PYTHON

ZBIÓR ZADAŃ Z ROZWIĄZANIAMI

Tomasz Jaśniewski

Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

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

ISBN: 978-83-289-1143-7

Printed in Poland.

Spis treści

Ważne! Przeczytaj!

Zadania w niniejszym zbiorze opisywane są skalą punktową. Chociaż jest ona subiektywna, to starałem się, by oddawała stopień trudności. Punkty pomagają określić potrzebną wiedzę i ilość pracy, którą trzeba włożyć w rozwiązanie zadania. Przy numerach zadania będzie można zauważyć liczbę punktów do zdobycia, otoczoną nawiasami kwadratowymi:

Zadanie 70. [3]

Niekiedy jednak zauważymy następującą notację:

Zadanie 62. [7:3,4]

Oznacza ona, że za zadanie jest przyznawane wprawdzie łącznie 7 punktów, ale jego treść stawia przed nami więcej pomniejszych wyzwań, które mają niższy poziom trudności. W praktyce zadanie na przykład z notacją **[7]** będzie trudniejsze do rozwiązania niż zadanie z notacją **[7:3,4]**, trudniej będzie zaprojektować do niego rozwiązanie, wpaść na pomysł tegoż rozwiązania itp. Dla tej rozdrobnionej notacji punkty będą występować dodatkowo w treści zadania, co wskazuje dokładnie na to, ile jest punktów za konkretne podpytanie lub za zrealizowanie danego fragmentu zadania. Zatem dla notacji **[7:3,4]** w którymś miejscu treści zadania pojawi się **[3]**, a w którymś **[4]**.

Dodatkowo każde zadanie opatrzone jest zbiorem **tagów**, które pozwalają zorientować się w wymaganiach odnośnie do zadania, sugerując przy tym obszar wiedzy, jaką można wykorzystać do rozwiązania. Podkreślam — można. Nie oznacza to, że trzeba albo że z czegoś korzystać nie wolno. Chyba że w zadaniu jawnie się czegoś zabroni lub o coś się w nim poprosi. Przykładowo, notacja:

Zdanie 19. [4:1,2,1] <str> <list> <rng> <time>

oznacza, że w zadaniu będzie przydatna wiedza o **napisach**, **listach**, uzyskiwaniu liczb **losowych** czy operowaniu **czasem**. Wszystkie tagi są opisane w dalszej części tego zbioru. Jeżeli jakieś zadanie będzie wymagać tylko całkowicie podstawowych wymagań, pojawi się tag **<abc>**. Jednak nie będzie go, gdy wymagane będą jeszcze inne umiejętności. Można założyć, że związane z tym tagiem umiejętności są niezbędne w każdym zadaniu.

Jeżeli tag jest poprzedzony znakiem zapytania, np. **<?regex>**, oznacza to sugestię, jakie umiejętności mogą się przydać (choć nie ma o nich mowy w zadaniu), a często oznacza, że proponowane przeze mnie rozwiązanie zawiera elementy związane z tagiem **<regex>** lub innym.

Czasami zadanie nie kończy się żadnym konkretnym wynikiem, nie generuje jasnej odpowiedzi. Jest tylko projektem, pewną mechaniką, której działania oczekujemy. Takie zadanie nie ma żadnych dodatkowych oznaczeń poza już wspomnianymi. W przypadku pozostałych zadań, w których pojawia się konkretna odpowiedź, np. 19 albo „Baba-Jaga", umieszczony jest również znacznik **{!}** — wykrzyknik w nawiasach klamrowych. W rozwiązaniu do zadania jest również podany wynik. Jeżeli odpowiedź jest zbyt obszerna (co wiązałoby się z wydrukowaniem bardzo dużego zbioru danych), nie znajdzie się w tym zbiorze, jednak będzie ją można zobaczyć po uruchomieniu kodu dostępnego na serwerze FTP itp. Niekiedy będzie też dostępny plik wynikowy z rozwiązaniem.

Czasami zadanie nie zawiera konkretnej odpowiedzi, jednak jego poprawność jest łatwo zauważalna. Przykładowo: gdy zadanie polega na narysowaniu ze znaków # pewnej określonej figury (np. prostokąta), to wprawdzie rozmiar prostokąta może nie być określony, ale wynik jest łatwo weryfikowalny — wystarczy uruchomić program i już sam rzut okiem pozwala określić poprawność jego działania. Inny przykład: należy podawać z klawiatury pewne wartości i obserwować wynik działania programu, co w przypadku dość prostych zadań pozwala na bieżąco weryfikować poprawność rozwiązania. W takich i podobnych przypadkach zadania zostaną opatrzone znacznikiem **{~}**.

Zarówno znacznik **{!}**, jak i **{~}** pozwalają szybko wyłapać te zadania, których rozwiązanie łatwo zweryfikować. Jest to przydatne np. dla nauczycieli planujących sprawdzian. Polecam!

