwright.dev/docs/codegen

実演

デモの筋書き

手を止めて画面を見てください。写す必要はありません

1. **操作を記録する**
   作業記録アプリで日付と作業種別と内容と工数を入れ、追加ボタンを押すまでを記録します。30秒ほどの操作です
2. **読める形に直す**
   書き出されたコードをCopilotに渡し、テスト名と待つ条件を人が読める言葉に直します
3. **わざと壊す**
   追加ボタンのラベルを1文字変えて、テストが落ちるところを見せます
4. **落ちた出力を読む**
   失敗レポートとスクリーンショットを開き、どの行で止まったかを一緒に読みます

> 見どころは3番と4番です。テストが落ちる瞬間を見ないと、E2Eが壊れやすいという前のページの話が実感になりません。

失敗出力の読む場所

貼るのは1文の依頼と、出力から抜いた4か所だけです

落ちた出力は長いですが、読む場所は決まっています。上の3行が依頼文、下が午前の tests.html から抜いた4か所です。

```
次はブラウザ内で動かした単体テストの失敗出力です。
原因の見当を3つ、可能性の高い順に挙げてください。
直すのは実装かテストかも書いてください。コードは書き換えないでください。

[FAIL] 工数25は不合格になる
expected: 不合格 received: 合格
止まった場所: tests.html の check 呼び出し 7行目
添付: なし。ブラウザ内実行のためスクリーンショットは残りません
```

貼る範囲の決め方

出力を全文貼ると、端末名やフォルダ構成まで一緒に送ります

出力を全文貼る

ターミナルやレポートの内容をまるごとコピーして送ります。端末名、利用者名、社内のフォルダ構成、案件名が一緒に流れます。派遣先の環境では、ここが機密の線に触れます。

4か所に絞って貼る

テスト名、期待と実際、止まった場所、添付の有無だけを貼ります。パスは末尾のファイル名だけに書き換えます。返ってくる見当の精度は落ちません。

ワーク

失敗の見当出し

午前の FAIL を1件、自分の言葉で説明できる状態にします

どんな場面か

午前に出た FAIL が1件、まだ直っていません。現場では、落ちた原因を自分の言葉にできないまま相談すると、往復が2回3回に増えます。先に見当を3つ持ってから相談する形を、ここで1回やります。E2Eツールが無くても、この手順は今日から使えます。

やること

1. デスクトップの tpro-work フォルダにある tests.html をブラウザで開き、FAIL と書かれた行をコピーします。見つからないときは Ctrl+F で FAIL を検索します
2. VSCode に戻り、Ctrl+Alt+I でチャットパネルを開きます
3. 前のページの依頼文3行をそのまま打ち、続けてコピーした FAIL の行を貼って送ります
4. 返ってきた3つの見当から1つ選び、tpro-work フォルダに fail_note.md を作って、選んだ見当と、直すのは実装かテストかを1行ずつ書きます

できあがるもの

デスクトップの tpro-work フォルダの中に fail_note.md。2行だけの短いファイルです。

達成の目安

2行書けていて、その2行を口で説明できれば到達です

達成の目安

- tpro-work の fail_note.md に、原因の見当が1行、直す対象が実装かテストかが1行、合わせて2行あります
- 直す対象を選んだ理由が、午前に書いた受入基準の文と結び付いています。上限の値が仕様と実装で違う、という形で言えていれば到達です
- Copilot の返答をそのまま貼った状態になっていません。自分が選んだ1つだけが残っています

つまずいたら

- FAIL の行が見つからない人は、tests.html を開いたブラウザで Ctrl+F を押し、FAIL で検索します
- 見当が1つしか返らない場合は、見当を3つ挙げてください、ともう一度だけ送ります。依頼文を作り直す必要はありません
- 見当が全部テストの書き方の話になっている人には、実装を疑う可能性を1つ入れてください、と声をかけます
- 新しいファイルが作れない人は、エクスプローラーの tpro-work の行を右クリックして新しいファイルを選びます

