Informatyka w Edukacji, XVI
UMK Toruń, 2019

MATURA Z PYTHONEM

Agnieszka Samulska
Ośrodek Edukacji Informatycznej i Zastosowań Komputerów
02-026 Warszawa, ul. Raszyńska 8/10
agnieszka.samulska@gmail.com

Abstract. Python can be widely used during the The Matura Exam in computer science. It can be used to write an algorithm in the theoretical part, and to implement solutions of tasks in the practical part..Despite typical programming task, it can be used for solving database problems and for preparing simulations. The article also presents how to use Python in educational classes.

1. Wstęp

Język programowania Python jest wymieniany jako jeden z kilku najbardziej pożądanych na rynku pracy. Jest językiem stosunkowo łatwym do opanowania, cechującym się prostotą i dbałością o czytelność kodu, w którym wcięcia grają kluczową rolę. Doskonale nadaje się do nauki programowania. Coraz częściej jest wybierany jako pierwszy język tekstowy do nauki algorytmiki. Oprócz zastosowań edukacyjnych wykorzystywany jest przede wszystkim w praktyce. Posłużył m.in. do napisania systemu do przeprowadzania konkursów programistycznych SIO2[1].

Od roku szkolnego 2018/2019 na maturze z informatyki można wybrać język programowania Python. Wszystkie prezentowane przykłady pochodzą z egzaminu maturalnego z maja 2018 roku, przeprowadzonego według nowej formuły[2] [1].

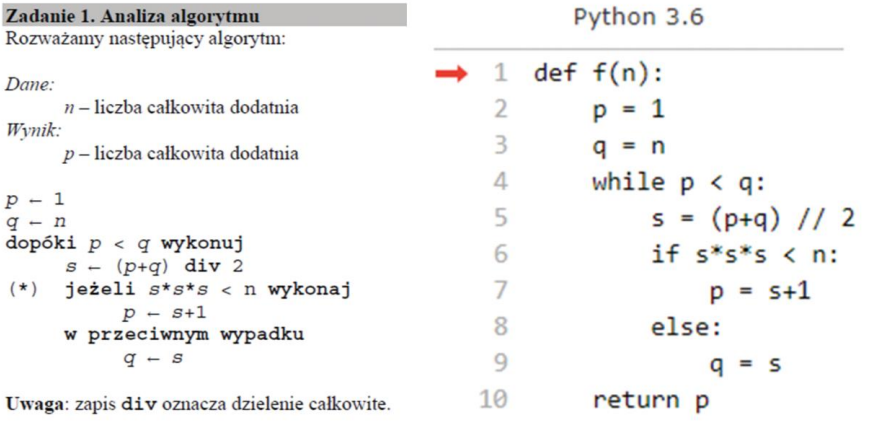
2. Analiza algorytmów

Podczas egzaminu, poszukując odpowiedzi na problemy zamieszczone w pierwszej części arkusza maturalnego, posługujemy się kartką papieru i długopisem oraz

[1] Informacje o projekcie znajdują się na stronie https://sio2project.mimuw.edu.pl. Więcej informacji o aplikacjach, w których wykorzystano ten język programowania, znajduje się na stronie https://en.wikipedia.org/wiki/List_of_Python_software

[2] Treści zadań dostępne są w serwisie Centralnej Komisji Egzaminacyjnej https://cke.gov.pl/egzamin-maturalny/egzamin-w-nowej-formule/arkusze/2018-2

prostym kalkulatorem. Na lekcjach z uczniami możemy wykorzystać narzędzie Python Tutor[3] [2], do sprawdzenia, czy przeprowadzona przez nas analiza algorytmu jest poprawna. Wizualizację można przeprowadzić między innymi w języku Python. Zauważmy, że zapis algorytmu w języku Python jest zbliżony do pseudokodu i wymaga jedynie niewielkiego dostosowania do składni języka.



