Zeszyt ćwiczeń

Vibe coding w praktyce: GitHub Copilot, agenci kodujący i MCP

Kurs VITA: vita.edu.pl/courses/vibe-coding-github-copilot-i-mcp
24 ćwiczenia, po jednym do każdej lekcji. Na końcu znajdziesz przykładowe odpowiedzi. Najpierw spróbuj odpowiedzieć samodzielnie, dopiero potem zajrzyj do wzorów. Postacie, firmy i sytuacje w ćwiczeniach są fikcyjne. Narzędzia i ich nazwy zmieniają się, więc sprawdzaj aktualną dokumentację. Zeszyt jest materiałem edukacyjnym, nie poradą prawną ani konsultacją specjalisty, i nie zastępuje prawnika, inspektora ochrony danych ani specjalisty ds. bezpieczeństwa.
Stan na: listopad 2026.

Ćwiczenia

Moduł 1. Czym jest vibe coding i gdzie się kończy

Lekcja 1.1. Vibe coding: co to jest i dla kogo jest ten kurs

Napisz własnymi słowami definicję vibe codingu w jednym lub dwóch zdaniach, a potem opisz trzy sytuacje z własnej pracy lub zainteresowań, w których chcesz użyć AI do kodu. Dla każdej określ, czy wystarczy Ci uruchomienie i sprawdzenie wyniku, czy zapisujesz kod i wracasz do niego później, i czy dotyczy on cudzych danych lub pieniędzy.

Lekcja 1.2. Kiedy vibe coding działa: prototyp, skrypt, narzędzie wewnętrzne

Weź zadanie programistyczne, które chcesz oddać AI (może być wymyślone), i odpowiedz na pięć pytań: czy jest małe, czy wynik zobaczysz gołym okiem, czy pracujesz na kopii danych, czy korzysta z niego mniej niż kilka osób i czy omija cudze pieniądze oraz dane osobowe. Policz odpowiedzi "tak". Przy pięciu możesz pracować szybko, przy czterech lub mniej zapisz, co dodajesz do kontroli.

Lekcja 1.3. Dług techniczny: przypadek, w którym nikt nie rozumie własnej aplikacji

Wybierz program lub skrypt (własny albo wymyślony), który powstał z pomocą AI i jest używany regularnie. Odpowiedz na cztery pytania: kto potrafi go uruchomić bez Ciebie, czy w repozytorium jest opis działania, czy istnieje przynajmniej jeden test oraz co się stanie, jeśli dane wejściowe zmienią format. Do każdej odpowiedzi "nie wiem" lub "nie" dopisz jedno konkretne działanie na najbliższy tydzień.

Moduł 2. GitHub Copilot w edytorze: uzupełnianie, czat, agent, instrukcje repozytorium

Lekcja 2.1. GitHub Copilot: jak zacząć w VS Code i co oznaczają plany

Otwórz w VS Code dowolny mały plik z własnym kodem albo wklej kod do dowolnego czatu z AI. Poproś o wyjaśnienie,

co robi kod, z zastrzeżeniem, że nie wolno niczego zmieniać. Wypisz trzy konkretne twierdzenia z odpowiedzi i obok każdego zapisz, czy zostało sprawdzone w kodzie i jak. Na końcu wskaż jedno twierdzenie, którego nie umiesz zweryfikować, i napisz, kogo lub co zapytasz.

Lekcja 2.2. Uzupełnianie i czat: jak pytać o kod, żeby dostać użyteczną odpowiedź

Weź błąd z własnego programu albo wymyśl taki: plik CSV z kolumną "data" w formacie 31.12.2026 i funkcja, która wczytuje datę przez datetime.strptime z formatem "%Y-%m-%d". Napisz pytanie do czatu złożone z pięciu elementów: cel, kod, komunikat błędu, ograniczenia i format odpowiedzi. Potem oceń odpowiedź, którą dostaniesz: czy zmiana jest tylko w jednej funkcji, czy rozumiesz każdy jej wiersz i jak ją sprawdzisz.

Lekcja 2.3. Tryb agenta i instrukcje repozytorium: copilot-instructions.md i AGENTS.md

Napisz plik instrukcji dla własnego projektu (lub wymyślonego): sześć krótkich zdań. Uwzględnij język i jego wersję, jedną regułę o danych (np. waluta lub format), sposób sprawdzania zmian, zakaz dodawania bibliotek bez pytania, zakaz ruszania wskazanych plików oraz prośbę o plan przed zmianami. Zapisz też jedno zadanie dla agenta z celem, kryterium gotowości i granicą.

Moduł 3. Pierwszy projekt z AI krok po kroku

Lekcja 3.1. Specyfikacja na jednej stronie: co zapisać, zanim poprosisz o kod

Wybierz małe zadanie programistyczne (własne lub wymyślone, np. program łączący pliki z fakturami albo zliczający zgłoszenia według działu) i napisz specyfikację na jednej stronie w ośmiu punktach: cel, wejście, wyjście, reguły, błędy, poza zakresem, ograniczenia i kryteria akceptacji. W kryteriach podaj co najmniej jedną liczbę policzoną ręcznie z przykładowych danych.

