Extracted from:

Go Brain Teasers

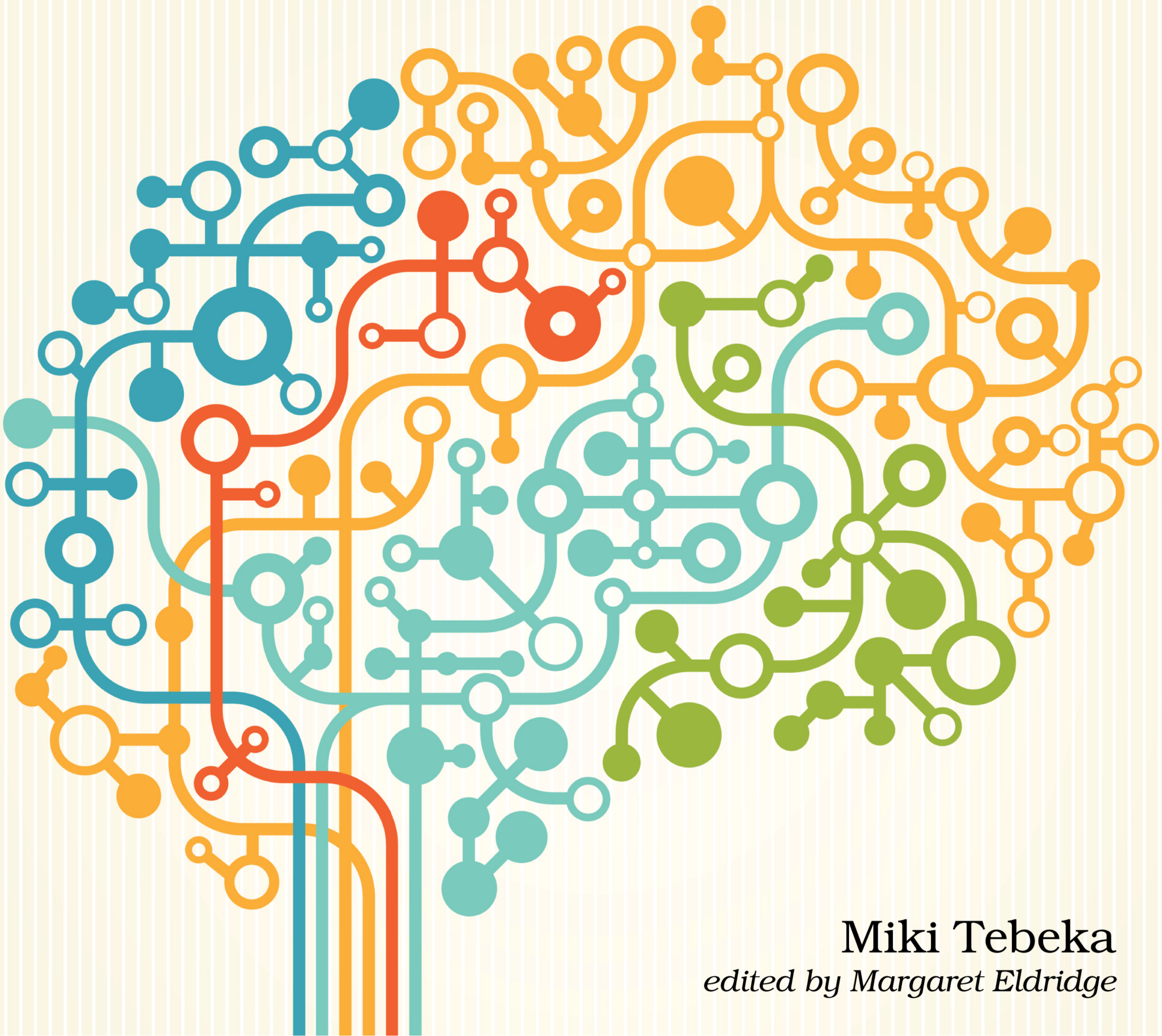
Exercise Your Mind

The Pragmatic Bookshelf
Raleigh, North Carolina

The Pragmatic Programmers

Go Brain Teasers

Exercise Your Mind



Miki Tebeka

edited by Margaret Eldridge

Go Brain Teasers

Exercise Your Mind

Miki Tebeka

The Pragmatic Bookshelf
Raleigh, North Carolina

The team that produced this book includes:

CEO: Dave Rankin
COO: Janet Furlow
Managing Editor: Tammy Coron
Development Editor: Margaret Eldridge
Copy Editor: Jennifer Whipple
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics
Founders: Andy Hunt and Dave Thomas

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-899-4
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—September 2021

To Sharon, who suffered me in quarantine,
and the twenty years before that.

A Funky Number?

num.go

```go
package main

import (
	"fmt"
)

func main() {
	fmt.Println(0x1p-2)
}
```

Guess the Output

Try to guess what the output is before moving to the next page.

This code will print: 0.25

Go has several number types. The two main ones are

Integer
These are whole numbers. Go has int8, int16, int32, int64, and int. There are also all the unsigned ones such as uint8 and so on.

Float
These are real numbers. Go has float32 and float64.

There are other types such as complex and the various types defined in math/big.

When you write a number literal, such as 3.14, the Go compiler needs to parse it to a specific type (float64, in this case). The Go spec[2] defines how you can write numbers. Let's have a look at some examples.

num_lit.go
```go
package main

import (
	"fmt"
)

func main() {
	// Integer
	printNum(10)    // 10 of type int
	printNum(010)   // 8 of type int
	printNum(0x10)  // 16 of type int
	printNum(0b10)  // 2 of type int
	printNum(1_000) // 1000 of type int <1>

	// Float
	printNum(3.14)   // 3.14 of type float64
	printNum(.2)     // 0.2 of type float64
	printNum(1e3)    // 1000 of type float64
	printNum(0x1p-2) // 0.25 of type float64

	// Complex
	printNum(1i)     // (0+1i) of type complex128
	printNum(3 + 7i) // (3+7i) of type complex128
	printNum(1 + 0i) // (1+0i) of type complex128
}

func printNum(n interface{}) {
	fmt.Printf("%v of type %T\n", n, n)
}
```

2. https://golang.org/ref/spec#Lexical_elements

`_` serves as the thousands separator. It makes big numbers much more readable for us humans.

`1e3` is known as *scientific notation*.

`0x1p-2` is called a *hexadecimal floating-point literal* in the Go specification and is following the IEEE 754 2008 specification. To calculate the value, do the following:

- Compute the value before the `p` as a hexadecimal number. In this example it's `0x1` = 1.
- Compute the value after the `p` as *2 to the power of that value*. In this example it's `2-2` = 0.25.
- Finally, multiply the two numbers, in this example, `1 * 0.25` = 0.25.

Further Reading

Lexical Elements in the Go Specification
    golang.org/ref/spec#Lexical_elements

Scientific Notation on Wikipedia
    en.wikipedia.org/wiki/Scientific_notation

IEEE 754 on Wikipedia
    en.wikipedia.org/wiki/IEEE_754

Integer Literals
    golang.org/ref/spec#Integer_literals

Floating-Point Literals
    golang.org/ref/spec#Floating-point_literals

Imaginary Literals
    golang.org/ref/spec#Imaginary_literals