Andrzej „SunRiver” Gromczyński

Programowanie w Go

Poznaj prosty i wydajny język od Google

Helion

Redaktor prowadzący: Tomasz Gojowy

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: *helion.pl* (księgarnia internetowa, katalog książek)

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

ISBN: 978-83-289-2213-6

Printed in Poland.

SPIS TREŚCI

DLA KOGO JEST PRZEZNACZONA TA KSIĄŻKA?

Najwięcej z lektury tej książki wyniosą:

Początkujący programiści, którzy chcą się nauczyć języka Go od podstaw. W książce znajdą oni wprowadzenie do struktury programów, zmiennych, funkcji, wskaźników, interfejsów oraz narzędzi ekosystemu Go.

Średnio zaawansowani programiści, którzy chcą poszerzyć swoją wiedzę na temat współbieżności, testowania, obsługi błędów i zaawansowanych wzorców programistycznych w Go.

Przyszli twórcy gier i aplikacji multimedialnych, którzy chcą użyć Go do tworzenia gier 2D i 3D, z naciskiem na biblioteki takie jak raylib-go, go-sdl2, beep oraz Ebitengine. Znajdą tu liczne przykłady dotyczące animacji, grafiki i fizyki, jak też być może natchnienie do własnych projektów, które będzie można uruchamiać w różnych systemach operacyjnych.

Warto jednak podkreślić, że nie jest to szczegółowa dokumentacja języka Go, ale raczej uzupełnienie wprowadzenia do jego świata. Przedstawiam tu podejście oparte na własnych doświadczeniach, z myślą o spłaszczeniu krzywej uczenia się. Przez analizowanie przykładowych kodów, od prostych do bardziej zaawansowanych projektów, nauka staje się łatwiejsza i bardziej przystępna. Konkretne przykłady pokazują sposoby na uzyskanie określonych efektów, które, mam nadzieję, pobudzą Twoje zmysły i pozwolą Ci szybko wystartować z językiem Go. Zapraszam też do odwiedzenia forum Lothar-TeaM, dostępnego pod adresem *http://forum.lothar-team.pl*. (Jego nazwa nawiązuje do grupy zapaleńców z czasów Commodore 64 i Amigi, której miałem zaszczyt być członkiem). Znajdziesz na nim przyjazną atmosferę i wiele ciekawych projektów opartych na różnych mikrokontrolerach oraz dużo informacji o programowaniu, nie tylko w Go, ale też w C, C++, Pythonie i C#. Tam też znajdziesz kody z tej książki i dodatkowe objaśnienia. Odpowiem także chętnie na każde pytanie. Zapraszam serdecznie.

W tym miejscu chciałbym serdecznie podziękować Ci za to, że przyjąłeś zaproszenie do programowania w języku Go. Dziękuję też mojej żonie Dorocie za wsparcie i to, że wytrzymała ze mną w czasie, gdy pisałem tę książkę, jak również przyjaciołom z naszego forum: Nefariousowi, Marasowi, Xbary'emu, l3n1n i wielu innym, których w tym miejscu nie wymieniłem. Specjalne podziękowania należą się koledze Foreste, który znosi moje żarty, takie jak choćby specjalna gra z naszym liskiem Foreste w roli głównej.

Dziękuję Wam — bez Was ta książka by nie powstała.

SunRiver

WSTĘP

Gratuluję Ci zakupu tej książki i decyzji o rozpoczęciu swojej przygody z językiem Go! Jesteś na dobrej drodze, aby poznać jedno z najbardziej nowoczesnych i efektywnych narzędzi, które może znacząco wpłynąć na Twoją karierę programisty.

Wielu programistów, podobnie jak Ty, szuka prostszego, bardziej intuicyjnego sposobu na tworzenie wydajnych aplikacji. Typowym problemem, z jakim mogą się spotkać, jest złożoność języków programowania i narzędzi, które wymagają obszernej wiedzy i skomplikowanej konfiguracji. Może być trudno znaleźć narzędzie, które nie tylko zapewnia prostotę, ale też jest na tyle potężne, aby sprostać współczesnym wymaganiom. Jeśli zastanawiasz się, jak szybko i efektywnie tworzyć aplikacje, nie tracąc godzin na konfigurację czy debugowanie, to ta książka jest dla Ciebie.

Pokażę Ci, jak:

- szybko rozpocząć pracę z Go, nawet jeśli dopiero zaczynasz programowanie,
- budować GUI,
- prosto i efektywnie korzystać z bibliotek graficznych,
- napisać intro jak z dawnej demosceny,
- napisać grę w stylu retro.

