Ćwiczenie 10
Agenci AI w pracy z kodem

10.1. Rozpoczęcie pracy

1. Otworzyć terminal linux i sprawdzić:

```
git --version
python3 --version
curl --version
```

   Wymagany Git 2.28 lub nowszy (`git init -b`) i Python 3.

2. Otworzyć `https://opencode.ai/docs/` i sprawdzić bieżącą instrukcję instalacji. Instalator dla własnego konta Linux pobrać do unikalnego pliku, przeczytać,
   a dopiero potem wykonać:

```
installer="$(mktemp -t ndp-opencode.XXXXXX)"
curl -fsSL https://opencode.ai/install -o "$installer"
less "$installer"
bash "$installer"
rm -f -- "$installer"
opencode --version
```

   W przypadku błędu usunąć plik tymczasowy przez `rm -f -- "$installer"`. Nie uruchamiać instalatora przez `sudo`. Gdy instalacja jest blokowana na stanowisku uczelnianym, ale użyć przygotowanego środowiska z załączników.

3. Pobrać z **załączników PDF** pliki `calc.py` oraz `test_calc.py`. Jeżeli aplikacja nie obsługuje załączników, pobrać pakiet `starter/` z strony z materiałami do zajęć.

```
mkdir -p ~/ndp-lab10/tests
cd ~/ndp-lab10
# Skopiuj calc.py do katalogu projektu.
# Skopiuj test_calc.py do katalogu tests/.
ls -l calc.py tests/test_calc.py
```

   Pliki pochodzą z `lab-10/starter/` w źródłach prowadzącego i nie wymagają bibliotek spoza standardowej instalacji Pythona.

4. Uruchomić testy przed modyfikowaniem kodu:

```
python3 -m unittest discover -s tests -v
```

   Oczekiwane: **5 testów, 1 PASS i 4 ERROR**. Brak `sub`, `mul` i `div` zgłaszany jest jako `AttributeError` (a nie `ImportError`). Zachować wynik do późniejszego porównania.

10.2. Repozytorium Git i reguły agenta

5. Zainicjować repozytorium i pierwszy commit:

```
cd ~/ndp-lab10
git init -b main
printf '.env\n.env.*\n*.log\n__pycache__/\n.opencode/\n' > .gitignore
git add calc.py tests/test_calc.py .gitignore
git commit -m "Add initial calculator and tests"
```

Katalog `.opencode/` jest celowo ignorowany w tym ćwiczeniu; reguły projektu będą śledzone w pliku `AGENTS.md`, a konfiguracja uprawnień w `opencode.json`. Nie dodawać tokenów ani haseł. W razie braku tożsamości Git ustawić `user.name` i `user.email` lokalnie w repozytorium.

6. Utworzyć `AGENTS.md` w katalogu projektu:

```
# NDP Lab 10
- Modify only files in the current repository.
- Use Python 3 standard library; no extra packages.
- Preserve numeric addition and all starter tests.
- Implement sub(a,b), mul(a,b), div(a,b).
- Division by zero must raise ZeroDivisionError.
- Run: python3 -m unittest discover -s tests -v
- Do not copy private data into prompts or reports.
```

Zatwierdzić plik:

```
git add AGENTS.md
git commit -m "Add agent rules"
```

`AGENTS.md` jest opisem zasad, a nie systemową blokadą działań.

10.3. Konfiguracja i pierwszy kontakt z OpenCode

7. **Przed uruchomieniem agenta** utworzyć w katalogu projektu plik `opencode.json` z ograniczeniami narzędzi:

```
cat > opencode.json <<'JSON'
{
  "$schema": "https://opencode.ai/config.json",
  "permission": {
    "edit": "ask",
    "bash": "ask",
    "webfetch": "deny",
    "websearch": "deny",
    "external_directory": "deny"
  }
}
JSON
git add opencode.json
git commit -m "Restrict agent permissions"
```

`ask` wymaga zgody na edycję i wykonanie polecenia; nie włączać automatycznego zatwierdzania. Ustawienia nie chronią przed udostępnieniem treści promptu dostawcy modelu. Nie wklejać danych osobowych, tajemnic ani cudzego niepublicznego kodu.

