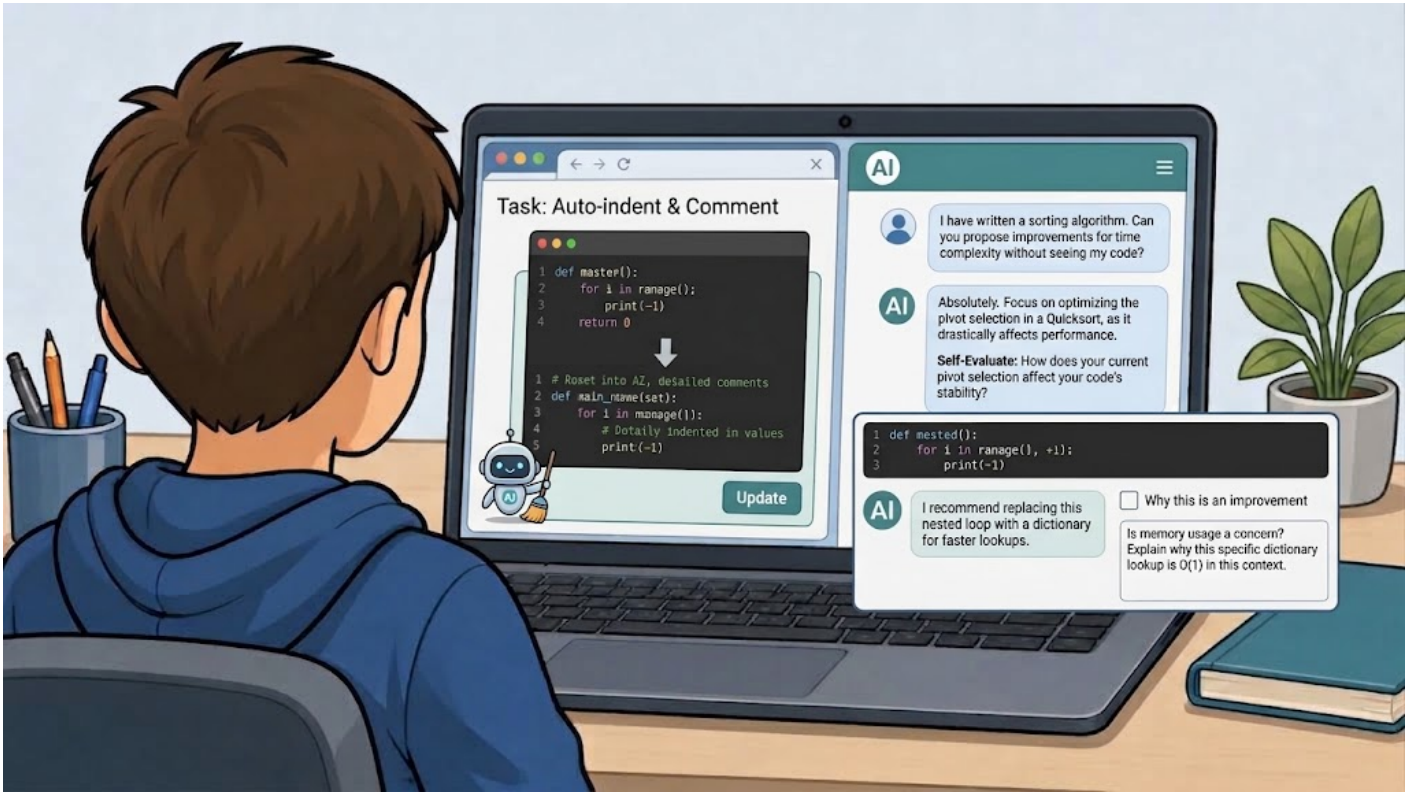
2.10 Programowanie: wsparcie w wybranych zadaniach

Nauka programowania bez pisania kodu przez SI

Przedmiot: Programowanie · **Klasa:** 9 (wiek 14–15) · **Czas trwania:** 1 lekcja podwójna (90 min) · **SI:** Wsparcie **Z** SI

Kompetencje związane z SI (Ramy DUAL.AI.TEACHer) – poziom nauczyciela	**Przedmiotowe cele kształcenia – poziom ucznia**
AF-TL-2b *(Podstawy i zastosowania SI × Nauczanie i uczenie się, poziom 2 – refleksyjne wdrażanie)*: „Nauczyciele potrafią organizować elementy lekcji, które osadzają wykorzystanie SI w jasnych rolach pedagogicznych oraz strategiach weryfikacji elementów wyniku.” **AF-FC-2b** *(Podstawy i zastosowania SI × Wspieranie kompetencji cyfrowych (SI) uczniów, poziom 2 – refleksyjne wdrażanie)*: „Nauczyciele potrafią wdrażać wskazówki, które pomagają uczniom rozwijać praktyczne umiejętności w zakresie SI, dostosowując zadania do wcześniejszej wiedzy uczniów oraz dostępnych narzędzi SI.”	Po zakończeniu lekcji uczniowie potrafią: • wiedzieć, jak wykorzystywać SI do powtarzalnych lub pomocniczych zadań w programowaniu, takich jak formatowanie wcięć w kodzie czy pisanie komentarzy • poprosić model SI o zaproponowanie ulepszeń kodu bez podawania samego kodu, aby móc samodzielnie ocenić swoją wiedzę • krytycznie ocenić model SI, gdy pytają go, w jaki sposób zoptymalizowałby dany kod

