**What are Packages?**

Packages are the medium to collect and then export items.

Packages in Go refers to the collection of Go files residing under the same directory serving a specific purpose. Packages can be considered as the single point of reference for a function, variable or feature.

For example, we need to we have so many functions for doing just for playing with numbers, (addition, squaring, multiplication, cube, division, remainder, finding prime, LCM, HCF and so on).

So, what we can do is, create a package for all these maths operations and place them as a collection.

Chart

Description automatically generated

Now anywhere in code, if we need to operate on number, we can use this package (math) and call the respective functionality and serve the purpose, thus creating a package help to achieve DRY (Don’t Repeat Yourself) principle and further results in code reusability.

Hence, we can say Packages are the medium to collect the specific variables, functionalities, or feature in form of Go files and making it available to be used anywhere inside the project.

**Packages in Go**

There are two types of packages in Go: **Executable** and **Utility.**

1. **Executable Package:** The “main” package is the only executable package in Go. This is the package which will be running. This package contains the main function which depict the start of the program. An executable is created in $GOBIN directory on installation of “main” package.
2. **Utility Package:** These are not self-executable package. Any package other than main package is utility package. This package contains the functionality which either enhances, support or adds on the working of “main” package.

**Ways to import package in go**

Usually name of package preceded by the import keyword is used to the import a package.

import “package\_name”

1. **Importing a single** **package**: It can be done by mentioning the import keyword followed by the package name.

Suppose if we need to make use of I/O functionality to print data in the terminal, we use “fmt” package.

import “fmt”

1. **Importing multiple package**: Suppose if we need to make use of string operation and maths operation along with the printing functionality, we will enclose all the package name in the parenthesis after import keyword.

**import**  (

“fmt”

“string”

“math”

)

1. **Alias Import:** Importing a package with user defined name comes under this category.

**Import m “math”**

1. **Nested Import:** Consider a scenario where user want to operate on complex numbers and user is aware that “cmplx” sub-package of math package can be handy in it, so user import math as the package.

Now here it is important to understand that “math” package have other package too inside it (big, bits, rand) which are of no use to user, but as user has imported “math”, they also get imported with it. So, in such case, nested import comes to picture.

In nested import, we specify the specific path of the sub package needed:

import “math/cmplx”

Now in this case only “cmplx” will only be imported, not the other sub-packages of the math.

1. **Dot Import:** We can use a dot before importing the entire package into the same namespace. So, we don’t have to call the functions using the package name. We can directly call them. This has some drawbacks associated with it since it mixes up the packaging and can cause namespace collisions.

package main

import (

    . "fmt"

)

func main() {

    Println("Check this") // calling function without package name

}

1. Blank Import: Try running the following code:

package main

import (

    "fmt"

    "math"

)

func main() {

    fmt.Println("Check this")

}

Does this code gets executed? Answer is no. But why?

Golang doesn’t allow unused package. So, if we have imported the package but not used it, code will not get executed. So, in such cases we use blank identifier or underscore before the package name.

package main

import (

    "fmt"

    \_ "math"

)

func main() {

    fmt.Println("Check this")

}

So, in such case even if the package is unused, code will still get executed.

**String Package in go lang**

String package in go lang is collection of different type of function using which one can easily manipulate UTF-8 encoded string.

How to use String package?

To use string package, one need to import it, so it can be done as follow

Import “string”

Let see string package in action:

package main

import (

    "fmt"

    "strings"

)

func main() {

    fmt.Println(strings.Compare("hello", "hey"))

    fmt.Println(strings.Compare("hello", "hello"))

    fmt.Println(strings.Compare("hello", "hallo"))

}

Output:

Graphical user interface, application

Description automatically generated

So, if you see here, we have used inbuilt sub package of strings package, i.e., Compare.

Compare takes two inputs and have lexicographical comparison in both and return a integer as output. Following is the possibility of the output using Compare:

1. If string1 == string2, output will be 0
2. If string1> string2, output will be 1
3. If string1<string2, output will be -1

Commonly used string function in Golang:

Most used case of String functions are:

1. To compare
2. To find substring
3. To find index of substring
4. To join to string
5. To replace
6. To split and Trim
7. Casing

As we have already seen Compare in action, let’s look at other use cases:

**To find substring:**

**Contains (string, substr):** We have a function which take two inputs, string and substring, hence return the Boolean value if the substring is part of string or not.

package main

import (

    "fmt"

    "strings"

)

func main() {

    fmt.Println(strings.Contains("batman", "man")) //man in batman

    fmt.Println(strings.Contains("batman", "bar")) //no bar in batman

    fmt.Println(strings.Contains("batman", ""))    // checking for no empty space in batman

    fmt.Println(strings.Contains("batman", " "))   // checking for empty space in batman

    fmt.Println(strings.Contains("", ""))          // comparing two no empty space

    fmt.Println(strings.Contains("", " "))         //comparing no empty space with empty space

}

Output:

Application

Description automatically generated with low confidence

Similar to Contain, we have two more variation, [ContainsAny](https://pkg.go.dev/strings#ContainsAny) and [ContainsRune](https://pkg.go.dev/strings#ContainsRune)

