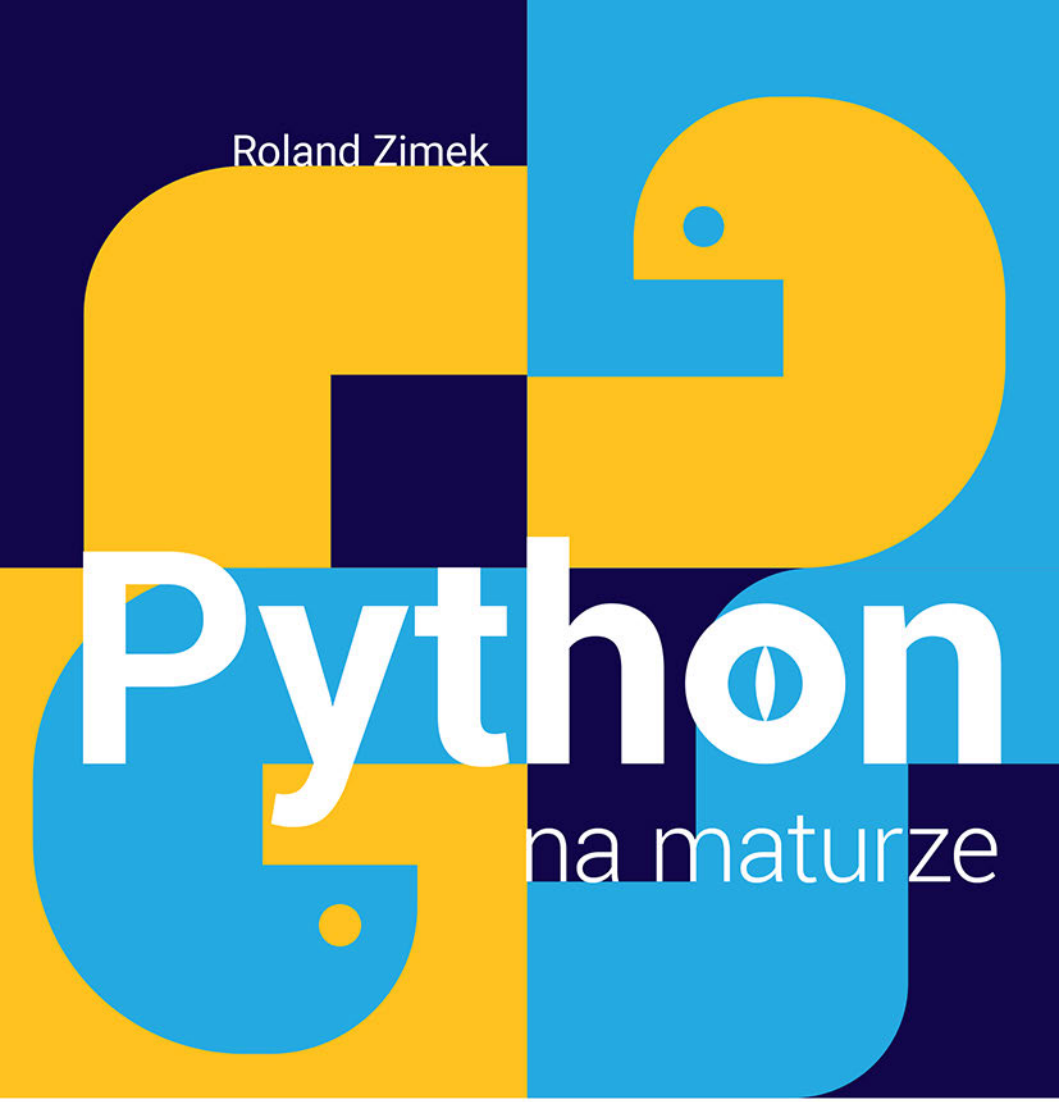
Roland Zimek



Python na maturze

Rozwiązania i analiza wybranych zadań programistycznych

Helion EDUKACJA

Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki
Grafika na okładce została wykorzystana za zgodą Shutterstock.com

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/pytmat
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-7744-8

Printed in Poland.