8. Uruchomić `opencode`; wybrać przez `/connect` tylko **rzeczywiście bezpłatny** model, a następnie przez `/models` odczytać jego identyfikator. Pomocniczo:

```
cd ~/ndp-lab10
opencode
# W osobnym terminalu:
opencode --version
opencode auth list
opencode models
```

   Dane logowania mogą znajdować się w `~/.local/share/opencode/auth.json`; nie kopiować tego pliku do projektu. **Ścieżka bez modelu:** jeżeli nie ma bezpłatnej usługi, wykonać punkty 9–16 ręcznie, a zamiast promptów sporządzić własny plan i implementację; uzyskać te same wyniki testów i historię Git. W `comparison.md` wpisać `model: brak (tryb reczny)`. Opis Plan/Build oprzeć na dokumentacji, nie deklarować pracy agenta.

9. W trybie **Plan** (jeśli model jest dostępny) wysłać jeden, nierozdzielony prompt ASCII:

```
Przeczytaj AGENTS.md i testy. Zaplanuj zmiany bez edycji plikow.
```

   W ścieżce ręcznej samodzielnie wypisać brakujące funkcje i testy. Następnie wykonać:

```
git status --short --ignored
git diff --check
```

   Nie zakładać stałej ścieżki przechowywania planów OpenCode. Tryb Plan może wywoływać narzędzia do odczytu, a zapisy planów roboczych nie są równoznaczne ze zmianami kodu projektu.

10.4. Implementacja i testowanie

10. Przełączyć się na **Build** albo wykonać implementację samodzielnie. Prompt ASCII:

```
Dodaj sub, mul i div. Nie zmieniaj testow. Uruchom unittest.
```

   Wymagania: działania na liczbach, zachowanie `ZeroDivisionError` dla zera, brak nowych bibliotek. Przed akceptacją każdego działania `edit` lub `bash` skontrolować treść żądania. Nie dopuszczać `sudo` ani operacji poza projektem.

11. W terminalu sprawdzić zmiany i ponowić testy:

```
python3 -m unittest discover -s tests -v
git diff --check
git diff -- calc.py tests/test_calc.py
git status --short
```

   Oczekiwane: **5/5 PASS**. Odrzucić niezamówione zmiany testów, instalację zależności, edycję plików poza projektem i modyfikacje z sekretami. Jeśli testy zostały niepotrzebnie zmienione, przywrócić tylko ten plik:

```
git restore tests/test_calc.py
```

12. Do `tests/test_calc.py` dodać jedną metodę testową sprawdzającą ujemną liczbę i dzielenie zmiennoprzecinkowe. Dla wyników float użyć `assertAlmostEqual`, np.:

```
self.assertAlmostEqual(calc.div(-0.3, 0.1), -3.0)
```

Ponowić testy: oczekiwane **6/6 PASS**. W `README.md` opisać uruchamianie kodu, testów oraz model (lokalny/chmurowy) albo tryb ręczny.

13. Zatwierdzić przejrzany kod i wyniki:

```
git add calc.py tests/test_calc.py README.md
git commit -m "Implement and test calculator"
git log --oneline -4
git status --short
```

W historii powinny już być osobne commity z plikami `AGENTS.md` i `opencode.json`.

10.5. Kontrolowana poprawka i weryfikacja

14. W Plan lub ręcznie ocenić zachowanie dla argumentów nienumerycznych. Dla `add`, `sub`, `mul` i `div` przyjąć kontrakt: tylko `int` i `float` (z wyłączeniem `bool`); inne typy zgłaszają `TypeError`. Sam Python nie egzekwuje takiego kontraktu (np. `mul("ab",3)` powiela napis).

15. Utworzyć gałąź roboczą:

```
git switch -c test/calculator-cases
```

Dodać **jedną metodę testową** dla `TypeError` na argumentach tekstowych we wszystkich czterech działaniach (można użyć `subTest`). Poprawić kod bez zmiany wcześniejszych testów i wykonać:

```
python3 -m unittest discover -s tests -v
git diff --check
git diff
git add calc.py tests/test_calc.py
git commit -m "Validate numeric argument types"
git switch main
git merge --ff-only test/calculator-cases
git branch -d test/calculator-cases
```

Oczekiwane: **7/7 PASS**; w przypadku konfliktu lub nieudanego testu nie scalać zmian.

16. Utworzyć `comparison.md`: model (lub ręczny), zadanie, wynik początkowy (1/5), wynik końcowy (7/7). Dodać dwa wnioski: po co sprawdzać `git diff` oraz czym różnią się tryby Plan i Build. Nie umieszczać tokenów ani prywatnych promptów.

