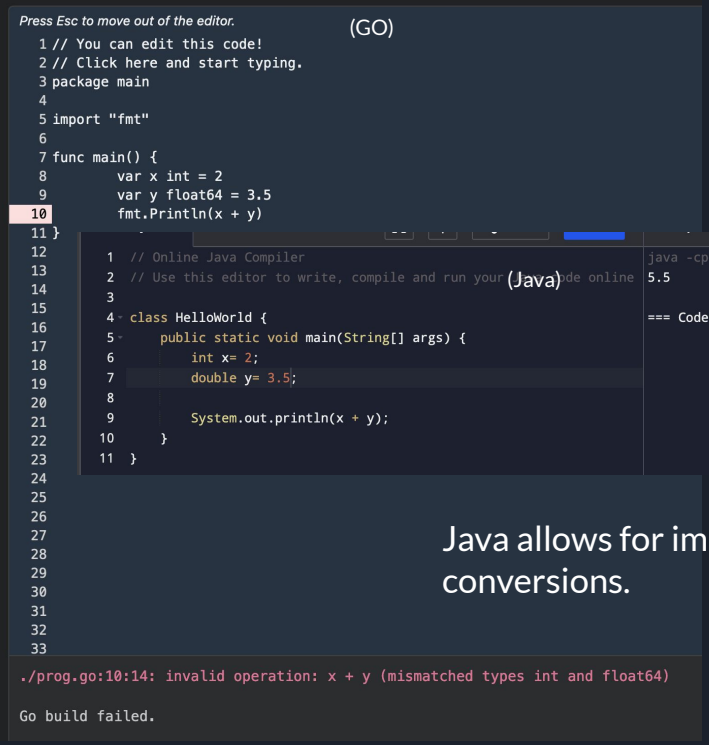
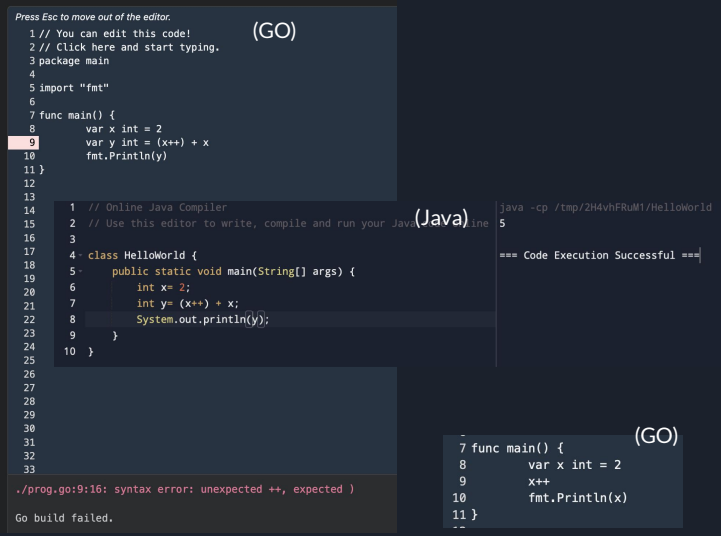
Go Lang Special Features and Control Statements

By: Caleb, Darrel, Jonah, Andy

Golang differences in arithmetic expression



Java allows for implicit type conversions.



Go requires incrementation to be its own statement. Java can integrate that into arithmetic.

Short circuit? Strict circuit?

Golang supports short circuit.

In the example to the left, Golang's compiler did not evaluate what the value of testFunction() may be. It saw that a was false and gave its response.

Press Esc to move out of the editor.

```
// You can edit this code!
// Click here and start typing.
package main

import "fmt"

func main() {
        a := false

        // Short-circuit evaluation: the second condition is not evaluated
        if a && testFunction() {
                fmt.Println("Both are true")
        } else {
                fmt.Println("Short-circuited")
        }
}

func testFunction() bool {
        fmt.Println("testFunction is called!")
        return true
}
```

```
Short-circuited

Program exited.
```

