Uniwersytet Mikołaja Kopernika w Toruniu
Wydział Matematyki i Informatyki

Alicja Kluczek
nr albumu: 324587
kierunek: informatyka

Praca magisterska

Nauka programowania dla klas 7-8 zintegrowana z platformą Szkopuł.

Opiekun pracy dyplomowej
prof. dr hab. Anna Beata Kwiatkowska

Toruń 2025

Spis treści

Wstęp **5**

1. Z czego składa się kurs? Filary programu. **7**

2. Automatyczny system sprawdzający - Szkopuł **11**

Automatyczny system sprawdzający - Szkopuł **11**

Bibliografia **15**

Wstęp

Współczesna edukacja informatyczna w szkołach podstawowych powinna rozwijać praktyczne umiejętności uczniów i przygotowywać ich do świadomego korzystania z technologii. Celem niniejszej pracy magisterskiej jest stworzenie kursu programowania dla klas 7–8, zintegrowanego z platformą Szkopuł, obejmującego 24 jednostek lekcyjnych, zestawy zadań i quizów, a także materiały ewaluacyjne – w tym klasówki i projekty zespołowe. Kurs jest zgodny z podstawą programową i może być wykorzystywany w ramach nauczania w szkole.
Oprócz części dydaktycznej praca obejmuje również rozwój funkcjonalności platformy – wdrożenie modułu Omówienia, który umożliwi dodawanie i kontrolę dostępu do komentarzy do zadań. Całość została zaprojektowana z myślą o praktycznym podejściu do nauczania informatyki i może zostać rozszerzona o elementy sztucznej inteligencji, np. w projektach semestralnych.

Rezultatem pracy będzie:

- Działający kurs opublikowany na Szkopuł Kursy,
- Komplet konspektów i materiałów do każdej opisanej lekcji,
- Nowy komponent „Omówienia” platformy Szkopuł.

Rozdział 1.

Z czego składa się kurs? Filary programu.

Program rozpoczyna się od nauki podstaw programowania, tak aby uczniowie mogli rozwijać umiejętność myślenia komputacyjnego na dalszych etapach kursu. Każda jednostka dydaktyczna składa się z części teoretycznej (w tym elementów „unplugged") oraz zadań praktycznych realizowanych przy użyciu języka Python.

Ćwiczenia informatyki „unplugged" są według nowoczesnych badań co najmniej tak efektywne, jak standardowe metody nauki [**?**].

Plan zajęć obejmuje 28 godzin lekcyjnych; podana liczba godzin ma charakter orientacyjny i może być dostosowywana w zależności od potrzeb grupy.

Myślenie algorytmiczne, podstawy programowania (12 godzin)

Moduł ten koncentruje się na wprowadzeniu uczniów w abstrakcyjne pojęcia programistyczne, takie jak zmienne, instrukcje warunkowe, pętle oraz funkcje. Zajęcia te przeplatane są mini-projektami, które umożliwiają uczniom praktyczne zastosowanie zdobytej wiedzy oraz wyrażenie własnej kreatywności.

Takie podejście zwiększa szanse na zaangażowanie uczniów, którzy niekoniecznie wykazują naturalne predyspozycje matematyczne. Abstrakcyjne koncepty nadal odgrywają istotną rolę, ponieważ rozwijają zdolność adaptacji do dynamicznie zmieniających się technologii informatycznych [**?**].

Badania wskazują również, że dziewczęta częściej interesują się praktycznym zastosowaniem technologii niż jej konstrukcją techniczną [**?**]. Włączenie projektowych i aplikacyjnych elementów do zajęć może więc pozytywnie wpłynąć na ich aktywność i zaangażowanie w naukę informatyki.

1. Podstawy Pythona – zmienne, operatory, `print()`, `input()`. Po co programować?

2. Myślenie komputacyjne od kuchni – specyfikacja, algorytm. Co to znaczy rozwiązać problem?

3. Instrukcje warunkowe – `if`, `elif`, `else`. Jak kierować zachowaniem komputera?

4. Pętle `for` i `while` – iteracje, przerwania (`break`, `continue`). Jak sobie poradzić z powtarzalną pracą? [**podwójna godzina**]

5. Zastosowanie praktyczne: Quiz. Jak użyć tego co umiem? [**podwójna godzina**]

6. Funkcje w Pythonie – argumenty, wartości zwracane, funkcje wbudowane.

7. Matematyka z pomocą Pythona – Jak napisać program rozwiązujący moją pracę domową?

8. Lista, słownik – Kiedy zmienna nie wystarcza.

9. Zastosowanie praktyczne – Sterowanie „robotem". Biblioteka `turtle`. [**podwójna godzina**]

Kryptografia (4 godziny)

