BLOCZKI POMOCNE W NAUCE PROGRAMOWANIA

Andrzej Polewczyński
WMiI UMK, ap@mat.umk.pl

Abstract. Historically, computer programs were created by typing text on a computer keyboard. The basic advantage of drag-and-drop programming is that it eliminates the need to memorize complex programming syntax and allows you to focus on thinking and concepts. Entry into the required level of computer science is often preceded by overcoming the initial difficulties associated with the use of algorithms and programming language notation. Support for this path may be the introduction to mainstream programming but supported by the visualization of important concepts, constructions, and methods that are important for further learning.

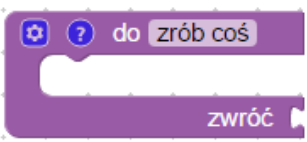
1. Wstęp

Nauczyciel w trakcie omawiania trudniejszych dla uczniów zagadnień musi stosować zróżnicowane środki i metody nauczania. Sięganie po nowe pomysły przekazywania istotnych dla tematu informacji, oraz próbowanie nowych strategii nauczania jest naturalnym działaniem nauczyciela. Wraz z rozwijającymi się technologiami wzrasta liczba technik ułatwiających wizualizację problemów, zjawisk, zasad działania, metod i zasad projektowania. Fakt dominacji przekazu wizualnego oraz dostępne na rynku edukacyjnym narzędzia dość mocno wspierają dziś metodyki nauczania programowania. Właśnie dzięki różnym metodom i narzędziom wizualnym, uczniowie nie muszą nudzić się w trakcie wyjaśniania zagadnień ważnych, przeznaczonych nie tylko dla tych, którzy chcą rozwijać swoje zainteresowania powstawaniem programów i własnoręcznym ich tworzeniem. Znane są fakty, że dla niektórych uczniów pewną barierę jest łączenie abstrakcyjnego myślenia z jego wyrażaniem w postaci programów. O ile rozumienie potrzeby wyrażania algorytmów jest na niezłym poziomie u większości przeciętnych licealistów, to umiejętność kodowania tych algorytmów już niekoniecznie. Jak minimalizować występowanie takich problemów? Pomocnym może być właśnie skupienie uwagi na myśleniu i jego wyrażaniu sposób bardziej obrazowy zwłaszcza na pewnym etapie nauki programowania? Gdy nauczyciel chce skupić uwagę na ważniejszych aspektach rozwiązywania problemów informatycznych posługuje się różnymi „rekwizytami”. Taką rolę spełniają nie od dziś narzędzia lub środowiska

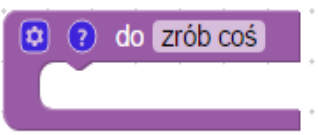
kodowania wizualnego. Należy podkreślić jednak, że dobrze spełniają one swoją rolę na wstępnym etapie wprowadzania ucznia do zasad i metod programowania. Bardzo prostym i doskonale nadającym się do pierwszych zastosowań w kontekście omawiania nowych treści jest znany już od kilku lat Google Blockly Code Editor. Kodowanie wizualne ma dziś coraz szersze zastosowanie w różnych dziedzinach, w których projektowaniem zajmują się niekoniecznie informatycy. Korzystanie z podobnego wizualnego narzędzia staje się pewnym standardem umożliwiającym kodowanie procesów i urządzeń.

2. Podprogramy

Jednym z ważniejszych i jednocześnie trudniejszych dla ucznia zagadnień, ale ułatwiających dalszą edukację informatyczną jest budowanie podprogramu w postaci funkcji czy funkcji proceduralnej. Zwykle zależy nam na tym, aby użyteczne działania były powtarzalne i uniwersalne. Zawsze interesuje nas efekt końcowy realizowanego pomysłu. Wyniki zainicjowanych procesów zależą od zakresu i przedmiotu działań oraz ich przebiegu. Działania oraz ich cel trzeba precyzyjnie określić. Celem zaplanowanych obliczeń matematycznych jest wynik, który trzeba po prostu zwrócić. Zwrócenie wyniku obliczeń może być też celem pośrednim, potrzebnym dla realizacji innych, dalszych działań. Zarys podobnej sytuacji możemy przedstawić w postaci bloku „zrób coś" i „zwróć" wartość jak na rys. 1.