Przedstawię też wiele innych przydatnych przykładów.

Go sprawił, że moja praca stała się bardziej efektywna i przyjemna. Dlatego zdecydowałem się podzielić tą wiedzą z Tobą, abyś mógł korzystać z zalet Go w swoich projektach. Na własnej skórze przekonałem się, jak Go uprościł moje projekty. Kiedy po raz pierwszy wdrożyłem aplikację narzędziową napisaną

w Go, zaskoczyła mnie jej stabilność, wydajność i to, jak mało zasobów zużywała w porównaniu z innymi językami. Podobne efekty widziałem u innych programistów, którzy przeszli na Go — ich aplikacje działały szybciej, a oni sami byli bardziej zadowoleni z efektów swojej pracy.

Nie czekaj zbyt długo z rozpoczęciem nauki. Postęp techniczny nie zwalnia, a każda chwila, którą tracisz na niewydajne rozwiązania, to czas, który mógłbyś poświęcić na doskonalenie swoich umiejętności w Go. Im szybciej zaczniesz, tym szybciej zobaczysz efekty — Twoje aplikacje będą bardziej responsywne, łatwiejsze do skalowania i gotowe na przyszłe wyzwania. Zachęcam Cię do natychmiastowego rozpoczęcia nauki i pełnego wykorzystania wiedzy zawartej w tej książce. Twoja podróż z Go właśnie się zaczyna — już teraz wejdź na kolejny poziom programowania!

Gotowy? Zaczynamy!

CZĘŚĆ I

PODSTAWY JĘZYKA GO

1

WPROWADZENIE DO JĘZYKA GO

Język Go, znany również jako Golang, to dzieło zespołu inżynierów Google — Roberta Griesemera, Roba Pike'a i Kena Thompsona — którzy ukończyli je w 2007 roku. Jego powstanie było odpowiedzią na rosnącą potrzebę języka, który byłby szybki, łatwy w użyciu, a zarazem dobrze dostosowany do współczesnych wyzwań związanych z tworzeniem oprogramowania, zwłaszcza w środowiskach sieciowych i wielowątkowych. W 2009 roku język udostępniono publicznie i od tego momentu zdobywa coraz większą popularność wśród programistów. Oczywiście świat komputerów zdominowały języki programowania takie jak C i C++. Ich dominacja w tworzeniu systemów o wysokiej wydajności od lat się nasilała, ale miały pewne wady, które zniechęcały młodych adeptów sztuki programistycznej — głównie chodziło o skomplikowaną składnię, długie czasy kompilacji oraz trudności związane z zarządzaniem pamięcią. Z drugiej strony, chociaż języki takie jak Python czy Ruby zapewniały prostotę i elastyczność, brakowało im wydajności i obsługi złożonych operacji wielowątkowych. Mimo to i one znalazły swoje miejsce w szeregu, więc można by się pokusić o pytanie, czy nowy język był nam potrzebny.

Otóż okazuje się, że tak. W środowisku programistów przyda się powiew świeżości i może nawet uda mu się skusić zarówno tych starszych, jak i nowych.

W sumie mógłbym tu napisać, że za moich czasów, gdy zaczynałem przygodę z programowaniem, istniały 10 grupy programistów:

- ci, którzy rozumieli kod binarny,
- i pozostali.

Później świat się zmienił i zmieniła się technologia. Od wydania StormC na komputery Amiga pokochałem ten język i można powiedzieć, że trochę nim przesiąkłem. Przez lata programowałem w różnych językach, ale zawsze z lubością wracałem do C, choć mógłbyś pomyśleć, że do assemblera. I choć

Python znacząco zmienił podejście do programowania za sprawą swojej prostoty, też go polubiłem. Ma on jednak swoje wady, więc, muszę to przyznać, byłem nieco rozdarty między dwoma światami: poważnego języka C i prostego Pythona.

Dlatego moim zdaniem Go stworzono po to, by połączył te światy. Jest szybki jak C, ale prosty jak Python. Projektanci Go wybrali prostotę jako jedno z głównych założeń projektowych — zarówno w składni, jak też podejściu do zarządzania pamięcią i współbieżności.