Zadania są dość luźno poukładane, jednak ich stopień trudności generalnie wzrasta. Do rozwiązań stosuję zasadę: proste zadania -> proste techniki, trudniejsze zadania -> bardziej skomplikowane techniki. Czasami do zadania jest kilka rozwiązań, w których staram się ukazać różne pomysły lub wykorzystać inne biblioteki/funkcje. Niektóre zadania nie pozwalają na wykorzystywanie pewnych udogodnień języka Python. Celem jest zmuszenie do samodzielnego zaprojektowania algorytmu rozwiązującego zadanie. Rozwijamy się, gdy dogłębnie zaczynamy rozumieć pewne procesy. Nawet gdy później wykorzystujemy gotowe rozwiązania, to w procesie nauki niezastąpione jest takie samodzielne projektowanie algorytmów.

Jeśli chodzi o dane do zadań, dane wprowadzane z klawiatury itp., **zawsze zakładamy ich poprawność i nie musimy przeprowadzać żadnych testów, chyba że w zadaniu wprost zostaniesz poproszony o kontrolę wprowadzanych danych**, o sprawdzenie, czy są poprawne, lub gdy taki test będzie zasugerowany. Niekiedy w proponowanych rozwiązaniach sprawdzam poprawność danych, mimo że w zadaniu o to się nie prosi. Wybaczcie. Chciałem choć w kilku miejscach pokazać proste metody weryfikacji, np. poprawnego wpisania liczby.

Wprawdzie nie ma konieczności rozwiązywania zadań po kolei, to jednak w proponowanych rozwiązaniach kwestie omówione w zadaniu wcześniejszym, nie są omawianie w późniejszym. Sporo rozwiązań opatrzonych jest komentarzem, nie występuje on jednak zawsze. Opisem opatruję często zadania prostsze, gdyż osoby początkujące muszą zrozumieć to, co się kryje pod liniami kodu. Osoby, które już swobodniej posługują się językiem, raczej nie potrzebują wyjaśnień dotyczących

tego, co robi jakiś ciąg zagnieżdżonych funkcji. Ostatecznie — to zbiór zadań, a nie dokumentacja języka Python.

Jeżeli w zadaniach podawany jest jakiś zakres, np. od 1 do 100, domyślnie jest to zakres włącznie z 1 i 100. Zapisuję go często w postaci <1;100>. Jeżeli będzie inaczej, w zadaniu będzie to wyraźnie zaznaczone.

Niektóre rozwiązania są dość obszerne, jednak nie tworzyłem do nich modułów. Chciałem zachować pewną kompaktowość zgodnie z ideą: jedno zadanie jeden plik **.py* z rozwiązaniem.

W zadaniach w większości wykorzystywane są biblioteki standardowe: Python Standard Library. Staram się wykorzystywać podstawowe środki języka i nie używać dodatkowych bibliotek, które można doinstalować. W procesie edukacji lepiej samemu rozwiązać problem, projektując algorytm/klasę/mechanikę niż wykorzystywać od razu gotowe rozwiązanie, które wprawdzie daje dobry wynik, ale nie bardzo wiadomo, jak do niego doszło.

Wszystkie rozwiązania sprawdzałem i uruchamiałem na wersji Python 3.12.

Korzystałem ze środowiska PyCharm w wersji 2023.2.5.

Tagi

- **<abc>** — wszelkie podstawowe instrukcje sterujące i podstawowe operacje. Instrukcja warunkowa; pętle; znajomość typów `int`, `float`, `bool`, `str`; instrukcja `match`; porównania; działania matematyczne; znajomość podstawowych operatorów; operacje logiczne; operacje prostego wyświetlania na konsoli (`print`) i wprowadzania danych z klawiatury; proste konwersje np. z `int` na `str` i odwrotnie; importowanie potrzebnych funkcjonalności z bibliotek; rozumienie pojęcia referencji; bardzo podstawowe operacje na tekstach. Każde zadanie wymaga tych umiejętności, dlatego ten tag pojawi się tylko tam, gdzie te umiejętności/wymagania są jedynymi.
- **<mat>** — pewne umiejętności z zakresu matematyki (co najwyżej poziom podstawowy szkoły średniej, choć może zdarzyć się wyjątek).
- **<str>** — znajomość różnych bardziej zaawansowanych operacji na napisach/tekstach.
- **<we/wy>** — wszelkie podstawowe operacje wejścia/wyjścia związane z obsługą plików, w tym zapis i odczyt danych. W praktyce zawiera również tag **<str>**. Zadania oznaczone tym tagiem bardzo często będą od nas wymagać obróbki danych pobranych z pliku, umieszczania ich w kontenerach, analizy, segregowania, sortowania itp. Można więc uznać, że z tym tagiem stowarzyszony jest również tag **<kont>**.
- **<kont>** — kontenery. Obsługa krotek, zbiorów, list, słowników i znajomość funkcji, które pozwalają na nich operować: dodawać, usuwać elementy, przeglądać kontenery za pomocą odpowiednich pętli itp.;
- **<list>**, **<set>**, **<dict>**, **<tuple>** — czasami tak określam konkretny kontener.
- W ramach tego znacznika posługujemy się funkcjami np. `map`, `filter` czy `sum`.
- **<sd>** — potrzebna jest wiedza o strukturach danych: stosie, kolejce, kolejce priorytetowej, liście jednokierunkowej, dwukierunkowej, drzewie binarnym, grafie itp.
- **<rng>** — umiejętność generowania liczb losowych.
- **<regex>** — znajomość wyrażeń regularnych.
- **<f>** — tworzenie funkcji.
- **<fy>** — tworzenie funkcji z `yield`, **generatorów**, **wyrażeń generujących**.
- **<c>** — tworzenie klas, bardzo podstawowe korzystanie z konstruktora i prostych metod.
- **<C>** — tworzenie klas, wykorzystanie zaawansowane (np. przeciążanie operatorów, destruktory, metody statyczne).
- **<time>** — umiejętność obsługi czasu, kalendarza itp.
- **<+tag>** — w niektórych zadaniach wykorzystuję wiedzę wykraczającą poza standardową bibliotekę. Przykładowo, w pewnych zadaniach wykorzystuję format tekstowego zapisu bitowych plików graficznych *.PBM*. Aby ich używać, trzeba poszerzyć zakres swojej wiedzy. Niekiedy zamiast ogólnego **<+>** pojawi się bardziej szczegółowy opis, np. **<+.pbm>** albo **<+fractions>** czy **<+enum>**,

