Exercise on Errors

Here is an exercise you can solve to test your understanding!

WE'LL COVER THE FOLLOWING

Assignment

Copy your `Sqrt` function from the earlier exercises and modify it to return an `error` value. (This online assignment may be useful)

`Sqrt` should return a non-nil error value when given a negative number, as it doesn't support complex numbers.

Create a new type

```
type ErrNegativeSqrt float64
```

and make it an error by giving it a

```
func (e ErrNegativeSqrt) Error() string
```

method such that `ErrNegativeSqrt(-2).Error()` returns

`cannot Sqrt negative number: -2`.

Note: a call to `fmt.Print(e)` inside the Error method will send the program into an infinite loop. You can avoid this by converting e first: `fmt.Print(float64(e))`. Why?

Change your `Sqrt` function to return an `ErrNegativeSqrt` value when given a negative number.

Given below is some starter code, add your code to implement the required function:

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import (
        "fmt"
  "strconv"
  "encoding/json"
)

type ErrNegativeSqrt float64

func (e ErrNegativeSqrt) Error() string {
        return fmt.Sprintf("cannot Sqrt negative number")
}

//Your function should return nil for error when the number can be squared
//When the number cannot be squared, the function should return 0 along with the error
func Sqrt(x float64) (float64, error) {
        //write code here

  return -1, nil //edit to return the correct answer
}
```

Tip: When doing an inferred declaration of a float, you can omit the decimal value and do the following:

```
z := 1.
// same as
// z := 1.0
```