

Go 언어 기본 문법: 변수, 상수, 타입, 제어문, 함수

모든 실습은 VSCode에서 진행하며, Go-Learning 워크스페이스 내 02-기본문법 폴더에 예제 파일을 생성하는 것을 기준으로 설명함.

Java 개발자를 위한 Go 문법 핵심 차이점

항목	Go	Java
패키지 선언	package main	package com.example;
임포트	import "fmt"	import java.util.*;
접근제어자	없음(대소문자: 대문자=공개, 소문자=비공개)	public/private/protected
예외 처리	error 반환, try-catch 없음	try-catch-finally
함수/메서드	함수(클래스 필요 없음), 메서드(리시버)	클래스 내 메서드
실행 진입점	func main()	public static void main(String[] args)

Go는 클래스가 없고, 패키지 구조와 접근제어자가 다르다. 특히 package main과 func main()은 실행 가능한 프로그램의 진입점이다. 예외 처리도 error 타입을 반환하는 방식으로 동작한다.

1. 변수 (Variables)

Java에서 변수를 선언할 때는 타입 이름 = 값; 형식을 따름. Go는 타입 추론과 짧은 선언(:=) 등 다양한 방식을 제공한다.

구분	Go	Java
기본 선언	var a int = 10	int a = 10;
타입 추론	var a = 10	(Java 10+) var a = 10;
짧은 선언	a := 10 (함수 내에서만)	해당 없음

Go의 := (짧은 선언)은 var와 타입 지정을 생략할 수 있어 함수 내부에서 변수를 선언하고 초기화할 때 매우 유용하게 사용됨. 단, 이 방식은 함수 내에서만 사용 가능하며, 이미 선언된 변수에는 사용할 수 없음.

실습 1: 캐릭터 이름과 레벨 변수 선언하기

간단한 게임 캐릭터의 이름과 레벨을 변수로 선언하고 출력하기

실습전 주의사항

Go 언어는 같은 폴더에 두 개의 main 패키지가 존재할 경우 컴파일 에러가 발생하기 때문에 main 패키지를 각각 독립적인 폴더에 넣어야 한다.

그러므로 다음과 같은 실습 구조를 갖도록 디렉토리를 구성해야 한다.

```
02-기본문법          <---- 단원명
|
|--01-variables       <---- 예제명
|   main.go          <---- 소스파일 (파일이름은 상관 없음)
```

```
└─02-constants
    main.go
```

실습 예제: **/02-기본문법/01-variables/main.go**

```
package main

import "fmt"

func main() {
    // 1.1 `var` 키워드를 사용한 변수 선언 및 초기화
    // Java: `String characterName = "Hossam";`와 유사. Go는 타입이 뒤에 온다.
    var characterName string = "Hossam"
    // Java: `int characterLevel = 99;`와 유사.
    var characterLevel int = 99

    // 1.2 `var` 키워드와 타입 추론을 사용한 변수 선언
    // 초기화 값으로 타입을 추론한다. Java 10+의 `var exp = 85.5;`와 유사.
    var exp = 85.5

    // 1.3 짧은 선언(`:=`)을 사용한 변수 선언 및 초기화
    // 함수 내부에서만 사용 가능하며, `var` 키워드와 타입 지정을 생략한다.
    // Go에서 가장 흔하게 사용되는 변수 선언 방식이다.
    isAlive := true

    // 1.4 변수 값 출력
    // `fmt.Println` 함수는 인자들을 공백으로 구분하여 출력하고 줄바꿈한다.
    fmt.Println("캐릭터 이름:", characterName)
    fmt.Println("레벨:", characterLevel)
    fmt.Println("경험치:", exp)
    fmt.Println("생존 여부:", isAlive)
}
```

실행 방법:

- VSCode에서 Go 확장 설치 후, 파일 상단 ▶ 버튼 클릭 또는 터미널에서 **go run 01-variables.go** 실행
- 여러 파일 관리 시, **go mod init**으로 모듈 환경 구성하면 import가 편리함

2. 상수 (Constants)

상수는 변하지 않는 값. Go에서는 **const** 키워드를 사용하며, Java의 **final**과 유사하다.
Go 상수는 타입 없이 선언할 수도 있고, 여러 개를 그룹으로 선언할 때 **iota**를 활용한다.

