Pointers and Addresses

Created	@Jan 28, 2021 10:16 PM
Tags	Lesson

The Point of Pointers and Addresses

- Go is a pass-by-value language
  - We pass functions the *value* of an argument
- The `&` operator gets the address of a variable
  - `x := "My very first address"`
    `fmt.Println(&x)`
  - Prints: `0x414020` the address of `x` - a number formatted in hexadecimal

Pointers

- Pointers are used to store addresses
  - `var pointerForInt *int`
- The `*` operator signifies that this variable will store an address

```
var pointerForInt *int

minutes := 525600

pointerForInt = &minutes

fmt.Println(pointerForInt) // Prints 0xc000018038
```

- Pointers can also be implied implicitly like other variables `pointerForInt := &minutes`

Dereferencing

- We use the `*` operator again on a pointer to dereference it and access the actual variable

```
lyrics := "Moments so dear"
pointerForStr := &lyrics

*pointerForStr = "Journeys to plan"

fmt.Println(lyrics) // Prints: Journeys to plan
```

Changing Values in Different Scopes

- Using pointers can allow us to access variables outside of their scope:

- This does not incorporate the change to `num`

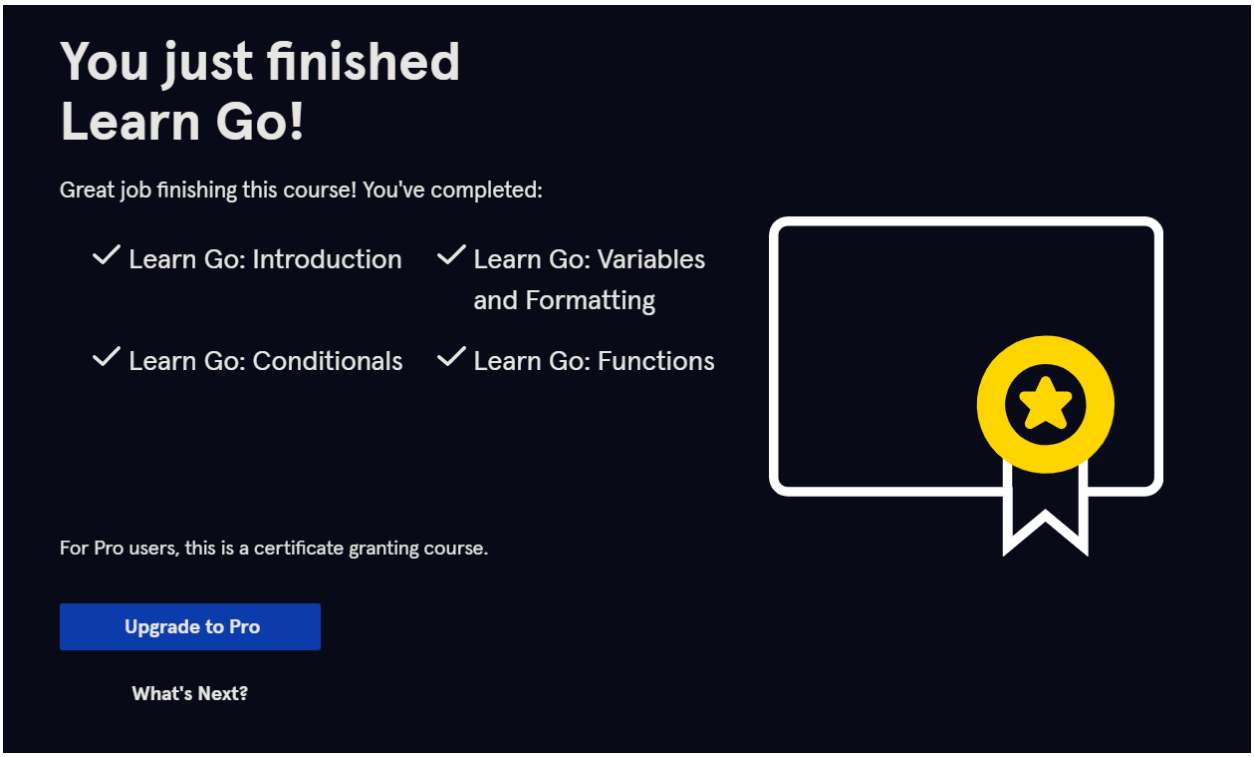
```
func addHundred(num int) {
  num += 100
}

func main() {
  x := 1
  addHundred(x)
  fmt.Println(x) // Prints 1
}
```

- When a pointer is passed, the change is made

```
func addHundred (numPtr *int) {
  *numPtr += 100
}

func main() {
  x := 1
  addHundred(&x)
  fmt.Println(x) // Prints 101
}
```


You just finished Learn Go!
Great job finishing this course! You've completed:
Learn Go: Introduction
Learn Go: Variables and Formatting
Learn Go: Conditionals
Learn Go: Functions
For Pro users, this is a certificate granting course.
Upgrade to Pro
What's Next?