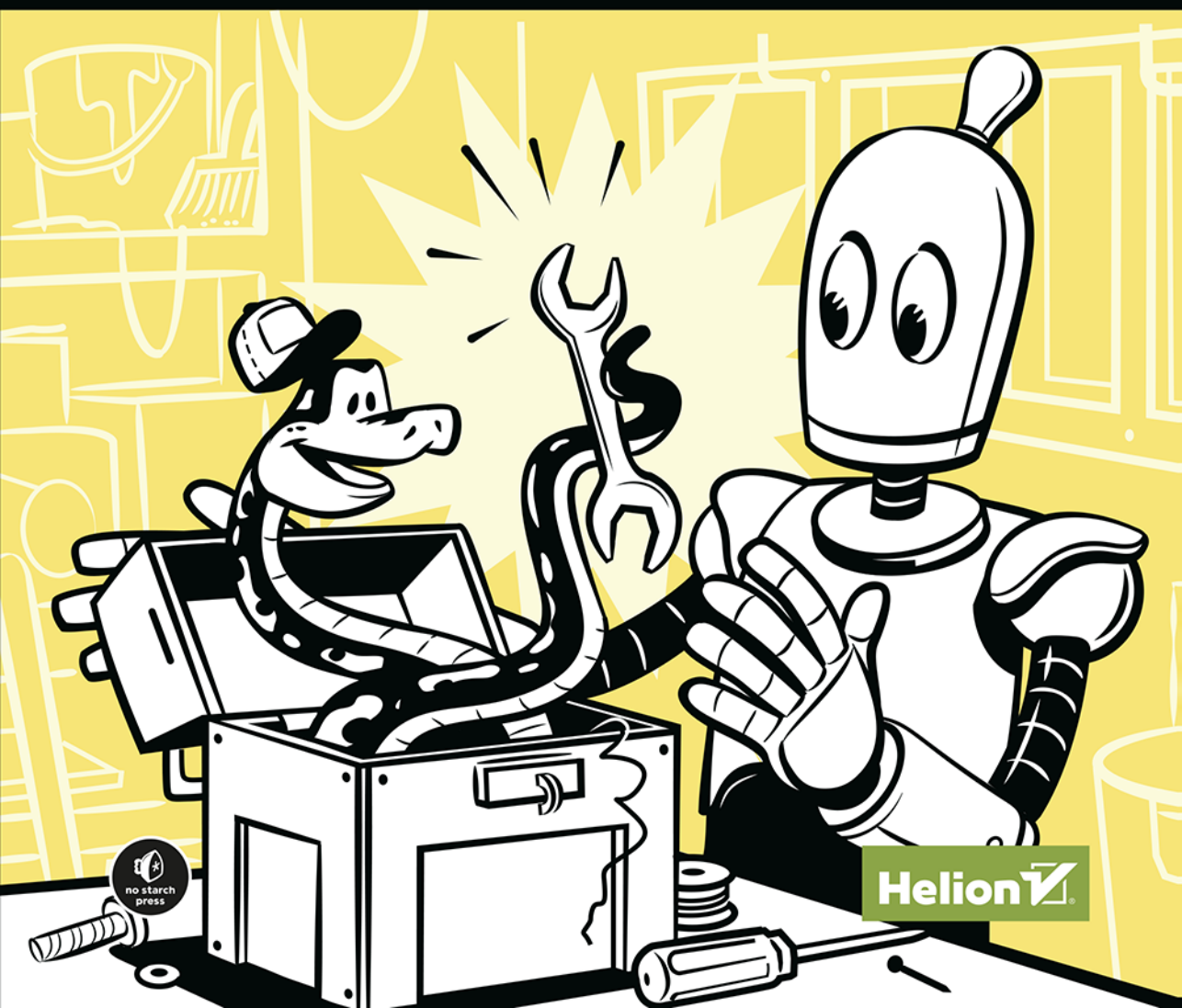


WIELKA KSIĘGA MAŁYCH PROJEKTÓW W PYTHONIE

81 ŁATWYCH PRAKTYCZNYCH PROGRAMÓW

AL SWEIGART



no starch
press

Helion

Tytuł oryginału: The Big Book of Small Python Projects: 81 Easy Practice Programs

Tłumaczenie: Anna Mizerska

ISBN: 978-83-283-8861-1

Copyright © 2021 by Al Sweigart. Title of English-language original: The Big Book of Small Python Projects: 81 Easy Practice Programs, ISBN 9781718501249, published by No Starch Press Inc. 245 8th Street, San Francisco, California United States 94103.

The Polish-language edition Copyright © 2022 by Helion S.A. under license by No Starch Press Inc. All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion S.A.

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <https://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<https://ftp.helion.pl/przyklady/wiksma.zip>

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<https://helion.pl/user/opinie/wiksma>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

WSTĘP	11
PROJEKT #1. BAJGLE: Odgadnij trzycyfrową liczbę w oparciu o podpowiedzi	25
<i>Poćwicz używanie stałych</i>	
PROJEKT #2. PARADOKS DNIA URODZIN: Określ prawdopodobieństwo, że dwie osoby w różnej wielkości grupach mają taką samą datę urodzin	30
<i>Użyj wbudowanego w Pythonie modułu <code>datetime</code></i>	
PROJEKT #3. BITMAPOWA WIADOMOŚĆ: Wyświetl wiadomość na ekranie w postaci dwuwymiarowej bitmapy	34
<i>Stosuj wielolinijkowe łańcuchy znaków</i>	
PROJEKT #4. OCZKO: Klasyczna gra karciana przeciwko krupierowi ze sztuczną inteligencją	38
<i>Naucz się używać znaków <code>Unicode</code> i punktów kodowych</i>	
PROJEKT #5. ANIMACJA LOGO DVD: Symulacja kolorowego logo DVD odbijającego się na ekranie	46
<i>Korzystaj ze współrzędnych i kolorowego tekstu</i>	
PROJEKT #6. SZYFR CEZARA: Prosty szyfr używany tysiące lat temu	52
<i>Zamieniaj litery na liczby i na odwrót, by wykonywać obliczenia na tekście</i>	
PROJEKT #7. ŁAMACZ SZYFRU CEZARA: Program odczytujący wiadomości napisane szyfrem Cezara bez znajomości klucza	56
<i>Zaimplementuj algorytmy typu <i>brute force</i></i>	
PROJEKT #8. GENERATOR KALENDARZA: Twórz kartki z kalendarza dla danego roku i miesiąca	59
<i>Użyj modułu <code>datetime</code> oraz typu <code>timedelta</code></i>	
PROJEKT #9. MARCHEWKA W PUDEŁKU: Prosta gra na blefowanie dla dwóch graczy	64
<i>Twórz obrazy za pomocą kodów <code>ASCII</code></i>	

PROJEKT #10. CHO-HAN: Hazardowa gra w kości z feudalnej Japonii	70
<i>Poćwicz używanie liczb losowych i słowników</i>	
PROJEKT #11. GENERATOR CHWYTLIWYCH NAGŁÓWKÓW:	
Generator śmiesznych nagłówków	74
<i>Poćwicz operacje na łańcuchach znaków i generowanie tekstu</i>	
PROJEKT #12. PROBLEM COLLATZA: Odkryj najprostszy niemożliwy	
do rozstrzygnięcia problem matematyczny	79
<i>Poznaj operator modulo</i>	
PROJEKT #13. GRA W ŻYCIE CONWAYA: Klasyczny automat komórkowy,	
którego proste zasady pozwalają na złożone zachowanie	82
<i>Korzystaj z danych słownikowych i współrzędnych ekranu</i>	
PROJEKT #14. ODLICZANIE: Licznik w stylu wyświetlacza siedmiosegmentowego	86
<i>Poćwicz importowanie stworzonych przez siebie modułów</i>	
PROJEKT #15. GŁĘBOKA JASKINIA: Animacja tunelu, który nigdy się nie kończy	89
<i>Powielaj łańcuchy znaków i stosuj prostą matematykę</i>	
PROJEKT #16. DIAMENTY: Algorytm rysowania różnej wielkości diamentów	92
<i>Poćwicz swoje umiejętności tworzenia algorytmów rysujących</i>	
PROJEKT #17. MATEMATYKA I KOSTKI: Graficzna gra matematyczna	
z rzutami kostką	96
<i>Stosuj słowniki do zapisywania współrzędnych ekranu</i>	
PROJEKT #18. RZUT KOSTKĄ: Narzędzie do odczytywania rzutów kostką	
używaną w grze <i>Dungeons & Dragons</i> w celu generowania losowych liczb	102
<i>Analizuj tekst w celu wyszukania kluczowych łańcuchów znaków</i>	
PROJEKT #19. ZEGAR CYFROWY: Zegar z wyświetlaczem w stylu kalkulatora	106
<i>Generuj liczby, które pasują do informacji z modułu <code>datetime</code></i>	
PROJEKT #20. STRUMIEŃ CYFROWY: Przewijany wygaszacz ekranu,	
który przypomina film <i>Matrix</i>	109
<i>Eksperymentuj z różnymi prędkościami animacji</i>	
PROJEKT #21. WIZUALIZACJA DNA: Niekończąca się podwójna helisa	
stworzona za pomocą znaków ASCII przedstawiająca łańcuch DNA	112
<i>Korzystaj z szablonów tekstowych i losowo wygenerowanego tekstu</i>	

PROJEKT #22. KACZĄTKO: Mieszaj i dobieraj łańcuchy znaków, by za pomocą znaków ASCII stworzyć różne kaczuski	115
<i>Wykorzystaj programowanie zorientowane obiektowo w celu stworzenia modelu kacuszki</i>	
PROJEKT #23. ZNIKOPIS: Rysuj linie za pomocą kursora	121
<i>Korzystaj ze współrzędnych ekranu i względnych ruchów w różnych kierunkach</i>	
PROJEKT #24. ROZKŁAD NA CZYNNIKI: Wyznacz wszystkie czynniki danej liczby	127
<i>Użyj operatora modulo i modułu math</i>	
PROJEKT #25. SZYBKI STRZAŁ: Sprawdź swój refleks, by przekonać się, czy jesteś najszybciej klikającą osobą na Dzikim Zachodzie	130
<i>Poznaj bufor klawiatury</i>	
PROJEKT #26. FIBONACCI: Generuj liczby sławnego ciągu Fibonacciego	133
<i>Zaimplementuj podstawowy algorytm matematyczny</i>	
PROJEKT #27. AKWARIUM: Kolorowe, animowane akwarium wykonane ze znaków ASCII	136
<i>Korzystaj ze współrzędnych ekranu, kolorów tekstu i struktur danych</i>	
PROJEKT #28. FLOODER: Próba wypełnienia całej planszy jednym kolorem	144
<i>Zaimplementuj algorytm flood fill</i>	
PROJEKT #29. POŻAR LASU: Symulacja rozprzestrzeniania się pożaru w lesie	151
<i>Stwórz symulację z możliwością doboru parametrów</i>	
PROJEKT #30. CZWÓRKI: Gra planszowa, w której dwóch graczy stara się ułożyć w rzędzie cztery płytki	156
<i>Stwórz strukturę danych, która naśladuje grawitację</i>	
PROJEKT #31. ODGADNIJ LICZBĘ: Klasyczna gra w odgadywanie liczby	162
<i>Poznaj podstawowe koncepcje programowania</i>	
PROJEKT #32. NAIWNIAK: Zabawny program, który zajmie na długo naiwne osoby	165
<i>Sprawdzaj wprowadzone przez użytkownika dane i stosuj pętle</i>	
PROJEKT #33. ŁAMACZ HASEŁ: Odkryj hasło na podstawie wskazówek	167
<i>Upiększ grę, by była ciekawsza</i>	
PROJEKT #34. WISIELEC I GILOTYNA: Klasyczna zgadywanka słowna	173
<i>Wykonuj operacje na łańcuchach znaków i używaj znaków ASCII</i>	