Lekcja 3.2. Podział na małe zadania: kolejność, kryteria gotowości, commit po każdym kroku

Weź specyfikację z lekcji 3.1 (własną lub wymyśloną) i podziel ją na od czterech do sześciu zadań. Dla każdego zapisz jedno zdanie, kryterium gotowości, które da się sprawdzić bez czytania kodu, oraz opis commitu. Ustaw zadania w kolejności od danych do wyniku.

Lekcja 3.3. Iteracje: raport zamówień z pliku CSV krok po kroku

Zapisz własny plan trzech iteracji dla programu, który czyta plik CSV z polskim zapisem liczb. Dla każdej iteracji wpisz: co dopisujesz, jaki błąd może się pojawić, jak ten błąd rozpoznasz po komunikacie i jakie zdanie dodasz do promptu po jego wystąpieniu. Jeden z błędów ma być taki, który nie zatrzymuje programu i można go zauważyć dopiero po porównaniu wyniku z liczeniem ręcznym.

Moduł 4. Czytanie, testowanie i poprawianie kodu z AI

Lekcja 4.1. Czytanie kodu z AI: sześć pytań przed zatwierdzeniem zmiany

Weź dowolną funkcję napisaną przez AI (może być wymyślona w czacie na prośbę o przeliczanie kwot z faktur VAT w złotych na podstawie pliku CSV z polskiego systemu księgowego). Przejdź przez sześć pytań z lekcji i zapisz po jednej linii odpowiedzi do każdego. Następnie przygotuj jeden plik z trzema wierszami, które mają przełamać funkcję: kwotę ze spacją, pustą kwotę i polską literę w nazwie klienta. Uruchom funkcję i zapisz, co się stało oraz czy przyjmujesz kod, poprawiasz go, czy odrzucasz.

Lekcja 4.2. Testy od AI: jak o nie poprosić i jak sprawdzić, czy coś wykrywają

Wybierz funkcję, która sumuje kwoty faktur w złotych (może być wymyślona w czacie). Poproś AI o testy z wartościami oczekiwanymi wpisanymi na stałe. Potem skopiuj kod funkcji, zepsuj w kopii jedną linię (zamień dodawanie na przypisanie albo zmień znak porównania), uruchom testy i zapisz, które padły. Jeśli żaden test nie padł, dopisz jeden,

który wykryje ten błąd. Na końcu cofnij zepsucie w kopii.

Lekcja 4.3. Debugowanie i halucynacje: funkcja, której nie ma, i przykład sprzed roku

Poproś AI o krótki skrypt, który wczytuje plik CSV zamówień (średnik, przecinek dziesiętny) i wypisuje sumę kwot. Uruchom go. Dla każdej użytej funkcji lub importu sprawdź w dokumentacji Pythona albo biblioteki, czy istnieje w Twojej wersji. Zapisz trzy rzeczy: jaką wersję Pythona masz, który element sprawdzasz jako pierwszy i czy w kodzie jest coś nieistniejącego lub nieaktualnego. Jeśli nic, zapisz, jak to potwierdzasz.

Moduł 5. Bezpieczeństwo: sekrety, zależności, prompt injection, licencje

Lekcja 5.1. Klucze API, hasła i pliki .env: czego agent nie może zobaczyć ani opublikować

Wymyśl mały projekt w Pythonie, który wysyła SMS-y z przypomnieniem o wizycie w gabinecie w Krakowie i potrzebuje klucza do bramki SMS. Opisz w pięciu punktach, jak ułożysz projekt: gdzie wartość klucza trafi, co wpiszesz do `.gitignore`, co do `.env.example`, jak program odczyta klucz i co zrobisz, jeśli klucz przypadkiem pojawi się w commicie.

Lekcja 5.2. Zależności i prompt injection: gdy treść z zewnątrz próbuje sterować agentem

Wymyśl projekt, w którym agent ma czytać zgłoszenia klientów z formularza (na przykład opinie o sklepie z kawą w Lublinie) i przygotować dla właściciela krótkie streszczenie. Wypisz, jakie uprawnienia dasz agentowi, jakich narzędzi mu nie dasz, jak oddzielisz te dane od kodu firmy i co zrobisz, jeśli w jednej z opinii znajdziesz tekst, który wygląda jak polecenie do AI.

Lekcja 5.3. Kod z AI w firmie: licencje, prawa autorskie i dane, które wolno wkleić do czatu

Przygotuj dla swojej firmy lub wymyślonej firmy listę pięciu pytań o kod z AI i dane. Każde pytanie niech wskazuje dokument, w którym szukasz odpowiedzi, i osobę, do której je skierujesz. Dodaj jedno zdanie o tym, jakie dane testowe będziesz podawać agentowi zamiast prawdziwych.