1.1. Główne cechy języka Go

Język Go, jak już pewnie się domyślasz, jest dosyć specyficzny. Można w nim natrafić na wiele podobieństw z innymi językami programowania, jednak ma też kilka unikalnych cech, które sprawiają, że wyróżnia się na tle pozostałych. Przede wszystkim jego prosta, wręcz minimalistyczna składnia znacznie ułatwia szybkie pisanie i rozumienie kodu. Zapewnia też bezpośrednią obsługę współbieżności poprzez tzw. *goroutines*, czyli lekkie wątki zarządzane przez *runtime* Go. Dzięki nim łatwo jest pisać programy, które równocześnie wykonują wiele zadań. Inną ważną cechą jest tzw. ***garbage collector*** — pozwala on automatycznie zarządzać pamięcią, co eliminuje ryzyko błędów typowych dla ręcznego zarządzania, takich jak wycieki pamięci. Dzięki temu zamiast pilnować pamięci, programista może skupić się na logice aplikacji. Ponadto Go jest językiem o statycznej typizacji, co dla programisty oznacza, że typy muszą być jawnie deklarowane. I choć może się to wydawać ograniczeniem, zapewnia też pewne udogodnienia, takie jak możliwość dedukcji typu (ang. *type inference*). W odróżnieniu od takich języków jak Python, C# czy Java, Go jest językiem kompilowanym, podobnie jak C/C++. Można więc śmiało powiedzieć, że w odróżnieniu od wielu innych języków zaprojektowano go z myślą o szybkim procesie kompilacji. Nawet duże projekty kompilują się błyskawicznie, co znacznie przyspiesza cykl rozwoju aplikacji. Dodatkowo język Go oferuje bardzo bogaty zestaw wbudowanych narzędzi, takich jak `go fmt` do automatycznego formatowania kodu, `go test` do testowania jednostkowego oraz `go mod` do zarządzania zależnościami. Te wszystkie cechy sprawiają że programowanie w tym języku może być łatwe i przyjemnie, ale też tworzone w nim programy, od kodu do pliku binarnego, zapewniają prostotę i przejrzystość kodu przed kompilacją i nie ustępują wydajnością tym pisanym w innych językach.

1.2. Typowe zastosowania Go

Język Go zyskał powszechne uznanie programistów dzięki swoim unikalnym cechom, które czynią go idealnym rozwiązaniem do tworzenia systemów rozproszonych, aplikacji sieciowych oraz mikrousług. Jego prostota, wydajność oraz wbudowana obsługa współbieżności sprawiają, że jest chętnie wybierany do tworzenia wysokowydajnych serwerów HTTP, narzędzi do przetwarzania danych w czasie rzeczywistym, a także platform chmurowych.

Jednym z najważniejszych atutów Go jest model współbieżności oparty na wspomnianych *goroutines*, które umożliwiają efektywne zarządzanie wieloma zadaniami jednocześnie przy minimalnym zużyciu zasobów. Dzięki temu Go jest wyjątkowo przydatny w aplikacjach, w których wymagana jest obsługa wielu równoległych operacji, np. w serwerach obsługujących setki tysięcy połączeń sieciowych jednocześnie. Stał się także fundamentem dla wielu znanych projektów open source, co przyczyniło się do dalszego wzrostu jego popularności. Do najważniejszych z nich należą Docker, Kubernetes i Prometheus.

Jeśli nie słyszałeś o tych projektach, to przybliżę Ci je możliwie zwięźle, byśmy mogli jak najszybciej przejść do poznawania języka i programowania. Docker to znana platforma do konteneryzacji, która zrewolucjonizowała sposób wdrażania aplikacji i zarządzania nimi. Kubernetes to system do zarządzania kontenerami na dużą skalę. Prometheus zaś to narzędzie do monitorowania i alertowania.

Już te trzy przykłady dobitnie świadczą o zdolności Go do obsługi złożonych, skalowalnych systemów.

Za sprawą wymienionych właściwości Go stał się popularnym wyborem inżynierów DevOps oraz architektów systemów rozproszonych. Jego prostota w połączeniu z wydajnością i bogatym ekosystemem bibliotek sprawia, że często się go używa do budowania nowoczesnych aplikacji opartych na mikroserwisach i narzędzi do zarządzania chmurą oraz automatyzacji infrastruktury. W tej książce jednakże pokażę Ci zupełnie inne podejście do nauki języka, jak i do programowania.

2

PODSTAWY JĘZYKA GO

Zrozumienie podstaw Go to pierwszy krok na drodze do efektywnego programowania w tym języku. Zapewnia on prostą, a zarazem wydajną składnię, która łączy w sobie cechy języków wysokopoziomowych z możliwością tworzenia wydajnych i skalowalnych aplikacji.