PROJEKT #35. SIATKA HEKSAGONALNA: Generuj wzory za pomocą programu i znaków ASCII	179
<i>Wykorzystaj pętle do stworzenia powtarzalnych wzorów</i>	
PROJEKT #36. KLEPSYDRA: Prosty silnik spadającego piasku	182
<i>Symuluj grawitację i wykrywaj kolizję</i>	
PROJEKT #37. GŁODNE ROBOTY: Unikaj zabójczych robotów w labiryncie	188
<i>Stwórz prostą sztuczną inteligencję sterującą ruchami robota</i>	
PROJEKT #38. OSKARŻAM! Detektywistyczna gra, w której trzeba ustalić, kto kłamie, a kto mówi prawdę	195
<i>Wykorzystaj struktury danych do generowania związków pomiędzy podejrzanymi, miejscami i wskazówkami związanymi z przedmiotami</i>	
PROJEKT #39. MRÓWKA LANGTONA: Automat komórkowy, gdzie mrówki poruszają się zgodnie z prostymi zasadami	203
<i>Odkryj, jak proste zasady tworzą złożone graficzne wzory</i>	
PROJEKT #40. HAKERSKI SLANG: Przetłumacz wiadomości na h4k3r\$]<i \$!@ng	209
<i>Analizuj tekst i wykonuj operacje na łańcuchach znaków</i>	
PROJEKT #41. SZCZĘŚLIWE GWIAZDY: Gra kościana	212
<i>Poćwicz stosowanie znaków ASCII i prawdopodobieństwa</i>	
PROJEKT #42. MAGICZNA KULA: Program odpowiadający „tak”/„nie” na Twoje pytania dotyczące przyszłości	219
<i>Dodaj ozdobniki do podstawowego tekstu, by wyglądał bardziej atrakcyjnie</i>	
PROJEKT #43. MANKALA: Starożytna gra planszowa z Mezopotamii dla dwóch osób	223
<i>Twórz rysunki za pomocą znaków ASCII i używaj szablonów tekstowych do rysowania planszy</i>	
PROJEKT #44. LABIRYNT 2D: Spróbuj uciec z labiryntu	230
<i>Odczytaj dane labiryntu z plików tekstowych</i>	
PROJEKT #45. LABIRYNT 3D: Spróbuj uciec z labiryntu... w 3D!	236
<i>Zmieniaj wielolinijkowe łańcuchy znaków, aby wyświetlać trójwymiarowe widoki</i>	
PROJEKT #46. SYMULATOR MILIONA RZUTÓW KOSTKĄ: Odkryj prawdopodobieństwo na podstawie wyników miliona rzutów zestawem kostek	245
<i>Dowiedz się, jak komputery wykonują obliczenia na dużej liczbie danych</i>	

PROJEKT #47. GENERATOR SZTUKI MONDRIANA: Twórz geometryczne rysunki w stylu Pieta Mondriana	248
<i>Zaimplementuj algorytm generujący dzieła sztuki</i>	
PROJEKT #48. PARADOKS MONTY’EGO HALLA: Symulacja paradoksu Monty’ego Halla znanego z teleturnieju telewizyjnego	255
<i>Zbadaj prawdopodobieństwo z kozami ze znaków ASCII</i>	
PROJEKT #49. TABLICZKA MNOŻENIA: Wyświetl tabliczkę mnożenia o wymiarach 12×12 pól	262
<i>Poćwicz rozmieszczanie tekstu</i>	
PROJEKT #50. 99 BUTELEK: Wyświetl powtarzające się zwrotki piosenki	264
<i>Wykorzystaj pętle i szablony z łańcuchów znaków, by utworzyć tekst</i>	
PROJEKT #51. 99 BUUTELLEK: Wyświetl powtarzające się zwrotki piosenki, której tekst staje się z każdym wersem coraz bardziej zniekształcony	267
<i>Wykonuj operacje na łańcuchach znaków w celu zniekształcenia tekstu</i>	
PROJEKT #52. SYSTEMY LICZBOWE: Zbadaj liczby w systemie binarnym i szesnastkowym	271
<i>Użyj funkcji Pythona do zamiany zapisu liczb między różnymi systemami liczbowymi</i>	
PROJEKT #53. UKŁAD OKRESOWY PIERWIASTKÓW: Interaktywna baza danych pierwiastków chemicznych	275
<i>Odczytaj pliki w formacie CSV, by wgrać dane do programu</i>	
PROJEKT #54. ŚWIŃSKA ŁACINA: Tłumacz wiadomości na inśkąśwaj acinętaj	279
<i>Przeszukuj tekst i wykonuj operacje na łańcuchach znaków</i>	
PROJEKT #55. LOTERIA: Symulacja przegranej w loterii tysięcy razy	282
<i>Odkryj prawdopodobieństwo za pomocą liczb losowych</i>	
PROJEKT #56. LICZBY PIERWSZE: Wyznacz liczby pierwsze	287
<i>Poznaj zasady matematyczne i skorzystaj z modułu math</i>	
PROJEKT #57. PASEK POSTĘPU: Przykładowy pasek postępu do wykorzystania w innych programach	290
<i>Twórz animacje z wykorzystaniem klawisza Backspace</i>	
PROJEKT #58. TĘCZA: Prosta animacja tęczy dla początkujących	294
<i>Stwórz bardzo prostą animację</i>	
PROJEKT #59. PAPIER, KAMIEŃ, NOŻYCE: Klasyczna gra z użyciem dłoni dla dwóch osób	297
<i>Zaimplementuj proste zasady gry w postaci programu komputerowego</i>	

PROJEKT #60. PAPIER, KAMIEŃ, NOŻYCE (WERSJA ZWYCIĘZCY): Wersja gry, w której gracz nie może przegrać	301
<i>Stwórz złudzenie losowości</i>	
PROJEKT #61. SZYFR ROT13: Najprostszy szyfr do kodowania i odkodowania wiadomości	305
<i>Zamieniaj litery na liczby i na odwrót, by wykonywać obliczenia na tekście</i>	
PROJEKT #62. OBRACAJĄCY SIĘ SZEŚCIAN: Animacja obracającego się sześcianu	308
<i>Dowiedz się, jak wykonywać obroty w przestrzeni trójwymiarowej, oraz poznaj algorytmy rysujące</i>	
PROJEKT #63. KRÓLEWSKA GRA Z UR: Gra z Mezopotamii licząca sobie 5000 lat	315
<i>Używaj znaków ASCII i szablonów z łańcuchów znaków, by rysować planszę do gry</i>	
PROJEKT #64. WYŚWIETLACZ SIEDMIOSEGMENTOWY: Wyświetlacz taki jak ten używany w kalkulatorach i kuchenkach mikrofalowych	324
<i>Twórz moduły do wykorzystania w innych programach</i>	
PROJEKT #65. LŚNIĄCY DYWAN: Generuj za pomocą programu dywan jak z filmu <i>Lśnienie</i>	328
<i>Twórz powtarzające się wzory za pomocą pętli</i>	
PROJEKT #66. PROSTY SZYFR PODSTAWIENIOWY: Bardziej zaawansowana wersja szyfru Cezara	331
<i>Wykonuj różne operacje na tekście</i>	
PROJEKT #67. SINUSOIDALNA WIADOMOŚĆ: Wyświetlanie przewijanej wiadomości w kształcie fali	336
<i>Stwórz animację z użyciem funkcji trygonometrycznych</i>	
PROJEKT #68. PRZESUWANKA: Klasyczna układanka 4×4	339
<i>Wykorzystaj strukturę danych do przedstawienia stanu układanki</i>	
PROJEKT #69. WYŚCIG ŚLIMAKÓW: Szybkie wyścigi ślimaków!	344
<i>Mierz odstęp między ślimakami wykonanymi za pomocą znaków ASCII</i>	
PROJEKT #70. SOROBAN — Japoński abakus: Komputerowa symulacja przyrządu do liczenia używanego długo przed powstaniem komputerów	348
<i>Narysuj liczydło za pomocą znaków ASCII i szablonów tekstowych</i>	
PROJEKT #71. POWTARZANIE DŹWIĘKÓW: Staraj się zapamiętywać coraz dłuższe sekwencje dźwięków	354
<i>Odtwarzaj pliki dźwiękowe z poziomu programu napisanego w Pythonie</i>	

PROJEKT #72. TEKST KANCIASTOPORTY: Przetłumacz wiadomość na tEkSt KaNcIaStOpOrtY	358
<i>Zmieniaj wielkości liter w łańcuchach znaków</i>	
PROJEKT #73. SUDOKU: Klasyczna łamigłówka z gazet	361
<i>Stwórz łamigłówkę za pomocą struktury danych</i>	
PROJEKT #74. ZAMIANA TEKSTU NA MOWĘ: Spraw, by Twój komputer do Ciebie przemówił!	368
<i>Wykorzystaj systemowy silnik zamiany tekstu na mowę</i>	
PROJEKT #75. TRZY KARTY: Podstępna gra w trzy karty, w którą przegrał już niejeden turysta	371
<i>Wykonuj operacje na strukturze danych w oparciu o losowe ruchy</i>	
PROJEKT #76. KÓŁKO I KRZYŻYK: Klasyczna gra dla dwóch osób, w której gracze naprzemiennie stawiają X i O	376
<i>Stwórz strukturę danych i funkcje pomocnicze</i>	
PROJEKT #77. WIEŻE HANOI: Klasyczna łamigłówka z krążkami układanymi na sobie według określonych zasad	380
<i>Wykorzystaj stos w celu przedstawienia bieżącego stanu układanki</i>	
PROJEKT #78. PODCHWYTLIWE PYTANIA: Quiz z pozornie łatwymi pytaniami i zaskakującymi odpowiedziami	385
<i>Przeszukaj podany przez użytkownika tekst w celu znalezienia określonych słów kluczowych</i>	
PROJEKT #79. 2048: Przyjemna układanka polegająca na dopasowywaniu płytek	391
<i>Stwórz symulację grawitacji, by płytki „spadały” w dowolnym kierunku</i>	
PROJEKT #80. SZYFR VIGENÈRE’A: Szyfr tak zaawansowany, że nikomu nie udało się go złamać przez setki lat, dopóki nie powstały komputery	398
<i>Wykonuj bardziej zaawansowane operacje na tekście</i>	
PROJEKT #81. WIADRA Z WODĄ: Uzyskaj dokładnie 4 litry wody przez wylewanie i napełnianie trzech wiader	402
<i>Twórz sztukę za pomocą znaków ASCII i szablonów tekstowych</i>	
A. SPIS ETYKIET	407
B. TABELA ZNAKÓW	411

#11

Generator chwytliwych nagłówków



NASZA STRONA INTERNETOWA MUSI PRZYCIĄGAĆ LUDZI, BY ZOBACZYLI UMIESZCZONE NA NIEJ REKLAMY! ALE WYMYŚLENIE POMYSŁOWEJ, NIEPOWTARZALNEJ TREŚCI JEST ZBYT TRUDNE. NA SZCZĘŚCIE dzięki generatorowi chwytliwych nagłówków możemy sprawić, by komputer wymyślał szokujące, fałszywe nagłówki. Wszystkie są niskich lotów, ale czytelnicy nie zwracają na to uwagi. Ten program generuje tyle nagłówków, ile potrzebujesz, na bazie szablonów w stylu gry Mad Libs (gry polegającej na tworzeniu dziwnych i śmiesznych zdań na podstawie wzoru).

W tym programie jest dużo tekstu wykorzystywanego w nagłówkach, ale sam kod jest prosty i odpowiedni dla początkujących¹.

¹ Program jest w angielskojęzycznej wersji, gdyż polska gramatyka oparta jest na innych zasadach, które odgrywają tutaj istotną rolę. W ramach wyzwania możesz pokusić się o próbę przerobienia tego programu, tak by generował polskie nagłówki — *przyj. tłum.*

Działanie programu

Gdy uruchomisz program *clickbait.py*, uzyskasz następujący rezultat:

Generator chwytliwych nagłówków

Autor: Al Sweigart, al@inventwithpython.com

Nasza angielskojęzyczna strona musi przyciągać ludzi, by zobaczyli umieszczone na niej reklamy!

Wpisz liczbę nagłówków do wygenerowania:

> **1000**

This New York Video Game Didn't Think Robots Would Take Her Job. She Was Wrong.

What Cats Don't Want You To Know About Clowns

15 Gift Ideas to Give Your Telephone Psychic From Florida

--*ucięte*--

6 Reasons Why Telephone Psychics Are More Interesting Than You Think (Number 1 Will Surprise You!)

Are Millenials Killing the Dog Industry?

Are Millenials Killing the Plastic Straw Industry?

Jak to działa?

Ten program ma kilka funkcji w celu generowania różnych rodzajów chwytliwych nagłówków. Każda z nich pobiera losowe słowa ze STATES, NOUNS, PLACES, WHEN i innych list. Następnie funkcje umieszczają te słowa w szablonowych łańcuchach znaków za pomocą metody `format()`, przed zwróceniem gotowego łańcucha znaków. Jest to podobne działanie jak w grze Mad Libs, z tym że komputer wypełnia puste pola, dzięki czemu program może generować tysiące chwytliwych nagłówków w ciągu zaledwie kilku sekund.