Moduł 6. MCP (Model Context Protocol): serwery, narzędzia, własny serwer

Lekcja 6.1. MCP: co to jest i po co agentowi serwer, który udostępnia narzędzia

Wybierz jedną powtarzalną czynność, w której wklejasz do czatu AI dane z pliku lub systemu (na przykład sumy sprzedaży z pliku CSV w Opolu). Opisz, jak wyglądałby serwer MCP dla tej czynności: jedno narzędzie z nazwą, parametrem i wynikiem, jeden zasób, jeśli ma sens, oraz informację, czy narzędzie tylko czyta, czy też zapisuje. Dopisz, jakiej zgody wymagałoby wywołanie i dlaczego.

Lekcja 6.2. Serwery MCP w VS Code: komu ufasz, co zatwierdzasz i jakie uprawnienia dajesz

Wymyśl serwer MCP, który ktoś Ci poleca (na przykład do czytania zgłoszeń z systemu obsługi klienta w sklepie w Poznaniu). Napisz listę kontrolną przed dodaniem: pięć pytań o wydawcę, polecenie, dostęp do danych, sposób przekazania klucza i możliwość cofnięcia zmian. Do każdego pytania dopisz, jaką odpowiedź uznasz za wystarczającą, a jaką za powód do rezygnacji.

Lekcja 6.3. Własny serwer MCP w Pythonie: napisanie, test i pułapka starych przykładów

Dopisz do serwera trzecie narzędzie tylko do odczytu, które zwraca zamówienia wskazanego klienta po imieniu i nazwisku, a potem rozszerz skrypt testowy tak, żeby je wywołał. Uruchom całość na własnym komputerze (pakiet `mcp` instalujesz przez pip w środowisku wirtualnym) albo, jeśli nie masz Pythona, poproś o kod czat AI i przeanalizuj go zgodnie z pytaniami z lekcji 4.1. Zapisz, co zwróciło narzędzie dla klienta, który jest w pliku, i dla takiego, którego nie ma.

Moduł 7. Praca zespołowa i produkcja: Git, pull requesty, CI

Lekcja 7.1. Git przy pracy z AI: gałąź, małe commity, czytelna historia

Wymyśl małą zmianę w dowolnym projekcie, na przykład dodanie do raportu zamówień sumy w złotych z dwoma miejscami dziesiętnymi, i zapisz plan pracy w Gicie. Podaj nazwę gałęzi, trzy kroki, które zlecisz agentowi lub czatowi AI, oraz komunikat commitu po każdym kroku. Dopisz, co sprawdzisz w diffie przed pierwszym commitem.

Lekcja 7.2. Pull request z kodu AI: opis, przegląd i ochrona gałęzi

Weź dowolną zmianę z kodu, który powstał z pomocą AI (może być sztuczna: funkcja zamieniająca kwotę "1 250,00" na liczbę), i napisz do niej opis PR w sześciu liniach: co, po co, jak sprawdzić, udział AI, gdzie patrzeć i czego nie zmieniano. Potem zapisz trzy pytania, które zadasz jako recenzent, i jedną regułę ochrony gałęzi, którą włączyłbyś w repozytorium.

Lekcja 7.3. CI i granica wdrożenia: kiedy nie wypuszczać kodu z AI bez przeglądu

Przygotuj listę trzech kontroli przed wdrożeniem dla małej aplikacji, która liczy sumy zamówień w złotych, oraz decyzję dla trzech zmian: A) agent zmienia kolor nagłówka w raporcie, B) agent zmienia sposób zaokrąglania kwot, C) agent dodaje nowy pakiet do czytania plików. Dla każdej zmiany napisz, czy wystarczy zielone CI, czy potrzebny jest przegląd człowieka, i podaj jeden powód.

Moduł 8. Projekt końcowy: aplikacja z AI, plan przeglądu i wdrożenia, egzamin

Lekcja 8.1. Projekt końcowy: karta projektu i specyfikacja aplikacji

Wybierz własny projekt końcowy (mały, na kilka godzin pracy) i wypełnij kartę projektu w ośmiu polach, a potem napisz specyfikację na jedną stronę z jednym przykładem danych wejściowych i wynikiem dla tego przykładu. Podaj co najmniej trzy kryteria gotowości, które da się sprawdzić, i dwie rzeczy, których aplikacja nie robi.

Lekcja 8.2. Projekt końcowy: budowa w iteracjach i dziennik pracy z AI

Weź specyfikację z lekcji 8.1 i rozpisz ją na pięć iteracji. Do każdej napisz cel w jednym zdaniu i kryterium gotowości, które da się sprawdzić poleceniem. Potem wypełnij jeden wpis dziennika dla pierwszej iteracji w sześciu polach: data i krok, skrót promptu, wynik agenta, co sprawdzono, decyzja z komunikatem commitu, wniosek.

Lekcja 8.3. Projekt końcowy: plan przeglądu i wdrożenia

Napisz plan przeglądu i wdrożenia dla swojego projektu końcowego. Plan ma mieć cztery części kontroli (poprawność, bezpieczeństwo, proces, dane i prawa) po dwa punkty, nazwisko osoby zatwierdzającej, opis pierwszego wdrożenia i plan wycofania w trzech krokach. Dopisz jedną rzecz, której nie jesteś w stanie ocenić sam, i osobę, którą o to zapytasz.