Rysunek 1

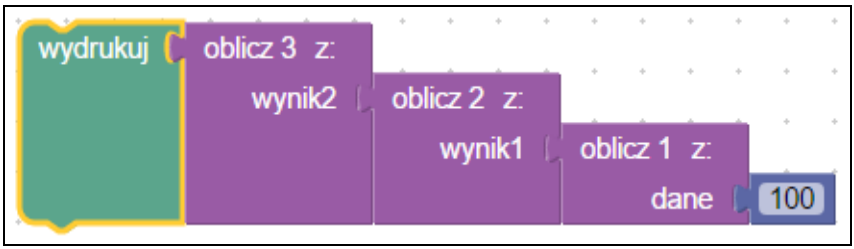


Rysunek 2

Znane są też operacje, których celem jest doprowadzenie do pewnych zmian. Operacje te mogą składać się z wielu działań prostych. W działaniach tych nie chodzi jednak o konkretną wartość, która ma być zwrócona – często chodzi o same zmiany – o przepis lub metody tych zmian. Po prostu „zrób coś" jak na rys. 2.

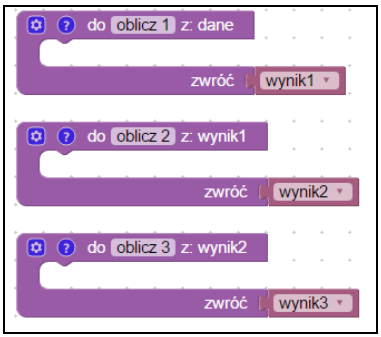
Okazuje się, że wymienione bloczki trafiają do wyobraźni ucznia bardziej niż ich odpowiedniki w formalnych językach programowania. Warunkiem koniecznym jest jednak odpowiednie zdefiniowanie zadania, wymuszające pewien schemat, do którego uczeń sam może zaproponować listę operacji elementarnych. Na przykład wydruk wyniku końcowego pewnych obliczeń **oblicz 3** zależy od wyniku operacji wykonanych w ramach **oblicz 2**, a wcześniej, wyniki **oblicz 2** są uzależnione od wyników **oblicz 1**. Obliczenia w ramach **oblicz 1** wykonane są na pewnych danych wejściowych **dane**. Sformułowanie schematu dochodzenia do wyniku narzuca

pewną sekwencję działań programisty, które można wyrazić symbolicznie w postaci wywołania jak na rys. 3. W naszym przykładzie dane wejściowe dla ciągu obliczeń to liczba 100.



Rysunek 3 Wywołanie współpracujących ze sobą podprogramów

Pamiętając o wadze zagadnienia specyfikacji każdego problemu informatycznego warto równolegle naszkicować przedstawiony schemat w formalnym języku programowania, uwzględniającym typy danych wejściowych i oczekiwanych wyników. Funkcje realizujące poszczególne fazy obliczeń mogą mieć swoją reprezentację w postaci bloczków jak na rys. 4. Można też skorzystać z funkcjonalności, którą daje Blockly Code Editor w postaci możliwości eksportu do tekstów kodów w popularnych językach: JavaScript, Python, PHP, Lua, Dart, lub XML.



Rysunek 4 Prototypy podprogramów kolejnych obliczeń

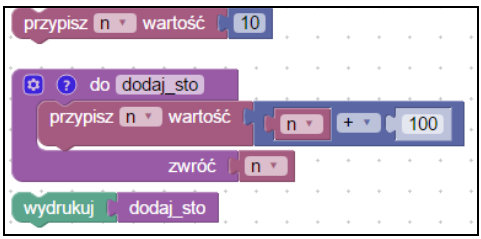
Przedstawienie propozycji schematu działań za pomocą bloczków wraz z zapisem w omawianym na lekcjach językiem programowania może przyczynić się do łatwiejszego przyswojenia tematyki. Temat warto przećwiczyć na konkretnym niezbyt trudnym zadaniu rozwiązywanym w kontekście przykładowego schematu zależności między funkcjami. Przykładem takim może być zadanie polegające na obliczeniu n-tej potęgi, do której należy ponieść liczbę, będąca największym wspólnym dzielnikiem dla dwóch liczb: minimalnej i maksymalnej –wyznaczonych z pewnego zbioru liczb całkowitych. Zadania tego typu mogą skupiać w sobie i być bazą

do omawiania kilku bardzo ważnych zagadnień takich jak: obsługa wielu danych elementarnych (odwołanie do elementów struktury danych w postaci listy-tablicy), iteracja i rekurencja, złożoność obliczeń.

```
int oblicz1(int dane)
{
    //obliczenia1
    return wynik1;
}
int oblicz2(int wynik1)
{
    //obliczenia2
    return wynik2
}
int oblicz3(int wynik2)
{
    //obliczenia3
    return wynik3
}
int main() {  cout << oblicz3(oblicz2(oblicz1(dane)));}
```