001: *"""Generator chwytliwych nagłówków, autor: Al Sweigart, al@inventwithpython.com*

002: *Generator chwytliwych nagłówków dla bezdusznej zawartości Twojej strony.*

003: *Kod pobrany ze strony <https://ftp.helion.pl/przyklady/wiksma.zip>.*

004: *Etykiety: długi, dla początkujących, zabawny, słowa"""*

005:

006: `import random`

007:

008: *# Definicja stałych:*

009: `OBJECT_PRONOUNS = ['Her', 'Him', 'Them']`

010: `POSSESSIVE_PRONOUNS = ['Her', 'Him', 'Their']`

011: `PERSONAL_PRONOUNS = ['She', 'He', 'They']`

012: `STATES = ['California', 'Texas', 'Florida', 'New York', 'Pennsylvania',`

013: `'Illinois', 'Ohio', 'Georgia', 'North Carolina', 'Michigan']`

014: `NOUNS = ['Athlete', 'Clown', 'Shovel', 'Paleo Diet', 'Doctor', 'Parent',`

015: `'Cat', 'Dog', 'Chicken', 'Robot', 'Video Game', 'Avocado',`

016: `'Plastic Straw', 'Serial Killer', 'Telephone Psychic']`

017: `PLACES = ['House', 'Attic', 'Bank Deposit Box', 'School', 'Basement',`

018: `'Workplace', 'Donut Shop', 'Apocalypse Bunker']`

019: `WHEN = ['Soon', 'This Year', 'Later Today', 'RIGHT NOW', 'Next Week']`

```

020:
021:
022: def main():
023:     print('Generator chwytliwych nagłówków')
024:     print('Autor: Al Sweigart, al@inventwithpython.com')
025:     print()
026:
027:     print('Nasza angielskojęzyczna strona musi przyciągać ludzi,  

    ↳by zobaczyli umieszczone na niej reklamy!')
028:     while True:
029:         print('Wpisz liczbę nagłówków do wygenerowania:')
030:         response = input('> ')
031:         if not response.isdecimal():
032:             print('Proszę podać liczbę.')
033:         else:
034:             numberOfHeadlines = int(response)
035:             break # Wyjdź z pętli, gdy zostanie podana wartość liczbowa.
036:
037:     for i in range(numberOfHeadlines):
038:         clickbaitType = random.randint(1, 8)
039:
040:         if clickbaitType == 1:
041:             headline = generateAreMillenialsKillingHeadline()
042:         elif clickbaitType == 2:
043:             headline = generateWhatYouDontKnowHeadline()
044:         elif clickbaitType == 3:
045:             headline = generateBigCompaniesHateHerHeadline()
046:         elif clickbaitType == 4:
047:             headline = generateYouWontBelieveHeadline()
048:         elif clickbaitType == 5:
049:             headline = generateDontWantYouToKnowHeadline()
050:         elif clickbaitType == 6:
051:             headline = generateGiftIdeaHeadline()
052:         elif clickbaitType == 7:
053:             headline = generateReasonsWhyHeadline()
054:         elif clickbaitType == 8:
055:             headline = generateJobAutomatedHeadline()
056:
057:         print(headline)
058:     print()
059:
060:     website = random.choice(['website', 'blog', 'Facebuuk', 'Googles',
061:                               'Facesbook', 'Tweedie', 'Pastagram'])
062:     when = random.choice(WHEN).lower()
063:     print('Post these to our', website, when, 'or you\'re fired!')
064:
065:
066: # Każda z tych funkcji zwraca różny rodzaj nagłówka:
067: def generateAreMillenialsKillingHeadline():
068:     noun = random.choice(NOUNS)
069:     return 'Are Millenials Killing the {} Industry?'.format(noun)
070:
071:
072: def generateWhatYouDontKnowHeadline():
073:     noun = random.choice(NOUNS)
074:     pluralNoun = random.choice(NOUNS) + 's'

```

```

075:     when = random.choice(WHEN)
076:     return 'Without This {}, {} Could Kill You {}'.format(noun, pluralNoun,
    ↪when)
077:
078:
079: def generateBigCompaniesHateHerHeadline():
080:     pronoun = random.choice(OBJECT_PRONOUNS)
081:     state = random.choice(STATES)
082:     noun1 = random.choice(NOUNS)
083:     noun2 = random.choice(NOUNS)
084:     return 'Big Companies Hate {}! See How This {} {} Invented a Cheaper
    ↪{}'.format(pronoun, state, noun1, noun2)
085:
086:
087: def generateYouWontBelieveHeadline():
088:     state = random.choice(STATES)
089:     noun = random.choice(NOUNS)
090:     pronoun = random.choice(POSSESSIVE_PRONOUNS)
091:     place = random.choice(PLACES)
092:     return 'You Won\'t Believe What This {} {} Found in {} {}'.format(state,
    noun, pronoun, place)
093:
094:
095: def generateDontWantYouToKnowHeadline():
096:     pluralNoun1 = random.choice(NOUNS) + 's'
097:     pluralNoun2 = random.choice(NOUNS) + 's'
098:     return 'What {} Don\'t Want You To Know About {}'.format(pluralNoun1,
    ↪pluralNoun2)
099:
100:
101: def generateGiftIdeaHeadline():
102:     number = random.randint(7, 15)
103:     noun = random.choice(NOUNS)
104:     state = random.choice(STATES)
105:     return '{} Gift Ideas to Give Your {} From {}'.format(number, noun, state)
106:
107:
108: def generateReasonsWhyHeadline():
109:     number1 = random.randint(3, 19)
110:     pluralNoun = random.choice(NOUNS) + 's'
111:     # Zmienna number2 powinna być większa niż zmienna number1:
112:     number2 = random.randint(1, number1)
113:     return '{} Reasons Why {} Are More Interesting Than You Think (Number {}
    ↪Will Surprise You!}'.format(number1, pluralNoun, number2)
114:
115:
116: def generateJobAutomatedHeadline():
117:     state = random.choice(STATES)
118:     noun = random.choice(NOUNS)
119:
120:     i = random.randint(0, 2)
121:     pronoun1 = POSSESSIVE_PRONOUNS[i]
122:     pronoun2 = PERSONAL_PRONOUNS[i]
123:     if pronoun1 == 'Their':
124:         return 'This {} {} Didn\'t Think Robots Would Take {} Job. {} Were
    ↪Wrong.'.format(state, noun, pronoun1, pronoun2)

```

```

125:     else:
126:         return 'This {} {} Didn\'t Think Robots Would Take {} Job. {} Was
           ↳Wrong.'.format(state, noun, pronoun1, pronoun2)
127:
128:
129: # Jeśli program został uruchomiony (a nie zaimportowany), rozpocznij grę:
130: if __name__ == '__main__':
131:     main()

```

Po przepisaniu kodu źródłowego i uruchomieniu go kilka razy spróbuj w ramach eksperymentowania wprowadzić kilka zmian w programie. Możesz również samodzielnie spróbować wykonać następujące zadania:

- Dodaj inne rodzaje chwytliwych nagłówków.
- Dodaj nowe kategorie słów, poza rzeczownikami (NOUNS), miejscami (PLACES) i tak dalej.

Odkrywanie programu

Postaraj się znaleźć odpowiedzi na poniżej podane pytania. Spróbuj wprowadzić zmiany w kodzie i uruchomić ponownie program, by zobaczyć, jaki efekt to przyniosło.

1. Jaki komunikat błędu otrzymasz, gdy usuniesz wyrażenie `numberOfHeadlines = int(response)` w linii 34. lub umieścisz przed nią znacznik komentarza?
2. Jaki komunikat błędu otrzymasz, gdy zamienisz `int(response)` na `response` w linii 34.?
3. Jaki komunikat błędu otrzymasz, gdy zmienisz kod w linii 19. na `WHEN = []`?

#12

Problem Collatza



PROBLEM COLLATZA, ZWANY RÓWNIEŻ PROBLEMEM $3n + 1$, TO NAJPROSTSZY, NIEMOŻLIWY DO UDOWODNIENIA PROBLEM MATEMATYCZNY. (ALE NIE MARTW SIĘ, PROGRAM SAM W SOBIE JEST WYSTARCZAJĄCO ŁATWY DLA POCZĄTKUJĄCYCH). Zaczynając od liczby n , zastosuj trzy zasady, by otrzymać kolejną liczbę w ciągu:

1. Jeśli liczba n jest parzysta, kolejna liczba n to $n : 2$.
2. Jeśli liczba n jest nieparzysta, kolejna liczba n to $n \cdot 3c + 1$.
3. Jeśli liczba n to 1, zatrzymaj. W przeciwnym razie powtórz.

To ogólne sformułowanie, ale jak dotąd nieudowodnione matematycznie — każda liczba początkowa wcześniej czy później będzie miała wartość 1. Więcej informacji na temat problemu Collatza znajdziesz na stronie https://pl.wikipedia.org/wiki/Problem_Collatza.

Działanie programu

Gdy uruchomisz program *collatz.py*, uzyskasz następujący rezultat:

Problem Collatza lub problem $3n + 1$
Autor: Al Sweigart, al@inventwithpython.com

Problem Collatza to ciąg liczb wygenerowany na podstawie
liczby początkowej n z zastosowaniem trzech zasad:

--ucięte--

Podaj liczbę początkową (większą niż 0) lub KONIEC:

> 26

26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Problem Collatza lub problem $3n + 1$

Autor: Al Sweigart, al@inventwithpython.com

--ucięte--

Podaj liczbę początkową (większą niż 0) lub KONIEC:

> 27

27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242, 121, 364,
182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700, 350, 175, 526, 263,
790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167, 502, 251, 754, 377,
1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479, 1438, 719, 2158, 1079, 3238,
1619, 4858, 2429, 7288, 3644, 1822, 911, 2734, 1367, 4102, 2051, 6154, 3077,
9232, 4616, 2308, 1154, 577, 1732, 866, 433, 1300, 650, 325, 976, 488, 244, 122,
61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

Jak to działa?

Operator modulo `%` pomaga określić, czy liczba jest parzysta, czy nieparzysta. Pamiętaj, że to operator wyznaczania reszty z dzielenia. Podczas gdy 23 podzielone przez 7 to 3, reszta 2, 23 modulo 7 to po prostu 2. Parzyste liczby dzielone przez 2 nie mają reszty z dzielenia, a dzielenie liczb nieparzystych przez 2 daje resztę równą 1. Jeśli `n` jest parzyste, warunek `if n % 2 == 0`: w linii 33. jest prawdziwy (`True`). Jeśli wartość `n` jest nieparzysta, warunek jest nieprawdziwy (`False`).

```
01: """Problem Collatza, autor: Al Sweigart, al@inventwithpython.com
02: Generuje liczby dla ciągu Collatza na podstawie podanej liczby początkowej.
03: Więcej informacji na stronie https://pl.wikipedia.org/wiki/Problem_Collatza.
04: Kod pobrany ze strony https://ftp.helion.pl/przyklady/wiksma.zip.
05: Etykiety: króciutki, dla początkujących, matematyka"""
06:
07: import sys, time
08:
09: print('Problem Collatza lub problem 3n + 1
10: Autor: Al Sweigart, al@inventwithpython.com
11:
12: Problem Collatza to ciąg liczb wygenerowany na podstawie
13: liczby początkowej n z zastosowaniem trzech zasad:
14:
15: 1) Jeśli liczba n jest parzysta, kolejna liczba n to n : 2.
16: 2) Jeśli liczba n jest nieparzysta, kolejna liczba n to n * 3c + 1.
17: 3) Jeśli liczba n to 1, zatrzymaj. W przeciwnym razie powtórz.
18:
19: To ogólne sformułowanie, ale jak dotąd nieudowodnione matematycznie –
20: każda liczba początkowa wcześniej czy później będzie miała wartość 1
21: ')
22:
23: print('Podaj liczbę początkową (większą niż 0) lub KONIEC:')
24: response = input('> ')
25:
26: if not response.isdecimal() or response == '0':
```

```
27:     print('Musisz podać liczbę całkowitą większą od 0.')
28:     sys.exit()
29:
30: n = int(response)
31: print(n, end='', flush=True)
32: while n != 1:
33:     if n % 2 == 0: # Jeśli n jest parzyste...
34:         n = n // 2
35:     else: # W innym przypadku n jest nieparzyste...
36:         n = 3 * n + 1
37:
38:     print(' ' + str(n), end='', flush=True)
39:     time.sleep(0.1)
40: print()
```

Odkrywanie programu

Postaraj się znaleźć odpowiedzi na poniżej podane pytania. Spróbuj wprowadzić zmiany w kodzie i uruchomić ponownie program, by zobaczyć, jaki efekt to przyniosło.