Spis treści

Wstęp 5

Egzamin maturalny z 2018 r. — Wega 7

Egzamin maturalny z 2015 r. — Liczby binarne 23

Egzamin maturalny z 2016 r. — Szyfr Cezara 37

Egzamin maturalny według starej formuły z 2019 r. — Działki 49

Egzamin maturalny według starej formuły z 2018 r. — Słowa oraz egzamin maturalny z czerwca 2016 r. — Systemy liczbowe 63

Egzamin maturalny z 2019 r. — Liczby 69

Egzamin maturalny z 2017 r. — Piksele 79

Próbny egzamin maturalny z 2020 r. — Luki w ciągu 89

Egzamin maturalny z 2020 r. — Pary 99

Egzamin maturalny z 2016 r. — Gra w życie 111

Egzamin maturalny z czerwca 2018 r. — Scalanie 119

Wstęp

Nauka programowania na dobre wkroczyła do szkół. Zgodnie z Rozporządzeniem Ministra Edukacji Narodowej z dnia 14 lutego 2017 r. w sprawie podstawy programowej[1] programowania uczą się już uczniowie pierwszej klasy. Oczywiście w początkowym okresie nauki wykorzystywane są głównie wizualne języki programowania. Od klasy siódmej uczniowie powinni jednak rozpocząć programowanie w języku tekstowym. Jaki język programowania wybrać spośród dziesiątków dostępnych w internecie? To zależy już od nauczyciela. Jednak istotną wskazówką winno być przyjrzenie się liście języków programowania wymienionych w komunikacie dyrektora Centralnej Komisji Egzaminacyjnej w sprawie listy systemów operacyjnych, programów użytkowych oraz języków programowania w przypadku egzaminu maturalnego z informatyki[2]. Od roku szkolnego 2017/2018 na liście tej pojawił się bardzo dynamicznie rozwijający się język Python. Obecnie znajduje się na trzecim miejscu rankingu TIOBE najbardziej popularnych języków programowania[3]. Niewątpliwą zaletą języka Python jest łatwość programowania, intuicyjność i przejrzystość kodu, połączona z ogromnymi możliwościami.

[1] *http://prawo.sejm.gov.pl/isap.nsf/DocDetails.xsp?id=WDU20170000356*

[2] *http://www.oke.krakow.pl/inf/filedata/files/20180820%20EM%20Komunikat%20o%20egzaminie%20z%20informatyki.pdf*

[3] *https://www.tiobe.com/tiobe-index/*

Wszystkie zawarte w książce zadania maturalne oraz arkusz zasad oceniania rozwiązań pochodzą z arkuszy maturalnych Centralnej Komisji Egzaminacyjnej i są własnością ich autorów oraz Centralnej Komisji Egzaminacyjnej. W niniejszej książce zostały przytoczone w celu umożliwienia czytelnikowi zapoznania się z ich treścią. Szczegółowe informacje dotyczące matur oraz treści arkuszy maturalnych są dostępne na stronie Centralnej Komisji Egzaminacyjnej: *https://cke.gov.pl/*.

Egzamin maturalny z 2020 r. — Pary

Poziom rozszerzony na maturze z 2020 r. zawiera zadanie 4., które należy rozwiązać, tworząc odpowiedni program. Treść tego zadania jest następująca:

Zadanie 4. PARY

Zadanie 4

W pliku `pary.txt` znajduje się 100 wierszy. Każdy wiersz zawiera parę danych składającą się z liczby całkowitej z przedziału od 3 do 100 i słowa (ciągu znaków) złożonego z małych liter alfabetu angielskiego o długości od 1 do 50 znaków. Liczba i słowo są oddzielone znakiem spacji.

Napisz **program(-my)**, dający(-e) odpowiedzi do poniższych zadań. Uzyskane odpowiedzi zapisz w pliku `wyniki4.txt`, poprzedzając każdą z nich numerem odpowiedniego zadania.

Uwaga: plik `przyklad.txt` zawiera przykładowe dane spełniające warunki zadania.