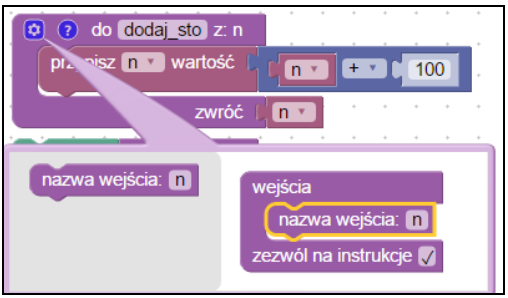
Rysunek 5 Prototypy podprogramów i wywołanie funkcji w C++

Przykład zaprezentowany przy użyciu bloczków powinien pokazywać korzyści płynące z projektowania podprogramów. Podprogramy bowiem, rozwiązują jakiś fragment całego większego problemu, ale mogą też być zastosowane do rozwiązania zupełnie innych zadań. Mają więc walor pewnej uniwersalności. Na uwagę zasługuje fakt, że podprogramy warto od razu konstruować tak, aby zapewniać w nich uniwersalność wejścia danych do obliczeń. Bardzo ważną jest też kwestia wyprowadzania wyników z jednego podprogramu i przekazania ich w charakterze danych dla innego. W formalnym języku programowania mamy różne metody przekazywania wartości do podprogramów i zwracania wyników. Należą do nich: referencja, stosowanie zmiennych globalnych(rys. 6), wskaźniki(w przypadku bloczków niejawnie za pomocą np. listy jak na rys. 8), zwracanie wartości przez sam podprogram (identyfikator funkcji). Do pewnej liczby takich i podobnych, ważnych zagadnień, można wprowadzić ucznia przy pomocy bloczków. Przykłady korzystania ze zmiennych o różnym zasięgu na rys. 6, 7, 8.



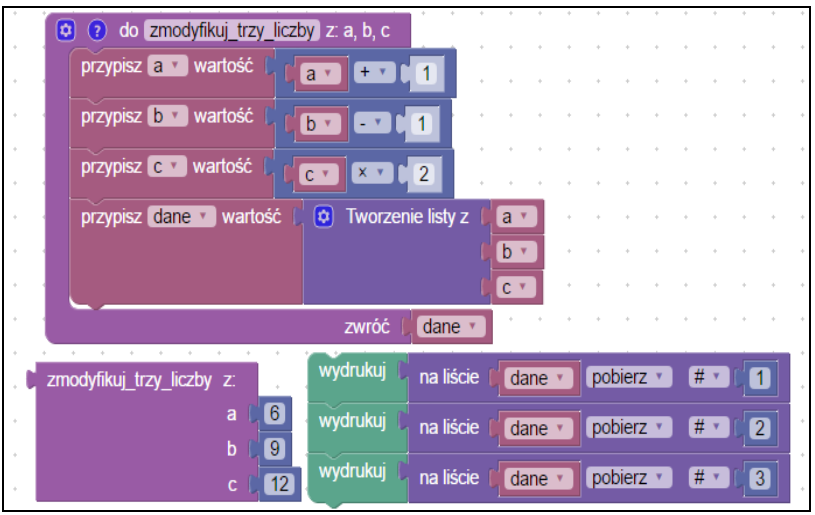
Rysunek 6 Zmienna globalna przetwarzana w podprogramie

Rys. 6 ilustruje użycie w podprogramie zmiennej **n** o przypisanej jej wartości 10, inicjowanej globalnie – w rozumieniu zasad znanych w językach programowania wysokiego poziomu – i poza funkcją. Wynik modyfikacji zmiennej **n** jest wyprowadzany za pomocą operacji zwróć czyli poprzez identyfikator funkcji **dodaj_sto**.