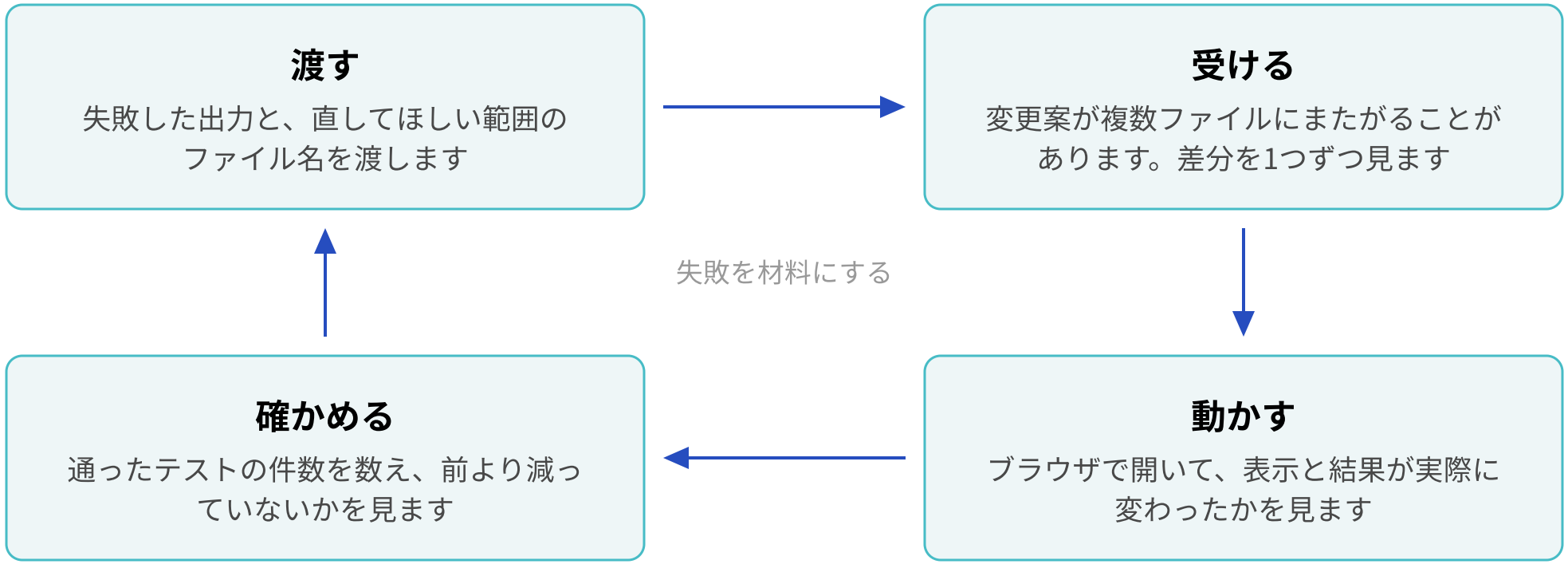
達成の目安（続き）

時間が余ったら

- 同じ FAIL を、期待と実際の行だけ抜いて貼り直し、返ってくる見当がどう変わるか比べます。fail_note.md には書き足しません
- fail_note.md の末尾に、この失敗を客先に報告するときの1文を書いてみます。ファイル名とパスを出さずに書けるかを試します

エージェントモードの回し方

状況を説明するより、落ちた出力を渡すほうが速く進みます



解説

GitHubとCopilotの違い

GitHub Copilot は、GitHubにコードを置かなくても使えます

GitHub

コードを預けて履歴と差分を残す場所です。3日間で一度も開いていません

GitHub Copilot

VSCodeの中で動くAIです。今日の依頼は全部ここに出しています

VSCode

Copilotが動く場所です。3日間の作業はこの中で完結しています

発展

技術の仕分け

今日の環境で試せる3つと、紹介にとどめる2つを分けます

技術	できること	今日の扱い
エージェントモード	複数ファイルの編集と確認を繰り返す	このあとの実技課題で使えます
カスタム指示ファイル	前提を常に効かせて指示を短くする	Day2で作った続きが使えます
コードレビュー支援	変更した差分に指摘をもらう	このあとの実技課題で使えます
E2E自動テスト（Playwright）	画面操作を記録して繰り返し実行する	講師デモのみ。Node.js が要ります
Model Context Protocol	AIから社内の検索やチケット管理を呼び出す	紹介のみ。接続先ごとに設定ファイルが要ります

> 最初に手を付けるならカスタム指示ファイルです。設定は一度で済み、効果が毎回出ます。

クイズ

理解度チェック２

Q. ブラウザ内のJavaScriptで単体テストを書く場合、Jestやpytestのようなテストランナーを使う場合と比べて、追加で自分が用意することになるものはどれですか。

A　テストに使う入力値

B　期待する結果の定義

C　合否を判定してまとめて表示する仕組み

D　テスト対象となる関数そのもの

正解は次のページで確認します。まず自分で考えてみてください。

解説

解説

正解はC。正解は、合否を判定してまとめて表示する仕組みです

C **合否を判定してまとめて表示する仕組み**

テストランナーが受け持つのは、複数のテストをまとめて実行し、結果を並べ、落ちた場所を指すところです。午前はそこを10行ほどの検証関数として自分で書きました。あの10行が、ランナーの一番外側にあたります。

A テストに使う入力値
入力値はどちらの環境でも自分で決めます。ランナーが用意してくれるものではありません

B 期待する結果の定義
期待値は仕様から人が決めます。AIが候補を出しても、正しいかを決めるのは人です

D テスト対象となる関数そのもの
どちらの環境でも自分で書くものなので、差になりません

環境が変わっても、入力値と期待値を決める仕事は残ります。楽になるのは実行と集計の部分だけです。

認定への接続

残りの4時間弱は、全部が認定の時間です

ここまでにやったこと

E2Eが囲む範囲の広さ、層ごとの配分、失敗出力の読み方と貼り方をやりました。手を動かしたのは fail_note.md の2行だけで、あとは講師機の画面でした。技術の仕分けでは、実技課題で使える3つと持ち帰りの2つを分けました。

これからやること

ここからは認定です。まず考え方と構成を6分で確認し、そのあとリテラシーチェックに入ります。資料は全部見て構いません。落とすための時間ではありません。

そのあとどこで効くか

チェックで空欄になった箇所は、実技課題のあとのフィードバック60分で講師と一緒に埋めます。今日のうちに全員が到達した状態で終わる設計です。

02

認定ガイダンスとリテラシーチェック

認定で何を見るのかと、チェックの進め方が分かる状態にします

認定は、合否の振り分けではありません

> **落とすための時間ではありません**
> **全員が到達した状態で今日を終えます**
>
> 届いていない項目は、このあとのフィードバック60分で講師と一緒に埋めます。実技課題の110分は、自分で手を動かす時間です