Zadanie 1. Analiza algorytmu
Rozważamy następujący algorytm:

Dane:
n – liczba całkowita dodatnia
Wynik:
p – liczba całkowita dodatnia

```
p ← 1
q ← n
dopóki p < q wykonuj
    s ← (p+q) div 2
(*) jeżeli s*s*s < n wykonaj
        p ← s+1
    w przeciwnym wypadku
        q ← s
```

Uwaga: zapis **div** oznacza dzielenie całkowite.

Python 3.6

```
1  def f(n):
2      p = 1
3      q = n
4      while p < q:
5          s = (p+q) // 2
6          if s*s*s < n:
7              p = s+1
8          else:
9              q = s
10     return p
```

Rysunek 1 Po prawej zapis algorytmu w języku Python

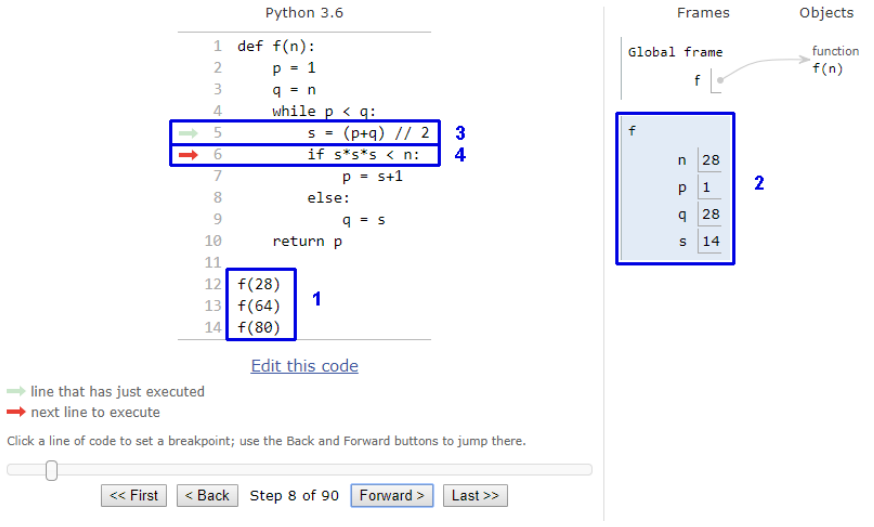
W prezentowanym zadaniu należało podać wyniki działania algorytmu dla wybranych wartości n. Wywołując zdefiniowaną funkcję dla podanych wartości n (1) możemy przeanalizować algorytm krok po kroku i prześledzić zmianę wartości poszczególnych zmiennych (2). Na bieżąco mamy informację o aktualnie wykonywanej instrukcji (3) oraz zaznaczoną kolejną instrukcję do wykonania (4).

Po zakończeniu algorytmu dla danej wartości n mamy podgląd na wartość będącą wynikiem funkcji.

3. Tworzenie algorytmów

Język Python, jak już wspomniano, ma składnię zbliżoną do pseudokodu, pod warunkiem ograniczenia się do stosowania instrukcji sterujących, operatorów arytmetycznych, operatorów logicznych i przypisań do zmiennych. Zatem jest dobrą alternatywą pseudokodu do zapisu algorytmów w części teoretycznej. Zadanie Krajobraz polegało na rozwiązaniu dwóch problemów algorytmicznych. Pierwszy z nich to poszukiwanie minimum, czyli skrajnie lewego szczytu (zad. 2.1.), drugi – sortowanie szczytów względem ich widoczności z punktu położenia obserwatora (zad. 2.2.) [3].

[3] http://pythontutor.com/



Rysunek 2 Analiza algorytmu – śledzenie wartości zmiennych

```
01. def skrajnie_lewy(n, X, Y):
02.     x, y = X[0], Y[0]
03.     for i in range(1, n):
04.         if X[i] / Y[i] < x / y:
05.             x, y = X[i], Y[i]
06.     return x, y
```

Rysunek 3 Rozwiązanie zadania 2.1.