10.6. Lokalne modele w Dockerze (dodatkowe)

17. Jeśli dostępny jest Docker, utworzyć `compose.yaml`:

```
services:
  ollama:
    image: ollama/ollama:0.40.13
    environment:
      OLLAMA_CONTEXT_LENGTH: "32768"
    ports:
      - "127.0.0.1:11434:11434"
```

```
    volumes:
      - ollama-data:/root/.ollama
volumes:
  ollama-data:
```

Uruchomić `docker compose up -d` i `docker compose ps`. Przypiętą wersję obrazu sprawdzić w oficjalnym Docker Hub przed wykonaniem. Nie nadawać studentowi uprawnień do grupy `docker` na współdzielonym komputerze; preferować własną VM lub rootless Docker. Port pozostaje lokalny.

18. Wybrać model oznaczony jako obsługujący *tools* w katalogu `https://ollama.com/library/`. Przykład: `qwen2.5-coder:7b`, alternatywnie `qwen3:4b` przy mniejszej pamięci RAM:

```
docker compose exec ollama ollama pull qwen2.5-coder:7b
docker compose exec ollama ollama list
curl -fsS http://127.0.0.1:11434/v1/models
```

W wariancie alternatywnym pobrać zamiast tego `qwen3:4b`. Weryfikować działanie narzędzi empirycznie; sama etykieta modelu nie gwarantuje poprawnych wywołań. Kontekst `32768` zwiększa zapotrzebowanie na pamięć, dlatego przy braku zasobów pominąć rozszerzenie.

19. W pliku `~/.config/opencode/opencode.json` dodać wpis `provider`:

```
{
  "$schema": "https://opencode.ai/config.json",
  "provider": {
    "ollama": {
      "npm": "@ai-sdk/openai-compatible",
      "name": "Ollama (local)",
      "options": {"baseURL": "http://127.0.0.1:11434/v1"},
      "models": {"qwen2.5-coder:7b": {"name": "Qwen coder"}}
    }
  }
}
```

To tylko **przykład fragmentu konfiguracji**, a nie polecenie nadpisania dotychczasowych dostawców lub kluczy. W razie wyboru `qwen3:4b` trzeba podmienić identyfikator. Sprawdzić dokumentację bieżącej wersji OpenCode: `https://opencode.ai/docs/providers/`.

10.7. Zadania dodatkowe

20. **Porównaj 2 darmowe modele**, identyczny stan początkowy kodu i **co najmniej 3** niezależne przebiegi na model. Zanotować czas, wynik testów i liczbę tur z nową poprawką (patch) po nieudanym teście. Wyniki w `ext-models.md`. Dla każdego przebiegu użyć osobnej kopii lub worktree.

21. **Embeddingi;** Dla lokalnej Ollamy pobrać `bge-m3` i wysłać pięć zdań w żądaniu:

```
curl -fsS http://127.0.0.1:11434/api/embed \
  -H 'Content-Type: application/json' \
  -d '{"model":"bge-m3","input":["Kot spi.",
       "Kotek odpoczywa.","Pies biega.",
       "Szczeniak biegnie.","Samochod hamuje."]}'
```

Obliczyć podobieństwa cosinusowe dwóch par pokrewnych (zdania 1–2 i 3–4) oraz jednej niepodobnej (1–5). Pozostałych siedmiu z dziesięciu możliwych par nie trzeba analizować.

Dane i wyniki w `ext-embeddings.md`.

22. **Rozpoznawanie mowy;** Nagrać 20-30 sekund swojej mowy; przetwarzać lokalnie, bez przekazywania danych do usługi chmurowej. Jeśli istnieje dostępny serwer STT zgodny z API transkrypcji, porównać wynik z transkryptem wzorcowym. Policzyć `WER=(S+D+I)/N`, gdzie `N` jest liczbą słów wzorca, a `S`, `D`, `I` to podstawienia, usunięcia i wstawienia. Odległość edycyjną wyznaczyć na sekwencjach słów. Wyniki i narzędzie zapisać w `ext-stt.md`.

10.8. Zakończenie ćwiczenia

23. Uruchomić wszystkie testy i wyeksportować potwierdzenie historii (bez `.git`):

```
python3 -m unittest discover -s tests -v
git log --oneline --stat > history.txt
git status --short
```

Przedstawić `calc.py`, `tests/test_calc.py`, `AGENTS.md`, `opencode.json`, `README.md`, `comparison.md` i `history.txt`. Wymagane: **7/7 PASS**, co najmniej trzy commity, brak niezamierzonych zmian i sekretów. Pliki `ext-*.md` są nieobowiązkowe i nie wpływają na wykonanie części podstawowej. Nie przesyłać katalogu `.git/`, `auth.json` ani modeli.