Rysunek 7 Parametry wejściowe podprogramu

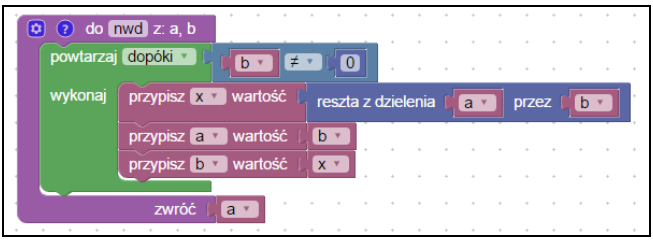
Na rys. 7 zmienna **n** jest parametrem wejściowym i znana jest tylko na terenie funkcji **dodaj_sto**. Wynik modyfikacji zmiennej **n** jest również wyprowadzany za pomocą opcji zwróć – czyli przez identyfikator funkcji.



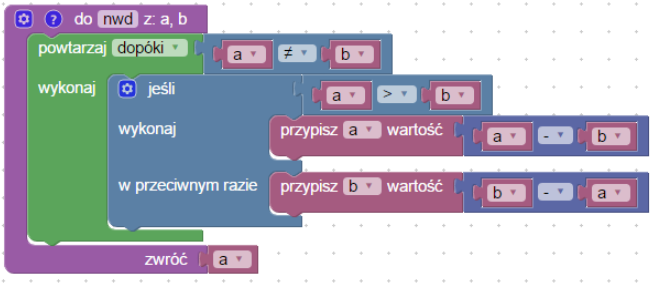
Rysunek 8 Zwracanie zmodyfikowanych wartości trzech zmiennych

Na rys. 8 mamy ilustrację sytuacji zwracania wartości funkcji przez wskazanie elementu zmiennej **dane** (nazwy listy). Nazwa listy wskazuje na strukturę złożoną z trzech elementów, zmodyfikowanych wartości liczb przypisanych początkowo zmiennym: a, b, c**,** oraz będących parametrami wejściowymi funkcji **zmodyfikuj_trzy_liczby.** Mamy więc przykładową odpowiedź na pytanie, jak wyprowadzić

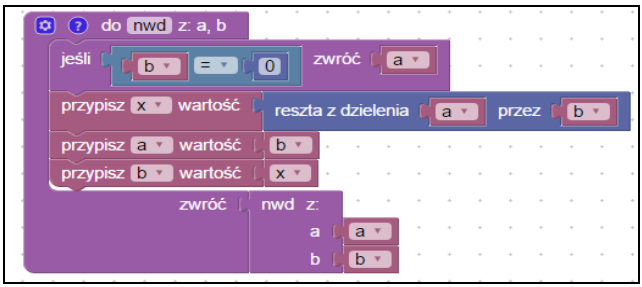
z podprogramu więcej niż tylko jedną wartość. W językach programowania znamy różne rozwiązania tego zagadnienia i analizujemy różne sytuacje oraz ich konsekwencje. Za pomocą bloczków można rozpocząć wyjaśnianie technik przydatnych przy pisaniu kodów w konkretnym języku programowania. Używanie formalnego języka w kontekście programowania funkcyjnego (proceduralnego) nie musi być procesem zniechęcającym ucznia – jeśli zaplanujemy dobrze przykłady korzystając też z techniki bloczków. Do konstrukcji tworzonych wizualnie warto też odwoływać się w trakcie implementacji w językach programowania takich jak Java, C/C++, Python. Wprowadzając kodowanie symboliczne i posługując się bloczkami posługujemy się zwykle znanymi problemami. Przykładowe algorytmy: NWD(rys. 9 i 10), NWW, czy algorytm ciągu Fibonacciego zapisywane są i omawiane na lekcji zarówno w realizacji iteracyjnej, jak i rekurencyjnej. Przy każdym z nich podajemy też przykłady rozwiązywalnych problemów, w których te algorytmy znajdują zastosowanie. Przykładowe zastosowanie NWD mamy w trakcie udzielania odpowiedzi na pytanie: „Czy dysponując naczyniami o pojemnościach A i B zawsze można napełnić nimi zbiornik określoną X objętością wody?" Do zbiornika można nalewać wodę tylko pełnymi czerpakami. Potrzebne jest spełnienie zależności X=n*NWD(A,B), gdzie n jest liczbą całkowitą n>0. Pojemność wynikowa musi być równa NWD(A,B) lub musi być jego wielokrotnością. Czynność wypełniania zbiornika wodą wykonywana jest w określonej liczbie kroków. Każde z naczyń A, B może być użyte wielokrotnie. Mamy tutaj też pole do dyskusji i wyboru najkorzystniejszego algorytmu, na wskazanie możliwych technik programowania, których trzeba używać, aby osiągnąć cel. Ilustrowanie zagadnień za pomocą bloczków ułatwia tego rodzaju dyskusje. Bloczki z powodzeniem mogą być używane przy wyjaśnianiu sporej liczby ważnych zagadnień, podstawowych jak i trudniejszych. Pozytywne rezultaty przynosi stosowanie bloczków przy konstruowaniu rozwiązań rekurencyjnych. Uczeń mając umiejętność operowania pojęciem podprogramu i świadomość wagi i roli parametrów podprogramu, z większą swobodą i zrozumieniem podejmie działania o charakterze rekurencyjnym. Pewnym eksperymentem może być nawet wprowadzenie rekurencji w środowisku Google Blockly Code przez umieszczenie jej na liście ćwiczeń pierwszego kontaktu z tematyki dotyczącej programowania funkcyjnego/proceduralnego.