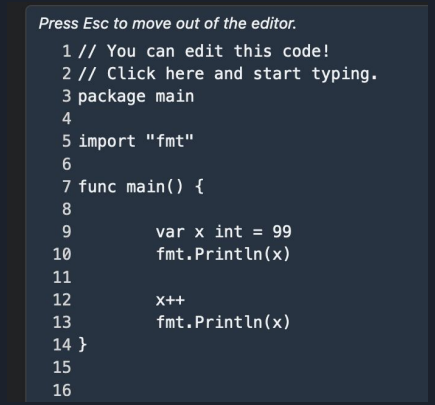
Press Esc to move out of the editor.

```
// You can edit this code!
// Click here and start typing.
package main

import "fmt"

func main() {
        a := true

        // Short-circuit evaluation: the second condition is not evaluated
        if a && testFunction() {
                fmt.Println("Both are true")
        } else {
                fmt.Println("Short-circuited")
        }
}

func testFunction() bool {
        fmt.Println("testFunction is called!")
        return true
}
```

```
testFunction is called!
Both are true
```

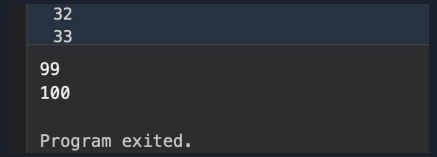
Single operand expressions in Golang?

Yes. Golang allows for there to be single operand expressions as long **as it is on its own line.**



Press Esc to move out of the editor.

```
1 // You can edit this code!
2 // Click here and start typing.
3 package main
4
5 import "fmt"
6
7 func main() {
8
9         var x int = 99
10        fmt.Println(x)
11
12        x++
13        fmt.Println(x)
14 }
15
16
```



```
32
33

99
100

Program exited.
```

Multiple assignment in Golang?

Yes, golang does support multiple assignment and it is shown. The result swaps the variables.

```
Press Esc to move out of the editor.
 1 // You can edit this code!
 2 // Click here and start typing.
 3 package main
 4
 5 import "fmt"
 6
 7 func main() {
 8
 9         var x int = 24
10         var y int = 25
11
12         fmt.Println(x, y)
13         x, y = y, x
14         fmt.Println(x, y)
15 }
16
17
```

```
31
32
33

24 25
25 24

Program exited.
```

Dangling else in GO

- The dangling else is resolved in GO by using { } to enclose all if/else blocks even if they only contain one line
- Else blocks will always be matched with the closest preceding if block

```
if condition {

        // do something

} else {

        // do something else

}
```

Conditional expressions

- Go does not support traditional ternary type conditional expressions.
- It enforces ambiguity and clarity over conciseness using if-else statements only

```
if x > 10 {
        y = 20
} else {
        y = 30
}
```

Loops

For Loop in Go

```
for i := 0; i < 10; i++ {
    // code block
}
```

The update of the count variable can affect the amount of iterations.

For instance, if we subtract 1 from “i” in the for loop code block, we have to iterate one extra time.

Go evaluates the condition i <10 on every condition

Loops

Go Lang does not support while loop

Go Lang supports break and continue statements

```go
package main

import "fmt"

func main() {
    for i := 0; i < 10; i++ {
        if i == 5 {
            fmt.Println("Breaking the loop at i =", i)
            break // Exit the loop when i equals 5
        }
        fmt.Println(i)
    }
}
```

```go
package main

import "fmt"

func main() {
    for i := 0; i < 10; i++ {
        if i%2 == 0 {
            continue // Skip even numbers
        }
        fmt.Println(i)
    }
}
```

Switch Statements

- Yes, GO has switch statements
- The main difference between java is that Go only runs the selected case, not all the cases that follow.
- More flexible
    - Cases do not need to be constants

```go
package main

import "fmt"

func main() {
    day := 3

    switch day {
    case 1:
        fmt.Println("Sunday")
    case 2:
        fmt.Println("Monday")
    case 3:
        fmt.Println("Tuesday")
    case 4:
        fmt.Println("Wednesday")
    case 5:
        fmt.Println("Thursday")
    case 6:
        fmt.Println("Friday")
    case 7:
        fmt.Println("Saturday")
    default:
        fmt.Println("Invalid day")
    }
}
```

Switch Statements

- Go does not have type matching
- Pattern matching allows you to test data structures in a concise and readable way, checking for the structure, type, or value of data
  - Can match on values or data structures
- Benefits of pattern matching is code is more
  - Concise
  - Readable
  - Safe