埋める作業は本人がやります。講師が代わりに書くことはしません。

認定の構成

3つの時間で、それぞれ見るところが違います

チェック 40min	機密の伏せ方、AIの出力を確かめる手順、テストの考え方。知識を見ます
実技課題 110min	ブラウザで動くものが作れているか、テスト観点を自分で出せているか。作れるかを見ます
フィードバック 60min	詰まった箇所をその場で解消できたか。1人3〜5分で全員に回ります

3つ足すと210分です。午後の残りは、ほぼ全部が認定の時間になります。

ワーク

リテラシーチェック

資料は全部見て構いません。覚えているかは見ません

どんな場面か

現場では調べながら仕事をします。覚えているかを測っても意味がないので、資料参照可の設計にしています。3日間でどこに何が書いてあったかをたどれるかを見ます。選択式20問を20分で回答します。

やること

1. 配布されたリテラシーチェックを開きます。選択式20問です
2. Day1からDay3のスライド、ハンズオンガイド、自分のメモ、tpro-work フォルダの中身は全部見て構いません。Copilot に聞くのはこの20分だけ控えてください
3. 分からない問題は空欄のまま次へ進み、戻る印を付けておきます。最後に3問だけ全体で確認します
4. 早く終わった人は、実技課題の要件のページを先に読んでおきます

できあがるもの

成果物はありません。20問すべてに選択が入っていれば達成です。回答方法と回収方法は当日の案内に従ってください。

チェックの達成目安

20問すべてに選択が入っていれば到達です。点数は出しません

達成の目安

- 20問すべてに選択が入っています。迷った問題も空欄にせず、いちばん近いものを1つ選んで印を付けてあります
- 答えの根拠にした資料の場所を、少なくとも3問について言えます。ページ番号でもファイル名でも構いません
- 機密の伏せ方の設問で、丸める対象が3つではなく4つあったことに自分で気づけています。数は次のページで確認します

つまずいたら

- どこに書いてあるか探せない人には、ページ番号を答えて構いません。探すこと自体は目的ではありません
- 残り5分で空欄が5問以上ある人には、選択肢を2つに絞ってどちらかを選ぶよう声をかけます。空欄のままより、選んで外したほうが次の確認で拾えます
- 時間内に終わらなかった場合はそのままで構いません。未回答の数で扱いは変わりません
- 設問の意味が読み取れないという申し出があれば、設問を言い換えて構いません。言い換えた内容は控えておき、運営に共有してください

チェックの達成目安（続き）

時間が余ったら

- 迷った問題を1問選び、なぜ迷ったかを1行だけメモしておきます。フィードバックの時間で講師に見せる材料になります
- 実技課題の必須要件を先に読み、最初に Copilot へ投げる1文を tpro-work のメモに下書きしておきます

解説

よく外す3問

この3問だけ確認します。個人の点数には触れません

機密の伏せ方
丸める対象は、客先名と製品名とプロジェクト名と取引先名の4つです。3つと答える人が毎回います。加えて、実データと実ファイルとスクリーンショットは使いません

AIの出力の確かめ方
読んで筋が通っていることは、確かめたことになりません。ブラウザで開いて動かし、結果を目で見るまでが1周です

期待値を決める人
仕様から期待値を決めるのは人です。AIは候補を出すところまでで、正しいかどうかの判断は代わりにやってくれません

空欄だった人は、この3問だけ埋めれば到達です。残りは実技課題とフィードバックで見ます。

ここまでと実技課題

ここから110分は、3日間でやったことをつなげる時間です

ここまでにやったこと

午前は受入基準からケース表を起こし、単体テストをブラウザで走らせました。午後はE2Eの動きを見て、認定の考え方とリテラシーチェックまで終わっています。

これからやること

ここから110分で、1枚のHTMLを自分で組みます。新しい技術は出しません。最初の20分で要件と時間配分をお渡しして、残り90分は各自の作業です。

そのあとどこで効くか

作ったものは、このあと60分のフィードバックで1人ずつ画面を見せてもらいます。動いていない状態のまま持ってきて構いません。詰まった箇所はその場で直します。

03

実技課題

3日間の内容を1枚のHTMLにつなげて、動くところまで持っていきます

補足

今日の流れと現在地

実技課題はここです。この先に全体休憩は入りません

作業の90分は各自のタイミングで席を外して構いません。全体で止める休憩はここから先ありません。

課題の場面

客先から受け取ったCSVを、開いて直して集計して報告します

> ## 受け取ったCSVの中身が
そのままでは使えない
>
> 派遣先に入って最初に頼まれるのは、この形の作業です。表記がそろっていない、単位が文字で混ざっている、必須の欄が空いている。今日はそれを見つけて集計するところまでを1枚のHTMLで組みます