Odpowiedzi dla danych z pliku `przyklad.txt` są podane pod treściami zadań oraz w pliku `odp_przyklad.txt`.

W pierwszej kolejności muszę wczytać dane z pliku `pary.txt`. Należy się jednak wcześniej zastanowić, w jaki sposób przechowywać pary liczba – słowo. Co prawda słowa nie będą potrzebne w zadaniu 4.1, ale nie ma sensu ich teraz pomijać, bo i tak będzie je trzeba ostatecznie wczytać. Możemy przechować pary liczba – słowo z użyciem jednej listy, zawierającej krótkie listy par:

```
[[68, "dnmjhmj"], [45, "uwdgttwklpoerstw"], [3, "asq"], …]
```

W tym przypadku musiałbym się odwoływać do konkretnych danych, podając zarówno indeks pary, jak i indeks elementu, na przykład `[0][0]` wskazuje na liczbę 68, a `[0][1]` na tekst „dnmjhmj".

Drugim sposobem może być przechowywanie liczby na jednej liście, a słów na drugiej. Chcąc otrzymać ich parę, wystarczy odczytać elementy o tych samych indeksach z obu list.

Moim zdaniem w przypadku tego zadania lepszym i bardziej przejrzystszym rozwiązaniem będzie drugi przykład. Tym bardziej że dopiero w zadaniu 4.3 będę potrzebował obu list jednocześnie.

Mogę utworzyć dwie listy na dwa różne sposoby. Pierwszy to:

```
with open("pary.txt") as plik:
    pary = [j.split() for j in plik]

liczby = [int(i[0]) for i in pary]
slowa = [i[1] for i in pary]
```

A drugi to:

```
with open("pary.txt") as plik:
    liczby, slowa = zip(*[j.split() for j in plik])

liczby = [int(i) for i in liczby]
```

W drugim za pomocą operatora * rozpakowuję listę zawierającą wiele argumentów, a korzystając z funkcji `zip()`, otrzymuję listę krotek, które przypisuję do dwóch list.

W każdym przypadku muszę jednak zadbać o to, by pierwsza lista została przekształcona na liczby.

Pierwszy przykład utworzenia dwóch list jest bardziej przejrzysty, więc pozostanę przy nim.

Zadanie 4.1

Mocna hipoteza Goldbacha mówi, że każda parzysta liczba całkowita większa od 4 jest sumą **dwóch nieparzystych** liczb pierwszych, np. liczba 20 jest równa sumie 3 + 17 lub sumie 7 + 13.

Każdą **liczbę parzystą** z pliku `pary.txt` przedstaw w postaci sumy dwóch liczb pierwszych. Wypisz tę liczbę oraz dwa składniki sumy w kolejności niemalejącej. Jeżeli istnieje więcej rozwiązań (tak jak dla liczby 20) należy wypisać składniki sumy o największej różnicy.

Wyniki podaj w oddzielnych wierszach, w kolejności zgodnej z kolejnością danych w pliku `pary.txt`. Liczby w każdym wierszu rozdziel znakiem spacji, np. dla liczby 20 należy wypisać 20 3 17.

Dla danych z pliku `przyklad.txt` prawidłową odpowiedzią jest:

```
24 5 19
6 3 3
6 3 3
```

Głównym tematem zadania 4.1 jest mało znana hipoteza Goldbacha. Ponieważ będziemy dużo operować na liczbach pierwszych, przypomnę, że są to liczby naturalne większe od 1, które mają dokładnie dwa dzielniki naturalne: jedynkę i siebie samą.

Aby sprawdzić, czy dana liczba jest liczbą pierwszą, zdefiniuję funkcję `pierwsza`:

```
def pierwsza(liczba):
    licznik = 1
    for i in range(2, liczba+1):
        if liczba % i == 0:
            licznik += 1
    return True if licznik == 2 else False
```

Zwróci ona wartość logiczną `True`, gdy liczba będzie liczbą pierwszą.

W głównej części programu będę brał każdą wartość z listy liczb i sprawdzał, czy jest to liczba parzysta. W kolejnej pętli `for` wyszukuję wszystkie liczby pierwsze mniejsze od tej liczby. Zapisuję je na liście `pierwsze`.