Rysunek 9 NWD iteracyjnie 1



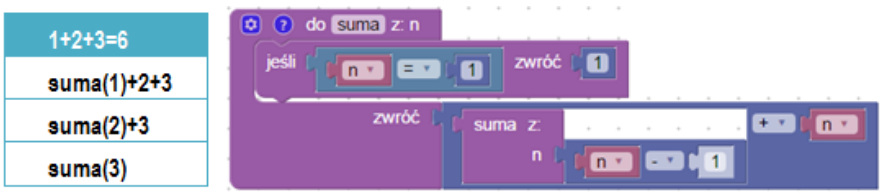
Rysunek 10 NWD iteracyjnie 2



Rysunek 11 NWD rekurencyjnie

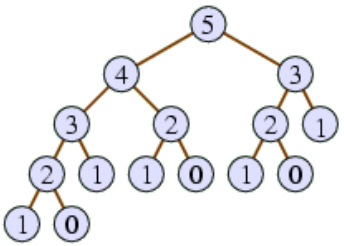
3. Rekurencja

Jak wspomniano wcześniej dobre opanowanie pojęcia podprogramu daje swobodę i łatwość w opanowaniu tematów trudniejszych, takich jak rekurencja. Jest wiele sposobów(i przykładów) na wyjaśnienie zasad, zalet i wad konstruowania funkcji rekurencyjnych. Na rysunku 12 przedstawiono rekurencyjne sumowanie kolejnych liczb naturalnych w zakresie od 1 do n. Jako ilustrację działania funkcji można wskazać przykładowy stos jej wywołań. Na szczycie stosu jest wynik obliczeń. Specyfikacja: **dane:** n∈N; **wynik:** suma kolejnych liczb całkowitych z zakresu od 1 do n. Zadanie rozwiązujemy nie korzystając z bloczków pętli.



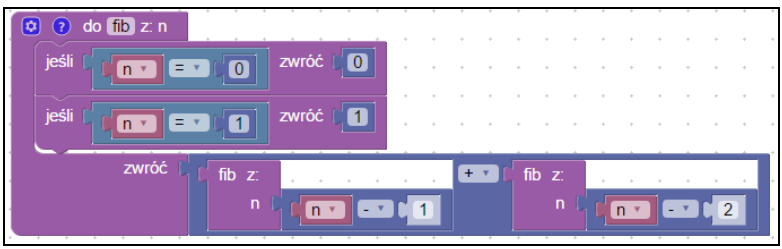
Rysunek 12 Suma rekurencyjnie

Przykład ten daje też pole do dyskusji na temat opłacalności takiego sumowania. Punktem wyjścia do dyskusji może być pytanie o możliwość przerwania funkcji zapisanej rekurencyjnie, albo problem zajętości pamięci przydzielanej na kolejną kopię wywoływanej funkcji (model stosu). Istotnym elementem ćwiczenia jest zwrócenie uwagi na możliwość zakończenia funkcji tylko wówczas, gdy n osiągnie wartość 1. Na uwagę zasługuje bloczek o nazwie „jeśli.. zwróć" stosowany w przypadku konieczności natychmiastowego zakończenia funkcji i zwrócenia wskazanej wartości funkcji w pewnej sytuacji. Wizualizacja algorytmów w postaci schematów czy drzew wywołań jest zawsze przydatna przy analizie działania algorytmów.