który oznacza wykorzystanie jakichś dodatkowych bibliotek języka Python, jednak niewymagających instalacji.

- **<algorytm>** — zadania zakładają znajomość pewnych algorytmów i technik algorytmicznych, np. algorytmów sortujących, metod zachłannych, metod dziel i zwyciężaj. W zadaniach nie muszą być wykorzystywane znane (nazwane) algorytmy, ale mogą one zawierać ich części lub być do nich podobne. Mogą to być też zadania, które według mojej subiektywnej oceny wymagają tego, aby wpaść na sprytny pomysł, zaprojektować nietrywialne rozwiązanie.
- **<symulacja>** — zadania, w których należy przeprowadzić symulację kolejnych zdarzeń, by otrzymać wynik. Często towarzyszy im znacznik **<rng>**.
- **<json>** — wiedza o używaniu formatu JSON, znajomość podstawowych funkcji biblioteki json.
- **<blob!>** — tego nie da się wyjaśnić. To trzeba przeżyć. A przynajmniej rozwiązać.
- **<err>** — obsługa błędów i wyjątków itp.
- **<CMOK>** — zadanie, które subiektywnie uważam za słodziutkie i ciekawe. Po jego rozwiązaniu oprócz satysfakcji czujesz, że wszystko będzie dobrze.
- **<?tag>** — dowolny tag, np. **<?regex>**, poprzedzony znakiem zapytania, oznacza, że w rozwiązaniu do tego zadania wykorzystałem pewne techniki/biblioteki, które nie są wymagane w zadaniu, ale nie są też zabronione. W tych rozwiązaniach możesz poznać nowe funkcjonalności i ciekawe sztuczki.
- **<p>** — ten tag oznacza projekt. Rozumiem to jako bardzo otwarte zadanie, prezentujące pewną funkcjonalność, którą należy zrealizować swobodnie i twórczo. Proponowane rozwiązania stają się wręcz delikatną sugestią, gdyż zadanie można zrealizować dowolnymi środkami, rozbudowywać je i poszerzać o nowe, nieuwzględnione w zadaniach funkcjonalności.

Zadanie 69. [3] <rng> <we/wy> {~}

Losuj liczby całkowite z przedziału <0;1000> tak długo, aż zostanie wylosowana liczba 1000. Każdą liczbę wylosowaną podzielną przez 10 dodaj do pliku *69_liczby10.txt* z zachowaniem zasady: jedna liczba w wierszu.

Zadanie 70. [3] <we/wy> <list> {~}

Utwórz program, który odczyta wszystkie liczby z pliku tekstowego *70_liczby.txt* i wyświetli je. Umieść liczby z pliku na liście. Wyświetl listę. Liczby zawarte w pliku spełniają następujące reguły:

- każda liczba jest typu całkowitego (`int`);
- w jednym wierszu znajduje się jedna liczba;
- w każdym wierszu pliku jest liczba (nie ma pustych wierszy, ale gdyby się jednak trafiły, program powinien je ignorować/omijać).

Program powinien działać dla dowolnego pliku z wymienionymi zasadami, nie tylko przykładowego pliku *70_liczby.txt*.

Zadanie 71. [4] <we/wy> <list> <str> {~}

Utwórz program, który odczyta wszystkie liczby z pliku tekstowego *71_liczby.txt* i wyświetli je. Liczby umieść na liście. Zasady obowiązujące w pliku:

- każda liczba jest typu całkowitego `int`;
- w jednym wierszu znajdują się dwie liczby oddzielone średnikiem;
- nie powinno być błędnych i pustych wierszy, ale puste wiersze należy pomijać.

Program powinien działać dla dowolnego pliku z wymienionymi zasadami, nie tylko przykładowego pliku *71_liczby.txt*.