1. Ile jest liczb w ciągu Collatza, który zaczyna się od 32?
2. Ile jest liczb w ciągu Collatza, który zaczyna się od 33?
3. Czy ciągi Collatza zaczynające się od potęg dwójki (2, 4, 8, 16, 32, 64, 128 i tak dalej) zawsze składają się tylko z liczb parzystych (z wykluczeniem ostatniej 1)?
4. Co się stanie, gdy podasz 0 jako liczbę początkową?

#13

Gra w życie Conwaya



GRA W ŻYCIU CONWAYA TO SYMULACJA AUTOMATU KOMÓRKOWEGO, KTÓRY OPIERA SIĘ NA PROSTYCH ZASADACH, BY TWORZYĆ CIEKAWY WZORY. ZOSTAŁA WYMYŚLONA PRZEZ MATEMATYKA JOHNA CONWAYA W 1970 ROKU, A SPOPIULARYZOWAŁ JĄ MARTIN GARDNER NA ŁAMACH CZASOPISMA „Scientific American”, W PROWADZONEJ PRZEZ SIEBIE RUBRYCE POŚWIĘCONEJ GROM MATEMATYCZNYM. DZISIAJ JEST BARDZO LUBIANA PRZEZ PROGRAMISTÓW I INFORMATYKÓW, JEDNAK JEST RACZEJ CIEKAWĄ WIZUALIZACJĄ NIŻ PRAWDZIWĄ GRĄ. NA DWUWYMIAROWEJ PLANSZY ZNAJDUJE SIĘ SIATKA KOMÓREK, A KAŻDA Z NICH KIERUJE SIĘ TRZEMA PROSTYMI ZASADAMI:

- Żyjące komórki z dwoma lub trzema sąsiadami pozostają przy życiu w kolejnym kroku symulacji.
- Martwe komórki z dokładnie trzema sąsiadami ożywają w kolejnym kroku symulacji.
- Pozostałe komórki umierają lub pozostają martwe w kolejnym kroku symulacji.

Czy komórka będzie żywa, czy martwa w kolejnym kroku symulacji całkowicie zależy od jej aktualnego stanu. Komórki nie pamiętają wcześniejszych stanów. Prowadzone są badania związane ze wzorami tworzonymi przez te proste zasady. Profesor Conway zmarł z powodu komplikacji po koronawirusie w kwietniu 2020 roku. Więcej informacji na temat jego gry w życie znajdziesz na stronie https://pl.wikipedia.org/wiki/Gra_w_%C5%BCycie, a o Martinie Gardnerze dowiesz się więcej ze strony https://pl.wikipedia.org/wiki/Martin_Gardner.

Działanie programu

Gdy uruchomisz program *conwaysgameoflife.py*, uzyskasz następujący rezultat:

```
00 0 0 0 0 0 0000 000 0
0 0 0 000 0 0 00 000 0 0 00 00000
00000 0 0 0 0 0 0 00 000 0 0 0 00
00 0 0 000 00 0 00 00 0 00 00 00 00
00 0 0 0 0 0 000 0 0000 00000 00 0 0
00 0 0 0 0 0 00 0 0 00 0 00 0 00 0
000 0 0 0 0 0 000 00 0 00 0 000
0 0 0 0 0 0 0 0 0 0 0000 0 0 0 0 0
0 0000 0 0000 0 0 0 00 00000
0 0 000 00 00 000 00 000 0 0 00 00
0 0 0 0 0 00 0 0 0 0 0 0 00 000000
0 0 00 0 000 00 00 0 0 0 0 00 000 0 0
0 0 0 00 000 00 0 0 0 0 00 0 000000
0000 0 0 0 0 0 0 0 0 0 0 00 00 00
0 0 0 0 0 00000 0 0 0 0 00 0 0 000 0
0 000 0 0 00 00 00 00 0 0 000 00 0 0
0 00 00 00 0 000 00 0 000 0 0 0 00
000 000 0 0 0 0 0 0000 0 000 0 0
0 00 000 0 0 0 0 0 0 0 0
00 0 0 00 0 0 0 000 0000 000
```

Jak to działa?

Stan komórek jest zapisywany w słownikach *cells* i *nextCells*. Oba słowniki mają klucze w postaci krotki (x,y) (gdzie x i y to liczby całkowite). Znak '0' oznacza żywą komórkę, a znak '.' martwą. Linie od 40. do 44. wyświetlają na ekranie graficzną reprezentację tych słowników. Słownik *cells* zawiera informacje o aktualnym stanie komórek, a słownik *nextCells* o stanie komórek w kolejnym kroku symulacji.

```
01: """Gra w życie Conwaya, autor: Al Sweigart, al@inventwithpython.com
02: Klasyczna symulacja automatu komórkowego. Naciśnij Ctrl+C, by zatrzymać program.
03: Więcej informacji na stronie https://pl.wikipedia.org/wiki/Gra\_w\_%C5%BCycie.
04: Kod pobrany ze strony https://ftp.helion.pl/przyklady/wiksma.zip.
05: Etykiety: krótki, artystyczny, symulacja"""
06:
07: import copy, random, sys, time
08:
09: # Deklaracja stałych:
10: WIDTH = 79 # Szerokość siatki.
11: HEIGHT = 20 # Wysokość siatki.
12:
13: # (!) Spróbuj zmienić znak komórki typu ALIVE (żywa) na # lub inny:
14: ALIVE = '0' # Znak przedstawiający żywą komórkę.
15: # (!) Spróbuj zmienić znak komórki typu DEAD (martwa) na . lub inny:
```

```

16: DEAD = ' ' # Znak przedstawiający martwą komórkę.
17:
18: # (!) Spróbuj zmienić ALIVE na |, a DEAD na -.
19:
20: # Słowniki cells i nextCells przechowują stan gry.
21: # Ich klucze (x, y) to krotki, a wartości to znaki zapisane
22: # albo w stałej ALIVE, albo DEAD.
23: nextCells = {}
24: # Umieść losowe martwe i żywe komórki w słowniku nextCells:
25: for x in range(WIDTH): # Przejdź w pętli przez każdą kolumnę.
26:     for y in range(HEIGHT): # Przejdź w pętli przez każdy wiersz.
27:         # 50/50 szansy, że pierwsza komórka będzie żywa lub martwa.
28:         if random.randint(0, 1) == 0:
29:             nextCells[(x, y)] = ALIVE # Dodaj żywą komórkę.
30:         else:
31:             nextCells[(x, y)] = DEAD # Dodaj martwą komórkę.
32:
33: while True: # Główna pętla programu.
34:     # Każdy przebieg pętli to krok symulacji.
35:
36:     print('\n' * 50) # Oddziel każdy krok nowymi liniami.
37:     cells = copy.deepcopy(nextCells)
38:
39:     # Wyświetl komórki na ekranie:
40:     for y in range(HEIGHT):
41:         for x in range(WIDTH):
42:             print(cells[(x, y)], end='') # Wyświetl # lub spację.
43:         print() # Wyświetl znak nowej linii na końcu wiersza.
44:     print('Naciśnij Ctrl+C, by wyjść.')
45:
46:     # Oblicz następny krok na podstawie aktualnego stanu komórek:
47:     for x in range(WIDTH):
48:         for y in range(HEIGHT):
49:             # Pobranie współrzędnych (x, y) sąsiednich komórek, nawet jeśli
50:             # zawijają się na drugą krawędź:
51:             left = (x - 1) % WIDTH
52:             right = (x + 1) % WIDTH
53:             above = (y - 1) % HEIGHT
54:             below = (y + 1) % HEIGHT
55:
56:             # Policz żywych sąsiadów:
57:             numNeighbors = 0
58:             if cells[(left, above)] == ALIVE:
59:                 numNeighbors += 1 # Górny lewy sąsiad jest żywy.
60:             if cells[(x, above)] == ALIVE:
61:                 numNeighbors += 1 # Górny sąsiad jest żywy.
62:             if cells[(right, above)] == ALIVE:
63:                 numNeighbors += 1 # Górny prawy sąsiad jest żywy.
64:             if cells[(left, y)] == ALIVE:
65:                 numNeighbors += 1 # Lewy sąsiad jest żywy.
66:             if cells[(right, y)] == ALIVE:
67:                 numNeighbors += 1 # Prawy sąsiad jest żywy.
68:             if cells[(left, below)] == ALIVE:
69:                 numNeighbors += 1 # Dolny lewy sąsiad jest żywy.
70:             if cells[(x, below)] == ALIVE:
71:                 numNeighbors += 1 # Dolny sąsiad jest żywy.

```

```

72:         if cells[(right, below)] == ALIVE:
73:             numNeighbors += 1 # Dolny prawy sąsiad jest żywy.
74:
75:         # Ustaw stan komórki według zasad gry w życie Conwaya:
76:         if cells[(x, y)] == ALIVE and (numNeighbors == 2
77:             or numNeighbors == 3):
78:             # Żywe komórki z dwoma lub trzema sąsiadami pozostają żywe:
79:             nextCells[(x, y)] = ALIVE
80:         elif cells[(x, y)] == DEAD and numNeighbors == 3:
81:             # Martwe komórki z dokładnie trzema sąsiadami ożywają:
82:             nextCells[(x, y)] = ALIVE
83:         else:
84:             # Pozostałe komórki umierają lub pozostają martwe:
85:             nextCells[(x, y)] = DEAD
86:
87:     try:
88:         time.sleep(1) # Dodaj 1-sekundową pauzę, by zmniejszyć efekt migania.
89:     except KeyboardInterrupt:
90:         print("Gra w życie Conwaya")
91:         print('Autor: Al Sweigart, al@inventwithpython.com')
92:         sys.exit() # Po naciśnięciu Ctrl+C zakończ program.

```

Po przepisaniu kodu źródłowego i uruchomieniu go kilka razy spróbuj pokusperymetować. Komentarze opatrzone znakiem (!) to sugestie zmian, jakie możesz wprowadzić. Możesz również samodzielnie postarać się wykonać następujące zadania:

- Zmień procent prawdopodobieństwa, że pierwsza komórka będzie żywa.
- Dodaj możliwość odczytywania stanu pierwszej komórki z pliku tekstowego, tak by użytkownik mógł sam podać stan pierwszej komórki.

Odkrywanie programu

Postaraj się znaleźć odpowiedzi na poniżej podane pytania. Spróbuj wprowadzić zmiany w kodzie i uruchomić ponownie program, by zobaczyć, jaki efekt to przyniosło.

1. Co się stanie, gdy zamienisz `WIDTH = 79` w linii 10. na `WIDTH = 7`?
2. Co się stanie, gdy usuniesz wyrażenie `print('\n' * 50)` z linii 36. lub gdy umieścisz przed nim znacznik komentarza?
3. Co się stanie, gdy zamienisz wyrażenie `random.randint(0, 1)` w linii 28. na `random.randint(0, 10)`?
4. Co się stanie, gdy zamienisz wyrażenie `nextCells[(x, y)] = DEAD` w linii 85. na `nextCells[(x, y)] = ALIVE`?

#14

Odliczanie



TEN PROGRAM WYŚWIETLA CYFROWY LICZNIK, KTÓRY ODLICZA DO ZERA. ZAMIAST WYŚWIETLAĆ PO PROSTU CYFRY, MODUŁ `SEVSEG.PY` Z PROJEKTU 64. „WYŚWIETLACZ SIEDMIOSEGMENTOWY” PRZYGOTOWUJE GRAFIKĘ dla każdej cyfry. Najpierw musisz utworzyć ten plik, by program odliczający w dół zadziałał. Następnie ustaw licznik na dowolną liczbę sekund, minut czy godzin. Ten program jest podobny do projektu 19. „Zegar cyfrowy”.

Działanie programu

Gdy uruchomisz program `countdown.py`, uzyskasz następujący rezultat:

```
  _ _  *  _ _  *  _ _  
  _ _  *  _ _  *  _ _
```

Naciśnij `Ctrl+C`, by zatrzymać program.

Jak to działa?