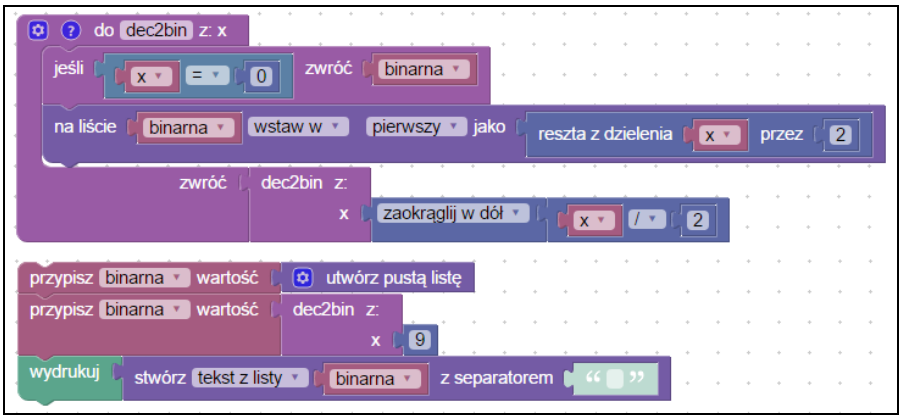

```
unsigned int fib(int n)
{
    if(n == 0) return 0;
    if(n == 1) return 1;
    return fib(n-1)+fib(n-2);
}
```

Rysunek 13



Rysunek 14 Wartość n-tego wyrazu ciągu Fibonacciego

Przy okazji tematyki wywołań rekurencyjnych warto też pokazać możliwości kształtowania i zwracania wyniku obliczeń rekurencyjnych w postaci bardziej złożonej struktury, której rozmiar jest dynamicznie dopasowany do liczby obliczanych wartości elementarnych. Blockly Code Editor oferuje wspomnianą wcześniej strukturę listy. Omawiane w szkole konwersje liczbowe w ujęciu rekurencyjnym są dobrą okazją, aby połączyć oba tematy. Uzyskanie wyniku w postaci ciągu bitów (zer i jedynek) polega na stworzeniu pustej listy, a następnie doklejaniu na jej początku kolejnego znaku 0 lub 1. W wyniku tej operacji otrzymujemy ukształtowaną liczbę w systemie o podstawie p=2. Kod można naturalnie uogólnić dla systemów o podstawie np. z zakresu $2 \leq p \leq 16$. Elementy struktury indeksowanej można przekształcić w tekst, który łatwiej się drukuje(stwórz tekst z listy).



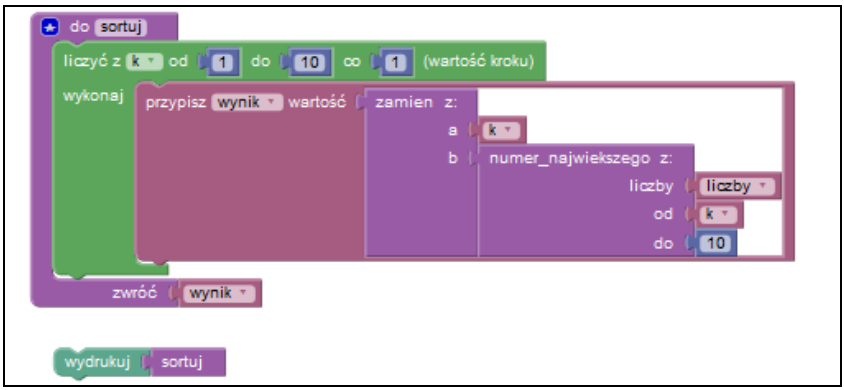
Rysunek 15

Wynik programu (rys.15) jest reprezentacją binarną liczby x (p=10) podanej jako parametr wejściowy funkcji dec2bin.