Zadanie 72. [5] <we/wy> <list> <str> {~}

Utwórz program, który odczyta wszystkie liczby i teksty z pliku *72_dane.txt*, a następnie je wyświetli. Zasady obowiązujące w pliku:

- każda liczba jest typu całkowitego `int`;
- w jednym wierszu znajdują się trzy liczby oddzielone średnikiem, następnie jest dowolnie długi napis oddzielony od liczb średnikiem, np.:

```
1;231;3423;dowolnie długi napis
321;345;67;inny długi napis
-351;211;0;potwornie długi napis, inny niż wszystkie
```

- nie powinno być wierszy błędnych ani pustych, ale pusty wiersz należy pomijać.

Pobrane liczby umieść na liście i wyświetl. Pobrane napisy umieść w jednym długim napisie niezawierającym znaków końca wiersza. Wyświetl ten napis. Program powinien działać dla dowolnego pliku z wymienionymi zasadami, nie tylko przykładowego pliku *72_dane.txt*.

Zadanie 73. [6] <we/wy> <list> <str> <?err> {~}

Utwórz program, który odczyta wszystkie liczby i teksty z pliku *73_dane.txt*, a następnie je wyświetli. Zasady obowiązujące w pliku:

- każda liczba w pliku jest typu `float` (np. `123.21`);
- w jednym wierszu znajdują się liczby i napisy oddzielone średnikiem, jednak ich kolejność i liczba w wierszu nie są znane i mogą być różne (nie ma schematu), np.:

```
1;231;3423;dowolnie długi napis;345456;inny napis
napis;321;345;67;inny długi napis;8893;krótki tekst;123123;1212;12;tekst
-31.21;napis kosmiczny;0.001;-1.0;2
```

Wyświetl pobrane liczby w kolejności odczytu. Wyświetl sumę tych liczb w zaokrągleniu do dwóch miejsc po przecinku. Pobrane napisy umieść na liście z napisami, każdy napis otocz znakami `#`. Pokaż zawartość listy. Program powinien działać dla dowolnego pliku z wymienionymi zasadami, nie tylko przykładowego pliku *73_dane.txt*.

Zadanie 74. [10:6,4] <p> <we/wy> <str> <json> <c> <f> <list>

Utwórz własną klasę **C** zawierającą własności typu: `int`, `float`, `bool` i `str`. Następnie zaprojektuj metody/mechaniki, które:

- zapiszą (lub dopiszą) w pliku zmienną/obiekt utworzonej klasy (musi istnieć możliwość zapisu wielu takich elementów w jednym pliku);
- będą odczytywać z pliku dane, na podstawie których utworzone zostaną obiekty klasy C (w pliku może być wiele obiektów klasy C).

Załadowane z pliku dane/obiekty umieść na liście.

Sugeruję przy zapisie do pliku stosować regułę: jeden obiekt klasy C umieszczony jest w jednej linii pliku. Sposób zapisania elementu jest dowolny — zakłada radosną kreatywność. Smutna jest niedopuszczalna **[6]**.

Gdy zaprojektujesz swój sposób zapisu obiektów do pliku i odczytu ich z niego, spróbuj dodać takie metody/mechaniki, które zrealizują te same cele, ale wykorzystując do tego format **JSON** (patrz biblioteka `json`) **[4]**.

Zadanie 75. [3] <f> <str> {~}

Utwórz funkcję, która zwraca `True`, jeśli otrzymany w argumencie napis (`str`) jest **palindromem**, lub `False` w przeciwnym razie. Palindrom to napis, który zapisany od tyłu i od przodu jest taki sam. Przyjmijmy, że napis pusty jest palindromem.

Zadanie 76. [3] <f> <str> {~}

Utwórz funkcję, która jako argument otrzymuje napisy `A` i `B`. Zwraca liczbę wystąpień napisu `A` w napisie `B`.

Zadanie 77. [6,3:3] <f> {~}

Napisz własną funkcję podobną w działaniu do konwersji napisu na liczbę całkowitą:

```
int('40')  # ma działać jak to polecenie, ale tak nie możesz zrobić
```

która to funkcja zamienia napis `str` wyglądający jak liczba na liczbę typu `int` i ją zwraca **[3]**. Zaprojektuj także funkcję, która wcześniej sprawdzi, czy możliwa jest zamiana tekstu na liczbę. Spróbuj nie wykorzystywać przechwytywania wyjątków, jak w rozwiązaniu zadania 73. **[3]** Zwróć uwagę na to, żeby poprawnie zamienić napisy zawierające poprzedzający minus, np. `'-123'` zamień na `-123`.

Zadanie 78. [4] <f> <str> <?mat> {~}

Napisz własną funkcję, która liczbę całkowitą (`int`) zamienia na napis i zwraca ten napis (`str`). Nie możesz wykorzystać konwersji Pythona:

```
str(34)  # tak nie możesz zrobić
```

Zadanie 79. [10:5,5] <f> <algorytm> {~}

Napisz własną funkcję konwertującą, która pobiera napis reprezentujący liczbę binarną, np. `"11110"`, a następnie zwraca liczbę całkowitą dziesiętną (tu 30), która tę liczbę reprezentuje. Utwórz drugą funkcję, która wykonuje operację odwrotną: jako argument otrzymuje liczbę całkowitą (np. 30), a następnie zwraca napis zawierający jej reprezentację binarną (tu `"11110"`). Ograniczamy się tylko do liczb nieujemnych. Nie można korzystać z gotowych narzędzi i mechanizmów konwertujących — tworzymy swój własny **[5]**.