Ostatecznie sprawdzam dla każdej otrzymanej w ten sposób liczby pierwszej, czy po zsumowaniu jej z nie mniejszą od niej liczbą pierwszą otrzymam w wyniku sprawdzaną liczbę.

Jeżeli porównanie okaże się prawdziwe, sprawdzam, czy aktualnie znaleziona para liczb pierwszych ma większą różnicę niż znalezione wcześniej. Jeżeli tak, to zapamiętuję tę różnicę i parę liczb pierwszych.

```
def pierwsza(liczba):
    licznik = 1
    for i in range(2, liczba+1):
        if liczba % i == 0:
            licznik += 1
    return True if licznik == 2 else False

for i in liczby:
    if not i % 2:
```

```
p = []
for j in range(2, i+1):
    if pierwsza(j):
        p += [j]

j_k = []
roznica = -1

for j in range(len(p)):
    for k in range(j, len(p)):
        if p[j]+p[k] == i:
            if roznica < p[k]-p[j]:
                roznica = p[k]-p[j]
                j_k = [p[j], p[k]]
print(i, j_k)
```

Powyższy program, choć daje poprawne wyniki, wykonuje bardzo dużo niepotrzebnych obliczeń i porównań. Przede wszystkim dla każdej liczby parzystej tworzy listę liczb pierwszych mniejszych od niej samej. Jednak liczby pierwsze będą zawsze te same, a jedyną różnicą będzie ich liczba.

Kolejnym niepotrzebnym działaniem jest sprawdzanie, czy znaleziona para liczb pierwszych ma większą różnicę niż znalezione wcześniej. Sprawdzając bowiem liczby pierwsze od najmniejszej, zawsze otrzymam parę o największej różnicy.

Na przykład dla liczby 48 otrzymam następującą listę liczb pierwszych:

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

Dla liczb pierwszych 2 i 3 nie znajdę właściwej pary, dopiero sprawdzając liczbę pierwszą 5 z większą od niej liczbą pierwszą 43, otrzymam odpowiednią parę. Każda kolejna para liczb pierwszych musi mieć mniejszą różnicę.

Podobnie też nie szereguję znalezionej pary liczb pierwszych w kolejności niemalejącej, gdyż pierwsza ze znalezionych liczb pierwszych będzie mniejsza bądź równa drugiej.

Program da się znacząco zoptymalizować poprzez pozbycie się generowania liczb pierwszych dla każdej liczby. Zgodnie z założeniami zadania liczby są z zakresu od 3 do 100. Wystarczy, że raz utworzę listę liczb pierwszych nie większych niż ten zakres i będę z niej korzystał dla każdej ze sprawdzanych liczb.

Część programu po definicji funkcji `pierwsza` będzie następująca:

```
p = [i for i in range(2, 100) if pierwsza(i)]

for i in liczby:
    if not i % 2:
        j_k = []
        roznica = -1

        for j in range(len(p)):
            for k in range(j, len(p)):
                if p[j]+p[k] == i:
                    if roznica < p[k]-p[j]:
                        roznica = p[k]-p[j]
                        j_k = [p[j], p[k]]
        print(i, j_k)
```

Program można jeszcze trochę zoptymalizować, by zamiast porównywać każdą liczbę pierwszą z kolejnymi na liście sprawdzał, czy różnica pomiędzy liczbą a liczbą pierwszą też jest liczbą pierwszą.

Tak więc ostatni fragment programu będzie następujący:

```
        for j in range(len(p)):
            if pierwsza(k := i-p[j]):
                if roznica < k-p[j]:
                    roznica = k - p[j]
                    j_k = [p[j], k]
        print(i, j_k)
```

Kolejnym sposobem modyfikacji programu może być uproszczenie definicji funkcji `pierwsza`. Obecnie liczy ona, ile dzielników ma liczba. Gdy będą to dokładnie dwa dzielniki, to zawraca logiczną wartość `True`.