**To find index of substring**:

Index (string, substring): Index returns the index of the first instance of substring in string, else it returns -1. We also have LastIndex which returns the index of last instance of substring in string else -1.

package main

import (

    "fmt"

    "strings"

)

func main() {

    fmt.Println(strings.Index("batman", "ma"))

    fmt.Println(strings.Index("batman", "a"))

fmt.Println(strings.LastIndex("batman", "a"))

    fmt.Println(strings.Index("batman", "new"))

}

Output:

Graphical user interface, application, Teams

Description automatically generated

Similar to index we also have [IndexAny](https://pkg.go.dev/strings#IndexAny), [IndexByte](https://pkg.go.dev/strings#IndexByte), [IndexFunc](https://pkg.go.dev/strings#IndexFunc) , [IndexRune](https://pkg.go.dev/strings#IndexRune) , [LastIndexAny](https://pkg.go.dev/strings#LastIndexAny), [LastIndexByte](https://pkg.go.dev/strings#LastIndexByte), [LastIndexFunc](https://pkg.go.dev/strings#LastIndexFunc).

**To join to string**

Join (elems, separator): Join take two inputs: elements and separator. It concatenates the elements of its first argument to create a single string. The separator string separator is placed between elements in the resulting string

package main

import (

    "fmt"

    "strings"

)

func main() {

    legends := []string{"goku", "vegeta", "gohan"}

    fmt.Println(strings.Join(legends, "\_"))

}

Output:



**To replace:**

Replace (string, old\_substr, new\_substr, count) : Replace takes four parameter: original string, substring to be replaced, new substring to replace by and how many occurrence to replace. Replace, marks the changes, and return a new copy of string.

package main

import (

    "fmt"

    "strings"

)

func main() {

    fmt.Println(strings.Replace("Bruce wayne is Batman", "a", "B", 1))

    fmt.Println(strings.Replace("Bane, Bruce is Batman", "B", "A", 2))

}

Output:

Graphical user interface, text

Description automatically generated

We also have [ReplaceAll](https://pkg.go.dev/strings#ReplaceAll) in strings.

**To Split and Trim:**

**Split (string, separator):** Split slices string into substrings separated by separator and returns a slice of the substrings between those separators.

If string does not contain separator and separator is not empty value, Split returns a slice of length 1 whose only element is string itself.

If separator is empty, Split splits after each character. If both string and separator are empty, Split returns an empty slice.

package main

import (

    "fmt"

    "strings"

)

func main() {

    fmt.Printf("%q\n", strings.Split("Goku\_Gohan\_Vegeta", "\_"))

    fmt.Printf("%q\n", strings.Split("Bruce wayne is Batman", "a"))

    fmt.Printf("%q\n", strings.Split("Batman", ""))

    fmt.Printf("%q\n", strings.Split("", "a"))

}

**Output:**

A picture containing text, black, scoreboard

Description automatically generated

Strings also have function like: [SplitAfter](https://pkg.go.dev/strings#SplitAfter), [SplitAfterN](https://pkg.go.dev/strings#SplitAfterN), [SplitN](https://pkg.go.dev/strings#SplitN)

Trim (string, cutset): Trim returns a slice of the string with all leading and trailing Unicode code points contained in cutset removed.

package main

import (

    "fmt"

    "strings"

)

func main() {

    fmt.Println(strings.Trim("12Hello, World12", "12"))

    fmt.Println(strings.Trim("12Hello12, 12World12", "12"))

}

Output:



Strings also have function : [TrimFunc](https://pkg.go.dev/strings#TrimFunc), [TrimLeft](https://pkg.go.dev/strings#TrimLeft), [TrimLeftFunc](https://pkg.go.dev/strings#TrimLeftFunc), [TrimPrefix](https://pkg.go.dev/strings#TrimPrefix), [TrimRight](https://pkg.go.dev/strings#TrimRight), [TrimRightFunc](https://pkg.go.dev/strings#TrimRightFunc), [TrimSpace](https://pkg.go.dev/strings#TrimSpace), [TrimSuffix](https://pkg.go.dev/strings#TrimSuffix).

**Casing:**

**ToLower (string) :** This function return the string into the lower characters.

package main

import (

    "fmt"

    "strings"

)

func main() {

    a := "BatMan is Bruce"

    fmt.Println(strings.ToLower(a))

    fmt.Println(strings.ToUpper(a))

    fmt.Println(strings.Title(a))

    fmt.Println(strings.ToTitle(a))

}

Output:

Text, application

Description automatically generated

Strings also offer function as: [ToLowerSpecial](https://pkg.go.dev/strings#ToLowerSpecial), [ToUpperSpecial](https://pkg.go.dev/strings#ToUpperSpecial), [ToTitleSpecial](https://pkg.go.dev/strings#ToTitleSpecial)

Count (string, substring) : It return the count of number of instance substring has occurred in the string.

package main

import (

    "fmt"

    "strings"

)

func main() {

    fmt.Println(strings.Count("Make me burger", "e"))

}

Output:



To further read on string, one can visit: [Strings](https://pkg.go.dev/strings#section-documentation).