Golang Variables

A variable can be declared at both package level and main level.

```
package main

import "fmt"

var c, python, java bool

func main() {
	var i int
	fmt.Println(i, c, python, java)
}
```

The `var` statement declares a bunch of variables as `bool`.
`var` declaration can include initializers, one per variable.
If an initializer is present, the type can be omitted; the variable will take the type of the initializer.

Short hand variable declarations

```
package main

import "fmt"

func main() {
	var i, j int = 1, 2
	k := 3
	c, python, java := true, false, "no!"

	fmt.Println(i, j, k, c, python, java)
}
```

The assignment operator `:=` changes the type of the variable implicitly when this is used.
This can only be used inside a function. Outside a function every statement begins with a keyword, so this construct cannot be used.

Basic datatypes of Go

Go's basic types are

```
bool
```

```
string
```

```
int  int8  int16  int32  int64
uint uint8 uint16 uint32 uint64 uintptr
```

```
byte // alias for uint8
```

```
rune // alias for int32
// represents a Unicode code point
```

```
float32 float64
```

```
complex64 complex128
```

Here is an example demonstrating all the types:

```go
package main

import (
	"fmt"
	"math/cmplx"
)

var (
	ToBe   bool       = false
	MaxInt uint64     = 1<<64 - 1
	z      complex128 = cmplx.Sqrt(-5 + 12i)
)

func main() {
	fmt.Printf("Type: %T Value: %v\n", ToBe, ToBe)
	fmt.Printf("Type: %T Value: %v\n", MaxInt, MaxInt)
	fmt.Printf("Type: %T Value: %v\n", z, z)
}
```

Default values for declared variables

All variables which are just declared without an initial value are assigned 0.
For numeric types its `0`.
For String types its an empty string `""`
For boolean type it is `False`.

Explicit type conversions

The expression `T(v)` converts the value `v` to the type `T`.

Some numeric conversions:

```
var i int = 42
var f float64 = float64(i)
var u uint = uint(f)
```

Or, put more simply:

```
i := 42
f := float64(i)
u := uint(f)
```

Unlike in C, in Go assignment between items of different type requires an explicit conversion.

Type inference

When declaring a variable without specifying an explicit type (either by using the `:=` syntax or `var =` expression syntax), the variable's type is inferred from the value on the right hand side.

When the right hand side of the declaration is typed, the new variable is of that same type:

```
var i int
j := i // j is an int
```

But when the right hand side contains an untyped numeric constant, the new variable may be an `int`, `float64`, or `complex128` depending on the precision of the constant:

```go
i := 42           // int
f := 3.142        // float64
g := 0.867 + 0.5i // complex128
```

Constants

Constants are declared like variables, but with the `const` keyword.

Constants can be character, string, boolean, or numeric values.

Constants cannot be declared using the `:=` syntax.

Numeric constants

Numeric constants are high-precision *values*.
An untyped constant takes the type needed by its context.
Try printing `needInt(Big)` too.
(An `int` can store at maximum a 64-bit integer, and sometimes less.)

```go
package main

import "fmt"

const (
	// Create a huge number by shifting a 1 bit left 100 places.
	// In other words, the binary number that is 1 followed by
100 zeroes.
	Big = 1 << 100
	// Shift it right again 99 places, so we end up with 1<<1, or
2.
	Small = Big >> 99
)

func needInt(x int) int { return x*10 + 1 }
func needFloat(x float64) float64 {
	return x * 0.1
}

func main() {
	fmt.Println(needInt(Small))
	fmt.Println(needFloat(Small))
	fmt.Println(needFloat(Big))
}
```