Przykładowe odpowiedzi

Moduł 1. Czym jest vibe coding i gdzie się kończy

Lekcja 1.1. Vibe coding: co to jest i dla kogo jest ten kurs

Przykładowa odpowiedź: Definicja: opisuję po polsku, co program ma robić, a AI pisze kod; ja go uruchamiam i sprawdzam, a w ważnych miejscach czytam. Sytuacja 1: skrypt, który łączy trzy pliki CSV z faktur do jednego arkusza; uruchamiam raz w miesiącu, kod zapisuję w repozytorium, dane to faktury z NIP klientów, więc czytam kod i nie wysyłam danych do narzędzi bez zgody firmy. Sytuacja 2: mały kalkulator rat do własnej nauki; wystarczy uruchomienie i kilka przykładów, kod nie trafia do nikogo. Sytuacja 3: formularz zamówień dla klientów sklepu; to cudze dane i płatności, więc potrzebuję przeglądu przez osobę, która zna się na bezpieczeństwie.

Do wydruku: [Zeszyt ćwiczeń do kursu (PDF)](/downloads/Zeszyt-cwiczen-vibe-coding.pdf) zawiera wszystkie 24 ćwiczenia z kursu wraz z przykładowymi rozwiązaniami.

Lekcja 1.2. Kiedy vibe coding działa: prototyp, skrypt, narzędzie wewnętrzne

Przykładowa odpowiedź: Zadanie: skrypt, który z eksportu CSV ze sklepu internetowego z Katowic zlicza zamówienia według miasta. Małe: tak. Wynik widać w tabeli: tak. Praca na kopii: tak, eksport jest plikiem, który mogę pobrać ponownie. Mało osób: tak, korzystam tylko ja. Bez cudzych danych: nie, plik zawiera imiona i adresy klientów. Wynik: cztery tak, więc dodaję kontrolę: usuwam kolumny z adresami przed wklejeniem czegokolwiek do narzędzia AI, sprawdzam sumy z oryginałem i pytam inspektora ochrony danych, czy w ogóle wolno mi używać takiego pliku w narzędziu AI.

Lekcja 1.3. Dług techniczny: przypadek, w którym nikt nie rozumie własnej aplikacji

Przykładowa odpowiedź: Program: skrypt Doroty do zamówień powyżej 5000 zł. Kto uruchomi bez niej: nikt, więc dopisuję trzy zdania instrukcji uruchomienia i przekazuję je Piotrowi z działu IT. Opis działania: brak, więc proszę AI o szkic README i poprawiam go po przeczytaniu kodu. Test: brak, więc piszę test dla kwoty "5 250,00 zł" i dla pustej kwoty. Zmiana formatu: skrypt zgłasza nieczytelny błąd, więc dodaję komunikat z numerem wiersza i zdaniem "sprawdź format kolumny kwota".

Moduł 2. GitHub Copilot w edytorze: uzupełnianie, czat, agent, instrukcje repozytorium

Lekcja 2.1. GitHub Copilot: jak zacząć w VS Code i co oznaczają plany

Przykładowa odpowiedź: Twierdzenie 1: "Funkcja kwota zamienia przecinek na kropkę". Sprawdzone: w linii z replace widzę dokładnie tę zamianę. Twierdzenie 2: "Funkcja usuwa spacje z liczby". Sprawdzone: tak, jest replace(" ", ""). Twierdzenie 3: "Kod obsługuje liczby ujemne". Nie umiem zweryfikować, bo w kodzie nie ma żadnego przypadku z minusem; uruchamiam funkcję dla "-5,00" i sprawdzam wynik. Przy pytaniach o licencję lub zasady firmy zapytam administratora planu i dział prawny.

Lekcja 2.2. Uzupełnianie i czat: jak pytać o kod, żeby dostać użyteczną odpowiedź

Przykładowa odpowiedź: Cel: wczytać datę zamówienia z kolumny "data" w formacie 31.12.2026. Kod: wiersz d = datetime.strptime(w["data"], "%Y-%m-%d"). Błąd: ValueError: time data '31.12.2026' does not match format '%Y-%m-%d'. Ograniczenia: Python 3.11, tylko biblioteka standardowa. Odpowiedź: zmiana w jednej linii i jedno zdanie wyjaśnienia. Ocena odpowiedzi: powinna zmienić format na "%d.%m.%Y"; sprawdzam to, uruchamiając funkcję dla "31.12.2026" i dla błędnej daty "2026-12-31", żeby zobaczyć, że drugi przypadek nadal zgłasza czytelny błąd.

Lekcja 2.3. Tryb agenta i instrukcje repozytorium: copilot-instructions.md i AGENTS.md

Przykładowa odpowiedź: Instrukcje: 1) Python 3.11. 2) Kwoty w zł jako Decimal. 3) Pliki z eksportu mają średnik jako separator. 4) Po zmianie uruchom pytest i pokaż wynik. 5) Nie dodawaj bibliotek i nie ruszaj pliku konfiguracja.py bez pytania. 6) Przed zmianami wypisz plan. Zadanie: "Dodaj do raportu osobną sumę dla zamówień ze statusem anulowane. Gotowe, gdy pytest przechodzi i raport pokazuje tę sumę. Zmieniaj tylko plik raport.py i jego test."