W tym rozdziale omówimy zmienne, typy danych, funkcje, struktury sterujące oraz unikalne mechanizmy Go, takie jak *goroutines* i kanały. Przyjrzymy się także sposobowi obsługi błędów oraz deklarowania interfejsów i pakietów standardowych w Go. Niektórym zagadnieniom poświęcę więcej miejsca, gdyż są na tyle ciekawe, że warto się nad nimi bardziej pochylić. Postaram się przekazywać Ci informacje w taki sposób, żebyś łatwo i szybko był wstanie przejść do tworzenia programów samodzielnie. W tym celu pokażę Ci, jak można maksymalnie spłaszczyć krzywą uczenia się Go, by szybko opanować sztukę programowania w tym języku. Zacznijmy więc łagodnie, od poznania języka. Dodajmy: poznania go we właściwy sposób, czyli w kodzie. Wszystko, czego dowiesz się z tego rozdziału, będziesz mógł od razu zastosować. Oczywiście przykłady te są dosyć syntetyczne i w większości pozbawione głębszego sensu, ale dokładnie obrazują konkretne sytuacje i metody. Dzięki temu łatwo przejdziesz do prawdziwego programowania.

Gotowy? No to Go…

2.1. Struktura programu Go

Każdy program, nie tylko w języku Go, ma określoną strukturę, czyli zestaw instrukcji, które muszą być obecne w każdym kodzie. W Go takimi elementami są pakiet `main` i funkcja `main()`. Programu bez tej struktury nie można uruchomić jako aplikacji. Pakiety pełnią też ważną funkcję w organizowaniu kodu: zapewniają mu modularność i możliwość wielokrotnego użycia. Pokażę Ci to na prostym przykładzie:

```
package main
import "fmt"
func main() {
    fmt.Println("Witaj, świecie!")
}
```

Jak widzisz, powyższy kod przedstawia najprostszy program, jaki możemy napisać w języku Go. Najważniejsze jego elementy to:

- **`package main`** — to on określa główny pakiet aplikacji. Każdy program w Go musi mieć pakiet `main`.
- **`import`** — służy do importowania innych pakietów i modułów. W tym konkretnym przypadku jest to pakiet `fmt` (standardowy pakiet do formatowania tekstu).
- **`func main()`** — to główna funkcja naszego programu, która odpowiada za jego wykonanie.
- **`fmt.Println()`** — to jedyna instrukcja naszego programu. Wywołuje funkcję `Println()`, czyli „wydrukuj linię", z pakietu `fmt`, która wypisuje (drukuje na wyjście standardowe, którym jest wiersz poleceń) tekst zawarty w parametrze instrukcji.

Warto ten układ zapamiętać, gdyż każdy z naszych kodów będzie właśnie tak skonstruowany. Oczywiście pojawi się wiele elementów i instrukcji, ale szkielet zawsze będzie właśnie taki.

2.2. Zmienne

Zmienne w języku Go mają ciekawą, dość unikalną właściwość. Mianowicie można je deklarować na różne sposoby. Standardowo w języku Go typy zmiennych muszą być jawnie deklarowane, czyli stosuje się statyczną typizację, ale język ten umożliwia również dedukcję typów. W uproszczeniu: sam zgaduje, jakiego typu jest nasza zmienna. I choć wydaje się to bardzo skomplikowane, za chwilę zobaczysz, jak działa ten mechanizm i jak łatwy jest w użyciu.

Deklarowanie zmiennych

Słowo kluczowe `var` (ang. *variable*) służy właśnie do jawnego deklarowania zmiennych. Zwyczajowo zgodna z zasadami deklaracja zmiennej w kodzie przedstawia się następująco:

```
var x int = 10
var y float64 = 20.5
```

Jak widzisz, mamy tu przykład tzw. zmiennej zainicjalizowanej, czyli z podaną wartością początkową. Natomiast jeśli zastosujemy taki zapis:

```
var a int
var b flat32
var txt string
```

to wykonamy przypisanie zmiennej typu, ale bez przypisania wartości. To tzw. zmienne niezainicjalizowane. W takim przypadku zmienna otrzymuje wartość domyślną, tzw. *zero value*, odpowiednią do jej typu.

