# GO on the toilet

episode #16

## Optional parameters

Java offers type overloading. In Python keyword arguments can be passed to functions. Go doesn't allow either, yet passing optional arguments to the function can be handy from time to time. Let's see how this can be achieved with Go's type system. We will create some sort of **`Client`** without a need to pass the parameters.

```
type Client struct {
  Insecure bool
  Timeout int
}

// Set of Options like that will be applied to the Client. Since function receives
// pointer as a parameter it can change Client's state.
type Option func(*Client)

func AllowInsecure() Option { // Return an anonymous function of type Option, that
  return func(c *Client) {    // changes value of `Insecure` field on the Client.
      c.Insecure = true
  }
}

func WithTimeout(timeout int) Option { // Do the same here, but this time pass
  return func(c *Client) {             // a timeout parameter.
      c.Timeout = timeout
  }
}

func NewClient(opts... Option) *Client { // Here magic happens. The `constructor`
  c := &Client{}                         // accepts 0 or many Options (which are
  for _, opt := range opts {             // functions). Then it creates new client,
      opt(c)                             // loops over the Options and apply them
  }                                      // by invoking them.
  return c
}

func main() {
  // New Client with default parameters.
  t1 := NewClient()
  // New Client that allows insecure connections and have connection timeout 10s.
  t2 := NewClient(AllowInsecure(), WithTimeout(10))
}
```