Wymagania edukacyjne niezbędne do otrzymania przez ucznia śródrocznej i rocznej oceny klasyfikacyjnej z Podstaw programowania

Klasa 3 technikum

Wymagania edukacyjne niezbędne do otrzymania przez ucznia śródrocznej oceny klasyfikacyjnej

temat lekcji	zagadnienia	Wymagania edukacyjne na poszczególne oceny				
		ocena dopuszczająca uczeń	ocena dostateczna uczeń	ocena dobra uczeń	ocena bardzo dobra uczeń	ocena celująca uczeń
Rozdział 1: Zaawansowane konstrukcje językowe						
Klasy i obiekty w programowaniu obiektowym	**Blueprint, instancja, enkapsulacja.**	Zna definicję klasy i obiektu.	Potrafi stworzyć prostą klasę i obiekt.	Potrafi wykorzystać konstruktory, metody i atrybuty klasy.	Zrozumienie i implementacja enkapsulacji.	Zastosowanie kompozycji i agregacji w projektowaniu klas.
Dziedziczenie i polimorfizm	**Hierarchia, wielokształtność, reużywalność.**	Zna definicje dziedziczenia i polimorfizmu.	Potrafi stworzyć klasy pochodne i korzystać z dziedziczenia.	Implementuje polimorfizm w praktycznych przykładach.	Rozróżnienie między dziedziczeniem a kompozycją.	Zaawansowane wykorzystanie wielokrotnego dziedziczenia (jeśli język to obsługuje).
Wyjątki i obsługa błędów	**Błędy, wyjątki, obsługa.**	Zna podstawowe typy wyjątków.	Potrafi napisać kod, który obsługuje wyjątki przy pomocy try/except.	Zastosowanie różnych rodzajów wyjątków w odpowiednich sytuacjach.	Tworzy własne klasy wyjątków i implementuje zaawansowaną obsługę błędów.	Optymalizuje kod pod kątem obsługi błędów i wyjątków, korzystając z pełnego zestawu narzędzi dostarczonych przez język.

Moduły i pakiety	**Import, organizacja, przestrzeń nazw.**	Zna definicję modułu i pakietu w kontekście programowania.	Potrafi zaimportować moduł i skorzystać z dostarczonych w nim funkcji lub klasy.	Zastosowanie różnych technik importowania (np. import całościowy, selektywny) oraz rozumienie organizacji modułów w pakietach.	Potrafi tworzyć własne moduły i pakiety, oraz zarządzać zależnościami w projekcie.	Demonstruje głęboką znajomość mechanizmów importowania, zrozumienie ścieżek i lokalizacji modułów, oraz optymalizacja struktury projektu za pomocą modułów i pakietów.
Rozdział 2: Struktury danych						
Listy, krotki, słowniki i zbiory w Pythonie	**Kolekcje, mutowalność, klucze.**	Rozumie podstawowe definicje i zna różnice między wymienionymi strukturami danych.	Potrafi tworzyć i modyfikować podstawowe struktury danych, takie jak listy czy słowniki.	Zna i stosuje bardziej zaawansowane operacje na strukturach danych, takie jak list comprehensions czy metody specyficzne dla danego typu kolekcji.	Optymalizuje wydajność operacji na strukturach danych, np. poprzez wykorzystanie odpowiednich metod lub wybór odpowiedniej struktury dla danego zadania.	Demonstruje głębokie zrozumienie wewnętrznej organizacji i mechanizmów działania wybranych struktur danych oraz potrafi zastosować je w zaawansowanych projektach.
Algorytmy sortowania i wyszukiwania	**Porządkowanie, przeszukiwanie, efektywność.**	Zna podstawowe algorytmy sortowania (np. bubble sort) i wyszukiwania (np. wyszukiwanie	Implementuje i stosuje podstawowe algorytmy sortowania i wyszukiwania.	Zna i potrafi zaimplementować bardziej zaawansowane algorytmy, takie jak quicksort czy binary search.	Analizuje złożoność obliczeniową różnych algorytmów i potrafi wybrać optymalny	Optymalizuje i dostosowuje algorytmy do specyficznych wymagań, demonstrując głęboką wiedzę w