Jest to właściwa i podstawowa metoda deklarowania zmiennych w języku Go, w której po słowie kluczowym var podajemy nazwę zmiennej, a następnie jej typ i/lub, po operatorze przypisania =, wartość, jaką reprezentuje. Jednak jak już wspomniałem, język Go pozwala deklarować zmienne — mimo że nie jest to wskazane — bez podawania ich typu. Kompilator sam dedukuje typ zmiennej na podstawie przypisywanej wartości. Dedukcja typu następuje w momencie kompilacji. Zmienne zainicjalizowane w kodzie można więc zapisywać w następujący sposób:

```
var a = 10        // Kompilator sam rozpozna, że zmienna 'a' jest typu int
var b = "Witaj"   // Kompilator rozpozna, że zmienna 'b' to string
```

Do tej pory wszystko jest, jak widzisz, proste, jednak nieco zamieszamy, Go umożliwia bowiem jeszcze krótszy zapis: bez użycia słowa kluczowego `var`, ale wraz z automatycznym rozpoznaniem typu. Zapis ten powstaje z użyciem operatora `:=`, np.

```
x := 10        // Go automatycznie dedukuje, że x jest typu int
y := 20.5      // Dedukcja typu - float64
z := "Witaj"   // Dedukcja string
```

To bardzo wygodna forma, ponieważ w wielu przypadkach pozwala nie określać jawnie typu, co przyspiesza pisanie kodu. Dzięki zaś dedukcji typu Go zachowuje swoją statyczną typizację, ale zarazem pozwala pisać kod bardziej zwięzły i czytelny. A wszystko to bez ryzyka błędów charakterystycznych dla dynamicznie typowanych języków.

Jednak warto zauważyć, że w przeciwieństwie do części języków, w których zmienne są dynamicznie typowane, dedukcja typu w języku Go nie oznacza, że typ zmiennej może się zmieniać w trakcie działania programu. Raz określony typ zmiennej (nawet jeśli jest dedukowany) pozostaje stały. Na przykład:

```
a := 10      // a jest typu int
a = "Hello" // Wystąpi błąd kompilacji, ponieważ zmienna a nie może być stringiem
```

To właśnie sprawia, że Go pozostaje językiem statycznie typowanym, ale zapewnia przy tym podobną elastyczność jak języki dynamicznie typowane.

Oczywiście, żeby nie było za łatwo, w języku Go można zadeklarować wiele zmiennych w jednej linii, co jeszcze bardziej poprawia czytelność kodu, zwłaszcza gdy mamy do czynienia z grupą powiązanych zmiennych. Spójrz na poniższy przykład:

```
var a, b, c int        // Deklaracja trzech zmiennych typu int
var x, y = 1, "Witaj"  // x to int, y to string
```

Jeśli wolisz krótszy zapis, to możesz zrobić tak:

```
a, b := 10, 20       // a i b zostaną zadeklarowane jako int
x, y := "Witaj", 3.14 // x to string, y to float64
```

Typy danych

Język Go oferuje szeroki wachlarz wbudowanych typów danych, takich jak:

- **typy całkowite** — `int`, `int8`, `int16`, `int32`, `int64`,
- **typy zmiennoprzecinkowe** — `float32`, `float64`,
- **znaki** — `byte` (8-bitowy znak) oraz `rune` (32-bitowy Unicode),
- **typ logiczny** — `bool` (czyli *boolean*; może przyjmować wartości *true* lub *false*),
- **łańcuchy znaków** — `string`, który jest sekwencją bajtów o stałej długości.

Zasięg i domyślne wartości

W języku Go są dostępne różne sposoby deklarowania zmiennych. Różnią się one zakresem widoczności zmiennej i sposobem inicjalizacji:

1. **Zmienna zadeklarowana z użyciem `var`:**
   - Może być używana w funkcji, bloku kodu (np. `if`, `for`) lub na poziomie pakietu, w zależności od miejsca deklaracji.
   - Jej zakres obejmuje funkcję lub blok, w których została zadeklarowana.
   - Można ją zadeklarować bez inicjalizacji — wówczas otrzymuje wartość domyślną dla jej typu.
2. **Zmienna zadeklarowana za pomocą `:=`:**
   - Jest ograniczona do najbliższego bloku kodu, w którym została zadeklarowana (np. wewnątrz funkcji, pętli lub instrukcji warunkowej).

- Zawsze wymaga jednoczesnej inicjalizacji — nie można użyć `:=` bez przypisania wartości.
- `:=` jest dostępne tylko w funkcjach, a więc nie można go używać do deklarowania zmiennych globalnych.