Następnie zaprojektuj dwie uniwersalne funkcje konwertujące liczby jak wyżej, ale dla dowolnej podstawy systemowej (od 2 do 16, czyli nie tylko dla systemu dwójkowego, ale także dla innych, do szesnastkowego włącznie). Pierwsza funkcja, nazwana np. `Any2Dec()`, powinna zamieniać napis w dowolnym systemie, np. trójkowym, ósemkowym czy szesnastkowym, na liczbę dziesiętną, całkowitą, dodatnią. Druga funkcja, nazwana np. `Dec2Any()`, powinna zamieniać i zwracać liczbę nieujemną całkowitą na liczbę w dowolnym systemie od dwójkowego po szesnastkowy **[5]**.

Zadanie 80. [3] <str> <?regex> {~}

Ciąg tekstowy `T` składa się z losowych cyfr i liter alfabetu angielskiego. Znajdź wszystkie podciągi możliwie jak najdłuższe, składające się z samych cyfr. Przykład: dla ciągu `T = "abd65asd7891das1d"` będą to podciągi `"65"`, `"7891"`, `"1"`.

Zadanie 81. [5] <str> <algorytm> <?regex> {~}

Ciąg tekstowy `T` składa się z losowych cyfr i liter alfabetu angielskiego. Znajdź przynajmniej jeden najdłuższy podciąg ciągu `T`, który składa się z samych cyfr, przy czym te cyfry tworzą podciąg niemalejący, np. w ciągu `T = "dfgnqeiut98tna1223v0w3r123334asdsh"` istnieje przynajmniej jeden najdłuższy podciąg niemalejący składający się z samych cyfr `"123334"`. Uważaj, podciąg `"12333"` też jest niemalejący, ale nie jest najdłuższy!

Proponowane rozwiązania i odpowiedzi do zadań

W tej części zbioru znajdują się proponowane rozwiązania. Pod nimi mogą znajdować się:

- prawidłowe odpowiedzi — jeżeli nie są zbyt długie;
- fragmenty odpowiedzi i informacja, w jakim pliku znajduje się pełna odpowiedź — jeżeli odpowiedzi to duże ilości danych i ich drukowanie nie miałoby sensu;
- informacje o plikach z wizualizacją wyników — jeżeli zadanie wymaga utworzenia np. plików graficznych *.pbm*;
- nic — jeżeli zadanie ma charakter projektu, losowej symulacji itp. i nigdy nie generuje powtarzalnego wyniku.

Przypominam również, że rozwiązania do zadań są **propozycją**. Nie muszą być rozwiązaniem optymalnym. Niektóre rozwiązania wydają się nawet rozbudowane na siłę. Jest to celowe, ponieważ czasami walor edukacyjny jest najważniejszy i podczas rozwiązywania zadania chciałem zaprezentować jakąś przydatną funkcjonalność.

Podsumowując: jeżeli Twoje rozwiązanie jest lepsze, wykorzystujesz w nim inne funkcje/pomysły — to super. Jeżeli jednak przejrzysz proponowane rozwiązania, to możesz natknąć się w nich na pouczające wykorzystanie różnych mechanik języka.

Początkowe zadania są opatrzone znacznie większą ilością komentarzy. Zakładam, że osoby rozpoczynające naukę potrzebują trochę więcej wyjaśnień, a nawet pokazania kilku sposobów rozwiązania tego samego problemu. Moim zdaniem to pomaga w nauce, w bardziej kreatywnym dostrzeganiu rozwiązania i projektowaniu go. Z czasem jednak rozwiązania zadań mają coraz mniej komentarzy.

Uwaga! Pod proponowanymi rozwiązaniami (czyli pod kodem Pythona) znajdują się szczegółowe odpowiedzi (o ile zadanie takich wymaga), fragmenty tego, co rozwiązania wyświetlają, uwagi, wizualizacje. Często jednak nie są to pełne informacje, które wyświetla program, a tylko ich fragment, który pozwala zorientować się w poprawności wyniku. Wiąże się to z ilością danych wynikowych, która niekiedy jest duża. Niemniej każdorazowe uruchomienie programów pozwoli zobaczyć nie tylko wyniki, ale także różne obliczenia pośrednie, informacje dodatkowe, listingi wyjaśniające itp., itd.

Przykład:

(końcówka kodu z rozwiązaniem)

```
…
numer = 1
for w in range(0, 799):
  for k in range(0, 799):
    kod4 = ''
    kod4 += mapa_lasu[w][k:k + 2] + mapa_lasu[w + 1][k:k + 2]
    if kod4 == kod_chatki:
      print('Chatka nr ', numer,
            f' położona w punkcie wiersz:kolumna = {w + 1}:{k + 1}')
      numer += 1
```

Kilka początkowych linii rozwiązania:

Wynik

```
Chatka nr  1  położona w punkcie wiersz:kolumna = 5:95
Chatka nr  2  położona w punkcie wiersz:kolumna = 5:429
Chatka nr  3  położona w punkcie wiersz:kolumna = 11:219
Chatka nr  4  położona w punkcie wiersz:kolumna = 11:293
Chatka nr  5  położona w punkcie wiersz:kolumna = 11:671
Chatka nr  6  położona w punkcie wiersz:kolumna = 13:35
…
```

Wszystkie rozwiązania znajdują się tutaj: *https://ftp.helion.pl/przyklady/pyzbza.zip*.

