Introduction to GitHub Copilot

Define, inspect, test, decide

Session M1-01 | 3 hours

Agenda

Segment	Time
Proposal-and-evidence model	7 min
How context shapes a suggestion	10 min
Access and data preflight	10 min
Inline suggestion controls	11 min
Prepared utility demonstration	12 min
Follow-up edits	6 min
Access decision and lab handoff	4 min

The review workflow

```
Define observable behavior
          ↓
Check access, data, and scope
          ↓
Inspect the proposed change
          ↓
Run focused checks
          ↓
Accept, revise, or reject
```

The developer owns the result.

Proposal is not proof

Proposal	Evidence
Looks plausible	A focused test passes
Uses familiar syntax	The diff stays in scope
Explains its reasoning	A reviewer can reproduce the result
Matches one example	Edge and failure behavior are checked

Code that reads well can still be incomplete or wrong.

A practical system model

```
Task + approved context
          ↓
Selected Copilot surface
          ↓
Proposed output
          ↓
Local checks + human review
```

Use this to guide review. It does not describe a fixed service architecture.

Context changes the proposal

Useful context includes:

- function names and type hints;
- docstrings and examples;
- selected code and nearby files;
- tests and repository instructions.

Use context only when it is relevant and you may use it for the task.

Preflight before use

1. Check the account and surface.
2. Check the repository and data classification.
3. Limit the task to named files.
4. Name any metered-work owner and stop condition.

If access is unclear, stop the lab.

Access policy

The lab requires GitHub Copilot.

If access is unavailable, stop. Manual implementation does not demonstrate how to inspect, accept, revise, or reject an inline suggestion.

Accept, revise, reject

Accept when the proposal meets the contract and fits the repository.

Revise when the direction is useful but a detail is wrong.

Reject when it changes scope, invents behavior, or is hard to check.

Undo is a normal review action.

Review one suggestion

Before accepting, ask:

- Which requirement does each line support?
- What happens on empty or unusual input?
- Did it add a dependency?
- Did it change a caller or public name?
- Which check will fail if the logic is wrong?

Demo contract

```
def normalize_username(value: str) -> str:
    """Trim outer whitespace, lowercase the value,
    and join words with hyphens."""
```

```
assert normalize_username("  River Team  ") == "river-team"
```

The test checks how repeated whitespace is handled.

Demo: inspect before accepting

Review the proposed implementation against three rules:

1. Outer whitespace disappears.
2. Case becomes lowercase.
3. Each run of internal whitespace becomes one hyphen.

Then run the focused test.

A failing check is useful

This version looks reasonable but mishandles repeated spaces:

```
return value.strip().lower().replace(" ", "-")
```

The failing test catches the whitespace assumption while the change is still small.

Predicted follow-up edits still need review

```
Rename username to handle
          ↓
Inspect each proposed caller update
          ↓
Search for the old name
          ↓
Run the focused test
```

A predicted edit can miss a reference.

Access check

Check that every learner can request and review an inline suggestion before the lab starts. Fix access problems before handing off the exercise.

Lab handoff

You will extend one Python utility module:

1. complete `normalize_username` and `is_palindrome`;
2. decide how apostrophes count in `word_frequency`;
3. add `profile_summary`;
4. rename `username` to `handle`;
5. run the full test suite and review the diff.

Deliverable: completed code, five passing tests, and a decision record.