Widać to na przykładzie kodu:

```
package main
import "fmt"

func main() {
        // Deklaracja z var
        var a int           // Domyślna wartość to 0
        fmt.Println(a)      // Wyświetli: 0

        // Deklaracja z :=
        if true {
            b := 10         // Deklaracja i inicjalizacja w bloku if
            fmt.Println(b)  // Wyświetli: 10
        }
        // fmt.Println(b)      // Błąd: b nie istnieje poza blokiem if
}
```

W Go zmienna, która nie została zainicjalizowana jakąś wartością, otrzymuje dla swojego typu tzw. *zero value*. Oto przykłady domyślnych wartości dla popularnych typów:

- Dla typu liczbowego (`int`, `float64`, itp.) domyślna wartość to 0.
- Dla `bool` domyślna wartość to `false`.
- Dla typu `string` domyślna wartość to pusty string `""`.
- Dla wskaźników i interfejsów domyślna wartość to `nil`.

Deklaracja zmiennych globalnych

Znając więc ograniczenia zasięgu zmiennych, możemy stwierdzić, że potrzebujemy zmiennych globalnych, czyli takich, których możemy używać w obrębie całego programu. Możemy je zadeklarować poza funkcjami, ale tylko za pomocą pełnej formy deklaracji (czyli ze słowem kluczowym `var`). Krótsza forma `:=` jest dostępna wyłącznie wewnątrz funkcji dla zmiennych lokalnych.

```
package main
import "fmt"

var globalVar = 100  // Zmienna globalna dostępna w całym programie i każdej funkcji
func  global() {
   fmt,Println( "Wartość zmiennej globalnej w funkcji global ",globalVar)
}
```

```
func main() {
    fmt.Println("Wartość zmiennej globalnej w pętli głównej",globalVar)
   global()
}
```

Zapamiętaj:

- `var` służy do pełnego deklarowania zmiennych z możliwością przypisania wartości.
- Pomijanie typu w deklaracji zmiennej umożliwia mechanizm dedukcji typu na podstawie przypisanej wartości.
- `:=` to wygodna forma skrócona do jednoczesnego zadeklarowania i zainicjalizowania zmiennej, ale używana w lokalnym zasięgu funkcji.
- Można deklarować wiele zmiennych jednocześnie, co ułatwia organizowanie kodu.

W języku Go w kwestii zmiennych stawia się na prostotę i przejrzystość, co jest zgodne z ogólnym podejściem tego języka, które polega na unikaniu zbędnej złożoności przy jednoczesnym zachowaniu dużej wydajności i bezpieczeństwa.

Trzy, dwa, jeden, Go!

Wielu programistów szuka prostego, bardziej intuicyjnego sposobu na tworzenie aplikacji. Przeszkadza im złożoność języków programowania, nieczytelność ich składni i stopień skomplikowania konfiguracji. W trakcie poszukiwań narzędzi zapewniających prostotę i dużą wydajność trafiają często na stworzony przez inżynierów Google język Go i... okazuje się, że spełnia on te oczekiwania, jest prosty jak Python i wydajny jak Java. Jeśli i Ty chcesz tworzyć aplikacje szybciej i efektywniej, nie tracić przy tym czasu na skomplikowane konfiguracje czy debugowanie, ta książka jest dla Ciebie.

Dowiesz się z niej między innymi:

- **jak łatwo rozpocząć pracę z Go, nawet jeśli dopiero uczysz się programować**
- **jak prosto i efektywnie korzystać z bibliotek graficznych odpowiednich dla tego języka**
- **jak zrealizować w Go pierwszy sprawny projekt, na przykład grę w stylu retro**

Zacznij programować w języku Go i przekonaj się, jak uprości to Twoje projekty. Napisz i wdróż w nim aplikację — i daj się zaskoczyć jej stabilnością, wydajnością, a także tym, jak mało zasobów zużywa w porównaniu z aplikacjami pisanymi w innych językach.

Andrzej „SunRiver” Gromczyński

Pasjonat elektroniki, programowania systemów wbudowanych, metod numerycznych i sztucznej inteligencji. Przez wiele lat, zarówno w pracy, jak i w ramach hobby, skupiał się na elektronice i programowaniu mikrokontrolerów. Zawodowo zajmuje się serwisem urządzeń elektronicznych, automatyką przemysłową, tworzeniem nowych urządzeń elektronicznych i renowacją zabytkowych urządzeń pomiarowych. Prowadzi forum dla amatorów i pasjonatów elektroniki i programowania: *forum.lothar-team.pl*. W świecie internetowych entuzjastów elektroniki znany jako SunRiver, jest aktywnym uczestnikiem wielu forów internetowych poświęconych szeroko rozumianej elektronice.

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 978-83-289-2213-6

9 788328 922136

Cena: 69,00 zł