Po zaimportowaniu modułu `sevseg` możesz wywołać funkcję `sevseg.getSevSegStr()`, by uzyskać kilkulinijkowy łańcuch znaków, który tworzy cyfrę w stylu wyświetlacza siedmiosegmentowego. Jednak program odliczający do zera musi dodatkowo wyświetlić dwukropek utworzony z gwiazdek między godzinami, minutami

i sekundami. Wymaga to rozdzielenia trzech linii tych kilkunastu znakowych łańcuchów znaków na trzy osobne łańcuchy, co zostało osiągnięte za pomocą metody `splitlines()`.

```
01: """Odliczanie, autor: Al Sweigart, al@inventwithpython.com
02: Animacja odliczania do zera za pomocą cyfr w stylu wyświetlacza siedmiosegmentowego.
03: Naciśnij Ctrl+C, by zatrzymać.
04: Więcej informacji na stronie https://pl.wikipedia.org/wiki/Wy%C5%9Bwietlacz\_siedmiosegmentowy.
05: Moduł sevseg.py musi być w tym samym folderze.
06: Kod pobrany ze strony https://ftp.helion.pl/przyklady/wiksma.zip.
07: Etykiety: króciutki, artystyczny"""
08:
09: import sys, time
10: import sevseg # Zaimportuj nasz program sevseg.py.
11:
12: # (!) Zmień wartość na dowolną liczbę sekund:
13: secondsLeft = 30
14:
15: try:
16:     while True: # Główna pętla programu.
17:         # Wyczyść ekran przez wyświetlenie wielu znaków nowej linii:
18:         print('\n' * 60)
19:
20:         # Oblicz godziny/minuty/sekundy na podstawie zmiennej secondsLeft:
21:         # Na przykład: 7265 to 2 godziny, 1 minuta, 5 sekund.
22:         # Zatem 7265 // 3600 to 2 godziny:
23:         hours = str(secondsLeft // 3600)
24:         # A 7265 % 3600 to 65, natomiast 65 // 60 to 1 minuta:
25:         minutes = str((secondsLeft % 3600) // 60)
26:         # I 7265 % 60 to 5 sekund:
27:         seconds = str(secondsLeft % 60)
28:
29:         # Pobierz łańcuchy znaków będące graficzną reprezentacją cyfr z modułu sevseg:
30:         hDigits = sevseg.getSevSegStr(hours, 2)
31:         hTopRow, hMiddleRow, hBottomRow = hDigits.splitlines()
32:
33:         mDigits = sevseg.getSevSegStr(minutes, 2)
34:         mTopRow, mMiddleRow, mBottomRow = mDigits.splitlines()
35:
36:         sDigits = sevseg.getSevSegStr(seconds, 2)
37:         sTopRow, sMiddleRow, sBottomRow = sDigits.splitlines()
38:
39:         # Wyświetl cyfry:
40:         print(hTopRow + ' ' + ' ' + mTopRow + ' ' + ' ' + sTopRow)
41:         print(hMiddleRow + ' * ' + mMiddleRow + ' * ' + sMiddleRow)
42:         print(hBottomRow + ' * ' + mBottomRow + ' * ' + sBottomRow)
43:
44:         if secondsLeft == 0:
45:             print()
46:             print(' * * * * BUM * * * *')
47:             break
48:
49:         print()
50:         print('Naciśnij Ctrl+C, by zatrzymać program.')
51:
```

```
52:         time.sleep(1) # Zrób jednosekundową pauzę.
53:         secondsLeft -= 1
54: except KeyboardInterrupt:
55:     print('Odliczanie, autor: Al Sweigart, al@inventwithpython.com')
56:     sys.exit() # Po naciśnięciu Ctrl+C zatrzymaj program.
```

Po przepisaniu kodu źródłowego i uruchomieniu go kilka razy spróbuj poeksperymentować. Możesz również samodzielnie postarać się wykonać następujące zadania:

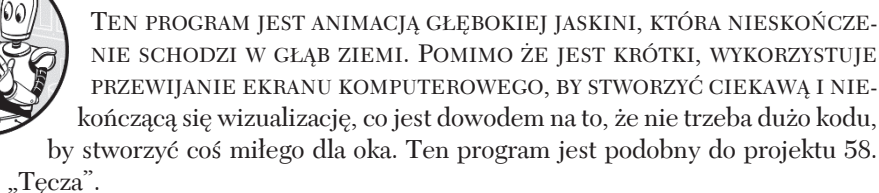
- Zapytaj użytkownika, od ilu licznik ma odliczać do zera.
- Daj użytkownikowi możliwość wpisania wiadomości, która ma się wyświetlić, gdy licznik dojdzie do zera.

Odkrywanie programu

Postaraj się znaleźć odpowiedzi na poniżej podane pytania. Spróbuj wprowadzić zmiany w kodzie i uruchomić ponownie program, by zobaczyć, jaki efekt to przyniosło.

1. Co się stanie, gdy zamienisz `secondsLeft = 30` w linii 13. na `secondsLeft = 30.5`?
2. Co się stanie, gdy zamienisz 2 w liniach 30., 33. i 36. na 1?
3. Co się stanie, gdy zamienisz wyrażenie `time.sleep(1)` w linii 52. na `time.sleep(0.1)`?
4. Co się stanie, gdy zamienisz wyrażenie `secondsLeft -= 1` w linii 53. na `secondsLeft -= 2`?
5. Co się stanie, gdy usuniesz wyrażenie `print('\n' * 60)` w linii 18. lub umieścisz przed nim znacznik komentarza?
6. Jaki komunikat błędu otrzymasz, jeśli usuniesz wyrażenie `import sys` w linii 10. lub umieścisz przed nim znacznik komentarza?

Głęboka jaskinia



Gdy uruchomisz program *deepcave.py*, uzyskasz następujący rezultat:

```
--uciete--
```

Jak to działa?

Ten program wykorzystuje to, że wyświetlanie znaków nowej linii powoduje, że poprzednie linie w końcu przesuwają się do góry ekranu. Przez wyświetlanie nieco innej przerwy w każdej linii program tworzy przewijaną animację, która sprawia wrażenie, że widz porusza się w dół.

Liczba hashtagów po lewej stronie jest zapisywana w zmiennej `leftWidth`. Liczba spacji w środku jest zapisywana w zmiennej `gapWidth`. Natomiast liczba hashtagów wyświetlana z prawej strony ekranu jest obliczana za pomocą wyrażenia `WIDTH - gapWidth - leftWidth`. Dzięki temu mamy pewność, że każda linia ma taką samą szerokość.

```
01: """Głęboka jaskinia, autor: Al Sweigart, al@inventwithpython.com
02: Animacja głębokiej jaskini, która nieskończenie schodzi w głąb ziemi.
03: Kod pobrany ze strony https://ftp.helion.pl/przyklady/wiksma.zip.
04: Etykiety: króciutki, dla początkujących, przewijanie, artystyczny"""
05:
06:
07: import random, sys, time
08:
09: # Deklaracja stałych:
10: WIDTH = 70 # (!) Spróbuj zmienić tę wartość na 10 lub 30.
11: PAUSE_AMOUNT = 0.05 # (!) Spróbuj zmienić tę wartość na 0 lub 1.0.
12:
13: print('Głęboka jaskinia, autor: Al Sweigart, al@inventwithpython.com')
14: print('Naciśnij Ctrl+C, by zatrzymać.')
15: time.sleep(2)
16:
17: leftWidth = 20
18: gapWidth = 10
19:
20: while True:
21:     # Wyświetl segment tunelu:
22:     rightWidth = WIDTH - gapWidth - leftWidth
23:     print(('#' * leftWidth) + (' ' * gapWidth) + ('#' * rightWidth))
24:
25:     # Sprawdź, czy podczas krótkiej przerwy nie została naciśnięta kombinacja Ctrl+C:
26:     try:
27:         time.sleep(PAUSE_AMOUNT)
28:     except KeyboardInterrupt:
29:         sys.exit() # Po naciśnięciu Ctrl+C zatrzymaj program.
30:
31:     # Dostosuj szerokość lewej strony:
32:     diceRoll = random.randint(1, 6)
33:     if diceRoll == 1 and leftWidth > 1:
34:         leftWidth = leftWidth - 1 # Zmniejsz szerokość lewej strony.
35:     elif diceRoll == 2 and leftWidth + gapWidth < WIDTH - 1:
36:         leftWidth = leftWidth + 1 # Zwiększ szerokość lewej strony.
37:     else:
38:         pass # Nic nie rób; szerokość lewej strony bez zmian.
39:
40:     # Dostosuj szerokość przerwy:
41:     # (!) Spróbuj usunąć znaczniki komentarza sprzed poniższych linii:
```

```
42:     #diceRoll = random.randint(1, 6)
43:     #if diceRoll == 1 and gapWidth > 1:
44:         #     gapWidth = gapWidth - 1 # Zmniejsz szerokość przerwy.
45:     #elif diceRoll == 2 and leftWidth + gapWidth < WIDTH - 1:
46:         #     gapWidth = gapWidth + 1 # Zwiększ szerokość przerwy.
47:     #else:
48:         #     pass # Nic nie rób; szerokość przerwy bez zmian.
```

Po przepisaniu kodu źródłowego i uruchomieniu go kilka razy spróbuj poeksperymentować. Komentarze opatrzone znakiem (!) to sugestie zmian, jakie możesz wprowadzić.

Odkrywanie programu

Postaraj się znaleźć odpowiedzi na poniżej podane pytania. Spróbuj wprowadzić zmiany w kodzie i uruchomić ponownie program, by zobaczyć, jaki efekt to przyniosło.

1. Co się stanie, gdy zamienisz (' ' * gapWidth) w linii 23. na ('.' * gapWidth)?
2. Co się stanie, gdy zamienisz random.randint(1, 6) w linii 32. na random.randint(1, 1)?
3. Co się stanie, gdy zamienisz random.randint(1, 6) w linii 32. na random.randint(2, 2)?
4. Jaki komunikat błędu otrzymasz, jeśli usuniesz wyrażenie leftWidth = 20 w linii 17. lub umieścisz przed nim znacznik komentarza?
5. Co się stanie, gdy zamienisz wyrażenie WIDTH = 70 w linii 10. na WIDTH = -70?
6. Jaki komunikat błędu otrzymasz, jeśli zamienisz wyrażenie PAUSE_AMOUNT = 0.05 w linii 11. na PAUSE_AMOUNT = -0.05?

#16

Diamenty



TEN PROGRAM PRZEDSTAWIA KRÓTKI ALGORYTM RYSUJĄCY RÓŻNEJ WIELKOŚCI DIAMENTY ZA POMOCĄ ZNAKÓW ASCII. ZAWIERA FUNKCJE, KTÓRE RYSUJĄ ALBO KONTUR DIAMENTU, ALBO WYPEŁNIONY DIAMENT O PODANYM PRZEZ CIEBIE ROZMIARZE. TE FUNKCJE SĄ DOBRYM ĆWICZENIEM DLA OSÓB POCZĄTKUJĄCYCH. POSTARAJ SIĘ ZROZUMIEĆ WZORZEC STOJĄCY ZA RYSUNKAMI DIAMENTÓW WRAZ ZE WZROSTEM ICH ROZMIARU.

Działanie programu

Gdy uruchomisz program *diamonds.py*, uzyskasz następujący rezultat:

Diamenty, autor: Al Sweigart, al@inventwithpython.com

```
 /\n \/\n\n /\n \/\n\n  /\n /\n \/\n  \/\n
```

```

  /\
 /\
//\
\\//
 V

  /\
 /\
//\
\\//
 V

  /\
 /\
//\
//\
\\//
\\//
 V
--ucięte--

```

Jak to działa?

Aby lepiej zrozumieć działanie tego programu, warto najpierw samemu narysować takie diamenty o różnych wielkościach w swoim edytorze, a następnie spróbować znaleźć wzór, według którego diamenty są rysowane, gdy stają się coraz większe. Ta technika pozwoli Ci odkryć, że każdy wiersz konturu diamentu ma cztery części: liczbę spacji na początku, zewnętrzny ukośnik, liczbę spacji w środku i zewnętrzny ukośnik wsteczny. Wypełnione diamenty zamiast spacji w środku mają kilka ukośników i ukośników wstecznych. Rozszyfrowanie tego wzoru pomogło mi napisać program *diamonds.py*.

```

01: r""" Diamenty, autor: Al Sweigart, al@inventwithpython.com
02: Program rysuje różnej wielkości diamenty.
03: Kod pobrany ze strony https://ftp.helion.pl/przyklady/wiksma.zip.
04:
05:
06:
07:
08:
09:
10:
11:
12: Etykiety: króciutki, dla początkujących, artystyczny"""
13:
14: def main():
15:     print('Diamenty, autor: Al Sweigart, al@inventwithpython.com')
16:
17:     # Wyświetl diamenty o wielkości od 0 do 6:
18:     for diamondSize in range(0, 6):
19:         displayOutlineDiamond(diamondSize)

```

```

20:         print() # Wyświetl znak nowej linii.
21:         displayFilledDiamond(diamondSize)
22:         print() # Wyświetl znak nowej linii.
23:
24:
25: def displayOutlineDiamond(size):
26:     # Wyświetl górną połówkę diamentu:
27:     for i in range(size):
28:         print(' ' * (size - i - 1), end='') # Odstęp z lewej strony.
29:         print('/', end='') # Lewa strona diamentu.
30:         print(' ' * (i * 2), end='') # Środek diamentu.
31:         print('\') # Prawa strona diamentu.
32:
33:     # Wyświetl dolną połówkę diamentu:
34:     for i in range(size):
35:         print(' ' * i, end='') # Odstęp z lewej strony.
36:         print('\', end='') # Lewa strona diamentu.
37:         print(' ' * ((size - i - 1) * 2), end='') # Środek diamentu.
38:         print('/') # Prawa strona diamentu.
39:
40:
41: def displayFilledDiamond(size):
42:     # Wyświetl górną połówkę diamentu:
43:     for i in range(size):
44:         print(' ' * (size - i - 1), end='') # Odstęp z lewej strony.
45:         print('/', end='') # Lewa strona diamentu.
46:         print('\') * (i + 1)) # Prawa strona diamentu.
47:
48:     # Wyświetl dolną połówkę diamentu:
49:     for i in range(size):
50:         print(' ' * i, end='') # Odstęp z lewej strony.
51:         print('\') * (size - i), end='') # Lewa strona diamentu.
52:         print('/') * (size - i)) # Prawa strona diamentu.
53:
54:
55: # Jeśli program został uruchomiony (a nie zaimportowany), rozpocznij grę:
56: if __name__ == '__main__':
57:     main()

```

Po przepisaniu kodu źródłowego i uruchomieniu go kilka razy spróbuj poeksperymentować. Możesz również samodzielnie postarać się wykonać następujące zadania:

- Stwórz inne kształty: trójkąty, prostokąty i romby.
- Zapisz kształty w pliku tekstowym, zamiast wyświetlać je na ekranie.

