OD PROGRAMOWANIA WIZUALNEGO DO TEKSTOWEGO

Maciej Borowiecki, Krzysztof Chechłacz
Ośrodek Edukacji Informatycznej i Zastosowań Komputerów
02-006 Warszawa, ul. Nowogrodzka 73
maciej.borowiecki@oeiizk.waw.pl, kch@oeiizk.waw.pl

Abstract. In this paper we discuss some problems of teaching programming. We will show four types of programming languages, and how to easy transform from one of them to the other. The focus is on the methods for transforming from visual programming languages to the text ones. In our opinion there is a need to start teaching text languages in older primary school classes. We will also show that the most important is to solve the problem by creating the right algorithm, implementing of solving is a more technical skill.

1. Wstęp

Nauczanie programowania we wczesnych etapach edukacyjnych zwykle rozpoczynamy od programowania wizualnego. Podczas tworzenia projektu unikamy w ten sposób problemów związanych z zawiłościami składni języka programowania. Gdy próbujemy połączyć elementy, które do siebie nie pasują, mechanizm bloczków nie pozwala nam na to. Ponadto szybko uzyskujemy efekt w postaci działającego projektu i w ten sposób zachęcamy uczniów do kolejnych aktywności.

Środowiska wizualne dostarczają zwykle dużą liczbę bloczków, jednak są one pogrupowane na kategorie i nie są widoczne wszystkie jednocześnie na ekranie. Uczeń rozwiązując bardziej skomplikowane zadania musi wykonać wiele operacji myszką, by wyszukać potrzebne bloczki i je użyć. Im więcej skomplikowanych zadań rozwiązujemy, tym łatwiej jest nam znaleźć potrzebne elementy (bo zapamiętujemy ich lokalizację), ale straty czasu wynikające z konieczności znalezienia i osadzenia w przestrzeni roboczej żądanego elementu stają się znaczące.

Pisząc program w środowisku tekstowym musimy znać składnię języka i sprawnie posługiwać się klawiaturą. Jest to podstawowym źródłem problemów, z jakimi spotykają osoby stawiające pierwsze kroki w programowaniu tekstowym. W sposób naturalny pojawiają się błędy składniowe. Wielu uczniów zniechęca się i wyraża chęć powrotu do programowania wizualnego. Często nie pomagają argumenty, że

programując tekstowo mamy większy wybór narzędzi (środowisk, języków mających większe możliwości) i choćby dlatego warto się tym zainteresować. W dalszej części pokazujemy propozycje przejścia od programowania wizualnego do tekstowego w sposób łagodny i akceptowalny dla uczniów. Co więcej, w ten sposób pokazujemy, że najważniejszy jest sam algorytm rozwiązania postawionego zadania, a sposób zapisu jest mniej istotny.

Wykorzystamy narzędzia, które umożliwiają napisanie programu w pewnym języku, a następnie automatyczne przełożenie kodu na inny język.

2. Algorytm Euklidesa

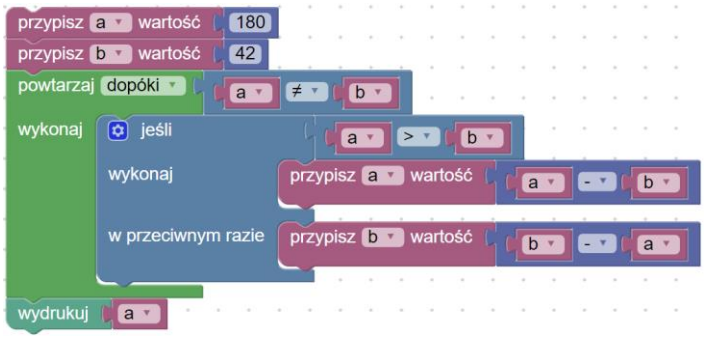
W nowej podstawie programowej znajdujemy wiele odwołań do różnych algorytmów. Niektóre z nich są nawet wymienione z nazwy. Przykładowo, weźmy algorytm Euklidesa. Umożliwia on wyliczenie największego wspólnego dzielnika dwóch liczb naturalnych. W najprostszej wersji polega on na tym, że póki liczby są różne, od większej z nich odejmujemy mniejszą. Gdy są równe, oznacza to, że otrzymaliśmy wynik.