구분	Go	Java
상수 선언	const PI = 3.14	final double PI = 3.14;
그룹 선언(iota)	const (...)	해당 없음

실습 2: 게임 설정값 상수로 관리하기

게임의 최대 레벨이나 서버 IP와 같이 변하지 않는 중요한 값들을 상수로 선언하기

실습 예제: **/02-기본문법/02-constants/main.go**

```
package main

import "fmt"

// 2.1 단일 상수 선언
// `const` 키워드를 사용하여 상수를 선언한다. Java의 `final` 키워드와 유사하다.
const MAX_LEVEL = 100
const SERVER_IP = "127.0.0.1"

// 2.2 `iota`를 사용한 그룹 상수 선언
// `iota`는 0부터 시작하여 `const` 블록 내에서 선언된 상수마다 1씩 자동으로 증가하는 특별한 상수 생성기이다.
// Java의 `enum`과 유사하게 열거형 상수를 정의할 때 유용하다.
const (
    STATE_IDLE = iota // 첫 번째 `iota`는 0. STATE_IDLE = 0
    STATE_WALK        // `iota`가 1 증가하여 STATE_WALK = 1
    STATE_RUN         // `iota`가 1 증가하여 STATE_RUN = 2
    STATE_JUMP        // `iota`가 1 증가하여 STATE_JUMP = 3
)

func main() {
    // 2.3 상수 값 출력
    fmt.Println("최대 레벨:", MAX_LEVEL)
    fmt.Println("서버 IP:", SERVER_IP)
    fmt.Println("캐릭터 상태 - 대기:", STATE_IDLE)
    fmt.Println("캐릭터 상태 - 점프:", STATE_JUMP)
}
```

`iota`는 열거형(enum)이 없는 Go에서 상태나 순서를 나타내는 상수를 정의할 때 매우 유용함.

3. 데이터 타입 (Data Types)

Go는 다양한 내장 데이터 타입을 제공한다. Java와 유사하지만, 타입 변환이 엄격하고 부호 없는 타입이 별도로 존재한다.

Go 타입	설명	Java 대응 타입
<code>bool</code>	논리값 (true/false)	<code>boolean</code>
<code>string</code>	문자열	<code>String</code>
<code>int, int8, int16, int32, int64</code>	부호 있는 정수	<code>int, byte, short, int, long</code>
<code>uint, uint8, uint16, uint32, uint64</code>	부호 없는 정수	해당 없음
<code>float32, float64</code>	부동소수점 숫자	<code>float, double</code>
<code>rune</code>	유니코드 문자를 나타내는 타입 (실제로는 <code>int32</code> 의 별칭)	<code>char</code>
<code>byte</code>	<code>uint8</code> 의 별칭, 바이트 데이터를 다룰 때 사용	<code>byte</code>

Go는 `int16`과 그에 대응하는 `uint16`도 지원함.

정수 타입 상세 설명:

- `int`와 `uint`: 가장 일반적으로 사용되는 정수 타입. 크기가 정해져 있지 않고, 프로그램을 컴파일하는 시스템의 아키텍처에 따라 크기가 결정됨. 32비트 시스템에서는 32비트(4바이트), 64비트 시스템에서는 64비트(8바이트) 크기를 가짐. 특별히 크기를 지정할 필요가 없다면 `int`를 사용하는 것이 일반적임.
- `int8`, `int16`, `int32`, `int64`: 각각 8, 16, 32, 64비트 크기의 부호 있는 정수 타입. 숫자는 비트 수를 의미하며, 비트 수가 클수록 더 큰 범위의 숫자를 표현할 수 있음.
 - `int8`: -128 ~ 127
 - `int16`: -32,768 ~ 32,767
 - `int32`: -2,147,483,648 ~ 2,147,483,647
 - `int64`: 매우 큰 정수
- `uint8`, `uint16`, `uint32`, `uint64`: 부호 없는 정수 타입. 0과 양수만 표현하며, 같은 크기의 부호 있는 정수 타입보다 두 배 큰 양수 범위를 가짐.

중요: Go는 타입에 매우 엄격하다. Java에서는 `int`와 `long` 간의 암시적 형 변환이 가능하지만, Go에서는 반드시 명시적으로 형 변환해야 한다.