Odkrywanie programu

Postaraj się znaleźć odpowiedzi na poniżej podane pytania. Spróbuj wprowadzić zmiany w kodzie i uruchomić ponownie program, by zobaczyć, jaki efekt to przyniosło.

1. Co się stanie, gdy zamienisz `print('\\')` w linii 31. na `print('@')`?
2. Co się stanie, gdy zamienisz `print(' ' * (i * 2), end='')` w linii 30. na `print('@' * (i * 2), end='')`?
3. Co się stanie, gdy zamienisz `range(0, 6)` w linii 18. na `range(0, 30)`?
4. Co się stanie, jeśli usuniesz wyrażenie `for i in range(size):` w linii 34. lub 49. albo umieścisz przed nim znacznik komentarza?

#17

Matematyka i kostki



TEN PROGRAM BĘDĄCY QUIZEM MATEMATYCZNYM POLEGA NA RZUCANIU SZEŚCIENNYMI KOSTKAMI, A TWOIM ZADANIEM JEST JAK NAJSZYBSZE POLICZENIE WYRZUCONYCH OCZEK. JEDNAK TEN PROGRAM TO COŚ WIĘCEJ niż tylko zmieniające się grafiki ścian kostek, ponieważ rysuje kostki za każdym razem w innym miejscu na ekranie. Grafika tworzona za pomocą znaków ASCII jest zabawnym dodatkiem do ćwiczenia arytmetyki.

Działanie programu

Gdy uruchomisz program *dicemath.py*, uzyskasz następujący rezultat:

Matematyka i kostki, autor: Al Sweigart, al@inventwithpython.com

Dodaj oczka wszystkich kości na ekranie. Masz 30 sekund, by podać jak największą liczbę odpowiedzi. Otrzymujesz 4 punkty za każdą poprawną odpowiedź i tracisz 1 punkt za każdą złą odpowiedź.

Naciśnij Enter, aby rozpocząć...

```
+-----+
|         |
|         |
|         |
|         |
|         |
|         |
+-----+
|         |
|         |
|         |
|         |
|         |
|         |
+-----+
```

```
+-----+
| 0  0 |
| 0  0 |
| 0  0 |
+-----+
```

```
+-----+
| 0 0 0 |
|       |
| 0 0 0 |
+-----+
```

```
+-----+ +-----+
| 0 0 0 | | 0       |
|       | |         |
| 0 0 0 | |         |
+-----+ +-----+
```

Podaj sumę: 25
--ucięte--

Jak to działa?

Kostka na ekranie jest przedstawiana za pomocą słownika canvas. W Pythonie krotki są podobne do list, z tym że ich zawartość nie może być zmieniana. Kluczami słownika są krotki (x, y), które są współrzędnymi położenia na ekranie lewego górnego rogu kostki, a wartościami krotki (odpowiedzialne za reprezentację graficzną) z krotki ALL_DICE. Jak widać w liniach od 28. do 80., każda krotka kostki zawiera listę łańcuchów znaków, które są graficzną reprezentacją ścian kostki, oraz liczbę całkowitą oznaczającą liczbę oczek na danej ścianie. Program korzysta z tej informacji, by wyświetlić kostkę i obliczyć sumę oczek na wszystkich kostkach.

Linie od 174. do 177. wyświetlają dane ze słownika canvas na ekranie w podobny sposób, jak program z projektu 13. „Gra w życie Conwaya” wyświetla komórki.

```
001: """Matematyka i kostki, autor: Al Sweigart, al@inventwithpython.com
002: Program, który sprawdza, jak szybko potrafisz dodawać wyrzucone na kostkach oczka.
003: Kod pobrany ze strony https://ftp.helion.pl/przyklady/wiksma.zip.
004: Etykiety: długi, artystyczny, gra, matematyka"""
005:
006: import random, time
007:
008: # Deklaracja stałych:
009: DICE_WIDTH = 9
010: DICE_HEIGHT = 5
011: CANVAS_WIDTH = 79
012: CANVAS_HEIGHT = 24 - 3 # Odejmij 3, by zostawić miejsce na dole na wprowadzenie sumy.
013:
014: # Czas trwania w sekundach:
015: QUIZ_DURATION = 30 # (!) Spróbuj zmienić tę wartość na 10 lub 60.
016: MIN_DICE = 2 # (!) Spróbuj zmienić tę wartość na 1 lub 5.
017: MAX_DICE = 6 # (!) Spróbuj zmienić tę wartość na 14.
018:
019: # (!) Spróbuj zmienić te wartości na inne:
020: REWARD = 4 # (!) Punkty przyznawane za poprawną odpowiedź.
021: PENALTY = 1 # (!) Punkty odejmowane za złą odpowiedź.
```

```

022: # (!) Spróbuj ustawić stałą PENALTY na liczbę ujemną,
023: # by gracz dostawał punkty za niepoprawną odpowiedź!
024:
025: # Program zawiesza się, jeśli nie wszystkie kości mieszczą się na ekranie:
026: assert MAX_DICE <= 14
027:
028: D1 = (['+-----+',
029:       '|         |',
030:       '|   0   |',
031:       '|         |',
032:       '+-----+'], 1)
033:
034: D2a = (['+-----+',
035:         '| 0     |',
036:         '|         |',
037:         '|     0 |',
038:         '+-----+'], 2)
039:
040: D2b = (['+-----+',
041:         '|         0 |',
042:         '|         |',
043:         '| 0     |',
044:         '+-----+'], 2)
045:
046: D3a = (['+-----+',
047:         '| 0     |',
048:         '|     0 |',
049:         '|         0 |',
050:         '+-----+'], 3)
051:
052: D3b = (['+-----+',
053:         '|         0 |',
054:         '|     0   |',
055:         '| 0     |',
056:         '+-----+'], 3)
057:
058: D4 = (['+-----+',
059:        '| 0 0  |',
060:        '|         |',
061:        '| 0 0  |',
062:        '+-----+'], 4)
063:
064: D5 = (['+-----+',
065:        '| 0 0  |',
066:        '|     0 |',
067:        '| 0 0  |',
068:        '+-----+'], 5)
069:
070: D6a = (['+-----+',
071:         '| 0 0  |',
072:         '| 0 0  |',
073:         '| 0 0  |',
074:         '+-----+'], 6)
075:
076: D6b = (['+-----+',
077:         '| 0 0 0 |',

```

```

078:         '|         |',
079:         '| 0 0 0 |',
080:         '+-----+', 6)
081:
082: ALL_DICE = [D1, D2a, D2b, D3a, D3b, D4, D5, D6a, D6b]
083:
084: print('''Matematyka i kości, autor: Al Sweigart, al@inventwithpython.com
085:
086: Dodaj oczka wszystkich kości na ekranie. Masz {} sekund,
087: by podać jak największą liczbę odpowiedzi. Otrzymujesz {} punkty za każdą
088: poprawną odpowiedź i tracisz {} punkt za każdą złą odpowiedź.
089: '''.format(QUIZ_DURATION, REWARD, PENALTY))
090: input('Naciśnij Enter, aby rozpocząć...')
091:
092: # Zapisuj liczbę poprawnych i niepoprawnych odpowiedzi:
093: correctAnswers = 0
094: incorrectAnswers = 0
095: startTime = time.time()
096: while time.time() < startTime + QUIZ_DURATION: # Główna pętla gry.
097:     # Wylosuj liczbę, innymi słowy, rzuć kostką:
098:     sumAnswer = 0
099:     diceFaces = []
100:     for i in range(random.randint(MIN_DICE, MAX_DICE)):
101:         die = random.choice(ALL_DICE)
102:         # die[0] zawiera listę łańcuchów znaków przedstawiających grafikę ścian kości:
103:         diceFaces.append(die[0])
104:         # die[1] zawiera całkowitą oznaczającą liczbę oczek na ścianie kości:
105:         sumAnswer += die[1]
106:
107:     # Zawiera krotkę (x, y) górnego lewego rogu każdej kości.
108:     topLeftDiceCorners = []
109:
110:     # Określ, gdzie powinna być wyświetlona kostka:
111:     for i in range(len(diceFaces)):
112:         while True:
113:             # Znajdź losowe miejsce na płótnie, gdzie zostanie wyświetlona kostka:
114:             left = random.randint(0, CANVAS_WIDTH - 1 - DICE_WIDTH)
115:             top = random.randint(0, CANVAS_HEIGHT - 1 - DICE_HEIGHT)
116:
117:             # Pobierz współrzędne x, y dla wszystkich czterech rogów:
118:             #         lewy
119:             #         v
120:             #górny > +-----+ ^
121:             #         | 0         | |
122:             #         |   0       | DICE_HEIGHT (5)
123:             #         |         0 | |
124:             #         +-----+ v
125:             #         <----->
126:             #         DICE_WIDTH (9)
127:             topLeftX = left
128:             topLeftY = top
129:             topRightX = left + DICE_WIDTH
130:             topRightY = top
131:             bottomLeftX = left
132:             bottomLeftY = top + DICE_HEIGHT
133:             bottomRightX = left + DICE_WIDTH

```

```

134:         bottomRightY = top + DICE_HEIGHT
135:
136:         # Sprawdź, czy ta kostka nie nachodzi na poprzednią.
137:         overlaps = False
138:         for prevDieLeft, prevDieTop in topLeftDiceCorners:
139:             prevDieRight = prevDieLeft + DICE_WIDTH
140:             prevDieBottom = prevDieTop + DICE_HEIGHT
141:             # Sprawdź każdy róg kości, by zobaczyć, czy znajduje się w środku
142:             # poprzedniej kości:
143:             for cornerX, cornerY in ((topLeftX, topLeftY),
144:                                     (topRightX, topRightY),
145:                                     (bottomLeftX, bottomLeftY),
146:                                     (bottomRightX, bottomRightY)):
147:                 if (prevDieLeft <= cornerX < prevDieRight
148:                     and prevDieTop <= cornerY < prevDieBottom):
149:                     overlaps = True
150:         if not overlaps:
151:             # Kości nie zachodzą na siebie, więc możemy wyświetlić kostkę w tym miej-
152:             scu:
153:             topLeftDiceCorners.append((left, top))
154:             break
155:
156:     # Narysuj kość na płótnie:
157:
158:     # Klucze (x, y) to krotki liczb całkowitych, które określają
159:     # położenie na ekranie:
160:     canvas = {}
161:     # Przejdź przez każdą kostkę:
162:     for i, (dieLeft, dieTop) in enumerate(topLeftDiceCorners):
163:         # Przejdź przez każdy znak tworzący grafikę ściany kości:
164:         dieFace = diceFaces[i]
165:         for dx in range(DICE_WIDTH):
166:             for dy in range(DICE_HEIGHT):
167:                 # Skopiuj ten znak do odpowiedniego miejsca na płótnie:
168:                 canvasX = dieLeft + dx
169:                 canvasY = dieTop + dy
170:                 # Zauważ, że w liście łańcuchów znaków dieFace x i y
171:                 # są zamienione:
172:                 canvas[(canvasX, canvasY)] = dieFace[dy][dx]
173:
174:     # Wyświetl płótno na ekranie:
175:     for cy in range(CANVAS_HEIGHT):
176:         for cx in range(CANVAS_WIDTH):
177:             print(canvas.get((cx, cy), ' '), end='')
178:         print() # Wyświetl znak nowej linii.
179:
180:     # Pozwól graczowi wpisać odpowiedź:
181:     response = input('Podaj sumę: ').strip()
182:     if response.isdecimal() and int(response) == sumAnswer:
183:         correctAnswers += 1
184:     else:
185:         print('Źle, poprawna odpowiedź to ', sumAnswer)
186:         time.sleep(2)
187:         incorrectAnswers += 1
188:
189: # Wyświetl wyniki końcowe:

```

```
189: score = (correctAnswers * REWARD) - (incorrectAnswers * PENALTY)
190: print('Poprawne odpowiedzi: ', correctAnswers)
191: print('Niepoprawne odpowiedzi:', incorrectAnswers)
192: print('Wynik:      ', score)
```

Po przepisaniu kodu źródłowego i uruchomieniu go kilka razy spróbuj poeksperymentować. Komentarze opatrzone znakiem (!) to sugestie zmian, jakie możesz wprowadzić. Możesz również samodzielnie postarać się wykonać następujące zadania:

- Zmień grafikę ścian kostek wykonaną za pomocą znaków ASCII.
- Dodaj ściany kostek z siedmioma, ośmioma i dziewięcioma oczkami.