Choć programowanie stanowi kluczowy element wprowadzający do informatyki, nie wyczerpuje całego spektrum tej dziedziny. Włączenie krótkich modułów tematycznych, takich jak kryptografia klasyczna, pozwala uczniom lepiej zrozumieć szerokość informatyki jako nauki. Zgodnie z podejściem stosowanym w szkołach średnich w Szwajcarii [**?**], kryptografia może być skutecznym sposobem rozwijania umiejętności analitycznego myślenia oraz krytycznego podejścia do informacji – nawet bez wcześniejszej wiedzy z zakresu informatyki. Szczególną rolę odgrywa tu motywacja uczniów: element rywalizacji między twórcami a łamaczami kodów stanowi silny czynnik angażujący. W badaniach nad tego typu modułami wykazano, że aż 70% uczniów wykazało chęć dalszego zgłębiania tematu, co czyni z kryptografii znakomity punkt wyjścia do dalszych zajęć.

1. Podstawy kryptografii – Poznajemy szyfr Cezara, `rot13`. Dlaczego są słabym zabezpieczeniem?

2. Kody ASCII – zmiana liter w liczbę.

3. Implementacja wybranej metody szyfrowania – podstawieniowej, przesunięć, *leet speak*, Rozier. [**podwójna godzina**]

Algorytmika z Pythonem (8 godzin)

W tej części kursu uczniowie zapoznają się z podstawowymi algorytmami zawartymi w polskiej podstawie programowej dla klas 7–8. Analiza algorytmów odbywa się zarówno na poziomie koncepcyjnym (na kartce), jak i implementacyjnym (w języku Python).

Ponieważ algorytmika jest dziedziną o wysokim stopniu abstrakcji, szczególną uwagę poświęca się wizualizacji działania omawianych algorytmów. W procesie nauczania wykorzystywane są elementy gier, zabaw oraz komputerowe narzędzia wizualizacyjne, które są kluczowe by uczniowie mogli dobrze zrozumieć mechanizmy stojące za poszczególnymi rozwiązaniami [**?**].

1. Własności liczbowe – podzielność, suma cyfr, iloczyn cyfr. [**podwójna godzina**]
2. Kod binarny, szesnastkowy.
3. Algorytm Euklidesa – zastosowanie.
4. Wyszukiwanie w zbiorze – Liniowo znajdujemy minimum, maksimum, k-ty element. [**podwójna godzina**]
5. Porządkowanie zbioru – sortowanie przez wybieranie. [**podwójna godzina**]

Wstęp do AI (4 godziny)

Ostatnia część kursu to odpowiedź na rosnącą potrzebę wdrożenia uczniów w tematykę sztucznej inteligencji. W związku z dynamicznym rozwojem tej dziedziny, zajęcia skupiają się na zbudowaniu prostego modelu i uczuleniu uczniów na typowy problem uczenia maszynowego (Garbage In - Garbage Out).

1. Czym jest sztuczna inteligencja? Implementujemy uproszczone ”drzewo decyzyjne”.
2. Moduł scikit-learn i sprytniejsze drzewa decyzyjne.
3. Trenowanie własnego modelu i eksploracja danych. [**podwójna godzina**]

Rozdział 2.

Automatyczny system sprawdzający - Szkopuł

Platforma Szkopuł[1] jest autorskim systemem automatycznego sprawdzania zadań programistycznych, stworzonym i rozwijanym przez zespół Uniwersytetu Warszawskiego. System jest wykorzystywany w edukacji informatycznej na różnych poziomach nauczania, od szkół podstawowych po studia wyższe. Jest również podstawowym systemem weryfikującym rozwiązania uczestników Olimpiady Informatycznej, najbardziej prestiżowego konkursu programistycznego dla uczniów szkół średnich w Polsce.

Ze względu na istniejące zaplecze techniczne i doświadczenie zespołu Szkopuł stanowi doskonałą platformę do wdrożenia kursu programowania dla klas 7-8. System jest oparty na technologii webowej (Django, Python) i umożliwia łatwe tworzenie oraz zarządzanie zadaniami i użytkownikami. Ponadto, Szkopuł posiada zintegrowaną stronę "Szkopuł Kursy", która umożliwia publikację gotowych kursów edukacyjnych, co idealnie wpisuje się w cele niniejszej pracy magisterskiej.

Działanie systemów automatycznego sprawdzania

Użytkownik po zalogowaniu do platformy otrzymuje dostęp do zbioru zadań programistycznych i ma możliwość przesyłania swoich rozwiązań. Każde zadanie jest wyposażone w zestaw przypadków testowych jawnych (widocznych dla użytkownika) oraz ukrytych (niewidocznych dla użytkownika).

Przypadki testowe jawne służą do wstępnej weryfikacji poprawności rozwiązania, są z reguły proste i mają na celu pomoc użytkownikowi w zrozumieniu treści zadania. Dodatkowo, stanowią podstawę do testowania rozwiązań podczas procesu nauki i debugowania kodu, kiedy uczniowie nie są jeszcze doświadczeni w samodzielnym tworzeniu kompleksowych testów. Zwyczajowo przypadki jawne są umieszczane na początku opisu zadania, aby użytkownik mógł je łatwo znaleźć i wykorzystać podczas pracy nad rozwiązaniem.