Moduł 3. Pierwszy projekt z AI krok po kroku

Lekcja 3.1. Specyfikacja na jednej stronie: co zapisać, zanim poprosisz o kod

Przykładowa odpowiedź: Zadanie: zliczanie zgłoszeń według działu. Cel: tabela liczby zgłoszeń na dział, uruchamiana raz w miesiącu przez Tomasza z działu IT w Rzeszowie. Wejście: zgloszenia.csv, UTF-8, separator średnik, kolumny numer;dzial;data. Wyjście: po jednej linii "dział: liczba", od największej liczby. Reguły: dział pisany bez rozróżniania wielkości liter. Błędy: pusty dział trafia do grupy "brak działu" i program wypisuje ostrzeżenie. Poza zakresem: wykresy i wysyłka mailem. Ograniczenia: Python, biblioteka standardowa. Gotowe, gdy dla pliku z 6 zgłoszeniami (3 sprzedaż, 2 magazyn, 1 bez działu) wynik to sprzedaż: 3, magazyn: 2, brak działu: 1.

Lekcja 3.2. Podział na małe zadania: kolejność, kryteria gotowości, commit po każdym kroku

Przykładowa odpowiedź: Zadanie: zliczanie zgłoszeń według działu. Z1: wczytaj zgloszenia.csv i zwróć listę wierszy; gotowe, gdy dla pliku z 6 wierszami lista ma 6 elementów; commit: "Dodaje wczytywanie pliku zgłoszeń". Z2: zlicz zgłoszenia według działu; gotowe, gdy sprzedaż ma 3, magazyn 2; commit: "Dodaje zliczanie według działu". Z3: puste pole działu trafia do grupy "brak działu"; gotowe, gdy test z pustym polem zwraca 1 w tej grupie; commit: "Obsługuje brak działu". Z4: posortuj wynik od największej liczby; gotowe, gdy pierwszą linią jest sprzedaż; commit: "Sortuje wynik malejąco". Z5: wypisz tabelę; gotowe, gdy program pokazuje trzy linie; commit: "Dodaje wyjście programu".

Lekcja 3.3. Iteracje: raport zamówień z pliku CSV krok po kroku

Przykładowa odpowiedź: Program: suma kwot faktur według miesiąca. Iteracja 1: wczytanie pliku; możliwy błąd: KeyError: 'kwota' przy złym separatorze; dopisuję do promptu "separator to średnik". Iteracja 2: zamiana kwoty na Decimal; możliwy błąd: ValueError przy zapisie "1 250,00"; dopisuję "kwoty mają przecinek i spację jako separator tysięcy". Iteracja 3: grupowanie według miesiąca; możliwy cichy błąd: faktura z datą 05.11.2026 trafia do maja, bo kod czyta dzień jako miesiąc (data 31.12.2026 zatrzymałaby program błędem, więc tę wychwycisz od razu); wynik wygląda sensownie, więc porównuję sumę każdego miesiąca z ręcznym liczeniem dla trzech faktur i dopisuję do promptu "format daty to dzień.miesiąc.rok".

Moduł 4. Czytanie, testowanie i poprawianie kodu z AI

Lekcja 4.1. Czytanie kodu z AI: sześć pytań przed zatwierdzeniem zmiany

Przykładowa odpowiedź: Przykład fikcyjny, firma Hurtownia Kwiatów Jaśmin w Gdańsku, funkcja liczy sumę brutto z kolumny kwota. 1) Zmieniony jeden plik, 22 linie, nic poza zakresem. 2) Funkcja otwiera plik, dla każdego wiersza zamienia przecinek na kropkę i dodaje kwotę do sumy. 3) Wejście to CSV ze średnikiem; kod zakłada brak spacji i brak pustych pól. 4) Wywróci się przy kwocie `1 250,00`, przy pustej kwocie i może przy znaku ł, bo plik jest otwarty bez kodowania. 5) Funkcje z biblioteki standardowej istnieją, potwierdzone w dokumentacji Pythona. 6) Plik testowy z trzema wierszami: kwota ze spacją zakończyła się błędem `ValueError`, pusta kwota również. Decyzja: poprawiam, czyli prosimy agenta o `Decimal`, usunięcie spacji, obsługę pustych pól i `encoding="utf-8"`, a potem czytam nowy diff od pytania pierwszego.

Lekcja 4.2. Testy od AI: jak o nie poprosić i jak sprawdzić, czy coś wykrywają

Przykładowa odpowiedź: Przykład fikcyjny, firma Meblex Wschód z Białegostoku, funkcja `suma_faktur(lista)` dodaje kwoty netto z pliku CSV. Zapytanie do AI: "Napisz testy pytest, wartości policz ręcznie: dla faktur 100,00 i 200,00 zł wynik to 300,00 zł." Zepsucie: w kopii zamieniam `suma += kwota` na `suma = kwota`. Uruchomienie: padł test z dwiema fakturami (spodziewane 300,00, otrzymane 200,00), test z jedną fakturą przeszedł. Wniosek: test z jedną fakturą nie wykryłby błędu sumowania, więc zostawiam test z dwiema i dopisuję test z pustą listą, która ma dać 0,00. Zepsucie cofam, uruchamiam testy ponownie i widzę komplet zielonych.

