Wymagania edukacyjne 2TP Pracownia podstaw oprogramowania

Podstawy programowania:

- **Ocena 2**:
    - Uczeń rozumie ogólne pojęcia, takie jak zmienne i proste instrukcje, ale ma trudności z zastosowaniem ich w praktyce.
    - Potrzebuje pomocy nauczyciela na każdym etapie tworzenia prostego programu.
    - Programy są niekompletne lub zawierają liczne błędy.
- **Ocena 3**:
    - Uczeń potrafi napisać prosty program, który zawiera zmienne, instrukcje warunkowe oraz pętle.
    - Program działa poprawnie, choć może zawierać błędy logiczne lub nieefektywne rozwiązania.
    - Uczeń rozumie strukturę prostych programów i potrafi samodzielnie poprawiać błędy, jednak wymaga czasem wsparcia w bardziej złożonych zadaniach.
- **Ocena 4**:
    - Uczeń tworzy poprawne programy z wykorzystaniem pętli, funkcji i podstawowych struktur kontrolnych, takich jak instrukcje warunkowe.
    - Potrafi samodzielnie zaplanować i napisać złożony program, który jest w pełni funkcjonalny i efektywny.
    - Zna i rozumie różne typy danych, a także potrafi wybrać odpowiednie struktury danych do realizacji zadania.
- **Ocena 5**:
    - Uczeń samodzielnie tworzy zaawansowane programy z zastosowaniem klas, obiektów oraz bardziej zaawansowanych elementów programowania obiektowego.
    - Potrafi optymalizować kod pod kątem wydajności oraz dba o jego czytelność.
    - Zna dobre praktyki programowania i stosuje je konsekwentnie w swoich projektach.

Programowanie obiektowe (OOP):

- **Ocena 2**:
    - Uczeń rozumie pojęcia takie jak klasa, obiekt, ale nie potrafi ich samodzielnie zastosować w programie.
    - Potrzebuje pomocy nauczyciela na każdym etapie tworzenia klas i obiektów.
- **Ocena 3**:

    - Potrafi napisać prostą klasę i stworzyć jej obiekt.
    - Uczeń rozumie podstawowe zasady OOP, takie jak enkapsulacja, ale stosuje je w ograniczonym zakresie.
    - Programy z użyciem klas działają, ale często mają ograniczoną funkcjonalność lub zawierają błędy.
- **Ocena 4**:
    - Uczeń potrafi tworzyć klasy, obiekty, dziedziczenie oraz stosować polimorfizm.
    - Programy są poprawnie napisane i potrafią realizować bardziej złożone zadania.
    - Uczeń zna zasady SOLID i próbuje stosować je w swoich projektach.
- **Ocena 5**:
    - Uczeń biegle stosuje zaawansowane techniki OOP, takie jak abstrakcje, interfejsy, wzorce projektowe.
    - Tworzy programy zgodne z najlepszymi praktykami programowania obiektowego.
    - Potrafi pisać złożone aplikacje, które są modularne i łatwe do rozbudowy.

Algorytmy i struktury danych:

- **Ocena 2**:
    - Potrafi opisać prosty algorytm (np. sortowanie) po szczegółowym wyjaśnieniu.
    - Ma trudności z implementacją algorytmów i wymaga pomocy na każdym etapie.
- **Ocena 3**:
    - Uczeń potrafi samodzielnie napisać podstawowy algorytm, taki jak sortowanie bąbelkowe lub wyszukiwanie liniowe.
    - Algorytmy są poprawnie napisane, choć mogą być nieefektywne.
    - Rozumie podstawowe struktury danych, takie jak tablice, listy.
- **Ocena 4**:
    - Uczeń tworzy bardziej złożone algorytmy, potrafi zastosować pętle zagnieżdżone i bardziej zaawansowane struktury danych, np. stosy i kolejki.
    - Potrafi samodzielnie ocenić złożoność algorytmu i optymalizować go pod kątem wydajności.
- **Ocena 5**:
    - Uczeń biegle stosuje zaawansowane algorytmy, takie jak drzewa, grafy, oraz potrafi implementować algorytmy optymalizacyjne.
    - Potrafi analizować złożoność czasową i pamięciową algorytmów.
    - Tworzy efektywne rozwiązania problemów, dbając o wydajność i elegancję kodu.

Debugowanie i testowanie:

- **Ocena 2**:
    - Uczeń potrafi odnaleźć i naprawić proste błędy w kodzie, ale wymaga ciągłej pomocy nauczyciela.
    - Nie stosuje narzędzi do debugowania i testowania kodu samodzielnie.
- **Ocena 3**:
    - Samodzielnie identyfikuje i naprawia podstawowe błędy w programach.
    - Potrafi korzystać z narzędzi do debugowania, choć ogranicza się do podstawowych funkcji.
    - Pisze proste testy jednostkowe dla swoich programów.
- **Ocena 4**:
    - Uczeń diagnozuje problemy w bardziej złożonych programach, korzystając z zaawansowanych funkcji debuggera.
    - Potrafi samodzielnie napisać testy jednostkowe i stosuje je w swoich projektach.
    - Zna narzędzia do automatyzacji testów i zaczyna je stosować.
- **Ocena 5**:
    - Uczeń biegle wykorzystuje narzędzia do debugowania i potrafi diagnozować oraz rozwiązywać złożone problemy w kodzie.
    - Samodzielnie pisze zaawansowane testy automatyczne, zapewniając pełne pokrycie testowe swoich programów.
    - Optymalizuje programy, zarówno pod kątem wydajności, jak i jakości kodu.