		liniowe).			algorytm dla danego problemu.	dziedzinie algorytmiki.
Stosy i kolejki	**LIFO, FIFO, priorytet.**	Zna definicje stosu i kolejki oraz potrafi wymienić ich podstawowe zastosowania.	Implementuje i operuje na prostych stosach i kolejkach.	Zastosowanie stosów i kolejek w bardziej skomplikowanych problemach, takich jak odwrotna notacja polska czy algorytm BFS.	Zna różne typy kolejek (np. kolejki priorytetowe) i potrafi je zaimplementować oraz stosować w odpowiednich sytuacjach.	Demonstruje głębokie zrozumienie i optymalizację operacji na stosach i kolej
Drzewa i grafy	**Węzły, krawędzie, połączenia.**	Zna podstawowe definicje drzew i grafów.	Tworzy i przeszukuje proste drzewa (np. binarne) oraz grafy.	Implementuje zaawansowane operacje na drzewach (np. balansowanie) oraz algorytmy grafowe (np. Dijkstra).	Analizuje złożoność operacji na drzewach i grafach, stosując optymalne strategie przeszukiwania i modyfikacji.	Projektuje i optymalizuje zaawansowane struktury danych bazujące na drzewach i grafach, demonstrując głęboką wiedzę w tej dziedzinie.
Rozdział 3: Zaawansowane techniki programowania						
Programowanie funkcyjne	**Czystość, lambda, mapowanie.**	Zna definicję programowania funkcyjnego oraz podstawowe cechy tego paradygmatu.	Potrafi korzystać z podstawowych funkcji wyższego rzędu, takich jak map, filter i reduce.	Stosuje bardziej zaawansowane konstrukcje programowania funkcyjnego, takie jak zagnieżdżone funkcje, domknięcia i curryfikacja.	Demonstruje zrozumienie konceptów takich jak monady, funktory i leniwe obliczenia (w zależności od języka).	Tworzy zoptymalizowane i czysto funkcyjne rozwiązania dla skomplikowanych problemów.

Wzorzec projektowy Singleton	**Unikalność, instancja, inicjalizacja.**	Zna definicję wzorca Singleton i potrafi wymienić jego podstawowe zastosowania.	Implementuje prosty wzorzec Singleton.	Zastosowanie wzorca Singleton w praktycznych projektach z uwzględnieniem wątków i synchronizacji.	Analizuje wady i zalety wzorca Singleton, zna alternatywne wzorce i podejścia.	Demonstruje zaawansowane zastosowania i modyfikacje wzorca Singleton w różnorodnych kontekstach aplikacji.
Wielowątkowość i współbieżność	**Wątki, procesy, równoczesność.**	Zna różnicę między wielowątkowością a współbieżnością.	Potrafi tworzyć podstawowe programy wielowątkowe i rozumie pojęcie blokady (lock).	Implementuje bardziej zaawansowane techniki współbieżności, takie jak semafory, barierki czy warunki.	Optymalizuje aplikacje wielowątkowe, unikając zakleszczeń i problemów związanych z dostępem do zasobów.	Projektuje i wdraża zaawansowane systemy oparte na wielowątkowości i współbieżności, z uwzględnieniem skalowalności i wydajności.
Generatory i wyrażenia generatorowe	**Leniwość, strumienie, yield.**	Zna podstawy działania generatorów i cel ich stosowania.	Potrafi tworzyć proste generatory za pomocą słowa kluczowego yield.	Korzysta z wyrażeń generatorowych do tworzenia efektywnych rozwiązań dla dużych kolekcji danych.	Optymalizuje użycie generatorów w kontekście zużycia pamięci i wydajności obliczeń.	Demonstruje zaawansowane zastosowania generatorów w złożonych projektach, łącząc je z innymi technikami programowania.
Wymagania edukacyjne niezbędne do otrzymania przez ucznia rocznej oceny klasyfikacyjnej (obejmują wymagania edukacyjne niezbędne do otrzymania przez ucznia śródrocznej oceny klasyfikacyjnej).						
Rozdział 4: Dostęp do bazy danych						
Podstawy baz danych SQL	**Tabele, zapytania, relacje.**	Zna podstawowe pojęcia związane z bazami danych, takie jak tabela,	Potrafi wykonać proste zapytania SQL, takie jak SELECT, INSERT i	Zastosowanie bardziej zaawansowanych zapytań,	Zrozumienie i implementacja relacji między tabelami, a także	Projektuje i wdraża skomplikowane schematy baz