Lekcja 4.3. Debugowanie i halucynacje: funkcja, której nie ma, i przykład sprzed roku

Przykładowa odpowiedź: Przykład fikcyjny, biuro rachunkowe Bilans Plus z Katowic, skrypt liczący sumę kwot z pliku faktury.csv. Wersja: Python 3.12.3. Skrypt używał `csv.DictReader(f, delimiter=";")` i `Decimal`. Pierwszy sprawdzany element: `delimiter`, który jest w opisie `csv.DictReader` w dokumentacji Pythona 3.12. Drugi: `Decimal(tekst.replace(" ", "").replace(",", "."))`, bo pierwsza wersja nie usuwała spacji i kwota `1 250,00` kończyła się błędem `decimal.InvalidOperation`. Nic nieistniejącego w kodzie nie było, ale wyszedł błąd danych. Potwierdzenie: uruchomienie na pliku z kwotą ze spacją, po poprawce wynik 1250.00.

Moduł 5. Bezpieczeństwo: sekrety, zależności, prompt injection, licencje

Lekcja 5.1. Klucze API, hasła i pliki .env: czego agent nie może zobaczyć ani opublikować

Przykładowa odpowiedź: Przykład fikcyjny, Gabinet Fizjoterapii Wisła w Krakowie. 1) Wartość klucza wpisuję tylko do

lokalnego pliku `.env` w linii `SMS_API_KEY=` i nie wklejam jej do czatu z AI. 2) Do `.gitignore` dopisuję linię `.env`. 3) Do `.env.example` wpisuję `SMS_API_KEY=` bez wartości i ten plik trafia do repozytorium. 4) Program odczytuje klucz przez `os.environ["SMS_API_KEY"]`, więc przy braku zmiennej zatrzymuje się z komunikatem `KeyError`. 5) Jeśli klucz trafi do commitu, unieważniam go w panelu bramki, tworzę nowy, aktualizuję `.env` i dopiero potem czyszczę repozytorium; przed każdym commitem uruchamiam skrypt sprawdzający na zmienionych plikach.

Lekcja 5.2. Zależności i prompt injection: gdy treść z zewnątrz próbuje sterować agentem

Przykładowa odpowiedź: Przykład fikcyjny, sklep Kawiarnia Pod Lipą w Lublinie. Uprawnienia: agent czyta jeden plik `opinie.csv` i pisze streszczenie w czacie. Bez narzędzi: wysyłania maili, żądań sieciowych, uruchamiania poleceń w terminalu i dostępu do folderu z kodem sklepu i plikiem `.env`. Oddzielenie: opinie analizuję w osobnym, pustym folderze, w którym leży tylko kopia pliku. Gdy znajdę w opinii tekst typu "zignoruj poprzednie polecenia", nie zatwierdzam żadnej akcji, zapisuję numer wiersza, usuwam wiersz z kopii pliku i zgłaszam go osobie, która odpowiada za formularz. Streszczenie traktuję jako szkic i czytam oryginalne opinie, zanim podejmę decyzję na jego podstawie.

Lekcja 5.3. Kod z AI w firmie: licencje, prawa autorskie i dane, które wolno wkleić do czatu

Przykładowa odpowiedź: Przykład fikcyjny, firma Softserwis Nadodrze z Wrocławia, 12 osób. 1) Czy umowa o pracę zmienia zasadę z art. 74 ust. 3 ustawy o prawie autorskim i prawach pokrewnych? Dokument: umowa o pracę, osoba: prawnik zewnętrzny. 2) Czy regulamin narzędzia pozwala wysyłać do niego kod klientów? Dokument: regulamin i ustawienia planu, osoba: administrator narzędzi. 3) Czy w firmie wolno wkleić do czatu dane osobowe klientów? Dokument: polityka bezpieczeństwa, osoba: inspektor ochrony danych. 4) Co robimy, gdy narzędzie pokaże odniesienie do licencji? Dokument: wewnętrzna procedura, osoba: kierownik zespołu i prawnik. 5) Czy w umowie z klientem jest zapis o używaniu AI? Dokument: umowa z klientem, osoba: kierownik projektu i prawnik. Dane testowe: zmyślone wiersze wygenerowane promptem, bez prawdziwych nazwisk i numerów PESEL.