ワーク

完成画面の並び

上から順に5つ並べば完成です。見た目は判定に入りません

1. **CSVを落とす枠**
   画面の一番上に破線の四角を置き、ここにCSVを落とす、と文字を入れます。必須1の入口です
2. **読み込んだ行の表**
   30行が5列で並びます。列の順番はCSVと同じで構いません。必須1はここまでで到達です
3. **警告の一覧**
   何行目のどの列がどう変か、を1行1件で並べます。必須2の出口です
4. **担当ごとの合計**
   担当3名の名前と合計時間の小さい表。必須3です
5. **テスト結果のリスト**
   先頭にPASSかFAILが付いた6行。必須4です。ページの一番下で構いません

この5つが縦に並んでいれば形は合っています。色やレイアウトは触らなくて構いません。

ワーク

必須要件と到達の判定

必須4つ。それぞれ自分で確かめられる判定を付けました

区分	やること	到達の判定
必須1	CSVをドロップして表に出す	30行すべてが表に並ぶ
必須2	汚れを検出して警告一覧に出す	仕込んだ4種のうち3種以上が出る
必須3	担当ごとの合計時間を出す	担当3名分が出て、手で足した数と一致する
必須4	検証関数で6件テストする	PASS/FAIL付きの6行が並び、正常系3件がPASS
任意	棒グラフ、絞り込み、CSV書き出し	判定には入りません
やらないこと	サーバー起動、追加インストール、実データ持ち込み	見つけたら止めます

ワーク

配布サンプルCSV

汚れは4種類だけ仕込んであります。探すのはこの4つです

デスクトップの tpro-work フォルダに入っている sample_work_log.csv の先頭5行です。全30行、担当は3名。文字コードはUTF-8です。

```
日付,担当,工程,作業内容,作業時間
2026/09/01,佐藤,設計,画面遷移の見直し,3.5
2026/9/2,鈴木,実装,一覧表示の修正,2
2026-09-03,佐藤,試験,結合テストの実施,3.5 h
2026/09/04,,調査,ログの調査,1.5
2026/09/05,田中,実装,入力チェックの追加,２

仕込んだ汚れ 1．日付が3通りの書き方で混在 2．作業時間が「3.5 h」の単位付き文字列
3．担当が空欄の行が1件 4．作業時間が全角数字の行が1件
```