> **Najważniejsze przesłanie:** Uczniowie muszą znać podstawy programowania, jeśli chcą oceniać wyniki podawane przez model. Dlatego na pierwszych etapach nauki programowania uczniowie powinni wykorzystywać modele SI do zadań pomocniczych i powtarzalnych związanych z programowaniem, ale nie do tworzenia kodu. Mogą też prosić model o możliwe ulepszenia kodu, aby sprawdzić swoje umiejętności programistyczne.



Rys. 1. Obraz wygenerowany za pomocą Gemini (3.6 Thinking), 8 września 2026 r.

Tre??

Nauczanie programowania zmienia się w kontekście, w którym systemy SI potrafią generować, wyjaśniać i recenzować kod. Choć narzędzia te dają nowe możliwości uczenia się, stawiają też ważne wyzwanie edukacyjne: uczniowie potrzebują wystarczającej wiedzy o programowaniu, aby ocenić, czy podpowiedź wygenerowana przez SI jest poprawna, wydajna czy w ogóle istotna dla danego zadania. Bez podstawowego rozumienia pojęć programistycznych uczniowie mogą przyjmować błędne rozwiązania, przeoczać błędy lub nie rozpoznawać lepszych alternatyw. Z tego powodu nauka podstaw programowania pozostaje niezbędna, nawet gdy dostępne są narzędzia SI.

Na wstępnych kursach programowania SI powinna być traktowana jako narzędzie wspierające uczenie się, a nie jako zastępczy programista. Uczniowie mogą skutecznie wykorzystywać SI do zadań powtarzalnych lub prostych, które nie zmniejszają wysiłku poznawczego związanego z nauką programowania. Przykłady to formatowanie kodu i wcięć, generowanie komentarzy dokumentacyjnych, wyjaśnianie komunikatów o błędach, podsumowywanie reguł składni czy pomoc w zrozumieniu przeznaczenia określonych konstrukcji programistycznych. Takie zastosowania pozwalają uczniom skupić uwagę na leżącej u podstaw logice, algorytmach i procesach rozwiązywania problemów, które są rdzeniem kompetencji programistycznych.

Innym produktywnym zastosowaniem SI jest przegląd kodu. Zamiast prosić model SI o napisanie rozwiązania, uczniowie mogą najpierw napisać własny kod, a następnie poprosić model o wskazanie możliwych ulepszeń. SI może zaproponować czytelniejsze nazwy zmiennych, lepszą organizację kodu, mniejszą redundancję, większą czytelność lub alternatywne podejścia, które

uczeń może ocenić samodzielnie. Zamienia to SI w partnera udzielającego informacji zwrotnej i pozwala uczniom porównać własne rozumowanie z zewnętrznymi podpowiedziami oraz ocenić mocne i słabe strony swoich rozwiązań.

Informacji zwrotnej wygenerowanej przez SI nie należy jednak przyjmować bezkrytycznie. Modele mogą zalecać zmiany zbędne, nieefektywne lub oparte na błędnych założeniach co do przeznaczenia kodu. Głównym celem tej lekcji jest zatem rozwijanie u uczniów umiejętności krytycznej oceny podpowiedzi SI. Uczniowie nauczą się pytać model SI nie tylko o to, jakie ulepszenia proponuje, ale także dlaczego uznaje te zmiany za korzystne. Porównując rozumowanie modelu z uznanymi zasadami programowania, uczniowie mogą ocenić, czy proponowana optymalizacja rzeczywiście ulepsza kod, czy stanowi jedynie inną decyzję projektową.

Ta lekcja traktuje SI jako narzędzie wspierające refleksję, samoocenę i poprawę jakości kodu, a nie jako generator kodu. Uczniowie ćwiczą pisanie własnych programów, wykorzystują SI do zadań pomocniczych, proszą o informację zwrotną na temat swojej pracy i krytycznie analizują jakość otrzymywanych odpowiedzi. Ostatecznym celem jest pomoc uczniom w tym, by stali się zarówno lepszymi programistami, jak i bardziej świadomymi użytkownikami systemów SI.