Zapis:

```
x, y = X[0], Y[0]
```

jest równoważny:

```
x = X[0]
y = Y[0]
```

Poniżej znajduje się jedno z rozwiązań drugiego problemu – algorytm sortowania bąbelkowego.

```
01. def posortuj(n, X, Y):
02.     for i in range(1, n):
03.         for j in range(0, n - i):
04.             if X[j] / Y[j] > X[j + 1] / Y[j + 1]:
05.                 X[j], X[j + 1] = X[j + 1], X[j]
06.                 Y[j], Y[j + 1] = Y[j + 1], Y[j]
07.     return X, Y
```

Rysunek 4 Rozwiązanie zadania 2.2.

Zauważmy też, że instrukcje w wierszach 5 i 6 służą do zamiany danych miejscami. Ich odpowiednikiem w pseudokodzie będzie:

```
tmp ← X[j]
X[j] ← X[j + 1]
X[j + 1] ← tmp
tmp ← Y[j]
Y[j] ← Y[j + 1]
Y[j + 1] ← tmp
```

Dla obu problemów uzyskujemy czytelny i zwięzły kod, pozbawiony nadmiarowych informacji takich, jak na przykład deklaracja typów zmiennych.

4. Algorytmy w praktyce

W drugiej części egzaminu należy wykazać się praktyczną umiejętnością rozwiązywania problemów, w tym sprawdzana jest umiejętność napisania programu. Posłużymy się dwoma przykładami. Pierwszy problem do rozwiązania jest czysto techniczny. Wymaga umiejętności identyfikacji w co czterdziestym słowie dziesiątego znaku. Oprócz implementacji algorytmu, będącego rozwiązaniem zadania, należy wykazać się umiejętnością operowania na plikach z danymi. W poniższym rozwiązaniu zaprezentowany został jeden ze sposobów odczytu i zapisu do pliku tekstowego.

```
01. def zad41():
02.     plik = open('sygnaly.txt', 'r')
03.     wynik = open('wynik4_1.txt', 'w')
04.     i = 1
05.     for wiersz in plik:
06.         if i % 40 == 0:
07.             wynik.write(wiersz[9])
08.         i = i + 1
09.     plik.close()
10.     wynik.close()
```

Rysunek 5 Rozwiązanie zadania 4.1.

Kolejny problem związany jest ze zliczaniem unikatowych liter w słowie. Algorytm zliczania można zapisać klasycznie, tworząc listę 26 liczb, przechowujących informację o wystąpieniach poszczególnych liter alfabetu. Początkowo lista jest wypełniona zerami. Analizując kolejne znaki słowa odnotowujemy fakt ich wystąpienia zamieniając właściwe 0 na 1. Wynikiem funkcji jest liczba wystąpień 1.

```
01. def ile(slowo):
02.     lista = [0 for i in range(26)]
03.     for znak in slowo:
04.         pom = ord(znak) - 65
05.         lista[pom] = 1
06.     return lista.count(1)
```

Rysunek 6 Zliczanie unikatowych liter w słowie

Do rozwiązania tego problemu można też wykorzystać strukturę jaką jest zbiór (`set`), gromadzący w tym przypadku informację o unikatowych znakach w słowie (wynikiem `set("AAAB")` jest `{'B', 'A'}` lub `{'A', 'B'}` – kolejność elementów zbioru nie jest ustalona). Wielkość zbioru jest rozwiązaniem problemu.

```
01. def ile(slowo):
02.     return len(set(slowo))
```

Rysunek 7 Zliczanie unikatowych liter w słowie z wykorzystaniem zbioru

Całość rozwiązania dopełni poszukiwanie słowa o maksymalnej liczbie unikatowych liter.

```
01. def zad42():
02.     plik = open('sygnaly.txt', 'r')
03.     wynik = open('wynik4_2.txt', 'w')
04.     maksimum = 0
05.     for wiersz in plik:
06.         pom = ile(wiersz[:-1])
07.         if maksimum < pom:
08.             slowo, maksimum = wiersz, pom
09.     wynik.write(slowo + str(maksimum))
10.     plik.close()
11.     wynik.close()
```

Rysunek 8 Rozwiązanie zadania 4.2.