Wystarczy jednak sprawdzić, czy w zakresie od 2 do wartości o 1 mniejszej od sprawdzanej liczby nie ma żadnych jej dzielników, i wtedy zwrócić wartość `True`. Dzięki temu można pozbyć się zmiennej `licznik`.

```
def pierwsza(liczba):
    if liczba < 2:
        return False
    for i in range(2, liczba):
        if liczba % i == 0:
            return False
    return True
```

Na początku funkcji sprawdzam, czy liczba jest mniejsza od 2. Jeżeli tak, to funkcja zwróci wartość `False`. Choć z treści zadania wynika, że będziemy

rozpatrywać tylko liczby od 3 do 100, wprowadziłem ten fragment do funkcji, aby była ona uniwersalna.

Dobrze by było uprościć jeszcze ostatni fragment programu. Obecnie wypisuję tę parę liczb pierwszych, dla której zapamiętana różnica jest największa. Wymaga to dodatkowych zmiennych i przypisywania do nich odpowiednich wartości.

Zamiast tego utworzę dla każdej liczby listę wszystkich par liczb pierwszych i wyświetlę jedynie tę o indeksie 0. Przypomnijmy sobie, że zawsze pierwsza para tych liczb będzie spełniała warunki zadania.

Ostatecznie program będzie następujący:

```
with open("pary.txt") as plik:
    pary = [j.split() for j in plik]

liczby = [int(i[0]) for i in pary]
slowa = [i[1] for i in pary]

def pierwsza(liczba):
    for i in range(2, liczba):
        if liczba % i == 0:
            return False
    return True

p = [i for i in range(2, 100) if pierwsza(i)]

for i in liczby:
    if not i % 2:
        wynik = [[i, p[j], k] for j in range(len(p)) if
        ↪pierwsza(k := i-p[j])]
        print(wynik[0])
```

Udało się skrócić zapis z 25 wierszy do 18. Jedynie przedostatni wiersz jest trochę złożony, dlatego przeanalizujmy jego działanie:

```
[[i, p[j], k] for j in range(len(p)) if pierwsza(k := i-p[j])]
```

Pętla dla każdej liczby pierwszej

```
[[i, p[j], k] for j in range(len(p)) if pierwsza(k := i-p[j])]
```

Różnica między nią a liczbą

```
[[i, p[j], k] for j in range(len(p)) if pierwsza(k := i-p[j])]
```

Wynik zapamiętany w zmiennej `k`

```
[[i, p[j], k] for j in range(len(p)) if pierwsza(k := i-p[j])]
```

Czy wynik jest liczbą pierwszą?

`[[i, p[j], k] for j in range(len(p)) if pierwsza(k := i-p[j])]`
Utworzenie elementu listy

`[[i, p[j], k] for j in range(len(p)) if pierwsza(k := i-p[j])]`
Utworzenie całej listy

Rozwiązanie zadania jest gotowe. Wyniki jednak są wyświetlane w postaci list:

```
[68, 7, 61]
[24, 5, 19]
[48, 5, 43]
...
```

Aby uzyskać zapis podany w treści zadania, wystarczy w funkcji `print()` użyć operatora * (gwiazdka), który rozpakuje wyświetlaną listę:

```
print(*wynik[0])
```

Dzięki temu na ekranie ujrzę rozwiązanie zadania w takiej postaci:

```
68 7 61
24 5 19
48 5 43
...
```

Zadanie 4.2

Dla każdego słowa z pliku `pary.txt` znajdź długość najdłuższego spójnego fragmentu tego słowa złożonego z identycznych liter. Wypisz znalezione fragmenty słów i ich długości oddzielone spacją, po jednej parze w każdym wierszu. Jeżeli istnieją dwa fragmenty o takiej samej największej długości, podaj pierwszy z nich. Wyniki podaj w kolejności zgodnej z kolejnością danych w pliku `pary.txt`.

Przykład:
dla słowa *zxyzzzz* wynikiem jest:
zzzz 4
natomiast dla słowa *kkkabbb* wynikiem jest:
kkk 3

Dla danych z pliku `przyklad.txt` odpowiedzi podano w pliku `odp_przy↪klad.txt`.