補足

実技課題で使う道具

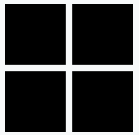
道具は3つだけです。追加インストールはありません

VSCode

kadai.html を書いて Ctrl+S で保存します

GitHub Copilot

Ctrl+Alt+I で開き、前提3行と依頼を貼ります

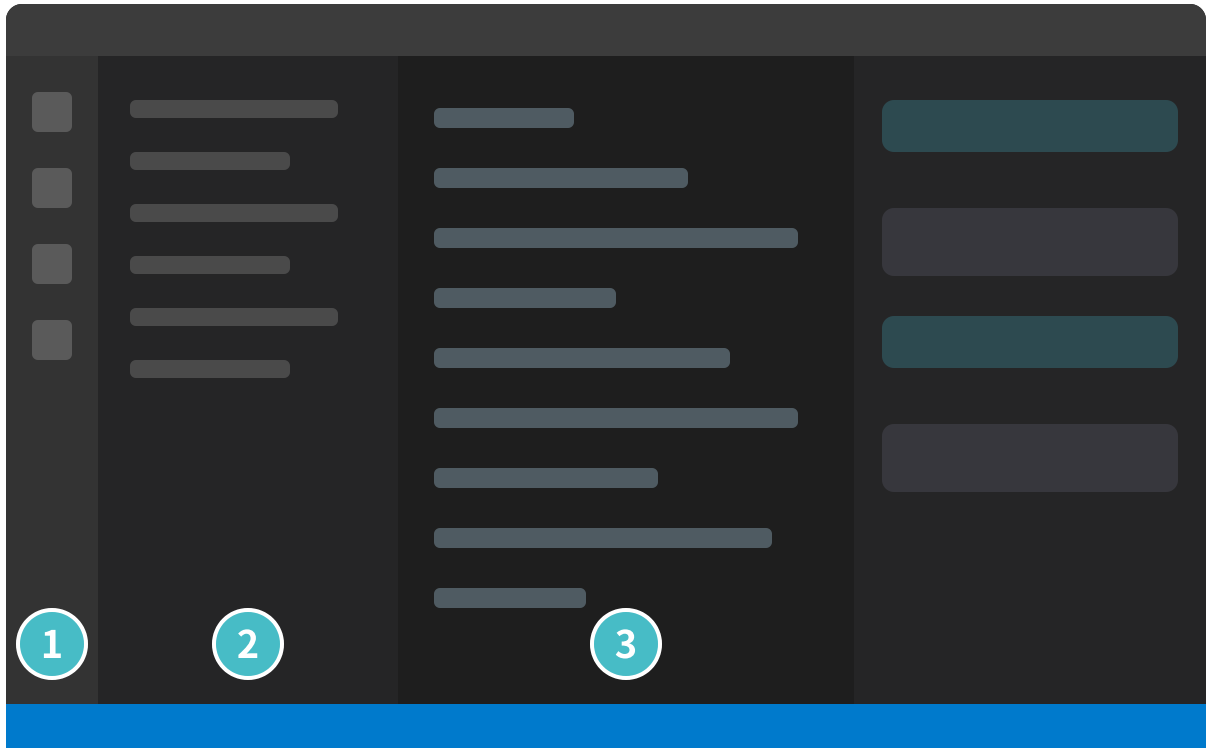


Edge

kadai.html を開いて、表と警告とテスト結果を見ます

作業中に見る4か所

置き場所はデスクトップの tpro-work フォルダに統一します

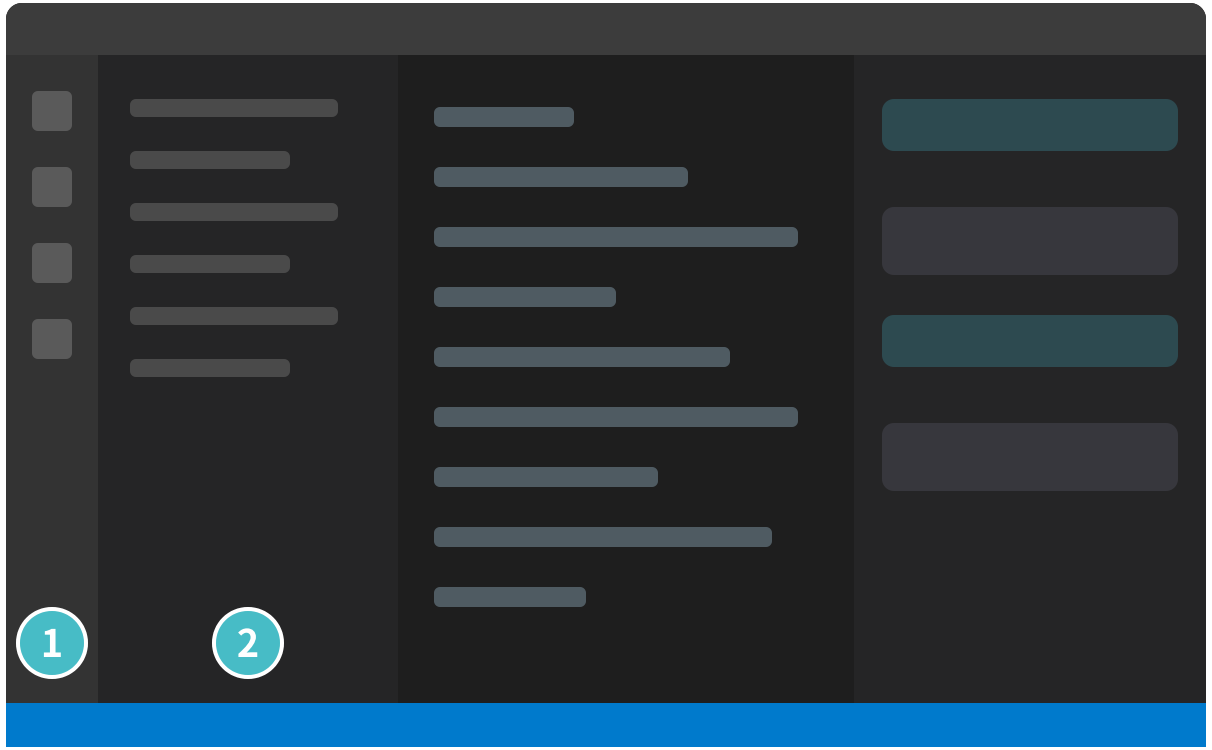


1 **アクティビティバー**
Copilot Chat のアイコン。閉じてしまったらここから開き直します

2 **エクスプローラー**
デスクトップの tpro-work フォルダを開いた状態にします。kadai.html と sample_work_log.csv と papaparse.min.js が同じ並びに見えていれば置き場所は正しいです

3 **エディタ**
kadai.html を開きます。Ctrl+S で保存してからブラウザを再読み込みします。保存前に再読み込みしても何も変わりません

作業中に見る4か所（続き）



1 **チャットパネル**
依頼文を貼る場所です。履歴はフィードバックの時間に見せるので消さないでください

2 **ステータスバー**
開いているファイルの文字コードが出ます。CSVの文字が化けたときはここがUTF-8かを見ます

ワーク

作業90分の区切り

110分のうちガイダンス20分、作業90分。90分を4つに割ります

観点を出す

[15min] 要件を読み、確かめることを箇条書きにします。ここではエディタを開きません

実装する

[45min] 依頼文を貼り、1つ足すたびにブラウザで開いて確かめます

テストを回す

[15min] 検証関数で6件流して、PASS/FAILを画面に出します

言語化する

[15min] 提出するテキストを書きます。実装が途中でもこの15分は使います

実装が終わっていなくても、残り30分でテストと言語化に移ります。講師が3回声をかけます。

ワーク

最初に貼る依頼文

前提3行を先に書いてから、やってほしいことを1つだけ書きます

実装の1手目です。この形をそのまま貼ってから、依頼の行だけを自分の言葉に差し替えていきます。

```
前提1: 1枚のHTMLだけで動かします。Node.js とビルドは使いません。
前提2: 読み込むCSVは同じフォルダの sample_work_log.csv です。列は 日付,担当,工程,作業内容,作業時間 の5列、全30行です。
前提3: CSVはドラッグ＆ドロップで受け取ります。fetch は使いません。同じフォルダの papaparse.min.js を使い、ライブラリは増やさないでください。

依頼: kadai.html に、CSVをドロップしたら全30行を表に出す処理を追加してください。
出力: 変更後の kadai.html の全文を1つのコードブロックで出してください。変更した箇所を最後に1行で説明してください。
```