Odkrywanie programu

Postaraj się znaleźć odpowiedzi na poniżej podane pytania. Spróbuj wprowadzić zmiany w kodzie i uruchomić ponownie program, by zobaczyć, jaki efekt to przyniosło.

1. Co się stanie, gdy zmienisz linię 82. na `ALL_DICE = [D1]`?
2. Co się stanie, gdy zamienisz `get((cx, cy), ' ')` w linii 176. na `get((cx, cy), '.')`?
3. Co się stanie, gdy zamienisz `correctAnswers += 1` w linii 182. na `correctAnswers += 0`?
4. Jaki komunikat błędu otrzymasz, jeśli usuniesz wyrażenie `correctAnswers = 0` w linii 93. lub 49. lub umieścisz przed nim znacznik komentarza?

#18

Rzut kostką



W GRACH DUNGEONS & DRAGONS I INNYCH GRACH FABULARNYCH UŻYWA SIĘ SPECJALNYCH KOSTEK Z 4, 8, 10, 12, A NAWET 20 ŚCIANKAMI. TE GRY MAJĄ RÓWNIEŻ SZCZEGÓLNE ZAPISY, WSKAZUJĄCE, KTÓRĄ KOSTKĄ NALEŻY RZUCIĆ. Na przykład $3d6$ oznacza rzut trzema kostkami sześciennymi, $1d10+2$ to rzut jedną kostką o dziesięciu ściankach i dodanie dwóch oczek do wyrzuconego wyniku. Ten program symuluje taki rzut kostkami, gdybyś zapomniał przynieść swoje kostki. Może również symulować rzut kostką, która fizycznie nie istnieje, na przykład kostką o 36 ściankach.

Działanie programu

Gdy uruchomisz program *diceroller.py*, uzyskasz następujący rezultat:

```
Rzut kostką, autor: Al Sweigart, al@inventwithpython.com
--ucięte--
> 3d6
Suma: 12 (Każda kostka 4, 6, 2)
> 1d10+2
Suma: 6 (Każda kostka 4, +2)
> 2d38-1
Suma: 23 (Każda kostka 19, 5, -1)
> 100d6
Suma: 323 (Każda kostka 2, 1, 1, 3, 1, 3, 2, 3, 5, 2, 5, 2, 6, 2, 4, 5, 6, 4, 4,
3, 2, 6, 2, 5, 2, 4, 2, 2, 2, 2, 3, 5, 3, 5, 2, 4, 3, 5, 5, 4, 5, 4, 6, 5, 4, 2,
1, 5, 5, 2, 2, 4, 2, 6, 1, 6, 1, 2, 5, 1, 5, 1, 6, 2, 1, 1, 1, 5, 5, 5, 3, 1, 2,
1, 4, 2, 2, 1, 2, 4, 6, 1, 2, 5, 6, 6, 1, 2, 5, 6, 1, 3, 3, 6, 1, 2, 5, 2, 3, 2)
--ucięte--
```

Jak to działa?

Większość kodu w tym programie ma za zadanie upewnić się, że podane przez użytkownika informacje mają poprawny format. Same rzuty kostką to po prostu wywołanie funkcji `random.randint()`, która nie ma obciążenia — dla każdej liczby całkowitej z podanego zakresu istnieje takie samo prawdopodobieństwo wylosowania. To sprawia, że funkcja `random.randint()` doskonale nadaje się do symulowania rzutów kostką.

```
01: """Rzut kostką, autor: Al Sweigart, al@inventwithpython.com
02: Program symuluje rzuty kostką według notacji Dungeons & Dragons.
03: Kod pobrany ze strony https://ftp.helion.pl/przyklady/wiksma.zip.
04: Etykiety: krótki, symulacja"""
05:
06: import random, sys
07:
08: print('Rzut kostką, autor: Al Sweigart, al@inventwithpython.com')
09:
10: Wpisz iloma i jakimi kośćcami chcesz rzucić. Zapis to liczba kości,
11: potem "d", a potem liczba ścianek tych kości.
12: Możesz również dodać lub odjąć oczka od sumy wyrzuconych punktów.
13:
14: Przykłady:
15: 3d6 to rzut trzema sześciennymi kostkami
16: 1d10+2 to rzut jedną 10-ścienną kostką i dodanie 2 oczek
17: 2d38-1 to rzut dwoma 38-ściennymi kostkami i odjęcie 1 oczka
18: KONIEC kończy program
19: '''
20:
21: while True: # Główna pętla programu:
22:     try:
23:         diceStr = input('> ') # Użytkownik podaje rzut zgodnie z formatem.
24:         if diceStr.upper() == 'KONIEC':
25:             print('Dziękujemy za grę!')
26:             sys.exit()
27:
28:         # Usunąć zapis rzutu kostką:
29:         diceStr = diceStr.lower().replace(' ', '')
30:
31:         # Znajdź "d" w rzucie podanym przez użytkownika:
32:         dIndex = diceStr.find('d')
33:         if dIndex == -1:
34:             raise Exception('Brakuje litery "d".')
35:
36:         # Uzyskaj liczbę kostek. ("3" w "3d6+1"):
37:         numberOfDice = diceStr[:dIndex]
38:         if not numberOfDice.isdecimal():
39:             raise Exception('Brakuje liczby kostek.')
40:         numberOfDice = int(numberOfDice)
41:
42:         # Znajdź, czy jest znak plus, czy minus:
43:         modIndex = diceStr.find('+')
44:         if modIndex == -1:
```

```

45:         modIndex = diceStr.find('-')
46:
47:         # Znajdź liczbę ścianek ("6" w "3d6+1"):
48:         if modIndex == -1:
49:             numberOfSides = diceStr[dIndex + 1 :]
50:         else:
51:             numberOfSides = diceStr[dIndex + 1 : modIndex]
52:         if not numberOfSides.isdecimal():
53:             raise Exception('Brakuje liczby ścianek.')
54:         numberOfSides = int(numberOfSides)
55:
56:         # Znajdź liczbę dodawanych/odejmowanych oczek (modyfikator) ("1" w "3d6+1"):
57:         if modIndex == -1:
58:             modAmount = 0
59:         else:
60:             modAmount = int(diceStr[modIndex + 1 :])
61:             if diceStr[modIndex] == '-':
62:                 # Zmień wartość modyfikatora na ujemną:
63:                 modAmount = -modAmount
64:
65:         # Zasymuluj rzut kostką:
66:         rolls = []
67:         for i in range(numberOfDice):
68:             rollResult = random.randint(1, numberOfSides)
69:             rolls.append(rollResult)
70:
71:         # Wyświetl sumę:
72:         print('Suma:', sum(rolls) + modAmount, '(Każda kostka ', end='')
73:
74:         # Wyświetl poszczególne rzuty:
75:         for i, roll in enumerate(rolls):
76:             rolls[i] = str(roll)
77:         print(' '.join(rolls), end='')
78:
79:         # Wyświetl wartość modyfikatora:
80:         if modAmount != 0:
81:             modSign = diceStr[modIndex]
82:             print(' {}'.format(modSign, abs(modAmount)), end='')
83:         print('')
84:
85:     except Exception as exc:
86:         # W razie błędu wyświetl informację użytkownikowi:
87:         print('Zapis rzutu w niepoprawnym formacie. Wpisz coś w stylu "3d6"
88:             ↪ lub "1d10+2".')
89:         print('Powód niepoprawnego zapisu: ' + str(exc))
90:         continue # Wróć do pytania o rzut.

```

Po przepisaniu kodu źródłowego i uruchomieniu go kilka razy spróbuj pokusperymetnować. Możesz również samodzielnie postarać się wykonać następujące zadania:

- Dodaj mnożnik jako kolejny modyfikator poza dodawaniem i odejmowaniem.
- Dodaj możliwość automatycznego usuwania kostki z najmniejszą liczbą oczek.

Odkrywanie programu

Postaraj się znaleźć odpowiedzi na poniżej podane pytania. Spróbuj wprowadzić zmiany w kodzie i uruchomić ponownie program, by zobaczyć, jaki efekt to przyniosło.

1. Co się stanie, gdy usuniesz wyrażenie `rolls.append(rollResult)` w linii 69. lub umieścisz przed nim znacznik komentarza?
2. Co się stanie, gdy zmienisz `rolls.append(rollResult)` w linii 69. na `rolls.append(-rollResult)`?
3. Co się stanie, gdy usuniesz wyrażenie `print(' ', '.join(rolls),end='')` w linii 77. lub umieścisz przed nim znacznik komentarza?
4. Co się stanie, gdy nie podasz żadnego rzutu kostką?

#19

Zegar cyfrowy



TEN PROGRAM WYŚWIETLA ZEGAR CYFROWY POKAZUJĄCY AKTUALNY CZAS. ZAMIAST WYŚWIETLAĆ PO PROSTU CYFRY, MODUŁ *sevseg.py* Z PROJEKTU 64. „WYŚWIETLACZ SIEDMIOSEGMENTOWY” PRZYGOTOWUJE GRAFIKĘ DLA KAŻDEJ CYFRY. TEN PROGRAM JEST PODOBNY DO PROJEKTU 14. „ODLICZANIE”.

Działanie programu

Gdy uruchomisz program *digitalclock.py*, uzyskasz następujący rezultat:

```
|_| * |_| * |_|
|_| * |_| * |_|
```

Naciśnij Ctrl+C, by wyjść.

Jak to działa?

Program z zegarem cyfrowym wygląda podobnie do tego z projektu 14. „Odliczanie”, nie tylko dlatego że oba korzystają z modułu *sevseg.py*, ale też dlatego że muszą za pomocą funkcji `splitlines()` podzielić wielolinijkowy łańcuch znaków zwrócony przez metodę `sevseg.getSevSegStr()`. Dzięki temu będziemy mogli dodać dwukropek utworzony z gwiazdek między godzinami, minutami i sekundami.