Zadanie 69.

```
from random import randint

n = -1
with open('69_liczby10.txt', 'w', encoding='utf-8') as f:
  while n != 1000:
    n = randint(0, 1000)
    if n % 10 == 0:
      f.write(str(n) + '\n')
```

Zadanie 70.

```
L = []
with open('70_liczby.txt', encoding='utf-8') as f:
  for linia in f:
    print(linia.strip())
    if len(linia.strip()) == 0: continue
    L.append(int(linia.strip()))
print(L)
```

Zadanie 71.

```
L = []
with open('71_liczby.txt', encoding='utf-8') as f:
  for linia in f:
    if len(linia.strip()) == 0: continue
    a, b = linia.split(';')
    print(a, b.strip())
    L.append(int(a))
    L.append(int(b))

print(L)
```

Wynik

```
430 130
380 130
770 290
650 430
890 380
210 1000
[430, 130, 380, 130, 770, 290, 650, 430, 890, 380, 210, 1000]
```

Zadanie 72.

```
L = []
N = ''
with open('72_dane.txt', encoding='utf-8') as f:
  for linia in f:
    if len(linia.strip()) == 0: continue  # pomijamy pustą
    x1, x2, x3, n = linia.split(';')
    print(x1, x2, x3)
    L.append(int(x1))
    L.append(int(x2))
    L.append(int(x3))
    N += n.strip()  # strip() usuwa białe znaki z początku i końca, zwraca kopię n
print(L, N, sep='\n')
```

Wynik

```
1 231 3423
321 345 67
-351 211 0
[1, 231, 3423, 321, 345, 67, -351, 211, 0]
dowolnie długi napisinny długi napispotwornie długi napis inny niż wszystkie
```

Zadanie 73.

```
N = []
suma = 0
with open('73_dane.txt', encoding='utf-8') as f:
  for linia in f:
    if len(linia) == 0: continue  # pomijamy pustą
    fragmenty = linia.split(';')
    for e in fragmenty:
      e = e.strip()
      # można sprawdzać, czy e składa się tylko z cyfr i przecinka/kropki,
      # ale zastosuję inną sztuczkę.
      try:
        e = float(e)  # próba konwersji
        print(e, end=' ')  # wyświetlam wycięte liczby float
        suma += e
      except ValueError:
        N.append(
          '#' + e + '#')  # strip() usuwa białe znaki z początku i końca, zwraca kopię n
print()
print(N)
print('Suma', round(suma, 2))
```

Wynik

```
1.0 231.0 3423.0 345456.0 321.0 345.0 67.0 8893.0 123123.0 1212.0 12.0 -31.21 0.001
-1.0 2.0
['#dowolnie długi napis#', '#inny napis#', '#napis#', '#inny długi napis#', '#krótki
tekst#', '#tekst#', '#napis kosmiczny#']
Suma 483053.79
```

Zadanie 74.

```
import json

class C:
  # klasa C
  file = 'obiektyKlasyC.txt'  # dane mojego pomysłu będą w tym pliku
  jsonFile = 'obiektyKlasyC.json'  # dane będą w tym pliku, jeżeli to będzie format JSON

  def __init__(self, ustawi: int = 0, ustawf: float = 0.0, ustawb: bool = True,
               ustaws: str = ''):
    """Konstruktor z 4 własnościami"""
    self.i = ustawi
    self.f = ustawf
    self.b = ustawb
    self.s = ustaws
```

```
  def zapisz(self, type='w'):
    """Zapis obiektu do pliku. Domyślnie nowy plik."""
    with open(self.file, type, encoding='utf-8') as f:
      f.write(str(self.i) + ':')
      f.write(str(self.f) + ':')
      f.write(str(self.b) + ':')
      f.write(self.s + '\n')
    print('Zapis wykonany.')

  def dodaj(self):
    """Dopisanie do pliku."""
    self.zapisz(type='a')
    print('Dopisek wykonany')

  def __str__(self):
    """Pozwoli wyświetlać nasz obiekt klasy C"""
    return ('###' +
            str(self.i) + ' ' + str(self.f) + ' ' + str(self.b) + ' ' +
            str(self.s) + ' ' + '###')

  @staticmethod
  def zaladujListe():
    """Ładowanie obiektów z pliku. Zwraca listę obiektów klasy C."""
    L = []
    print('Rozpoczynam ładowanie:')
    with open(C.file, 'r', encoding='utf-8') as f:
      for ln in f:
        ln = ln.strip()
        L.append(C(*ln.split(':', 3)))   # maxsplit=3 pozwoli na ':' w C.s
    return L

  @staticmethod
  def zaladujJsonListe():
    """Ładowanie obiektów z pliku. Zwraca listę obiektów klasy C."""
    L = []
    print('Rozpoczynam ładowanie:')
    with open(C.jsonFile, 'r', encoding='utf-8') as f:
      for ln in f:
        ln = ln.strip()
        L.append(C(*json.loads(ln)))
    return L

  def jsonZapisz(self, type='w'):
    """Zapis obiektu do pliku. Domyślnie nowy plik."""
    temp = json.dumps([self.i, self.f, self.b, self.s])
    print(temp)
    with open(self.jsonFile, type, encoding='utf-8') as f:
      f.write(temp + '\n')
    print('Zapis wykonany.')

  def jsonDodaj(self):
    """Dopisanie do pliku."""
    self.jsonZapisz(type='a')
    print('Dopisek wykonany')

o1 = C(10, 2.5, True, 'rabarbar:na:rowerze')
o2 = C(4, -5.5, False, 'marynowana rozwielitka')
o1.zapisz()   # utworzy nowy plik, zapisze o1
o2.dodaj()   # dopisze do pliku o2
print(*C.zaladujListe(), sep='\n')   # załadunek i prezentacja dzięki __str__()
```

```
# wersje JSON

o3 = C(4, -2.0, True, 'kropla piasku')
o4 = C(-2, 200.01, False, 'szum ciszy')
o3.jsonZapisz()  # zapis
o4.jsonDodaj()  # dopisanie
print(*C.zaladujJsonListe(), sep='\n')  # załadunek i prezentacja dzięki __str__()
```