Moduł 6. MCP (Model Context Protocol): serwery, narzędzia, własny serwer

Lekcja 6.1. MCP: co to jest i po co agentowi serwer, który udostępnia narzędzia

Przykładowa odpowiedź: Przykład fikcyjny, hurtownia ceramiki Glina i Spółka z Opola, plik zamowienia.csv. Narzędzie `suma_zamowien` z parametrem `status` (nowe, wyslane lub anulowane) zwraca sumę kwot w złotych jako tekst, np. `360.50`. Drugie narzędzie, `lista_zamowien`, zwraca wiersze o danym statusie. Zasób: sam plik zamówień do odczytu, jeśli agent ma go analizować w całości. Oba narzędzia tylko czytają, nic nie zapisują ani nie usuwają. Zgoda: edytor pyta mnie przed pierwszym wywołaniem każdego narzędzia, a ja zatwierdzam, bo widzę nazwę narzędzia i parametr. Gdyby serwer miał też narzędzie zapisujące, wymagałoby ono osobnej, uważnej zgody za każdym razem.

Lekcja 6.2. Serwery MCP w VS Code: komu ufasz, co zatwierdzasz i jakie uprawnienia dajesz

Przykładowa odpowiedź: Przykład fikcyjny, sklep Domowe Przetwory z Poznania, serwer do zgłoszeń klientów. 1) Wydawca: znam autora i widzę repozytorium z historią zmian; rezygnuję, jeśli nie da się ustalić, kto go wydał. 2) Polecenie: lokalny skrypt, który przeczytam; rezygnuję, jeśli pobiera i uruchamia nieznany pakiet bez wskazanej wersji. 3) Dostęp: tylko odczyt zgłoszeń z jednego folderu; rezygnuję, jeśli serwer chce dostępu do całego dysku lub poczty. 4) Klucz: przekazywany zmienną środowiskową, nie wpisany do pliku konfiguracji; rezygnuję, jeśli trzeba wpisać hasło w formularzu. 5) Cofanie: narzędzia tylko czytają, więc nie ma czego cofać; jeśli serwer zapisuje, najpierw próba na kopii danych i zgoda na każde wywołanie.

Lekcja 6.3. Własny serwer MCP w Pythonie: napisanie, test i pułapka starych przykładów

Przykładowa odpowiedź: Narzędzie nazywa się `zamowienia_klienta`, ma parametr `klient` (tekst) i opis "Zwraca zamówienia klienta o podanym imieniu i nazwisku". Funkcja czyta plik przez `wczytaj`, zostawia wiersze, w których pole `klient` jest równe parametrowi, i zamienia kwoty na tekst, tak jak dwa pierwsze narzędzia. W skrypcie testowym wywołuję `call_tool` z parametrem `{"klient": "Ewa Zielińska"}`. Po uruchomieniu na pliku z kursu lista narzędzi ma trzy pozycje, a dla Ewy Zielińskiej wynik to jedno zamówienie Z-003 na 240.00, status nowe. Dla nazwiska "Nikt Taki" wynik to pusta lista, bez błędu. Narzędzie nic nie zapisuje, więc nie wymaga dodatkowych zgód poza zwykłym zatwierdzeniem wywołania w edytorze.

Moduł 7. Praca zespołowa i produkcja: Git, pull requesty, CI

Lekcja 7.1. Git przy pracy z AI: gałąź, małe commity, czytelna historia

Przykładowa odpowiedź: Gałąź: `raport-suma-zl`. Krok 1: agent dodaje funkcję sumującą kwoty zamówień (commit "Dodaj funkcję sumy kwot zamówień"). Krok 2: test dla kwoty z przecinkiem i spacją, np. "1 250,00" (commit "Dodaj test sumy dla zapisu 1 250,00"). Krok 3: wypisanie sumy w raporcie w formacie z "zł" (commit "Wypisz sumę w raporcie w zł"). W diffie sprawdzę, czy agent nie zmienił plików, których nie było w prompcie, czy nie dopisał nowej zależności i czy kwoty nie są liczone na liczbach zmiennoprzecinkowych. Przed pushem przejrzę diff pod kątem haseł i pliku `.env`.

Lekcja 7.2. Pull request z kodu AI: opis, przegląd i ochrona gałęzi

Przykładowa odpowiedź: Przykład fikcyjny, firma Sklep Rowerowy Kołodziej z Gdańska. Co: funkcja zamieniająca kwotę z CSV na liczbę. Po co: zamówienia powyżej 999 zł były pomijane w raporcie. Jak sprawdzić: `pytest -q` i raport na pliku z zamówieniem na 1 250,00 zł. Udział AI: funkcja i test od agenta, przeczytane. Gdzie patrzeć: linia z zamianą przecinka na kropkę. Nie zmieniano: formatu pliku ani zależności. Pytania recenzenta: czy test zawiedzie, gdy usunę usuwanie spacji; co się stanie przy pustym polu; czy w diffie jest coś spoza opisu. Reguła: wymagany jeden zatwierdzający recenzent i przejście testów przed scaleniem do `main`.