Rozwiązanie należy zapisać w języku programowania. Na razie nie zastanawiamy się nad kwestią eleganckiego przekazania danych do programu, wrócimy do tego później. Przykładowe rozwiązanie, dla liczb 180 i 42, w najbardziej popularnym w Polsce środowisku programowania wizualnego – *Scratch* – może wyglądać następująco:

Rysunek 1 Algorytm Euklidesa w Scratchu

Podobnie będzie ono wyglądać w środowisku *Blockly* dostępnym na stronie *https://blockly-demo.appspot.com/static/demos/code/index.html*. Środowisko to jest jednym z demonstracyjnych projektów inicjatywy *Google Blockly*, w której powstała m.in. szeroko znana *Godzina kodowania*. Podobnie jak w *Scratchu*, możemy uży-

wać języka polskiego, mamy elementy – bloczki, które możemy łączyć tworząc własny projekt, a następnie taki projekt zapisać i uruchomić.



Rysunek 2 Algorytm Euklidesa w Google Blockly

W obu przypadkach otrzymujemy wynik 6. W istocie dzielnikami 180 są 1, 2, 3, 4, 5, 6, 9, 10, 12, 15, 18, 20, 30, 36, 45, 60, 90 oraz 180, dzielnikami 42 są 1, 2, 3, 6, 7, 14, 21 oraz 42 – liczba 6 jest największą, która występuje w obu przypadkach.

W *Blockly* mamy możliwość obejrzenia, jak wyglądałoby rozwiązanie zapisane w kilku innych językach. Wystarczy wybrać jedną z zakładek:

Rysunek 3 Wybór języka docelowego

Kilka ostatnich zakładek daje możliwość obejrzenia kodu w dość rzadko używanych językach lub kod jest mało czytelny. Zajmiemy się językami z dwóch pierwszych zakładek: *Python* i *JavaScript*.

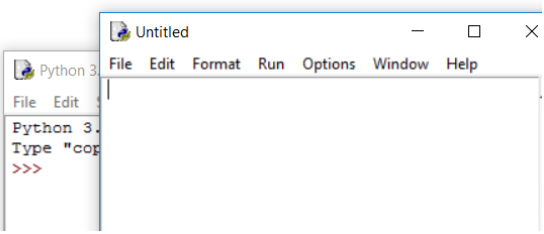
3. Python

Kod w języku *Python* jest krótki i czytelny, nie pozostawia wątpliwości co do sposobu działania; poniżej z pominiętymi dwoma pierwszymi wierszami (warto przedyskutować z uczniami dlaczego tak jest).

```
a = 180
b = 42
while a != b:
  if a > b:
    a = a - b
  else:
    b = b - a
print(a)
```

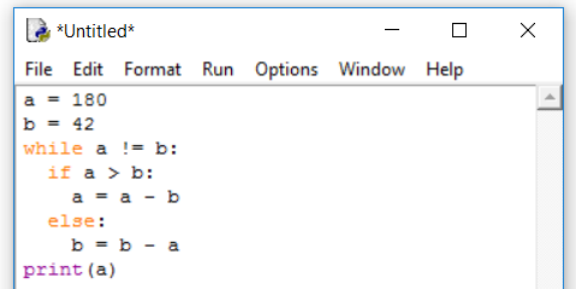
Rysunek 4 Kod w Pythonie wygenerowany automatycznie

Instalujemy wersję 3.* języka *Python* – do pobrania z *http://www.python.org* i wywołujemy IDLE, w którym otwieramy nowe (puste) okno:



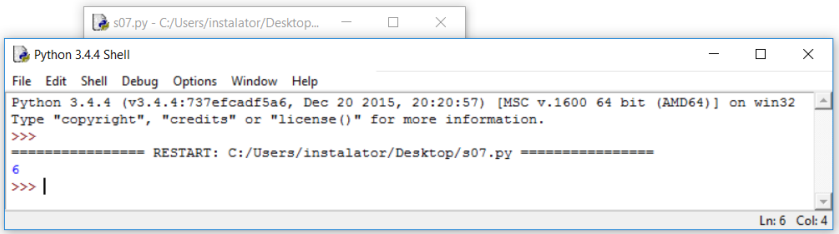
Rysunek 5 Okna w Pythonie

W tym oknie wklejamy uprzednio uzyskany kod.



Rysunek 6 Kod źródłowy w Pythonie

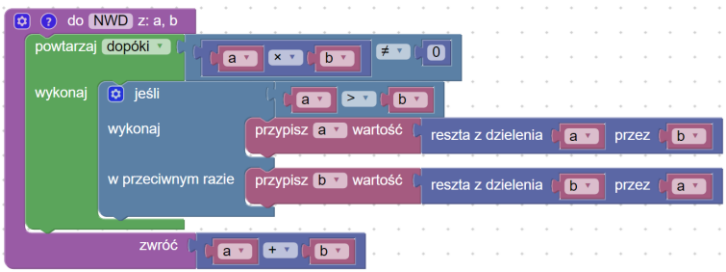
Od razu widać, jak elegancko *Python* koloruje składnię – dzięki temu wiemy, że słowa kluczowe wpisane zostały poprawnie. Program można uruchomić – wystarczy wybrać *Run Module (F5)* z menu *Run*. Zanim program zostanie uruchomiony, środowisko poprosi o jego zapisanie na dysku (warto przedyskutować z uczniami zalety i wady takiego rozwiązania). Po zapisaniu na dysku i translacji, program zostanie uruchomiony i wypisze wynik w głównym oknie.



Rysunek 7 Po uruchomieniu programu w Pythonie

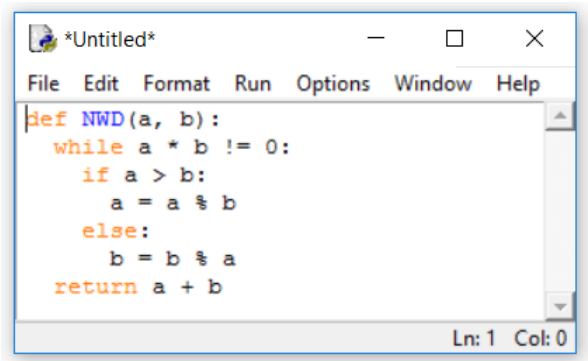
Tak więc nie znając *Pythona* udało się nam „napisać" i uruchomić program w tym języku. W sposób oczywisty, bez większej trudności, uczniowie poznali składnię najważniejszych instrukcji – przypisania wartości, warunkowej, pętli, a także zobaczyli sposób tworzenia wyrażeń i warunków logicznych. Warto w tym miejscu przeprowadzić dyskusję na temat znaczenia wcięć stosowanych w kodzie programu – *Python* wymusza ich stosowanie. Jest to sytuacja korzystna i nie rodzi dyskusji, bo to nie nauczyciel poleca stosowanie wcięć, ale jest to formalny wymóg składni języka. Uczniowie uczą się na podstawie przykładu i dzięki temu ich wiedza będzie trwalsza i użyteczniejsza.

W dalszej części zajęć z uczniami można rozbudowywać program. Na przykład korzystając z *Google Blockly* budujemy funkcję znajdującą największy wspólny dzielnik, a sam algorytm Euklidesa modyfikujemy tak, by zamiast odejmowania używał reszty z dzielenia. Budowanie funkcji jest możliwe w *Google Blockly*, natomiast *Scratch* nie dysponuje takimi możliwościami.