W zadaniu nie wyjaśniono jednej z możliwości, mianowicie co wypisać, jeżeli w słowie nie ma powtarzających się po sobie liter. Nie ma więc spójnego fragmentu. Czy takie słowo pominąć, czy wypisać pierwszy znak słowa jako fragment jednoznakowy? Przyjąłem to drugie rozwiązanie.

Program musi sprawdzać dla każdego słowa dwa spójne fragmenty: bieżący oraz pierwszy z nich.

Bieżący spójny fragment słowa znajduję, porównując każdy znak z poprzednim. Nie robię tego z oczywistych względów jedynie dla pierwszego znaku słowa, czyli znaku mającego indeks 0. Jeżeli znaki są takie same, zapamiętuję pozycję ostatniego z nich i liczbę powtórzeń. Jeżeli trafię na niepowtarzający się znak, to ustawiam wartości początkowe zmiennych.

Za każdym razem, gdy znajdę bieżący spójny fragment słowa, porównuję go z poprzednio znalezionym i zapamiętuję dłuższy z nich. Jeżeli więc nie znajdę dłuższego, lecz jedynie o takiej samej długości, to pamiętany jest ten pierwszy.

```
with open("pary.txt") as plik:
    pary = [j.split() for j in plik]

liczby = [int(i[0]) for i in pary]
slowa = [i[1] for i in pary]

for i in slowa:
    licznik = maxlicznik = 1
    poz = maxpoz = 0
    for j in range(1, len(i)):
        if i[j] == i[j-1]:
            licznik += 1
            poz = j-licznik+1
            if maxlicznik < licznik:
                maxlicznik, maxpoz = licznik, poz
        else:
            licznik, poz = 1, 0
    print(i[maxpoz:maxpoz+maxlicznik], maxlicznik)
```

Program można nieznacznie zmodyfikować, by poprawić jego czytelność. Zamiast zapamiętywać położenie i długość pierwszego spójnego fragmentu słowa będę jedynie zapamiętywał odpowiednie fragmenty. Muszę jeszcze zmodyfikować deklarację zmiennych, aby dla każdego słowa przypisać pierwszy znak.

```
for i in slowa:
    fragment = maxfragment = i[0]
    for j in range(1, len(i)):
        if i[j] == i[j-1]:
            fragment = i[j] * (len(fragment)+1)
            if len(maxfragment) < len(fragment):
                maxfragment = fragment
```

```
        else:
            fragment = i[0]
    print(maxfragment, len(maxfragment))
```

Trochę podobne zadania były już na maturze z 2019 r. („Liczby”, zadanie 4.3) i na maturze próbnej z 2020 r. („Luki w ciągu”, zadanie 4.2).

Zadanie 4.3

Para (*liczba1*, *słowo1*) jest **mniejsza** od pary (*liczba2*, *słowo2*), gdy:

– *liczba1* < *liczba2*,

albo

– *liczba1* = *liczba2* oraz *słowo1* jest leksykograficznie (w porządku alfabetycznym) mniejsze od *słowo2*.

Przykład:
para (*1*, *bbbb*) jest mniejsza od pary (*2*, *aaa*), natomiast para (*3*, *aaa*) jest mniejsza od pary (*3*, *ab*).

Rozważ wszystkie pary (*liczba*, *słowo*) zapisane w wierszach pliku `pary.txt`, dla których **liczba jest równa długości słowa**, i wypisz spośród nich taką parę, która jest mniejsza od wszystkich pozostałych. W pliku `pary.txt` jest jedna taka para.

Dla danych z pliku `przyklad.txt` odpowiedzią jest:

6 abbbbc

Choć treść zadania jest dosyć długa i trochę zawiła, samo rozwiązanie jest proste.

W pierwszej kolejności należy sprawdzić dla każdej pary, czy liczba odpowiada długości słowa. Dla par, które spełnią to kryterium, sprawdzam, czy liczba dla danego indeksu jest mniejsza od zapamiętanej lub taka sama, ale słowo musi być „alfabetycznie wcześniejsze” względem zapamiętanego. Jeżeli tak, to zapamiętuję taką parę.