> 2手目以降も同じ形です。前提3行は毎回そのまま、依頼の1行だけを差し替えます。

補足

提出物と伏せ方

出すのはテキスト1つだけ。コードと実データは提出できません

出すもの
furikaeri_<自分の名字ローマ字>.md というテキストファイル1つ。デスクトップの tpro-work フォルダの中に作ります

出さないもの
kadai.html と読み込んだCSVは提出しません。コードと実データは手元に残します

書く中身
観点の箇条書き、効いた依頼文、外した依頼文とその対処の3つ。A4で1枚に収まる分量で足ります

提出のしかた
提出フォームの冒頭に、客先名・製品名・案件名・取引先名を業界と工程に丸める注意書きが常に出ています。貼る前に一度読んでください。貼った内容は手元のファイルにそのまま残ります

ワーク

実技課題の作業

90分で kadai.html を組み、最後の15分で振り返りを書きます

どんな場面か

客先から作業記録のCSVを受け取り、中身の汚れを見つけて担当ごとに集計して報告する。常駐先に入って最初に回ってくるのはこの形の作業です。3日間で別々にやってきた読み込み・検出・集計・テストを、ここで1枚のHTMLにつなげます。

やること

1. デスクトップの tpro-work フォルダを開き、template.html を同じフォルダに複製して kadai.html という名前に変えます。
2. 最初の15分はエディタを開かず、確かめることを箇条書きにします。tpro-work の中に kanten.txt を作って書いても構いません。
3. kadai.html を VSCode で開き、チャットパネルに前ページの依頼文をそのまま貼ります。前提3行のあとに依頼を1つだけ書きます。

ワーク

実技課題の作業（続き）

やること

1. 返ってきたコードを kadai.html に貼り、Ctrl+S で保存してからブラウザで開き、sample_work_log.csv をドロップします。
2. 表が出たら、警告の一覧、担当ごとの合計、検証関数6件をこの順に足します。1つ足すたびに保存してブラウザを再読み込みします。
3. 残り15分で furikaeri_<自分の名字ローマ字>.md を作り、観点の箇条書き、効いた依頼文、外した依頼文とその対処の3つを書きます。

できあがるもの

デスクトップの tpro-work フォルダの中に kadai.html と furikaeri_<自分の名字ローマ字>.md の2つ。kanten.txt は残しても消しても構いません。

ワーク

実技課題の到達

数で確かめます。動いた気がする、では判定できません

達成の目安

- sample_work_log.csv の30行がすべて表に並んでいます。表の上に行数を出しておくと、数えずに確認できます。
- 仕込んだ汚れ4種のうち3種以上が警告一覧に出ています。警告が30件出ている場合は判定が緩すぎます。
- 担当3名の合計時間が画面に出ていて、電卓で足した数と一致します。

つまずいたら

- 汚れの判定を作り込みすぎて時間が無くなる: 4種のうち3種でよいので、日付の表記ゆれと担当の空欄の2つから始めます。
- 観点出しを飛ばして実装から始めている: 5分でよいので箇条書きに戻します。観点が無いと、必須4で何を6件テストするかが決まりません。
- 実装が45分で終わらない: 必須3を捨てて必須4に進みます。テストと言語化が残っているほうが到達に近いです。
- 画面が真っ白、CSVを落としても反応しない、文字が化ける、テスト結果が出ない: この4症状は詰まったときの見どころのページに、見る場所と直し方をまとめてあります。そのページを開かせてください。

ワーク

実技課題の到達（続き）

達成の目安

- テスト結果が6行出て、各行の先頭に PASS か FAIL が付いています。正常系3件は PASS です。
- furikaeri_<名字ローマ字>.md に、外した依頼文が1つ以上書かれています。

ワーク

実技課題の到達（続き）

時間が余ったら

- 任意要件の棒グラフを1つ足します。同じフォルダの chart.umd.js を使い、担当ごとの合計を棒にします。kadai.html の中だけで完結します。
- 警告の一覧をCSVで書き出すボタンを足します。書き出したファイルは tpro-work に置きます。
- 全角数字の作業時間を半角に直して集計に入れる処理を足します。直した行数を画面に出すと、直したことが自分で確かめられます。

ワーク

検証関数の骨格

午前に書いた10行をそのまま使います。追加インストールは要りません

必須4の中身です。kadai.html の script の中に貼って、下の呼び出しを6行書けば終わります。

```
<ul id="result"></ul>

function check(name, actual, expected) {
const ok = actual === expected;
const li = document.createElement('li');
li.textContent = (ok ? 'PASS' : 'FAIL') + ' / ' + name;
document.getElementById('result').appendChild(li);
}

check('全角数字は警告になる', isDirty('２'), true);
check('半角数字は警告にならない', isDirty('2'), false);
```

外れる依頼と通る依頼

外れる原因はほぼ前提の書き忘れです

外れる依頼

「CSVを読み込んで表示して」。返ってくるのは Node.js で動かす前提のコードや、fetch でファイルを読むコード。どちらも今日の環境では動きません。貼って開いて真っ白、で15分溶けます