Zwróćmy uwagę na zapis `wiersz[:-1]`. Wynikiem tego zapisu jest ciąg pozbawiony ostatniego znaku. Jest to niezbędny zabieg, ponieważ wiersz jest wczytywany wraz ze znakiem końca wiersza (`"\n"`). Taki pozostawiony znak w pierwszej wersji funkcji `ile` generuje błąd, ze względu na to, że `ord("\n") - 65` wykracza poza zakres indeksów zmiennej lista. W drugiej wersji funkcja `ile` zwraca błędny wynik – zawsze o 1 większy.

5. Przetwarzanie i tworzenie informacji

Pozostałe zadania w części praktycznej egzaminu trzeba rozwiązać za pomocą dostępnych narzędzi informatycznych. Może to być arkusz kalkulacyjny czy baza danych, ale nic nie stoi na przeszkodzie, żeby to zrobić w języku Python. W tym celu należy wykazać się umiejętnością przetwarzania informacji zawartych w pliku tekstowym w poszukiwaniu odpowiedzi. A w szczególności umiejętnością podziału wiersza danych na składowe czy wyodrębnienia fragmentu ciągu znaków. Pozostałe problemy sprowadzają się do klasycznych problemów algorytmicznych. W zadaniu 5.1. są nimi np. zliczenie poszczególnych elementów oraz wyszukanie maksimum wśród zliczonych elementów [4].

Poniżej przedstawiony jest sposób rozwiązania zadania, w którym wykorzystano listę (`list`) do zliczenia metrów sześciennych wody dopływającej do zbiornika w poszczególnych latach.

```
01. lista = [0 for i in range(10)] #10-cio elementowa lista wypełniona zerami
02. for wiersz in plik:
03.     data, woda = wiersz.split('\t') #podział wiersza na składowe
04.     rok = int(data[0:4]) #pierwsze cztery znaki daty zamienione na liczbę
05.     lista[rok - 2008] = lista[rok - 2008] + int(woda) #zliczanie
06. maks = max(lista) #znalezienie maksimum
```

Rysunek 9 Fragment rozwiązania zadania 5.1., w którym wykorzystano listę

Zastosowano do tego celu listę składaną, zawierającą 10 elementów (odpowiadających latom 2008-2017). Elementy listy to liczby całkowite indeksowane od zera, stąd konieczność uwzględnienia tego faktu w odwołaniu `lista[rok - 2008]`, gdzie rok to pierwsze cztery znaki daty zamienione na liczbę.

Lista początkowo wypełniona jest zerami. Dla każdego roku, zawartość listy jest korygowana o liczbę metrów sześciennych wody, jaka dopływa do zbiornika w ciągu doby. Analogicznie – wypisując rok, w którym zbiornik retencyjny został zasilony największą liczbą metrów sześcienny wody, do znalezionego indeksu elementu listy, odpowiadającego maksimum, dodajemy 2008.

```
01. def zad51():
02.     plik = open('woda.txt', 'r')
03.     wynik = open('wynik5_1.txt', 'w')
04.     lista = [0 for i in range(10)]
05.     for wiersz in plik:
06.         data, woda = wiersz.split('\t')
07.         rok = int(data[0:4])
08.         lista[rok - 2008] = lista[rok - 2008] + int(woda)
09.     maks = max(lista)
10.     wynik.write('Rok ' + str(lista.index(maks) + 2008)) #indeks powiększony o 2008
11.     plik.close()
12.     wynik.close()
```

Rysunek 10 Rozwiązanie zadania 5.1. z wykorzystaniem listy

Ten sam problem można rozwiązać w oparciu o słownik (`dict`). Jest to struktura danych różniąca się od listy tym, że zamiast indeksów zawiera klucze (`keys`). Kluczom odpowiadają wartości (`values`). Zatem elementami słownika (`items`) są pary klucz-wartość. Elementy słownika znajdują się w nim w losowej kolejności. W naszym zadaniu odpowiednikiem klucza będzie rok, a jego wartością liczbą metrów sześciennych wody. Co ciekawe, klucz nie musi być liczbą całkowitą – w naszym przypadku będzie ciągiem znaków (`slownik[rok]`, gdzie `rok` to np. `'2008'`). Podczas zwiększania wartości dla danego klucza należy zadbać o to, aby w słowniku istniała para klucz-wartość. Jeśli jej nie ma, dodajemy klucz z wartością zero.

```
01. if rok not in slownik:
02.     slownik[rok] = 0 #dodanie pary klucz-wartość
```

