Reading Input from the User

This lesson explains how to receive input from the user through the keyboard.

WE'LL COVER THE FOLLOWING

- Reading input from the keyboard

Apart from the packages `fmt` and `os`, we will also need to import and use the package `bufio` for buffered input and output.

Reading input from the keyboard

How can we read user input from the keyboard (console)? Input is read from the keyboard or standard input, which is `os.Stdin`. The simplest way is to use the `Scan-` and `Sscan-family` of functions of the package `fmt`, as illustrated in this program:

Environment Variables

Key:	Value:
GOROOT	/usr/local/go
GOPATH	//root/usr/local/go/src
PATH	//root/usr/local/go/src/bin:/usr/local/go...

```
package main
import "fmt"

var (
      firstName, lastName, s string
      i int
      f float32
      input = "56.12 / 5212 / Go"
      format = "%f / %d / %s"
)

func main() {
    fmt.Println("Please enter your full name: ")
    fmt.Scanln(&firstName, &lastName)
    // fmt.Scanf("%s %s", &firstName, &lastName)

    fmt.Printf("Hi %s %s!\n", firstName, lastName) // Hi Chris Naegels
    fmt.Sscanf(input, format, &f, &i, &s)
    fmt.Println("From the string we read: ", f, i, s)
}
```

Click the **RUN** button, and wait for the terminal to start. Type `go run main.go` and press ENTER.

The **Scan** functions require us to first define the variables in which the input is to be stored. So from **line 5** to **line** 7, we declare a total of **5** variables:

- At **line 5**, we have *three* string variables: `firstname`, `lastname`, and `s`.
- At **line 6**, we have *one* integer variable: `i`.
- At **line** 7, we have *one* float32 type variable: `f`.

At line 14, we use `fmt.Scanln` to read in a line from the keyboard until the user types ENTER. We asked the user to type in their full name so in the format `firstname` SPACE `lastname`. These strings will be stored in the same order in the parameters to `Scanln`: `fmt.Scanln(&firstName, &lastName)`. `Scanln` needs *references* to the storage variables, that's why it takes the parameters `&firstName` and `&lastName`. To verify that, we print out the variables at **line 16**.

Go has also the `Scanf`, `Fscanf`, and `Sscanf` functions that parse the arguments according to a format string, analogous to that of `Printf`. `Scanf` reads input from the keyboard, but `Sscanf` reads from another string.

At **line 17**, we see that `Sscanf` takes the following parameters, which were also defined in the variables section:

- **input**: the string to be read
- **format**: the format to read and parse input

This contains (a list of) the storage variables `&f`, `&i`, and `&s` for storing the input. So, the input e.g., **56.12 / 5212 / Go** is parsed according to format **%f / %d / %s** and stored in the variables `&f`, `&i`, and `&s`, respectively.

`Scanln` scans the text read from standard input, storing successive space-separated values into successive arguments; it stops scanning at a newline.

Scanf does the same, but it uses its first parameter as the format string to read the values in the variables. Sscan and friends work the same way but

read from an input string instead of reading from standard input. You can check on the number of items read in and the error when something goes wrong. However, it can also be done with a buffered reader from the package bufio, as is demonstrated in the following program:

Environment Variables

Key:	Value:
GOROOT	/usr/local/go
GOPATH	//root/usr/local/go/src
PATH	//root/usr/local/go/src/bin:/usr/local/go...

```
package main
import (
"fmt"
"bufio"
"os"
)

var inputReader *bufio.Reader
var input string
var err error

func main() {
  inputReader = bufio.NewReader(os.Stdin)
  fmt.Println("Please enter some input: ")

  input, err = inputReader.ReadString('\n')
  if err == nil {
    fmt.Printf("The input was: %s\n", input)
  }
}
```

Click the **RUN** button, and wait for the terminal to start. Type `go run main.go` and press ENTER.

The variable `inputReader` is a pointer to a `Reader` in `bufio`. This reader is created and linked to the standard input in the line: `inputReader := bufio.NewReader(os.Stdin)`. The `bufio.NewReader()` constructor has the signature: `func NewReader(rd io.Reader) *Reader`. It takes as an argument, any object that satisfies the `io.Reader` interface (any object that has a suitable `Read()` method) and returns a new buffered `io.Reader` that reads from the the given reader, `os.Stdin` satisfies this requirement.

The reader has a method `ReadString(delim byte)`, which reads the input until

The reader has a method `ReadString(delim byte)`, which reads the input until `delim` is found. The `delim` is included; what is read is put into a buffer.

`ReadString` returns the string read and nil for the error. When it reads until the end of a file, the string read is returned and `io.EOF`; if `delim` is not found, an error `err != nil` is returned. In our case, the input from the keyboard is read until the ENTER key (which contains '\n') is typed. Standard output `os.Stdout` is on the screen. `os.Stderr` is where the error-info can be written to, and it is mostly equal to `os.Stdout`.

The package `bufio` allows you to read input from the keyboard or any other source with its `Reader` type, which is declared at **line 8**, and created at **line 13**. The `Reader` has all kinds of methods to read in. The simplest is `ReadString`, which reads into the input as one big string until a *newline* character is found, and returns an error `err` if there was a problem (**line 16**). If no error is found, the input is printed at **line 18**.

In normal Go-code, the var-declarations are omitted and the variables are declared with `:=`, like:

```
inputReader := bufio.NewReader(os.Stdin)
input, err := inputReader.ReadString('\n')
```

We will apply this idiom from now on. The second example reads input from the keyboard with a few different switch-versions:

Environment Variables

Key:	Value:
GOROOT	/usr/local/go
GOPATH	//root/usr/local/go/src
PATH	//root/usr/local/go/src/bin:/usr/local/go...

```
package main
import (
"fmt"
"os"
"bufio"
)

func main() {
  inputReader := bufio.NewReader(os.Stdin)
  fmt.Println("Please enter your name:")
  input, err := inputReader.ReadString('\n')
  if err != nil {
    fmt.Println("There were errors reading, exiting program.")
    return
  }

  fmt.Printf("Your name is %s", input)

  // For Unix: test with delimiter "\n", for Windows: test with "\r\n"
  switch input {
    case "Philip\n": fmt.Println("Welcome Philip!")
    case "Chris\n": fmt.Println("Welcome Chris!")
    case "Ivo\n": fmt.Println("Welcome Ivo!")
    default: fmt.Printf("You are not welcome here! Goodbye!\n")
  }
  // version 2:
  switch input {
    case "Philip\n": fallthrough
    case "Ivo\n": fallthrough
    case "Chris\n": fmt.Printf("Welcome %s\n", input)
    default: fmt.Printf("You are not welcome here! Goodbye!\n")
  }

  // version 3:
  switch input {
    case "Philip\n", "Ivo\n": fmt.Printf("Welcome %s\n", input)
    default: fmt.Printf("You are not welcome here! Goodbye!\n")
  }
}
```

Click the **RUN** button, and wait for the terminal to start. Type `go run main.go` and press ENTER.

This example shows input obtained with `ReadString` at **line 11** being tested in a few *switch* constructs. Note how it is important to distinguish between the line-endings on Unix and Windows. For example, on Windows, you would code this as:

```
case "Philip\r\n": fmt.Println("Welcome Philip!")
```

Now that you know how to take input from the keyboard, the next lesson brings you a challenge to solve.