Zadanie 75.

```
# sposób 1
def czyPalindrom(s: str):
  return s[0::] == s[::-1]

print(czyPalindrom('kot'), czyPalindrom('kajak'), czyPalindrom('yyyYOYyyy'),
      czyPalindrom(''))

# sposób 2 (z własnym algorytmem sprawdzającym)
def czyPalindrom2(s: str):
  poczatek = 0
  koniec = len(s) - 1
  while poczatek <= koniec:
    if s[poczatek] != s[koniec]:
      return False
    poczatek += 1
    koniec -= 1
  return True

print(czyPalindrom2('kot'), czyPalindrom2('kajak'), czyPalindrom2('yyyYOYyyy'),
      czyPalindrom2(''))
```

Zadanie 76.

```
def ileAwB(A: str, B: str):
  ile = 0
  poz = -1
  while True:
    poz = B.find(A, poz + 1)
    if poz > -1:
      ile += 1
    else:
      break
  return ile

print(ileAwB('jo', 'jojowanie jest fajowe'))
# Nie wiesz, co to jojowanie? Ja też.
```

Zadanie 77.

```
def czyMozliwyInt(kandydat: str):
  if not len(kandydat):
    return False
  if kandydat[0] in ['-', '+']:
    znak = kandydat[0]
    kandydat = kandydat[1::]
  for i in kandydat:
    if i not in '0123456789':
      return False
  return True
```

```
def Str2Int(n: str):
  if not czyMozliwyInt(n):
    print('Konwersja niemożliwa')
    return
  znak = ''
  if n[0] in ['-', '+']:
    znak = n[0]
    n = n[1::]
  L = 0
  d = 0
  for i in n[::-1]:
    L += (ord(i) - 48) * 10 ** d
    d += 1
  if znak == '-':
    return -L
  else:
    return L

# przykłady
print(Str2Int('178'))
print(Str2Int('0222'))
print(Str2Int('-123'))
print(Str2Int('-0981'))
print(Str2Int(''))
```

Zadanie 78.

```
def Int2Str(n: int):
  plus = True
  if n < 0: plus = False
  n = abs(n)
  cyfry = '0123456789'
  wynik = ''
  while n > 0:
    r = n % 10
    wynik = cyfry[r] + wynik
    n //= 10
  if wynik == '': return 0
  if plus:
    return wynik
  else:
    return '-' + wynik
  return wynik

print(Int2Str(10))
print(Int2Str(214))
print(Int2Str(0))
print(Int2Str(-990))
```