通る依頼

前提3行を先に置いてから「CSVをドロップしたら全30行を表に出す処理を追加してください」。環境と読むデータと使い方が閉じているので、貼れば動くコードが返ってきます

補足

詰まったときの見どころ

症状ごとに見る場所が決まっています。当てずっぽうで直さないでください

症状	まず見る場所	直し方
画面が真っ白	F12 のコンソールの赤い行	script の src がファイル名と1文字違い。綴りを合わせます
CSVを落としても何も起きない	dragover のイベント登録	dragover 側にも preventDefault を入れます
日本語が化ける	ステータスバーの文字コード	配布の sample_work_log.csv に戻します。UTF-8です
テスト結果が出ない	ul の id と getElementById の文字列	id は result で固定。1文字でも違うと出ません
合計が合わない	単位付きと全角数字の行	数値に直してから足します。文字列のままだと連結されます

ここから60分、全員の画面を1人ずつ見ます

ここまでにやったこと

90分で kadai.html を組み、furikaeri_ に3つ書きました。必須4つのどこまで届いたかは人によって違います。全部埋まっていない人がいるのは想定どおりです。

これからやること

60分かけて全員の画面を見ます。動いている所を30秒、詰まった所を1つ、直し方をその場で。最後に修了証をまとめてお渡しします。

そのあとどこで効くか

この時間で直した内容が、明日の現場で最初に効きます。直した内容を furikaeri_ に1行足しておくと、次に同じ場所で止まったときに戻ってこられます。

04

フィードバックと修了

1人ずつ画面を見ながら、詰まった箇所をその場で直します

株式会社ギブリー (Givery, Inc.)
〒150-0036 東京都渋谷区南平台町15-13 帝都渋谷ビル8F

補足

この時間に見るもの

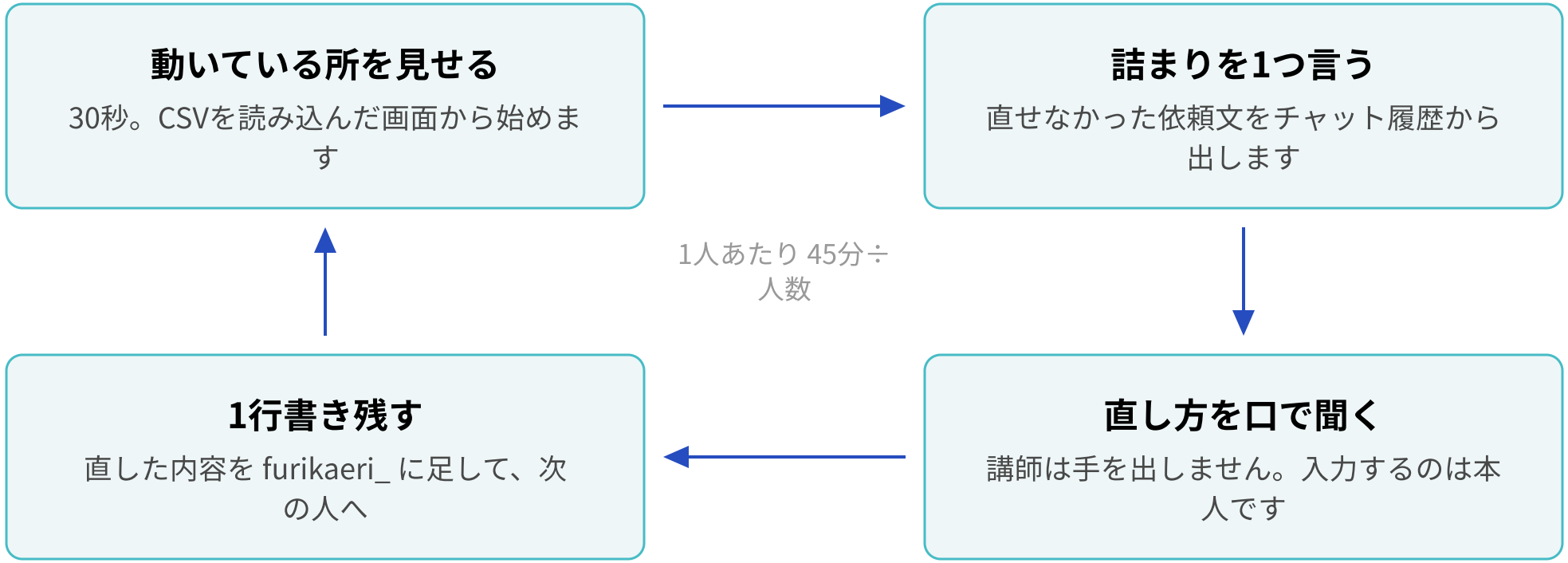
見るのは3つだけ。準備するものも3つだけです

見るもの	具体的に	用意しておくもの
動いている範囲	どこまで表に出て、どこから出ていないか	kadai.html をブラウザで開いた状態
詰まった箇所	直せなかった依頼文と、そのときの出力	チャットパネルの履歴。消さないでください
書いたテキスト	外した依頼文とその対処が書けているか	furikaeri_ の書きかけ。3行でも構いません