Porównaj ten kod z kodem programu odliczającego, by zobaczyć, jak są podobne i czym się różnią.

```
01: """Zegar cyfrowy, autor: Al Sweigart, al@inventwithpython.com
02: Wyświetla zegar cyfrowy pokazujący aktualny czas za pomocą cyfr
03: w stylu wyświetlacza siedmiosegmentowego. Naciśnij Ctrl+C, by zatrzymać program.
04: Więcej informacji na stronie https://pl.wikipedia.org/wiki/Wy%C5%9Bwietlacz\_siedmiosegmentowy.
05: Plik sevseg.py musi być w tym samym folderze.
06: Kod pobrany ze strony https://ftp.helion.pl/przyklady/wiksma.zip.
07: Etykiety: króciutki, artystyczny"""
08:
09: import sys, time
10: import sevseg # Importuje nasz program sevseg.py.
11:
12: try:
13:     while True: # Główna pętla programu.
14:         # Wyczyść ekran poprzez wyświetlenie wielu znaków nowej linii:
15:         print('\n' * 60)
16:
17:         # Pobierz aktualny czas z zegara systemowego:
18:         currentTime = time.localtime()
19:         # Używamy operacji % 12, więc nasz zegar będzie miał format 12-godzinny,
20:         # a nie 24-godzinny:
21:         hours = str(currentTime.tm_hour % 12)
22:         if hours == '0':
23:             hours = '12' # Zegar w formacie 12-godzinnym pokazuje 12:00, a nie 00:00.
24:         minutes = str(currentTime.tm_min)
25:         seconds = str(currentTime.tm_sec)
26:
27:         # Pobierz łańcuch znaków reprezentujących grafikę cyfry z modułu sevseg:
28:         hDigits = sevseg.getSevSegStr(hours, 2)
29:         hTopRow, hMiddleRow, hBottomRow = hDigits.splitlines()
30:
31:         mDigits = sevseg.getSevSegStr(minutes, 2)
32:         mTopRow, mMiddleRow, mBottomRow = mDigits.splitlines()
33:
34:         sDigits = sevseg.getSevSegStr(seconds, 2)
35:         sTopRow, sMiddleRow, sBottomRow = sDigits.splitlines()
36:
37:         # Wyświetl cyfry:
38:         print(hTopRow + ' ' + mTopRow + ' ' + sTopRow)
39:         print(hMiddleRow + ' * ' + mMiddleRow + ' * ' + sMiddleRow)
40:         print(hBottomRow + ' * ' + mBottomRow + ' * ' + sBottomRow)
41:         print()
42:         print('Naciśnij Ctrl+C, by wyjść.')
43:
44:         # Wykonuj pętle, dopóki zmieniają się sekundy:
45:         while True:
46:             time.sleep(0.01)
47:             if time.localtime().tm_sec != currentTime.tm_sec:
48:                 break
49:         except KeyboardInterrupt:
50:             print('Zegar cyfrowy, autor: Al Sweigart, al@inventwithpython.com')
51:             sys.exit() # Po naciśnięciu Ctrl+C zakończ program.
```

Odkrywanie programu

Postaraj się znaleźć odpowiedzi na poniżej podane pytania. Spróbuj wprowadzić zmiany w kodzie i uruchomić ponownie program, by zobaczyć, jaki efekt to przyniosło.

1. Co się stanie, gdy zmienisz `time.sleep(0.01)` w linii 45. na `time.sleep(2)`?
2. Co się stanie, gdy zamienisz 2 w liniach 27., 30. i 33. na 1?
3. Co się stanie, jeśli usuniesz wyrażenie `print('\n' * 60)` w linii 15. lub umieścisz przed nim znacznik komentarza?
4. Jaki komunikat błędu otrzymasz, jeśli usuniesz wyrażenie `import sys` w linii 10. lub umieścisz przed nim znacznik komentarza?

#20

Strumień cyfrowy



TEN PROGRAM NAŚLADUJE WIZUALIZACJĘ „STRUMIENIA CYFROWEGO” Z FILMU *MATRIX*. LOSOWE KROPLE BINARNEGO DESZCZU PŁYNĄ W GÓRĘ OD DOŁU EKRANU, TWORZĄC FAJNĄ ANIMACJĘ. (NIESTETY, Z POWODU sposobu, w jaki tekst się przemieszcza, gdy ekran przewija się w dół, nie jest możliwe, by strumień spadał w dół bez użycia modułu takiego jak *bext*).

Działanie programu

Gdy uruchomisz program *digitalstream.py*, uzyskasz następujący rezultat:

Strumień cyfrowy, autor: Al Sweigart, al@inventwithpython.com
Naciśnij Ctrl+C, by wyjść.

```

                                     1           0           0
                                     1           1           1
                               0       0       11       1       1 1
        1 0 1 1       11       0       1 1 11
        0 0 0 0       11       0       0 0 11
        1 0 0 10      11       1       0 0 00
        0 1 1 11      10       0       0 1 01
        0 0 0 11      01      1       1 1 01
1      1 1 1 11      1 11      0       1 1 10
0      00 0 10      1 00      0 01      1 0 1
0      10 0 11      1 11      1 11 0      0 1 1 1
0      01      11      1 1      0 1 11 1      0
1      10      0      1 0      0 1 0 1      1
1      1 0 1      1 1      0 0 1 1      10      0
1 0      0 0      0 0      0 1 1 1      110
1 0 1      1 0      0 1      0 1 0 1      100
--ucięte--
```

Jak to działa?

Ten program, podobnie jak projekt 15. „Głęboka jaskinia”, korzysta z przewijania uzyskanego za pomocą wywołań funkcji `print()` w celu stworzenia animacji. Każda kolumna jest przedstawiona przy użyciu liczby całkowitej zapisanej w liście `columns`: `columns[0]` to liczba całkowita dla kolumny pierwszej od lewej, `columns[1]` to liczba całkowita dla kolumny znajdującej się po jej prawej stronie i tak dalej. Program na początku ustawia wartość tych liczb na 0, co oznacza, że wyświetlany jest ' ' (pusty łańcuch znaków) zamiast strumienia w tej kolumnie. Program losowo zmienia każdą liczbę całkowitą na wartość z zakresu od `MIN_STREAM_LENGTH` do `MAX_STREAM_LENGTH`. Ta liczba całkowita zmniejsza swoją wartość o 1 po każdym wyświetleniu linii. Dopóki liczba całkowita dla tej kolumny jest większa niż 0, program wyświetla w niej losowo 1 lub 0. To tworzy efekt „strumienia cyfrowego”, który możesz zobaczyć na ekranie.

```
01: """Strumień cyfrowy, autor: Al Sweigart, al@inventwithpython.com
02: Wygaszacz ekranu w stylu wizualizacji z filmu "Matrix".
03: Kod pobrany ze strony https://ftp.helion.pl/przyklady/wiksma.zip.
04: Etykiety: króciutki, artystyczny, dla początkujących, przewijanie"""
05:
06: import random, shutil, sys, time
07:
08: # Deklaracja stałych:
09: MIN_STREAM_LENGTH = 6 # (!) Spróbuj zmienić tę wartość na 1 lub 50.
10: MAX_STREAM_LENGTH = 14 # (!) Spróbuj zmienić tę wartość na 100.
11: PAUSE = 0.1 # (!) Spróbuj zmienić tę wartość na 0.0 lub 2.0.
12: STREAM_CHARS = ['0', '1'] # (!) Spróbuj użyć innych znaków.
13:
14: # Gęstość mieści się w zakresie od 0.0 do 1.0:
15: DENSITY = 0.02 # (!) Spróbuj zmienić tę wartość na 0.10 lub 0.30.
16:
17: # Pobierz rozmiar okna terminala:
18: WIDTH = shutil.get_terminal_size()[0]
19: # Nie możemy wyświetlić ostatniej kolumny w systemie Windows
20: # bez automatycznego dodania znaku nowej linii, więc zmniejsz szerokość o 1:
21: WIDTH -= 1
22:
23: print('Strumień cyfrowy, autor: Al Sweigart, al@inventwithpython.com')
24: print('Naciśnij Ctrl+C, by wyjść.')
25: time.sleep(2)
26:
27: try:
28:     # Dla każdej kolumny, gdy licznik wynosi 0, nic nie jest wyświetlane.
29:     # W przeciwnym razie zachowuje się jak licznik określający liczbę 1 lub 0,
30:     # które powinny być wyświetlone w tej kolumnie.
31:     columns = [0] * WIDTH
32:     while True:
33:         # Ustaw licznik dla każdej kolumny:
34:         for i in range(WIDTH):
35:             if columns[i] == 0:
36:                 if random.random() <= DENSITY:
37:                     # Zrestartuj strumień w tej kolumnie.
```

```

38:             columns[i] = random.randint(MIN_STREAM_LENGTH,
39:                                         MAX_STREAM_LENGTH)
40:
41:         # Wyświetl spację lub znak 1/0.
42:         if columns[i] > 0:
43:             print(random.choice(STREAM_CHARS), end='')
44:             columns[i] -= 1
45:         else:
46:             print(' ', end='')
47:         print() # Wyświetl znak nowej linii na końcu wiersza.
48:         sys.stdout.flush() # Upewnij się, że tekst pojawił się na ekranie.
49:         time.sleep(PAUSE)
50: except KeyboardInterrupt:
51:     sys.exit() # Po naciśnięciu Ctrl+C zakończ program.

```

Po przepisaniu kodu źródłowego i uruchomieniu go kilka razy spróbuj poeksperymentować. Komentarze opatrzone znakiem (!) są sugestiami zmian, jakie możesz wprowadzić. Możesz również samodzielnie postarać się wykonać następujące zadania:

- Użyj innych znaków poza 0 i 1.
- Użyj innych kształtów poza liniami, na przykład prostokątów, trójkątów i diamentów.

Odkrywanie programu

Postaraj się znaleźć odpowiedzi na poniżej podane pytania. Spróbuj wprowadzić zmiany w kodzie i uruchomić ponownie program, by zobaczyć, jaki efekt to przyniosło.

1. Co się stanie, gdy zamienisz `print(' ', end='')` w linii 46. na `print('.', end='')`?
2. Co się stanie, gdy zamienisz `PAUSE = 0.1` w linii 11. na `PAUSE = -0.1`?
3. Co się stanie, gdy zamienisz `columns[i] > 0` w linii 42. na `columns[i] < 0`?
4. Co się stanie, gdy zamienisz `columns[i] > 0` w linii 42. na `columns[i] <= 0`?
5. Co się stanie, gdy zamienisz `columns[i] -= 1` w linii 44. na `columns[i] += 1`?

PROGRAM PARTNERSKI

— GRUPY HELION —



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

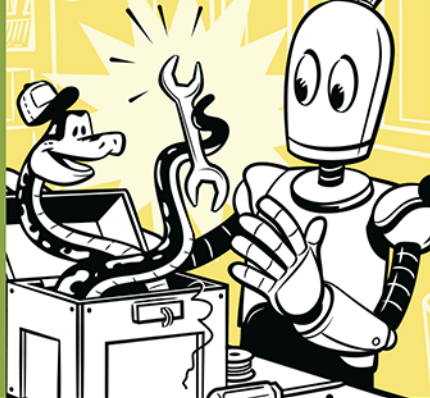
Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion 

**MASZ DOŚĆ
SAMOU CZKÓW?
ZACZNIJ PISAĆ
W PYTHONIE
PRAWDZI WY,
DZIAŁAJĄCY KOD!**



Programowanie wciąga. łatwo zapalić się do własnego pomysłu na świetny program, ale jeśli zabraknie umiejętności, nietrudno w poczuciu frustracji porzucić kod po napisaniu kilku linijek. Wiele osób, marząc o wykorzystywaniu imponujących możliwości Pythona, przepracowało cierpliwie liczne samouczki. Nie jest to zły sposób na rozpoczęcie przygody z programowaniem i przyswojenie składni języka. Ale do samodzielnego tworzenia kodu trzeba nieco innych umiejętności. Skąd jednak ma je wziąć kandydat na programistę, jeśli nie jest geniuszem komputerowym?

Tę książkę docenią ci, którzy opanowali już podstawową składnię Pythona i palą się do pisania własnych programów. Zawiera 81 projektów, które możesz napisać w tym języku. Programy składają się z maksymalnie 256 linii kodu i pozwolą Ci stopniowo nabierać umiejętności programisty — a zupełnie przy okazji dostarczą mnóstwo zabawy! Twoja nauka będzie polegała nie tylko na analizowaniu działania kodu i jego wpisywaniu w edytorze, ale także na samodzielnym eksperymentowaniu i modyfikowaniu kodu, aby dopasować program do własnych potrzeb i pomysłów. W efekcie — niepostrzeżenie, ćwicząc kodowanie nabierzesz biegłości i sporego doświadczenia w posługiwaniu się Pythonem!

Dzięki tej książce napiszesz:

- gry, w które zagrasz z komputerem lub przyjaciółmi
- realistyczne symulacje: pożaru lasu, miliona rzutów kostką i japońskiego liczydła
- piękne animacje: akwarium z rybkami i obracającego się sześcianu
- grę 3D, w której gracz porusza się po labiryncie
- programy szyfrujące z wykorzystaniem szyfrów ROT13 i Vigenère'a

Al Sweigart jest programistą o niezwykłym talencie dydaktycznym, autorem popularnych kursów programowania dla dzieci i dorosłych, w tym uwielbianego przez początkujących Udemy Python. Jest też członkiem organizacji Python Software Foundation, w ramach której zajmuje się rozwojem Pythona. Autor kilku cenionych podręczników do nauki programowania.

Helion

 **helion.pl**

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

Sprawdź nasze szkolenia!

SZKOLENIA



AKADEMIA IT & BUSINESS

HELIONSZKOLENIA.PL

KOD KORZYŚCI
Sięgnij po więcej! ▶



ISBN 978-83-283-8861-1



9 788328 388611

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 79,00 zł