[1] `https://szkopul.edu.pl`

Przypadki testowe ukryte służą do oceny poprawności i efektywności rozwiązania, zapewniając, że program działa zgodnie z wymaganiami zadania. Są one zazwyczaj bardziej złożone i obejmują różnorodne scenariusze, w tym przypadki brzegowe. Przypadków ukrytych nie udostępnia się uczniowi, ale nauczyciel ma do nich dostęp.

Po nadesłaniu kodu źródłowego, system kompiluje i uruchamia program w bezpiecznym środowisku izolowanym (tzw. piaskownicy). Następnie, program jest uruchamiany na kolejnych przypadkach testowych, a wyniki (zapisane w standardowym wyjściu) są porównywane z oczekiwanymi wynikami.

W przypadku niepowodzenia któregokolwiek z przypadków, użytkownik otrzymuje informację zwrotną:

- "Zła odpowiedź"(Wrong Answer) - gdy wynik programu różni się od oczekiwanego. Nauczyciel może zdecydować, czy ujawnić użytkownikowi szczegóły dotyczące nieudanego testu, na przykład oczekiwanego wyniki.
- "Przekroczenie limitu czasu"(Time Limit Exceeded) - gdy program nie zakończył działania w określonym czasie,
- "Przekroczenie limitu pamięci"(Memory Limit Exceeded) - gdy program przekroczył przydzielony limit pamięci,
- "Błąd wykonania"(Runtime Error) - gdy program zakończył się z powodu błędu podczas wykonywania, np. dzielenie przez zero.

Jeśli program zakończy działanie i zwróci poprawny wynik, użytkownik otrzymuje informację o sukcesie.

Przypadki testowe mogą być grupowane w zestawy testów, co pozwala na bardziej szczegółową ocenę rozwiązania. Użytkownik otrzymuje punkty za każdy zestaw testów, który jego rozwiązanie przejdzie pomyślnie.

Zwyczajowo, poprawne rozwiązanie otrzymuje 100% punktów, jednak nauczyciel może dostosować system punktacji według własnych potrzeb. Dla przykładu, rozwiązanie "wolne"(działające poprawnie, ale przekraczające limit czasu) może otrzymać 50% punktów, poprzez ustawienie odpowiednich wag dla poszczególnych zestawów testów.

Zalety platform automatycznego sprawdzania

Platformy automatycznego sprawdzania, takie jak Szkopuł, oferują szereg korzyści zarówno dla uczniów, jak i nauczycieli:

- Natychmiastowa informacja zwrotna: Uczniowie otrzymują szybkie informacje o poprawności swoich rozwiązań, co umożliwia im szybkie uczenie się na błędach i poprawę swoich umiejętności programistycznych
- Standaryzacja oceny: Automatyczne systemy zapewniają spójność i obiektywność w ocenie rozwiązań, eliminując subiektywne błędy ludzkie.

- Skalowalność: Systemy te mogą obsługiwać dużą liczbę użytkowników jednocześnie, co jest szczególnie przydatne w przypadku dużych klas lub kursów online.
- Różnorodność zadań: Szkopuł posiada bogatą (na moment pisania pracy, ponad 3000 zadań) bazę zadań, co pozwala na szeroki zakres tematyki i poziomu trudności.

Stos technologiczny platformy Szkopuł

Szkopuł (serwis internetowy) jest zbudowany w oparcu o technologię webową Django (Python). Do przechowywania danych wykorzystywana jest relacyjna baza danych PostgreSQL. Aplikacja jest hostowana na serwerach uczelni i korzysta z kontenerów Docker do izolacji środowisk uruchomieniowych dla przesyłanych rozwiązań. System obsługuje wiele języków programowania, w tym Python, C++, Java i Rust, co pozwala na elastyczność w tworzeniu zadań i dostosowanie do różnych potrzeb edukacyjnych. Należy jednak pamiętać, że limity czasowe i pamięci mogą się różnić w zależności od wybranego języka programowania.

Szkopuł Kursy to rozszerzenie platformy Szkopuł, które umożliwia tworzenie i publikację kompletnych kursów edukacyjnych. Serwis jest zbudowany w oparcu o framework MkDocs, który pozwala na łatwe tworzenie i zarządzanie treściami kursów w formacie Markdown.

Bibliografia

[1] On plugging ”unplugged”into CS classes, 2013

[2] Are Females Disinclined to Tinker in Computer Science? 2015

[3] L'enseignement de l'Informatique en France: Il est urgent de ne plus attendre, 2013

[4] A short introduction to classical cryptology as a way to motivate high school students for informatics, 2011

[5] Exploring the role of visualization and engagement in computer science education