실습 3: 다양한 타입의 변수 선언 및 형 변환

캐릭터의 여러 정보를 다양한 데이터 타입으로 선언하고, 필요한 경우 형 변환하기

실습 예제: `/02-기본문법/03-data-types/main.go`

```
package main

import "fmt"

func main() {
    // 3.1 정수 타입
    // `int`는 시스템 아키텍처에 따라 크기가 달라진다 (32비트 또는 64비트).
    var level int = 10
    // `int64`는 64비트 크기의 정수 타입이다. Java의 `long`과 유사하다.
    var monsterCount int64 = 100

    // Go는 타입에 매우 엄격하여 다른 타입 간의 직접적인 연산을 허용하지 않는다.
    // Java에서는 `int`와 `long` 간의 암시적 형 변환이 가능하지만, Go에서는 명시적으로 변환해야 한다.
    // fmt.Println(level + monsterCount) // 컴파일 에러 발생: `mismatched types int and int64`

    // 명시적 형 변환: `int64(level)`은 `level` 변수를 `int64` 타입으로 변환한다.
    fmt.Println("총합:", int64(level) + monsterCount)

    // 3.2 실수 타입
    // `float32`는 32비트 부동소수점 타입이다. Java의 `float`과 유사하다.
    var attackPower float32 = 35.5
    // `float64`는 64비트 부동소수점 타입이다. Java의 `double`과 유사하다.
    var defensePower float64 = 52.8

    // 다른 실수 타입 간에도 명시적 형 변환이 필수이다.
    fmt.Println("전투력:", float64(attackPower) * defensePower)

    // 3.3 문자열과 rune
    // `string` 타입은 UTF-8 인코딩된 문자열을 나타낸다. Java의 `String`과 유사하다.
    var message string = "Hello, World!"
    // `rune` 타입은 유니코드 코드 포인트(Unicode code point)를 나타내며, 실제로는 `int32`의 별칭이다.
    // 작은따옴표(``)를 사용하여 단일 문자를 표현한다. Java의 `char`와 유사하다.
    var firstChar rune = 'H'
}
```

```
fmt.Println(message)
// `rune`을 `string`으로 변환하여 출력한다. `string(firstChar)`는 `char`를 `String`으로
// 변환하는 것과 유사하다.
fmt.Println("첫 글자:", string(firstChar))
}
```

4. 제로 값 (Zero Values)

Go에서 변수를 선언하고 초기화하지 않으면, 각 타입의 "제로 값"으로 자동 초기화된다. Java에서 지역 변수는 명시적으로 초기화해야 하는 것과 대조적이다.

타입	제로 값
숫자 (int, float 등)	0
bool	false
string	"" (빈 문자열)
포인터, 함수, 인터페이스, 슬라이스, 맵, 채널	nil

실습 4: 제로 값 확인하기

실습 예제: [/02-기본문법/04-zero-values/main.go](#)

```
package main

import "fmt"

func main() {
    // 4.1 다양한 타입의 변수 선언 (초기화하지 않음)
    // Go는 변수를 선언하고 초기화하지 않으면 자동으로 해당 타입의 "제로 값"으로 초기화한다.
    // Java에서는 지역 변수를 초기화하지 않으면 컴파일 에러가 발생한다.
    var a int          // int 타입의 제로 값은 0
    var b float64      // float64 타입의 제로 값은 0.0
    var c bool         // bool 타입의 제로 값은 false
    var d string       // string 타입의 제로 값은 "" (빈 문자열)
    var e *int         // 포인터 타입의 제로 값은 nil (Java의 null과 유사)

    // 4.2 제로 값 출력
    fmt.Println("int의 제로 값:", a)
    fmt.Println("float64의 제로 값:", b)
    fmt.Println("bool의 제로 값:", c)
    fmt.Println("string의 제로 값:", d)
    fmt.Println("포인터의 제로 값:", e)
}
```

5. if문

Go의 제어문은 Java와 매우 유사하지만, 소괄호 ()를 생략하는 등 더 간결하다. 중괄호는 필수.

Go의 if문은 조건식에 소괄호를 사용하지 않음.

또한, if문 내에서만 사용할 수 있는 짧은 변수 선언을 지원하여 코드를 깔끔하게 유지할 수 있음.

구문 형식

if 문

```
if 조건 {
}
```

if ~ else 문

```
if 조건 {
} else {
}
```

if ~ else if ~ else 문

```
if 조건1 {
} else if 조건2 {
} else if 조건3 {
} else {
}
```

실습 5: 캐릭터 레벨에 따른 상태 출력

실습 예제: /02-기본문법/05-if-else/main.go

```
package main

import "fmt"

func main() {
    characterLevel := 99 // 캐릭터 레벨 변수 선언 및 초기화

    // 5.1 기본 if-else 문
    // Go의 `if` 조건문에는 소괄호 `()`를 사용하지 않는다. 중괄호 `{}`는 필수이다.
    // Java의 `if (characterLevel >= 100)`와 유사하다.
    if characterLevel >= 100 {
        fmt.Println("만렙 달성!")
    } else { // `else`는 `if`의 닫는 중괄호 `}`와 같은 줄에 와야 한다.
        fmt.Println("아직 만렙이 아님.")
    }

    // 5.2 `if` 문 내에서 변수 선언 및 사용 (if-short statement)
    // `if` 문 내에서만 유효한 변수를 선언하고 초기화할 수 있다. `exp` 변수는 `if` 블록 내에서만 사용
```