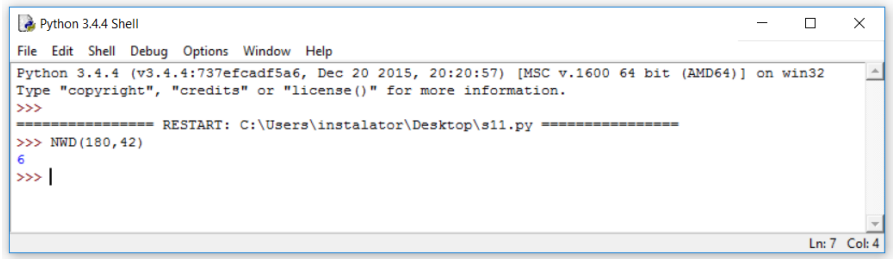
Rysunek 8 Kod funkcji w Google Blockly

Kod otrzymany w *Pythonie* będzie zawierał nowe informacje – o tym, jak wygląda struktura funkcji i jak zapisywać resztę z dzielenia.



Rysunek 9 Kod funkcji automatycznie wygenerowany i przeniesiony do Pythona

Wywołanie funkcji wymaga użycia nazewnika funkcji i podania parametrów wywołania. Efekt jest identyczny, jak poprzednio.



Rysunek 10 Wywołanie funkcji

Warto dać uczniom do wykonania inne zadanie polegające na zapisie pewnego algorytmu w wybranym przez siebie języku, np. algorytmu wyznaczenia dzielników zadanej liczby naturalnej bądź wyodrębnienia cyfr danej liczby. Praktyka pokazała, że mając do wyboru *Scratch* i *Python*, około połowy uczniów wybierała *Python*. Argumentowali to prostotą zapisu i stosunkowo niewielką ilością czynności do wykonania.

4. JavaScript

Spróbujmy teraz skorzystać z kodu *JavaScript*, który został wygenerowany z użyciem *Google Blockly* dla algorytmu Euklidesa:

Bloki | JavaScript

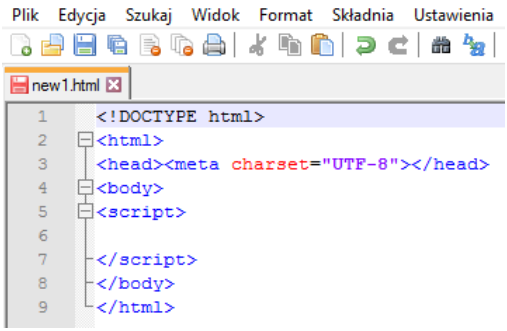
```
var a, b;


a = 180;
b = 42;
while (a != b) {
  if (a > b) {
    a = a - b;
  } else {
    b = b - a;
  }
}
window.alert(a);
```

Rysunek 11 Kod w JavaScript wygenerowany automatycznie

Najprościej będzie kod *JavaScript* zanurzyć w dokumencie *html*, a następnie tak utworzony dokument przekazać przeglądarce internetowej do zinterpretowania. Dla utworzenia dokumentu *html* możemy użyć programu *Notatnik++* z możliwością bezpośredniego przejścia – uruchomienia kodu *html* w przeglądarce internetowej.