Rysunek 11 Dodanie klucza z wartością 0

Maksimum, czyli największej liczby metrów sześciennych wody z rzeki Wirki, będziemy poszukiwać wśród wartości zapisanych w słowniku. Następnie wśród elementów słownika odszukujemy klucz odpowiadający znalezionemu maksimum:

```
01. def zad51():
02.     plik = open('woda.txt', 'r')
03.     wynik = open('wynik5_1.txt', 'w')
04.     slownik = {}
05.     for wiersz in plik:
06.         data, woda = wiersz.split('\t')
07.         rok = data[0:4]
08.         if rok not in slownik:
09.             slownik[rok] = 0
10.         slownik[rok] = slownik[rok] + int(woda)
11.     maks = max(slownik.values()) #poszukiwanie maksimum wśród wartości
12.     for x, y in slownik.items():
13.         if y == maks:
14.             rok = x
15.     wynik.write('Rok ' + rok)
16.     plik.close()
17.     wynik.close()
```

Rysunek 12 Rozwiązanie zadania 5.1. z wykorzystaniem słownika

6. Symulacje

Symulacja wymaga ścisłego trzymania się założeń podanych w treści polecenia. Kluczowe zapisy zostały wytłuszczone w arkuszu maturalnym. Wszelkie odstępstwa od założeń skutkują błędnymi wynikami. Stąd, w tego typu zadaniach, pojawia się informacja umożliwiająca zweryfikowanie rozwiązania.

Zacznijmy od zaimplementowania założeń samej symulacji:

- pomiar w dniu 2008-01-01 wskazywał 500 000 m^3 wody,
- ilość wypuszczanej wody ze zbiornika zaokrąglamy w górę,
- przepełnienie następuje w momencie przekroczenia 1 000 000 m^3 wody; w jego momencie wypuszczamy nadmiar wody powyżej 1 000 000 m^3,
- objętość wody w zbiorniku zmniejszamy o ilość wypuszczonej wody,
- aktualizujemy objętość wody w zbiorniku o dzienny dopływ.

Do sprawdzenia poprawności symulacji dodajmy instrukcję:

```
if data == '2008-02-01':
    print(pojemnosc)
```

Po uruchomieniu kodu i wywołaniu funkcji zad54() na ekranie konsoli pojawia się spodziewany wynik 338406.

```
01. from math import * #moduł niezbędny do korzystania z funkcji ceil
02. def zad54():
03.     plik = open('woda.txt', 'r')
04.     pojemnosc = 500000 #pojemność startowa
05.     for wiersz in plik:
06.         data, woda = wiersz.split('\t')
07.         if data == '2008-02-01':
08.             print(pojemnosc)
09.         do_wylania = ceil(0.02 * pojemnosc) #zaokrąglenie w górę
10.     if pojemnosc > 1000000:
11.             pojemnosc = 1000000 #wylanie nadmiaru wody po przepełnieniu
12.         pojemnosc = pojemnosc - do_wylania
13.         pojemnosc = pojemnosc + int(woda)
14.     plik.close()
```

Rysunek 13 Symulacja do zadania 5.4. wraz ze sprawdzeniem jej poprawności

Teraz można przystąpić do rozwiązania problemu opisanego w podpunkcie a). Zgodnie z treścią zadania podajemy dzień, w którym pierwszy raz wypuszczono nadmiar wody po przepełnieniu i kończymy przetwarzanie danych.

```
01. from math import *
02. def zad54a():
03.     plik = open('woda.txt', 'r')
04.     wynik = open('wynik5_4a.txt', 'w')
05.     pojemnosc = 500000
06.     for wiersz in plik:
07.         data, woda = wiersz.split('\t')
08.         do_wylania = ceil(0.02 * pojemnosc)
09.         if pojemnosc > 1000000:
10.             wynik.write(data)
11.             break
12.         pojemnosc = pojemnosc - do_wylania
13.         pojemnosc = pojemnosc + int(woda)
14.     plik.close()
15.     wynik.close()
```

Rysunek 14 Rozwiązanie zadania 5.4a.

