A Tour of Go Basics

1-1. Packages

Goのプログラムは、**パッケージ(package)で構成される。

- package 使用する言語で利用できる「オブジェクト」や「関数」の宣言などを、関連するものごとにまとめたものを**パッケージ**と呼ぶ。
- Goの場合は？
    - Goでプログラムを作成する場合、必ず**main**パッケージが存在する必要がある
    - また、**main**パッケージ中に**main**関数が定義される必要がある

簡単に言うと、Goのコードはパッケージの宣言から始まる！

```
# パッケージの宣言.
package main

# 使用したいパッケージをインポートする.
import (
    "fmt"
    "math/rand"
)

func main() {
    fmt.Println("My favorite number is", rand.Intn(10))
}
```

- fmt
    - 文字列の入出力と、フォーマット(書式設定)に関する機能を提供するパッケージ
    - `print`関数などを使うためには`fmt`のインポートが必要
- math/rand
    - 乱数生成に関するパッケージ
    - `math/rand`は弱い乱数、`crypto/rand`は強い乱数
    - 強弱は乱数精度(暗号化などに適しているかなど)

1-2. Imports

インポート自体はPythonでもあったので違和感はない。

- 複数インポートする場合は括弧でまとめる。

```
import (
  "fmt"
```

```
  "math"
)
```

- 複数のインポートステートメントで書く事も出来る

```
import "fmt"
import "math"
```

Goは先に示した括弧でまとめたスタイルの方が良い！

1-3. Exported names

- 最初の文字が大文字で始まる名前
    - 外部パッケージから参照できる名前(exported name)
- 小文字で始まる名前
    - エクスポートされていない名前
- 手順

1. パッケージのインポート
2. パッケージがエクスポートしている名前を参照可能になる

`math`パッケージの円周率に関するモジュールを使用したい場合はexported nameを指定する！

```
func main() {
    fmt.Println(math.Pi)
  # math.piはエラーになる
}
```

小文字から始まる方がいまいち理解出来てないが、一旦飛ばす。

1-4. Functions

関数は、0個以上の引数を取ることができる。

```
package main

import "fmt"

# ad関数の引数(x, y)がintであることを宣言
# () の後のintは戻り値の型を定義している
func add(x int, y int) int {
    return x + y
}
```

```
# 引数に指定した数の計算結果をintで返す
func main() {
    fmt.Println(add(42, 13))
}
```

上記のように、変数名の**後ろ**に型名を書く必要がある。

Goは、静的型付け言語である！！ 動的型付け言語しかやってこなかったので整理する。

- (架空説明的な言語で)C以外の言語の場合の型構文

```
x: int
p: pointer to int
a: array[3] of int
```

- Goの場合

```
x int
p *int
a [3]int
```

違いは、簡潔にするためにコロンを削除していることやいくつかのキーワードを削除している。 **Goの型宣言は、左から右へと読む！** (Cがらせん状に読み取られていることが指摘されている。)

1-5. Functions continued

2つ以上の引数が同じ型である場合には、最後の型を残して省略して記述できる

`func add(x int, y int)` -> `func add(x, y int)`

1-6. Multiple results

```
func swap(x, y string) (string, string) {
    return y, x
}
```

戻り値の第一引数と第二引数を交換するには、戻り値を交換するだけで簡単にできる `return y, x`

1-7. Named return values

戻り値となる変数に名前をつけることができる

```
func split(sum int) (x, y int) {
    x = sum * 4 / 9
```

```
    y = sum - x
    return
}
```

`split(sum int)`で関数を定義`(x, y int)`で`x`, `y`という名前を戻り値に付けている。

- 変数に名前を付けなかった場合

```
func split(sum int) (int, int) {
    return sum * 4 / 9, sum - (sum * 4 / 9)
}
```

実際に書いてみて分かったが、変数名に`x`, `y`と付ける事で、`y`の戻り値の計算が楽に書けていたことがわかった。(`y = sum - x`) 変数名を付ける事で可読性も上がるため、実務には必須な技術と思われる。

1-8. Variables

`var ~`で変数を宣言することができる

`var c, python, java bool` このように複数の変数名+最後に型を宣言する 結果、`c, python, java`の3つが全てbool型となる。 int型の場合は`var a, b int`とする。

1-9. Variables with initializers

initializers -> 初期化子 変数名に初期値を設定できるという解釈で良さそう。

`var i, j int = 1, 2` -> `var i, j = 1, 2` このように初期値を数値で指定することで、型宣言(int)を省略する事ができる。

1-10. Short variable declarations

短い変数宣言 **関数**の中では、`var`宣言の代わりに、`:=`の代入文を使い、暗黙的な型宣言ができる

なお、関数の外では型宣言が必要で、`:=`での暗黙的な宣言は利用できない！

`k := 3` `var k int = 3`(intは省略可能) 両方同じ型宣言をしている。

1-11. Basic types

- 重要だと思われる部分！
  - 整数の変数が必要な場合は基本的に`int`を使う
  - 変数宣言はまとめて宣言可能 参考: Go言語　プリミティブ型と型宣言

```
# まとめて宣言する場合
var (
  ToBe bool = false
  MaxInt unit64 = 1<<64 - 1
```

```
  z complex128 = cmplx.Sprt(-5 + 12i)
)
```

1-11-1. `int`, `uint`の違い

- int 32bitまたは64bitの符号**付き**整数 32bit -> -2147483648 〜 2147483647 64bit -> -9223372036854775808 〜 9223372036854775807
- uint 32bitまたは64bitの符号**無し**整数 32bit -> 0 〜 4294967295 64bit -> 0 〜 18446744073709551615
- 符号
    - 符号付き
        - 正の整数, 負の整数, 0 を表現できる
    - 符号無し
        - 正の整数, 0 だけを表現できる

1-11-2. その他の組み込み型

- byte unit8のエイリアス(0~255)
- rune int32のエイリアス(-2147483648 〜 2147483647)
- float 浮動小数点
- complex 実数部・虚数部をfloatで表現する複素数 難しそうなので一旦後回し。参考サイトだけ貼り付け。 虚数とは何か？複素数とは何か？が一気に分かりやすくなる記事

1-12. Zero values

変数に初期値を与えずに宣言すると、ゼロ値が与えられる。

```
var (
  i int
  f float64
  b bool
  s string
)
```

int, float -> 0 bool -> false string -> "" (空文字列)

1-13. Type conversions

型変換が可能

```
# iをintで宣言
var i int = 42
```

```
# iの値をfloatへ変換
var f float64 = float64(i)

# fの値をuintへ変換
var u unit = uint(f)
```

- よりシンプルに記述可能 var i int = 42 -> `i := 42` var f float64 = float64(i) -> `f := float64(i)` var u unit = uint(f) -> `u := unit(f)`

1-14. Type inference

どんな型か？は**右側の値(変数、または代入する値)**から推論できる。

`i := 42` -> 整数を代入しているのでint型 `i := 3.142` -> 小数点を代入しているのでfloat型 `i := 0.867 + 0.5i` -> complex型

1-15. Constants

定数は、`const`というキーワードで宣言する。

使える型は以下の4つのみ

- 文字(character)
- 文字列(string)
- 真偽値(boolean)
- 数値(numeric)

なお、定数は`:=`を使って宣言できない！

1-16. Numeric Constants

数値の定数は、高精度な値である。 = 型のない定数は、その状況によって必要な型を取ってくれる。

まだ理解が必要。