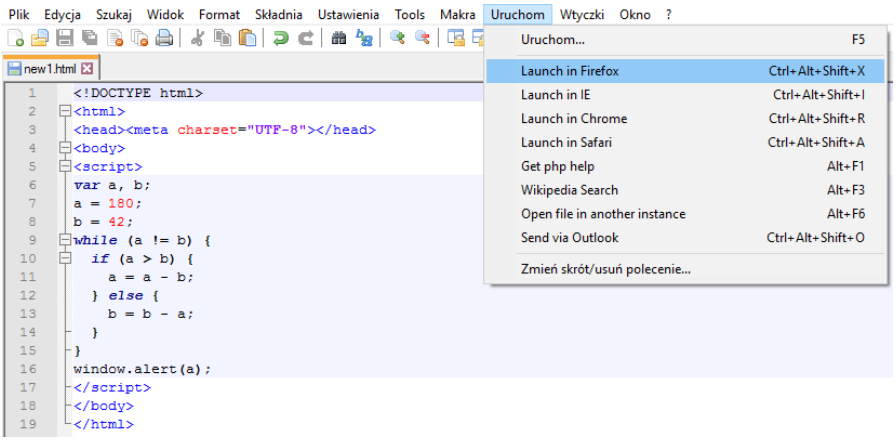
Można też użyć systemowego notatnika, ale wówczas niektóre czynności będą bardziej złożone.

W pierwszym kroku tworzymy szablon opisu strony internetowej. Utworzony skrypt będzie działać także wówczas, gdy ograniczymy się do piątego i siódmego wiersza, ale nie polecamy takiego rozwiązania, ponieważ tworzy to złe intuicje co do budowy plików opisujących strony internetowe.



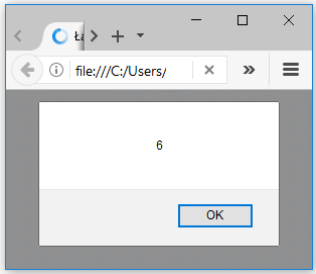
Rysunek 12 Szablon opisu strony internetowej

W miejscu pustego, szóstego wiersza umieszczamy kod z *Google Blockly*. Dokument zapisujemy (to ważne, bo program edycyjny komunikuje się z przeglądarką używając wersji pliku *html* zapisanej na dysku) i przekazujemy przeglądarce do zinterpretowania wybierając *Uruchom* i *Launch in (tu-nazwa- przeglądarki)*.



Rysunek 13 Opis strony internetowej po umieszczeniu w nim kodu

Przeglądarka zinterpretuje kod i wypisze wynik.



Rysunek 14 Wynik działania JavaScript

5. Podsumowanie

Wyżej przedstawiono rozwiązanie dotyczące automatycznej zamiany kodu zapisanego w środowisku programowania wizualnego na kod w języku programowania tekstowego. Pokazuje ono, że dość łatwo można zachęcić uczniów, nawet tych którzy przez długi czas programowali wizualnie, do podjęcia trudu tworzenia własnych programów z użyciem narzędzi programowania tekstowego. Wysiłek związany z nauczeniem się składni kolejnego języka jest nagradzany możliwością szybszego i wygodniejszego tworzenia aplikacji w trybie tekstowym.

Literatura

1. Borowiecki M., *Python w szkole od podstawówki do liceum*, EduFakty – Uczę Nowocześnie, nr 23 styczeń-luty 2013.
2. Borowiecki M., *Python na lekcjach informatyki w szkole ponadgimnazjalnej*, Informatyka w Edukacji, Toruń 2013.
3. Polewczyński A., *Nauka kodowania z Google Blockly*, Informatyka w Edukacji, Toruń 2014.
4. Summerfield M., *Python 3 Kompletne wprowadzenie do programowania, Helion*, Gliwice 2010.
5. Zając K., *Python jako alternatywa dla Pascala*, Informatyka w Edukacji, Toruń 2012.
6. Podstawa programowa kształcenia ogólnego dla szkoły podstawowej, http://www.dziennikustaw.gov.pl/DU/2017/356, ostatni dostęp 21.05.2017.
7. https://scratch.mit.edu, ostatni dostęp 21.05.2017.
8. https://blockly-demo.appspot.com/static/demos/code/index.html?lang=pl, ostatni dostęp 21.05.2017.
9. https://www.python.org, ostatni dostęp 21.05.2017