7. Tworzenie bazy i zapytań

Zanim zaczniemy z uczniami tworzyć bazy danych w Pythonie, warto zapoznać ich ze środowiskiem SQLite[4]. Jest to system zarządzania bazą danych, umożliwiający obsługę języka SQL, a co najważniejsze, posiadający API (Application Programming Interface) m.in. do Pythona. API to interfejs programistyczny, dzięki któremu możemy korzystać z funkcjonalności oferowanych przez daną aplikację czy program. W tym przypadku jest nim biblioteka środowiska Python o nazwie sqlite3 umożliwiająca tworzenie bazy danych tak, jak w środowisku SQLite. Wszystkie informacje dotyczące

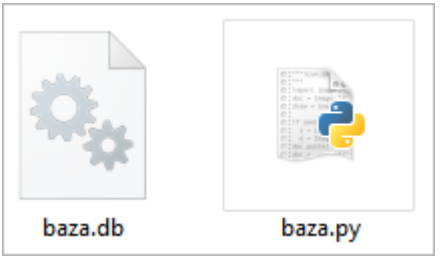
[4] Informacje dotyczące środowiska oraz program do pobrania znajdują się na stronie https://www.sqlite.org.

zawartości bazy danych, w tym tworzonych tabel oraz kwerend, znajdują się w jednym pliku wynikowym.

Moduł sqlite3[5] jest częścią standardowej biblioteki Pythona i może być wykorzystany do rozwiązywania zadań bazodanowych.

Rysunek 15 Dokumentacja modułu sqlite3

Wszystkie funkcje związane z tworzeniem bazy oraz zwierające rozwiązania poszczególnych zadań umieszczane są w pliku źródłowym *.py a cała baza danych (tabele wraz z zapytaniami) w pliku *.db. Oba te pliki stanowią komputerową realizację zadania bazodanowego na maturze w części praktycznej [5].



Rysunek 16 Komputerowa realizacja zadania bazodanowego

Proces tworzenia bazy danych sprowadza się do wygenerowania odpowiednich tabel (`CREATE TABLE`) a następnie wypełnienie ich danymi zgromadzonymi w plikach tekstowych (`INSERT`).

```
01. def tabela_komputery():
02.     obiekt.execute('''''CREATE TABLE komputery
03.                        (Numer_komputera INT primary key,
04.                        Sekcja CHARACTER(1),
05.                        Pojemnosc_dysku INT)''')
```

Rysunek 17 Przykład tworzenia tabeli

[5] Opis modułu znajduje się w dokumentacji https://docs.python.org/3.7/library/sqlite3.html.

Metoda `execute()` jest odpowiedzialna za wykonanie poleceń SQL. Tworzenie tabel i wypełnianie ich danymi niestety nie są czynnościami punktowanymi na maturze. Jednak są one niezmiernie ważne i bez nich nie możemy przystąpić do rozwiązywania problemów.

Do tworzenia zapytań używamy polecenia SELECT.

```
01. def zadanie6_1():
02.     obiekt.execute('''''SELECT Pojemnosc_dysku,
03.                        COUNT(Numer_komputera) AS liczba
04.                        FROM komputery
05.                        GROUP BY Pojemnosc_dysku
06.                        ORDER BY liczba DESC
07.                        LIMIT 10''')
08.     dane = obiekt.fetchall()
09.     for rekord in dane:
10.         print(rekord['Pojemnosc_dysku'], rekord['liczba'])
```

Rysunek 18 Zapytanie do zadania 6.1.