Lekcja 7.3. CI i granica wdrożenia: kiedy nie wypuszczać kodu z AI bez przeglądu

Przykładowa odpowiedź: Kontrole: 1) `pytest -q` w CI musi być zielony, 2) osoba inna niż autor czyta diff i opis PR, 3) raport uruchomiony ręcznie na pliku z zamówieniem na 1 250,00 zł daje oczekiwaną sumę. Zmiana A: wystarczy zielone CI i szybki rzut oka, bo nie dotyka liczenia ani danych. Zmiana B: przegląd człowieka, bo zaokrąglenia zmieniają kwoty w złotych, a testy mogły ich nie obejmować. Zmiana C: przegląd człowieka, bo nowa zależność trzeba sprawdzić, czy pakiet istnieje, kto go utrzymuje i co robi.

Moduł 8. Projekt końcowy: aplikacja z AI, plan przeglądu i wdrożenia, egzamin

Lekcja 8.1. Projekt końcowy: karta projektu i specyfikacja aplikacji

Przykładowa odpowiedź: Przykład fikcyjny. Nazwa: Kontrola zamówień. Cel: raport sum zamówień według statusu z pliku CSV sklepu Pasmanteria Wrona z Torunia. Użytkownik: właścicielka sklepu, raz w tygodniu. Wejście: `zamowienia.csv`, separator średnik, kwoty z przecinkiem, np. "1 250,00" w polu kwota. Wynik: linie w rodzaju "nowe: 360.50 zł". Granice: nie wysyła maili, nie zmienia pliku. Gotowe, gdy: `pytest -q` przechodzi, kwota "1 250,00" jest wczytana jako 1250,00, nieznany status zgłasza błąd z nazwą statusu. Ryzyka: pomyłka w kwotach, brak prawdziwych danych klientów w repozytorium. Narzędzia: Python, czat AI lub Copilot Free. Specyfikacja: przykład wiersza "Z-005;Maria Dąbrowska;1 250,00;wyslane" i oczekiwana suma dla statusu "wyslane".

Lekcja 8.2. Projekt końcowy: budowa w iteracjach i dziennik pracy z AI

Przykładowa odpowiedź: Przykład fikcyjny, projekt "Kontrola zamówień" dla sklepu z Torunia. Iteracje: 1) odczyt kwoty z zapisu "1 250,00", gotowe gdy test przechodzi; 2) wczytanie pliku CSV ze średnikiem, gotowe gdy pięć wierszy z pliku przykładowego daje pięć rekordów; 3) sumy według statusu, gotowe gdy suma dla "nowe" wynosi 360.50; 4) błąd dla nieznanego statusu, gotowe gdy test oczekuje ValueError; 5) wynik w złotych z dwoma miejscami dziesiętnymi, gotowe gdy `python raport.py zamowienia.csv` wypisuje linie "nowe: 360.50 zł". Wpis: 21.11.2026, krok 1. Prompt: funkcja kwota(), test dla "1 250,00", bez nowych zależności. Wynik agenta: funkcja i test, bez zmian w innych plikach. Sprawdzono: `pytest -q` zielony, diff przeczytany. Decyzja: przyjęte, commit "Dodaj odczyt kwoty z zapisu 1 250,00". Wniosek: w kolejnych promptach podawać dokładny przykład kwoty.

Lekcja 8.3. Projekt końcowy: plan przeglądu i wdrożenia

Przykładowa odpowiedź: Przykład fikcyjny, projekt "Kontrola zamówień" sklepu Pasmanteria Wrona z Torunia. Poprawność: `pytest -q` zielony; ręczne uruchomienie na pliku z zamówieniem 1 250,00 zł daje sumę 1339.00 zł dla statusu "wyslane". Bezpieczeństwo: brak plików `.env` w repozytorium (kontrola skryptem); jedna zależność, zapisana w README z powodem. Proces: gałąź `raport-statusy`, osobne commity po każdym kroku; pull request z opisem udziału AI i zielonym CI. Dane i prawa: w pliku przykładowym tylko zmyśleni klienci; zasady dla prawdziwych danych ustalam z inspektorem ochrony danych. Zatwierdza: Danuta Wrona. Pierwsze wdrożenie: uruchomienie raportu raz w tygodniu na kopii pliku, wyniki porównane ręcznie z sumami z arkusza. Wycofanie: 1) przestaję używać raportu, 2) wracam do arkusza, 3) wracam do ostatniego commitu oznaczonego "dobra wersja" i otwieram zgłoszenie z opisem błędu. Rzecz do zapytania: czy wolno wklejać prawdziwe nazwiska klientów do czatu AI, zapytam inspektora ochrony danych.

Do wydruku: wszystkie ćwiczenia z kursu razem z rozwiązaniami masz w [Zeszycie ćwiczeń (PDF)](/downloads/Zeszyt-cwiczen-vibe-coding.pdf). Po zakończeniu projektu możesz porównać pierwsze odpowiedzi z ostatnimi. Listę kontrolną przeglądu kodu i plan przeglądu do wydruku znajdziesz w pakiecie materiałów kursu: [Lista kontrolna przeglądu kodu z AI (PDF)](/downloads/Lista-kontrolna-przegladu-kodu-z-ai.pdf) i [Plan przeglądu i wdrożenia aplikacji (PDF)]

(/downloads/Plan-przegladu-i-wdrozenia-aplikacji.pdf).