Plan lekcji

Faza	Czas	Aktywność
Wprowadzenie	10 min	Nauczyciel przedstawia dwa krótkie rozwiązania tego samego problemu w kodzie: jedno napisane przez ucznia, drugie wygenerowane przez model SI. Uczniowie dyskutują, które rozwiązanie wydaje się czytelniejsze, wydajniejsze lub łatwiejsze do zrozumienia.
Wejście	10 min	Nauczyciel przedstawia właściwe zastosowania SI w nauczaniu programowania, podkreślając różnicę między zadaniami pomocniczymi, przeglądem kodu i generowaniem kodu.
Eksploracja	15 min	Uczniowie pracują w parach nad niewielkim zadaniem programistycznym i wskazują, które zadania można w uzasadniony sposób powierzyć asystentowi SI, a które powinny pozostać ich własną odpowiedzialnością.
Cykl ulepszania	35 min	Uczniowie piszą własne rozwiązanie i przechodzą 2–3 rundy cyklu napisz → zapytaj → oceń → popraw, wykorzystując SI wyłącznie do zadań pomocniczych i propozycji ulepszeń.
Refleksja	20 min	Dyskusja z całą klasą o przydatności i ograniczeniach informacji zwrotnej od SI, skupiona na tym, jak uczniowie ustalali, czy podpowiedzi były wartościowe.

Pytania przewodnie: stopniowe zwiększanie złożoności

Uczniowie wykonują tyle rund, ile pozwoli czas w trakcie 35-minutowego cyklu ulepszania. Każda runda przebiega według tego samego czterostopniowego procesu: napisz → zapytaj → oceń → popraw

1. **Runda 1 — wykorzystanie SI do zadań pomocniczych**
   - Napisz: wykonaj samodzielnie krótkie zadanie programistyczne.
   - Zapytaj: poproś SI o wsparcie w zadaniu niezwiązanym z samym programowaniem, na przykład o:
     - poprawę wcięć,
     - wygenerowanie komentarzy,
     - wyjaśnienie komunikatu o błędzie,
     - opisanie tego, co robi dana funkcja.
   - Oceń: czy wyjaśnienie SI trafnie opisuje kod? Czy wygenerowane komentarze są przydatne i zrozumiałe?
   - Popraw: wprowadź pomocne zmiany, upewniając się, że logika programu pozostaje w pełni twoim własnym dziełem.
2. **Runda 2 — proszenie o propozycje ulepszeń**
   - Napisz: przejrzyj ukończony program i wskaż fragmenty, które można by ulepszyć.
   - Zapytaj: poproś o propozycje ulepszeń, nie pozwalając SI przepisać kodu. Na przykład:

     > „Zaproponuj trzy sposoby poprawienia czytelności i łatwości utrzymania tego programu, ale nie podawaj kodu zastępczego.”

   - Oceń: które propozycje wydają się wartościowe? Które miałyby niewielki wpływ na program?
   - Popraw: wprowadź ulepszenia, z którymi się zgadzasz, i sprawdź, czy program nadal działa poprawnie.
2. **Runda 3 — krytyczna ocena porad dotyczących optymalizacji**
   - Napisz: wybierz fragment swojego kodu, który wykonuje powtarzalne zadanie lub korzysta z pętli.
   - Zapytaj: poproś o propozycje optymalizacji i wymagaj od SI uzasadnienia jej zaleceń.

     Przykładowe polecenie (prompt):

     > „Wyjaśnij, jak można zoptymalizować ten kod i dlaczego każda proponowana optymalizacja poprawiłaby wydajność lub czytelność. Nie podawaj zoptymalizowanego kodu.”

   - Oceń: czy proponowane optymalizacje są rzeczywiście korzystne? Które z nich są poparte jasnym uzasadnieniem? Które wydają się zbędne lub trudne do uzasadnienia?
   - Popraw: zdecyduj, które zalecenia wdrożyć, i wyjaśnij swoje rozumowanie.

Pytania do refleksji:

- Które podpowiedzi SI były najbardziej przydatne w ulepszeniu twojego kodu?
- Czy SI podała jakieś propozycje, z którymi się nie zgadzasz? Dlaczego?
- Na jakiej podstawie oceniasz, czy dana optymalizacja jest warta wdrożenia?
- Kiedy SI jest najbardziej pomocna w procesie programowania?
- Które zadania programistyczne powinien nadal wykonywać sam uczeń?
- Dlaczego wiedza o programowaniu jest niezbędna do oceny informacji zwrotnej wygenerowanej przez SI?
- Jak uczniowie mogą uniknąć nadmiernego uzależnienia od narzędzi SI podczas nauki programowania?

Pytanie zamykające (do fazy refleksji):

Ile wiedzy o programowaniu potrzeba, aby stwierdzić, czy podpowiedź SI rzeczywiście ulepsza program, i jakie ryzyko powstaje, gdy użytkownicy polegają na poradach wygenerowanych przez SI, sami nie rozumiejąc kodu?

Materia?y

- Jeden laptop na parę uczniów, ze środowiskiem programistycznym
- Dostęp do asystenta czatu SI lub lokalnie zainstalowanego modelu językowego
- Krótkie zadanie programistyczne dopasowane do poziomu doświadczenia uczniów
- Karta pracy zgodna z cyklem napisz → zapytaj → oceń → popraw
- Pytania do refleksji skupione na kompetencjach w zakresie SI i krytycznej ocenie
- Opcjonalne przykłady kodu ilustrujące dobre i złe praktyki programistyczne

---

Revision #1
Created 2026-09-17 21:16:07 UTC by Alexander
Updated 2026-09-17 21:16:08 UTC by Alexander