가능하다.

```
// 이는 Java에는 없는 Go의 특징적인 문법이다.
if exp := 85.5; exp > 90.0 {
    fmt.Println("경험치 충분, 레벨 업 가능!")
} else {
    fmt.Println("경험치가 부족함.")
}
}
```

6 for

Go에는 `for`문만 존재하며, Java의 `for`, `while`, `for-each`를 모두 대체함.

사용 방식	Go	Java
기본 for문	<code>for i := 0; i < 5; i++</code>	<code>for (int i = 0; i < 5; i++)</code>
while처럼	<code>for 조건 {}</code>	<code>while (조건) {}</code>
무한 루프	<code>for {}</code>	<code>for (;;) {}</code> 또는 <code>while (true) {}</code>

실습 6: 아이템 목록 출력하기

실습 예제: `/02-기본문법/06-for-loop/main.go`

```
package main

import "fmt"

func main() {
    // 6.1 기본 for문 (Java의 `for (int i = 0; i < 3; i++)`와 동일)
    fmt.Println("---- 기본 for문 ----")
    for i := 0; i < 3; i++ { // 초기화; 조건; 증감문
        fmt.Println(i)
    }

    // 6.2 `while`문처럼 사용 (Java의 `while (count < 3)`와 동일)
    // Go에는 별도의 `while` 키워드가 없으며, `for` 조건문만 사용하여 `while` 루프를 구현한다.
    fmt.Println("---- while처럼 사용 ----")
    count := 0
    for count < 3 { // 조건문만 사용
        fmt.Println(count)
        count++
    }

    // 6.3 무한 루프 (Java의 `while (true)` 또는 `for (;;)`)와 동일
    // `for {}` 형태로 사용한다.
    // for {
    //     fmt.Println("무한 루프!")
    //     time.Sleep(time.Second) // 1초 대기
    // }
}
```

7. switch

Go의 `switch` 문은 Java보다 더 유연하다.

`break`를 명시하지 않아도 각 `case`가 끝나면 자동으로 멈춘다. 여러 조건을 한 번에 처리하거나, 조건식 자체를 쓸 수 있다.

실습 7: 캐릭터 상태에 따른 메시지 출력

실습 예제: `/02-기본문법/07-switch/main.go`

```
package main

import "fmt"

func main() {
    state := "idle" // 캐릭터 상태 변수 선언

    // 7.1 기본 switch 문
    // Go의 `switch` 문은 Java의 `switch`와 유사하지만, `break`를 명시하지 않아도 각 `case`가 끝
    // 나면 자동으로 멈춘다.
    // Java의 `switch`는 `break`가 없으면 다음 `case`로 넘어간다 (fallthrough).
    switch state {
    case "idle": // `state`가 "idle"인 경우
        fmt.Println("캐릭터가 쉬고 있음.")
    case "walk", "run": // 여러 조건을 콤마(,)로 구분하여 한 번에 처리할 수 있다.
        fmt.Println("캐릭터가 이동 중임.")
    default: // 모든 `case`에 해당하지 않는 경우 (Java의 `default`와 동일)
        fmt.Println("알 수 없는 상태임.")
    }

    // 7.2 조건식을 사용한 switch (Java의 `if-else if` 체인과 유사)
    // `switch` 뒤에 변수 없이 `case`에 직접 조건식을 사용할 수 있다.
    level := 45 // 캐릭터 레벨 변수 선언
    switch { // 조건식 없이 `switch` 키워드만 사용
    case level < 30: // `level`이 30 미만인 경우
        fmt.Println("초보자 레벨임.")
    case level >= 30 && level < 50: // `level`이 30 이상 50 미만인 경우
        fmt.Println("중급자 레벨임.")
    default: // 위 조건들에 모두 해당하지 않는 경우
        fmt.Println("고수 레벨임!")
    }
}
```

8. 구조체 (Structs)

Go에는 클래스가 없지만, `struct`를 사용하여 관련 있는 데이터 필드를 묶어 사용자 정의 타입을 만들 수 있다. 이는 Java의 클래스나 데이터 객체와 유사하다.