Metoda `fetchall()` umożliwia uzyskanie wszystkich pasujących rekordów dla danego zapytania. Funkcję zadanie6_1() można rozbudować o wypisanie wiersza nagłówkowego z nazwami pól oraz przekierowanie wyników do pliku tekstowego. Ale są to kwestie czysto techniczne, nie wpływające na jakość otrzymanego wyniku.

8. Podsumowanie

Za wyborem języka Python na maturze przemawiają następujące argumenty:

1. łatwość konwersji pseudokodu na język Python;
2. czytelność zapisów algorytmów, pozbawiona zbędnych informacji – pod warunkiem ograniczenia się do stosowania instrukcji sterujących, operatorów arytmetycznych, operatorów logicznych i przypisań do zmiennych;
3. zwięzłość kodu będącego rozwiązaniem typowego zadania maturalnego w części praktycznej, po części będąca wynikiem możliwości stosowania funkcji bibliotecznych oraz specyficznej składni języka.

W przypadku zagadnień dotyczących przetwarzania i tworzenia informacji oraz symulacji, w prezentowanych zadaniach pojawiły się proste problemy algorytmiczne (poszukiwanie maksimum czy zliczanie elementów). Do ich rozwiązania wykorzystano podstawowe instrukcje języka programowania (instrukcję przypisania, warunkową oraz iterację) oraz strukturę danych w postaci listy (odpowiednik tablicy w języku C++). W jednym z zadań wykorzystano bardziej wyrafinowaną strukturę danych jaką jest słownik. Można ją zaprezentować uczniom podczas zajęć edukacyjnych, właśnie przy okazji zliczania elementów. Przedstawione rozwiązania są bardzo krótkie i czytelne, a implementacja ich, łącznie z wczytaniem i wypisaniem danych, nie przekracza kilku-

nastu linijek. Jeśli pominiemy w rozwiązaniu elementy stałe (otwieranie pliku do odczytu, otwieranie pliku do zapisu, zamykanie otwartych plików czy iterowanie po rekordach pliku), to do napisania pozostaje kilka linijek kodu. Uczniowi, sprawnie posługującemu się językiem programowania, implementacja rozwiązań zajmie nie więcej czasu niż rozwiązanie w arkuszu kalkulacyjnym. Szczególnie w przypadku symulacji warto napisać program.

Oczywistym wydaje się, że podstawą sukcesu w rozwiązywaniu zadań bazodanowych jest znajomość zagadnień związanych z projektowaniem relacyjnych baz danych oraz umiejętność posługiwania się poleceniami języka SQL. Zatem naukę warto rozpocząć od bardzo prostych przykładów, stopniowo zwiększając ich trudność. Z roku na rok zadania maturalne są coraz trudniejsze, a znajomość języka SQL badana jest także w części teoretycznej. Samo zaprogramowanie bazy danych w Pythonie nie jest problemem, trudność może sprawić jedynie brak łatwego dostępu do tabel czy tworzonych zapytań. Dlatego zanim przystąpimy do rozwiązywania trudniejszych zadań maturalnych warto wspomóc się wspomnianym na wstępie środowiskiem SQLite lub użyć wtyczki SQLite Manager w przeglądarce. Zalecane jest również rozpoczęcie nauki języka SQL od Akademii Khana [6].

Literatura

1. Serwis Centralnej Komisji Egzaminacyjnej: https://cke.gov.pl, ostatni dostęp 28.05.2017.
2. Borowiecki M., *Python na lekcjach informatyki w szkole ponadgimnazjalnej*, Informatyka w Edukacji, Toruń 2013.
3. Samulska A., *Maturalne potyczki z Pythonem*, W cyfrowej szkole, Nr 2/2018.
4. Samulska A., *Przetwarzanie i tworzenie informacji oraz symulacje w Pythonie*, W cyfrowej szkole, Nr 2/2019.
5. Samulska A., *Baza danych w Pythonie*, W cyfrowej szkole, Nr 1/2019.
6. Akademia Khana, https://pl.khanacademy.org/computing/computer-programming/sql, ostatni dostęp 28.05.2019.