Na początku programu muszę jeszcze zdefiniować zmienne pomocnicze w taki sposób, by były górnymi wartościami, które są podane w treści głównej zadania.

```
with open("pary.txt") as plik:
    pary = [j.split() for j in plik]

liczby = [int(i[0]) for i in pary]
slowa = [i[1] for i in pary]
```

```
l = 100
s = "z" * 50
for i in range(len(liczby)):
    if liczby[i] == len(slowa[i]):
        if liczby[i]<l or (liczby[i]==l and slowa[i]<s):
            l = liczby[i]
            s = slowa[i]
print(l, s)
```

I to wszystko. Każde kolejne zadanie było prostsze od poprzedniego.

Aby było jeszcze ciekawiej, rozwiązanie zadania da się zapisać w jednym wierszu:

```
with open("pary.txt") as plik:
    pary = [j.split() for j in plik]

wynik = [(3-len(i[0]))*"0"+i[0]+i[1] for i in pary if int(i[0]) ==
↪len(i[1])]
print(int(min(wynik)[:3]), min(wynik)[3:])
```

Dodatkową wartością takiego rozwiązania jest pozbycie się wierszy tworzących listy `liczby` i `słowa`.

Aby program stał się bardziej czytelny, zapiszę go bez użycia listy składanej:

```
wynik = []
for i in pary:
    if int(i[0]) == len(i[1]):
        wynik += [(3-len(i[0]))*"0"+i[0]+i[1]]
```

Zmienna `wynik` musi zostać zadeklarowana, jeżeli nie użyję listy składanej. Ale nie ona jest tu istotna. W pętli `for` i warunku `if` wybieram pary spełniające kryteria zadania. Jest to ten sam zapis co wcześniej.

Tak naprawdę znaczenie ma ostatni wiersz powyższego przykładu. Tworzę nim listę zawierającą tekstowy zapis liczby i słowa dla danej pary. Dodatkowo uzupełniam tekstowy zapis liczby tak, by była zapisywana na trzech miejscach (w razie potrzeby dodaję zera z lewej strony liczby). Ustawiam liczbę do trzech miejsc, gdyż największą liczbą może być 100, składająca się z trzech cyfr.

Na przykład para (liczba – słowo):