Zadanie 79.

```
def Bin2Dec(bin: str):
  wynik = 0
  pot = 0
  for i in bin[::-1]:
    wynik += (ord(i) - 48) * 2 ** pot   # ord(i) - 48 zamienia '0' na 0, '1' na 1
    pot += 1
  return wynik
```

```
print(Bin2Dec('11110'))  # 30
print(Bin2Dec('011110'))  # 30
print(Bin2Dec('0'))  # 0

def Dec2Bin(i: int):
  if i == 0: return '0'
  wynik = ''
  while i > 0:
    r = i % 2
    i //= 2
    wynik = chr(r + 48) + wynik   # chr(r+48) zamienia r=0 na '0', r=1 na '1'
  return wynik

print(Dec2Bin(30))  # '11110'
print(Dec2Bin(7))  # '111'
print(Dec2Bin(12))  # '1100'
print(Dec2Bin(0))  # '0'

# uniwersalna funkcja zamienia liczby w postaci 'ff' na dziesiętne 255 itp.
def Any2Dec(n: str, podstawa: int = 2):
  alfabetCyfr = '0123456789abcdef'   # cyfry systemów od binarnego po szesnastkowy
  wynik = 0
  pot = 0
  for cyfra in n[::-1]:
    wynik += alfabetCyfr.find(cyfra) * podstawa ** pot
    pot += 1
  return wynik

print(Any2Dec('11110', 2))   # bin-> 30
print(Any2Dec('77', 8))   # oct-> 63
print(Any2Dec('c8', 16))   # hex-> 200
print(Any2Dec('44', 5))   # piątkowy -> 24

# uniwersalna funkcja zamieniająca liczbę dziesiętną na postać w innym systemie (jako str)
def Dec2Any(n: int, podstawa: int = 2):
  if n == 0: return '0'   # zero to '0' w każdym systemie
  alfabetCyfr = '0123456789abcdef'   # cyfry systemów od binarnego po szesnastkowy
  wynik = ''
  while n > 0:
    r = n % podstawa
    wynik = alfabetCyfr[r] + wynik
    n //= podstawa
  return wynik

print(Dec2Any(200, 16))
print(Dec2Any(63, 8))
print(Dec2Any(30, 2))
print(Dec2Any(24, 5))
# zero to nawet w systemie dziewiątkowym wygląda tak samo...
print(Dec2Any(0, 9))
```

Zadanie 80.

```
T = input('Podaj tekst=')

# sposób 1 (prostymi środkami)
dl = 0
for poz, znak in enumerate(T):
```

```
  if znak in '0123456789':
    dl += 1
  if dl > 0 and znak not in '0123456789':
    print(T[poz - dl:poz])
    dl = 0
  if dl > 0 and poz == len(T) - 1:
    poz += 1
    print(T[poz - dl:poz])

# sposób 2 (wyrażenia regularne)
import re

print(*list(re.findall('[0-9]+', T)), sep='\n')
```

Zadanie 81.

```
T = input('Podaj tekst=')
# T = 'dfgnqeiut98tna1223v0w3r123334asdsh' # odkomentuj, by sprawdzić przykład z zadania

# sposób 1 (wyrażenia regularne)
import re

kandydaci = [[len(x), x] for x in list(
  filter(lambda x: len(x) > 0, re.findall('0*1*2*3*4*5*6*7*8*9*', T)))]
kandydaci.sort(reverse=True)
if (len(kandydaci)):
  print(kandydaci[0][1], 'o największej długości', kandydaci[0][0])

# sposób 2 (prostymi środkami)
cyfry = '0123456789'
poczatek = 0
dlugosc = 0
maxdl = 1
maxpocz = 0
for i in range(0, len(T)):
  znak = T[i]
  cyfra = True
  if znak not in cyfry:
    cyfra = False
  else:
    cyfra = True
  if cyfra and dlugosc == 0:
    poczatek = i
    dlugosc = 1
  elif cyfra and dlugosc > 0:
    if ord(T[i - 1]) - 48 <= ord(T[i]) - 48:
      dlugosc += 1
      if maxdl < dlugosc:
        maxdl = dlugosc
        maxpocz = poczatek
    else:
      poczatek = i
      dlugosc = 1
  else:
    dlugosc = 0
    poczatek = i
print(T[maxpocz:maxpocz + maxdl], 'o największej długości', maxdl)
```

Python w praktyce zadań i rozwiązań

Język programowania ogólnego przeznaczenia Python należy obecnie do najpopularniejszych na świecie. Skąd się bierze jego fenomen? Niewątpliwie kluczowe znaczenie ma tu bardzo czytelna składnia, mocno zbliżona do składni języka naturalnego. Czyni to Pythona dość łatwym do opanowania, także dla początkujących. Osoby bardziej doświadczone doceniają go za wszechstronność. Pythona można używać w różnych dziedzinach, takich jak analiza danych, sztuczna inteligencja, tworzenie stron internetowych, automatyka i automatyzacja, pisanie aplikacji mobilnych i wiele innych. Dodajmy do tego rozbudowany pakiet bibliotek standardowych i oto mamy (niemal) idealny język programowania.

A jak się go nauczyć? Najlepiej w praktyce. Rozwiązując zadania i weryfikując własne rezultaty z podanymi w tym zbiorze rozwiązaniami. Autor przygotował ten zeszyt ćwiczeń tak, by zawrzeć w nim wyzwania nieco prostsze, przeznaczone dla mniej zaawansowanych adeptów Pythona, jak i bardziej złożone, wymagające lepszej znajomości zagadnienia. Każde zadanie ma wyraźnie określony stopień trudności, a także jest opatrzone tagami pozwalającymi zorientować się w wymaganiach omawianej tematyki.

Ta książka pozwoli Ci zacząć przygodę z Pythonem, którego znajomość już dziś możemy określić kompetencją przyszłości, i to pożądaną nie tylko w branży IT.

PYTHON

Helion

helion.pl

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

KOD KORZYŚCI
Sięgnij po więcej! ▶

ISBN 978-83-289-1143-7

9 788328 911437

Cena: 59,00 zł