실습 8: 구조체의 정의와 사용

실습 예제: `/02-기본문법/08-struct/main.go`

```
package main

import "fmt"

type Player struct {
    Name string
}
```



```

    Level int
    HP    int
}

func main() {
    // 8-1. 구조체 선언 및 초기화
    p1 := Player{Name: "Hero", Level: 10, HP: 100}
    fmt.Println("플레이어 이름:", p1.Name)

    // 8-2. 필드 값 변경
    p1.Level = 11
    fmt.Println("플레이어 레벨:", p1.Level)
}

```

9. 함수 (Functions) & 메서드

Go 함수는 **func** 키워드로 정의하며, 여러 값을 반환할 수 있다. Java에서 여러 값을 반환하려면 객체나 배열을 사용해야 하지만, Go는 언어 차원에서 지원한다.

Go에는 클래스가 없고, 메서드는 리시버(receiver)를 통해 타입에 소속시킬 수 있다.

```

type Character struct {
    Name string
    Level int
}

// Character 타입에 소속된 메서드
func (c Character) Summary() string {
    return fmt.Sprintf("%s (%d)", c.Name, c.Level)
}

```

구분	Go	Java
기본 구조	<code>func name(p type) retType</code>	<code>retType name(type p)</code>
다중 반환	<code>func name() (int, string)</code>	(별도 클래스나 배열 필요)

실습 9: 캐릭터 정보 요약 함수 만들기

변수, 상수, 제어문, 함수를 모두 활용하여 캐릭터 정보를 받아 요약 메시지와 레벨업 가능 여부를 반환하는 함수.

실습 예제: `/02-기본문법/09-functions/main.go`

```

package main

import "fmt"

// 9.1 함수 정의 및 다중 반환 값
// `func` 키워드로 함수를 정의한다. 매개변수와 반환 타입이 명시된다.
// Go 함수는 여러 개의 반환 값을 가질 수 있다. (Java는 단일 값만 반환 가능)
// 입력: `name` (string), `level` (int), `exp` (float64)
// 반환: `summary` (string), `canLevelUp` (bool)
func getCharacterSummary(name string, level int, exp float64) (string, bool) {
    // 함수 내부에서 상수 선언

```

```

const MAX_LEVEL = 100
const EXP_REQUIRED_FOR_LEVEL_UP = 90.0

// `fmt.Sprintf`는 포매팅된 문자열을 반환한다. Java의 `String.format()`과 유사하다.
summary := fmt.Sprintf("이름: %s, 레벨: %d, 경험치: %.1f", name, level, exp)
// 레벨업 가능 여부를 판단하는 논리식
canLevelUp := level < MAX_LEVEL && exp >= EXP_REQUIRED_FOR_LEVEL_UP

// 두 개의 값을 반환한다.
return summary, canLevelUp
}

func main() {
    // 9.2 함수 호출 및 다중 반환 값 받기
    // `summary`와 `canLevelUp` 변수에 각각 반환 값을 할당한다.
    summary, canLevelUp := getCharacterSummary("Hossam", 99, 85.5)

    fmt.Println(summary)

    // 9.3 반환된 `canLevelUp` 값을 이용한 조건문 처리
    if canLevelUp { // `canLevelUp`이 `true`이면 실행
        fmt.Println("레벨 업이 가능함!")
    } else { // `canLevelUp`이 `false`이면 실행
        fmt.Println("레벨 업까지 경험치가 더 필요함.")
    }

    // 9.4 다른 인자로 함수를 한 번 더 호출
    summary2, canLevelUp2 := getCharacterSummary("Alice", 80, 95.1)
    fmt.Println(summary2)
    if canLevelUp2 {
        fmt.Println("레벨 업이 가능함!")
    } else {
        fmt.Println("레벨 업까지 경험치가 더 필요함.")
    }
}

```

10. 에러 처리 방식 비교

Java는 try-catch-finally로 예외를 처리하지만, Go는 함수에서 **error** 타입을 반환하는 방식이다. 이는 에러를 명시적으로 처리하여 예상치 못한 런타임 예외를 방지하고 코드의 안정성을 높이는 Go의 철학을 반영한다.

실습 10: 에러 처리

실습 예제: [/02-기본문법/10-error/main.go](#)

```

package main

import "fmt"

func divide(a, b int) (int, error) {
    if b == 0 {
        return 0, fmt.Errorf("0으로 나눌 수 없음")
    }
    return a / b, nil
}

```

```
func main() {  
    result, err := divide(10, 0)  
    if err != nil {  
        fmt.Println("에러 발생:", err)  
    } else {  
        fmt.Println("결과:", result)  
    }  
}
```