> 見た目、コードのきれいさ、必須が何個埋まったかは見ません。

1人あたりの進み方

4手で1人分が終わります。深い相談は最後に戻ってきます



持ち時間と回る順番

持ち時間は45分を人数で割った数です

持ち時間	45分 ÷ 受講者数。15名で3分、10名で4分半、5名で9分。開始前にホワイトボードに書いて全体に伝えます
回る順番	座席の端から機械的に進みます。挙手制にしません。声を出せない人が最後まで残ります
呼ばれる前	課題の続きか、任意要件に手を付けてください。手を止めて待つ必要はありません
余ったとき	1人あたり5分を超える人数のときは、任意要件の相談や現場での使い方の話に充てます

呼ばれるまでの過ごし方

手は止めなくて構いません。画面は閉じないでください

手は止めない
課題の続きで構いません。必須が埋まっている人は任意要件に進んでも構いません

閉じないもの
ブラウザ、チャットパネル、furikaeri_ の3つは開いたままにします

未完成でよい
動いていない状態のほうが話せることが増えます。作り直して待つ必要はありません

ワーク

画面確認と手直し

動かないものを見せられるかどうかで、直る速さが変わります

どんな場面か

現場で先輩に聞くときと同じ場面です。動いているところだけ見せても解決しません。動かないものと、そのとき自分が何をしたかを出せる人ほど、答えが速く返ってきます。この45分は、その出し方を練習する時間でもあります。

やること

1. kadai.html をブラウザで開き、sample_work_log.csv を読み込んだ状態にしておきます。
2. VSCode のチャットパネルを開き、直せなかった依頼文が残っている位置までスクロールしておきます。
3. furikaeri_<名字ローマ字>.md を開いておきます。白紙の人は3行だけ先に書いておきます。
4. 順番が来たら、動いている部分を30秒で見せ、次に直せなかった箇所を1つだけ挙げます。
5. 直し方を口で聞き、自分のキーボードで直してブラウザで結果を確かめます。
6. 直った内容を furikaeri_ に1行追記します。

できあがるもの

新しい成果物はありません。デスクトップの tpro-work フォルダの furikaeri_<名字ローマ字>.md に、この時間で直した内容が1行増えていれば達成です。

ワーク

この時間の到達

直った箇所が1つ、書き残しが1行。これで足ります

達成の目安

- 講師に見せる前に、ブラウザとチャット履歴と furikaeri_ の3つが開いています。
- 詰まった箇所を1つに絞って、言葉で説明できています。
- 直し方を自分のキーボードで入力し、ブラウザで結果を確かめています。
- furikaeri_ に、この時間で直した内容が1行増えています。

つまずいたら

- 何も動いていないので見せられない: そのまま見せます。真っ白の画面のほうが原因が早く分かります。
- 質問が「全部分かりません」になる: 直前に押した操作と、そのとき出たメッセージの2つだけ言います。
- 詰まりが多すぎて1つに絞れない: 表が出ていない、警告が出ていない、テストが出ていない、の順で上から1つ選びます。
- 順番待ちで手が止まる: 任意要件の棒グラフか、汚れの4種目に手を付けます。

ワーク

この時間の到達（続き）

時間が余ったら

- sample_work_log.csv を tpro-work の中で複製し、複製した側の日付を1行だけ壊して読み込み、警告が出るか確かめます。配布ファイルは書き換えません。
- furikaeri_ に「明日の現場で最初に試すこと」を1行足します。
- 直せなかった箇所が残っている人は、依頼文に前提の行を1つ足して投げ直します。

修了証

全員にお渡しします。点数での合否判定はありません

修了証は
全員にお渡しします

3日間の内容に自分の手で取り組んだことに対してお渡しするものです。必須要件が全部埋まっているかは
条件に入っていません。この時間の最後にまとめてお渡しします

修了から締めへ

残りは3日間を3手に圧縮して渡すだけです

ここまでにやったこと

全員の画面を見て、詰まった箇所をその場で直しました。修了証もお渡ししました。

これからやること

3日間を3手に圧縮して渡します。細かい手順は忘れても、この3手が残っていれば現場で回ります。

そのあとどこで効くか

手元に残るのは kadai.html と furikaeri_、それと3日間の配布資料です。次に詰まったときは、前提を3行書くところから戻ってきてください。

明日から使う3手

3日間の中身は、この3手に圧縮できます

前提を3行書く

使う言語、動かす場所、やってほしくないこと。この3行を先に置くだけで出力の外れ方が減ります。Day1のプロンプト設計です

→

動かして確かめる

読んで納得した時点では、まだ何も確かめていません。ブラウザで開くまでが1セットです。Day1のデバグ体験と今日のテストです

→

伏せてから貼る

客先名・製品名・案件名・取引先名は業界と工程に丸めます。貼る前に一度目で見る癖をつけてください。Day1のガバナンスです

3日間の到達点

3日間で書いたものは、全部ブラウザで動きました

追加インストールなし、サーバーなしの環境で全部動きました。同じ条件が現場にもあります。

クロージング

今日作ったものは手元に残ります。ここから先も同じやり方です

> **3日間で書いたものは**
> **全部ブラウザで動きました**
>
> kadai.html と furikaeri_、テスト観点、配布資料はすべて手元に残ります。次に詰まったときは、指示の最初に前提を3行足すところから戻ってきてください