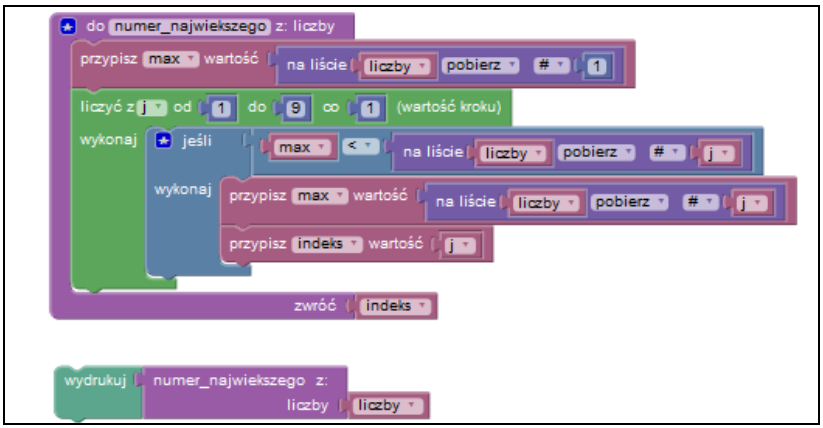
4. Dane

Istotnym faktem jest to, że za pomocą Google Blockly Editor możemy ilustrować też konsekwencje różnic między typami danych dzięki blokom oferującym działania na: liczbach, tekstach, listach. Mamy kategorię bloków kontroli logiki działań i pętli. Możemy więc wprowadzać i wyjaśniać podstawowe konstrukcje programistyczne i jednocześnie używać tych konstrukcji jako ilustracji przy rozwiązywaniu zadań bardziej złożonych. Niezwykle cenną dla ucznia jest umiejętność czytania i wyrażania tych samych kodów zapisanych w różnych notacjach. Bloczki stanowią notację bardzo konkretną i łatwą do opanowania przez wszystkich uczniów. Jednocześnie dają możliwość obrazowego wyrażania i analizy wielu typowych konstrukcji programistycznych metodą zstępującą(ang. top-down) lub wstępującą(ang. bottom-up). Punktem wyjścia mogą być przykładowo zadawane gotowe konstrukcje, które wymagają uszczegółowienia. Na rysunku 16 przedstawiono funkcję, w której wywołuje się dwie inne. Zadaniem ucznia może być zakodowanie funkcji: **zamień** oraz **numer_największego** na liście elementów w zakresie od k-tego do 10 elementu. Całość musi być poprzedzona odpowiednim wstępem – opisem algorytmu, symulacją działania na konkretnej partii danych.

Obiekt reprezentujący dane, ogólnie zwany elementem, może być składnikiem logicznie skonstruowanej grupy danych liczby (rys. 17). Zmienna jest miejscem przechowywania elementów danych. Zmienna przechowująca więcej niż jedną wartość elementarną daje czasami więcej możliwości. Mamy tutaj charakterystyczne odwoływanie się do elementów listy przez wyznaczenie numeru elementu.



Rysunek 16



Rysunek 17

5. Podsumowanie

Swoisty język graficznie reprezentowanych elementów i konstrukcji programistycznych, które ułatwiają kształtowanie umiejętności kodowania i dochodzenia do coraz wyższych jego poziomów, już na wczesnym etapie edukacji może mieć różne realizacje. Znakomicie widać je na rynku pomocy naukowych i zaawansowanych zabawek. Kodowanie wizualne ma jasną perspektywę coraz szerszego zastosowania w różnych poważnych projektach. Na wstępnych etapach nauki programowania podobne narzędzia sprawdzają się świetnie. Dostarczają nie tylko elementów zabawy i eksperymentu, ale przede wszystkim rozbudzają czasami uśpione zainteresowania i talent. Dla nauczyciela natomiast stanowią cenną pomoc dydaktyczną,

z której można korzystać na dowolnym etapie, w tych miejscach realizacji programu nauczania, w których podobne wsparcie jest korzystne dla ucznia.

Literatura

1. Baldwin R., *Teaching beginners to code,* http://cnx.org/content/col11498/1.20/
2. Harel D., *Rzecz o istocie informatyki*, WNT, Warszawa 1992
3. Sysło M.M., *Algorytmy*, Helion, Gliwice 2016.
4. https://blockly-demo.appspot.com
5. http://learn.code.org/