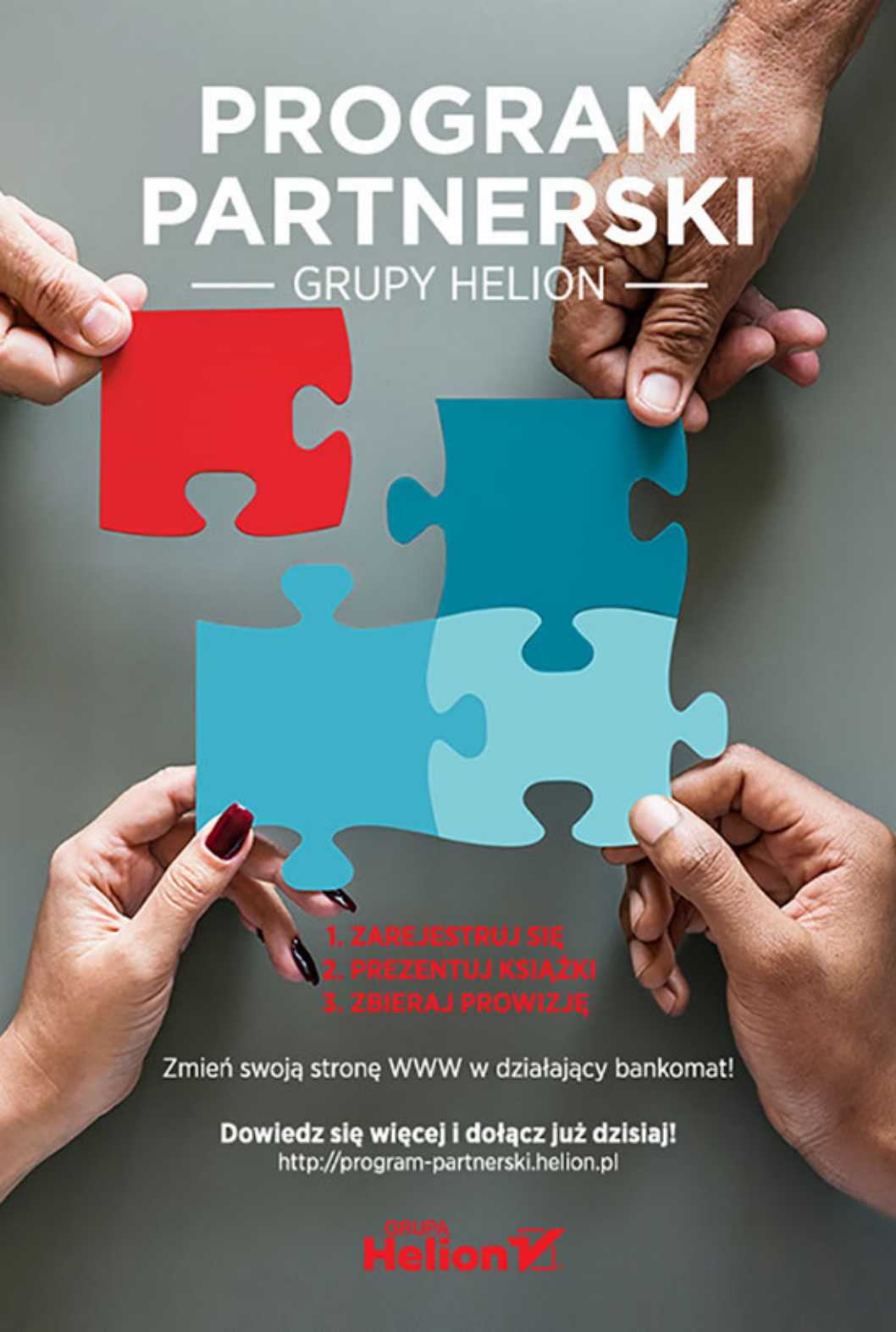
```
3 asd
```

zostanie zastąpiona tekstem:

```
003asd
```

Ponieważ otrzymam dzięki temu listę zawierającą same teksty, mogę użyć funkcji `min()`, aby wybrać alfabetycznie pierwszy z nich.

Na szczęście w kodzie ASCII znaki odpowiadające cyfrom są wcześniej niż litery, więc alfabetycznie też są mniejsze.


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA
Helion

Matura z Pythonem to nic trudnego!

- Ucz się!
- Analizuj!
- Programuj!

Nauka programowania na dobre zadomowiła się w szkołach, a umiejętności informatyczne są od lat sprawdzane na egzaminach maturalnych. Jej adepci zapewne zdają sobie sprawę, jakim wzięciem na rynku cieszą się osoby, które mogą się pochwalić znajomością Pythona. To język, który dzięki swoim możliwościom i wszechstronności znajduje zastosowanie w najróżniejszych dziedzinach nauki, przemysłu i biznesu.

Książka *Python na maturze. Rozwiązania i analiza wybranych zadań programistycznych* pozwoli Ci zapoznać się z prawdziwymi zadaniami maturalnymi z kilku ostatnich lat. Będziesz mógł je przeanalizować, a także opracować prawidłowe rozwiązania w języku Python. W efekcie nie tylko zdobędziesz i utrwalisz umiejętności programistyczne, lecz również — co nie mniej ważne — rozwiniesz zdolności w zakresie analizy problemów i wyboru właściwych rozwiązań.

- Treść zadań maturalnych
- Analiza problemów
- Praktyczne rozwiązania
- Uzasadnienie wybranych technik
- Typowe pułapki w zadaniach
- Składnia i instrukcje Pythona
- Zastosowanie języka w praktyce

Naucz się Pythona i zdaj maturę jak prymus!

Helion


helion.pl


HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-7744-8

Cena: 34,90 zł