		rekord i kolumna.	DELETE.	takich jak JOIN, GROUP BY i subzapytania.	optymalizacja zapytań.	danych oraz demonstruje zaawansowane techniki optymalizacji.
Połączenie z bazą danych	**Sterowniki, sesje, połączenie.**	Zna podstawowe informacje potrzebne do nawiązania połączenia z bazą danych.	Potrafi nawiązać połączenie z bazą danych za pomocą odpowiedniego sterownika lub biblioteki w wybranym języku programowania.	Implementuje operacje CRUD (Create, Read, Update, Delete) w aplikacji.	Zastosowanie technik takich jak transakcje i procedury składowane.	Optymalizuje połączenie i interakcje z bazą danych, gwarantując bezpieczeństwo i wydajność.
Operacje CRUD (Create, Read, Update, Delete)	**Zapis, odczyt, modyfikacja, usuwanie.**	Zna definicje operacji CRUD.	Wykonuje podstawowe operacje CRUD na danych w bazie.	Stosuje bardziej zaawansowane funkcje związane z operacjami CRUD, takie jak paginacja czy filtrowanie.	Integruje operacje CRUD z innymi funkcjonalnościam i aplikacji, takimi jak autoryzacja czy logowanie.	Optymalizuje i skaluje operacje CRUD w dużych bazach danych, dbając o spójność i integralność danych.
ORM (Object-Relational Mapping)	**Mapowanie, encje, relacje.**	Zna definicję ORM i podstawowe zalety jego stosowania.	Korzysta z prostego ORM do mapowania obiektów na rekordy bazy danych.	Implementuje zaawansowane funkcje ORM, takie jak relacje między encjami czy dziedziczenie.	Zastosowanie technik optymalizacji w kontekście ORM, takich jak leniwe ładowanie czy buforowanie.	Projektuje i wdraża skomplikowane systemy oparte na ORM, balansując między wydajnością a elastycznością.
Rozdział 5: Narzędzia i praktyki						

Debugowanie kodu	**Breakpoint, śledzenie, naprawa.**	Zna podstawowe techniki debugowania, takie jak stosowanie instrukcji wydruku do śledzenia błędów.	Używa wbudowanych narzędzi debugowania w środowisku programistycznym (np. breakpointy, krokowe wykonywanie kodu).	Analizuje i rozwiązuje bardziej skomplikowane błędy, stosując techniki takie jak profilowanie kodu.	Zastosowanie zaawansowanych technik debugowania, takich jak debugowanie zdalne.	Mistrzowska umiejętność wykrywania i naprawiania błędów w różnorodnych scenariuszach i technologiach.
Testy jednostkowe	**Asercje, mocki, pokrycie.**	Rozumie znaczenie testowania kodu i zna podstawowe pojęcia związane z testami jednostkowymi.	Tworzy proste testy jednostkowe dla funkcji lub metod.	Implementuje zestaw dobrze zorganizowanych testów jednostkowych, korzystając z frameworków testowych.	Stosuje techniki takie jak mockowanie i TDD (Test-Driven Development).	Osiąga pełne pokrycie kodu testami, gwarantując wysoką jakość i niezawodność oprogramowania.
Kontrola wersji z git	**Commit, gałęzie, merge.**	Zna podstawowe pojęcia kontroli wersji, takie jak commit, branch czy merge.	Regularnie korzysta z podstawowych funkcji git, takich jak git add, git commit i git push.	Zarządza wieloma gałęziami, rozwiązuje konflikty i korzysta z funkcji takich jak rebase.	Efektywnie korzysta z zaawansowanych funkcji git, takich jak cherry-pick, bisect czy stash.	Mistrzowska znajomość git, w tym tworzenie i zarządzanie dużymi repozytoriami, oraz stosowanie najlepszych praktyk w kontroli wersji.
Wprowadzenie do ciągłej integracji i ciągłej dostawy	**Automatyzacja, wdrożenie, cykl życia.**	Rozumie podstawowe pojęcia związane z CI/CD.	Konfiguruje proste środowisko CI do automatycznego budowania i testowania kodu.	Ustawia i zarządza kompletnym łańcuchem narzędzi CI/CD, włączając w to automatyczne	Optymalizuje procesy CI/CD, uwzględniając różne środowiska (np. dev, staging, production) i technologie.	Mistrzowska znajomość i implementacja zaawansowanych technik CI/CD, takich jak blue-green

				wdrażanie.		deployments, canary releases i inne.

UWAGI:

1. Ocenę wyższą otrzymuje uczeń spełniający łącznie wymagania edukacyjne określone dla ocen niższych np. ocenę dobrą otrzymuje uczeń spełniający wymagania edukacyjne na oceną dopuszczającą, dostateczną oraz dobrą. W ramach przedmiotu: “Programowania strukturalnego i obiektowego” uczeń powinien wykazać się umiejętnością wykorzystania wiedzy teoretycznej w praktyce.
2. Ocenę niedostateczną otrzymuje uczeń, który nie spełnia wymagań na poszczególne pozytywne oceny.
3. W przypadku nie zrealizowania tematów lekcji (zagadnień) w I okresie będą one realizowane po klasyfikacji śródrocznej. W tym przypadku obowiązują również wymagania edukacyjne dla tych tematów (zagadnień).