**1. 변수**

* **선언(Declaration)과 대입(Assignment)**
* 선언: 변수 및 상수의 자료형과 이름을 컴퓨터에게 알려줌
* 대입: 변수 및 상수의 실제 값을 컴퓨터에게 알려줌  
  - 정의(Definition)이라고도 함
* 선언과 대입을 동시에 할 수도 있다.
* 변수는 “변할 수 있는 값”을 의미함  
  - 문자나 문자열까지 변수에 포함함  
  - 상수(const) 또한 변수의 범위에 포함함
* **변수를 사용하는 이유**
* 어딘 가에 저장한 값을 다시 재사용하기 위해서
* 상태(속성, 데이터)의 변화를 기억하기 위해서  
  - 그래서 변수를 ‘mutable’이라고도 부름  
  - 그래서 상수를 ‘Immutable’이라고도 부름
* 매직 넘버에 명확한 이름으로 의미를 부여하여 모호성을 없앰

**2. 변수명에 관한 규칙**

* **변수의 이름은 의미 있게 지어야 함**
* 변수가 무엇을 의미하는지 명확하게 지어야 함
* 변수명에서 데이터가 무엇을 의미하는지 알 수 있어야 함
* **코딩 표준(스탠다드): 변수명은 명확하게 지을 것**
* 변수는 동작이 아닌 어떤 상태를 저장하는 것, 따라서 동사가 아닌 명사로 짓는 것이 보통임
* 명사로 정확하게 어떤 정보를 담는지 알려주는 단어를 사용  
  - 성적이라면 ‘score’  
  - 이름이라면 ‘name’
* 여러 명사가 들어간다면 두 번째 이상의 단어의 첫 글자는 대문자로 사용  
  - 수학 성적이면 ‘math**S**core’  
  - 사과 개수라면 ‘apple**C**ount’
* 즉, 변수의 작명은 Camel Casing 규칙을 따름
* **코딩 표준(스탠다드): 상수(CONST)**
* 상수의 경우, 명사를 사용하여 **대문자**로 작성함  
  ex) SCORE, NUMBER
* 여러 명사가 들어간다면 명사 사이에 언더바(\_) 기호를 사용함  
  ex) MATH\_SCORE, APPLE\_COUNT

**3. 기본 자료형(Primitive Types)**

* **기본 자료형**
* 컴퓨터가 이해할 수 있는 가장 자연스러운 데이터  
  - 전부 다 숫자로 이루어져 있음  
  - 디지털로 이루어져 있으므로 0과 1로 구성됨
* **자료의 크기**
* 최소 단위는 비트(bit)  
  - 1 혹은 0만 담을 수 있음  
  - 비트는 하나의 1 또는 0만 담을 수 있음
* 바이트(byte) = 8비트(bit)  
  - 각 비트는 마찬가지로 하나의 0 또는 1만 가질 수 있음
* 1024Byte = 1KB
* 1024KB = 1MB
* **정수형(Integer Types)**
* 0, 1, 2와 같은 정수를 담을 수 있는 자료형
* byte, short, int, long  
  - byte = 8비트  
  - short = 16비트  
  - int = 32비트  
  - long = 64비트
* 프로그래밍 언어마다 담을 수 있는 자료의 크기 차이는 존재
* **부동소수점형(Floating-Point Types)**
* 3.14와 같은 실수를 담을 수 있는 자료형
* float, double  
  - float = 32비트  
  - double = 64비트
* 프로그래밍 언어마다 담을 수 있는 자료의 크기 차이는 존재
* **문자형(Character Type)**
* **하나**의 문자를 담는 자료형  
  - 알파벳, 숫자, 특수문자 등  
  - 문자를 감쌀 때 **작은 따옴표**를 사용함
* char = 16bit
* **불리언형(Boolean Type)**
* 참(true)과 거짓(false)을 표현하는 자료형   
  - bool = 1bit
* C나 C++ 같은 일부 언어에서는 false를 0, true를 0이 아닌 값으로 간주하기도 함

**4. 부호 있는/없는 자료형(Signed/Unsigned Types)**

* **부호 있는 자료형**
* 음수와 양수를 모두 표현  
  - sbyte, short, int, long
* **부호 없는 자료형**
* 양수만 표현  
  - byte, ushort, uint, ulong
* **왜 byte만 부호 없는 쪽이 기본인가?**
* 이를 알기 위해서는 문자형(char)의 역사를 살펴봐야 함  
  - ASCII는 처음에는 총 128개-7비트로 표현이 가능했음 byte에서 1비트가 남음  
  - 다른 나라들이 자신의 언어를 남은 1비트에 추가하려는 노력이 발생  
  - 이때 최상위 비트(부호비트)가 1이 되어 음수로 표현하게 되어 버림  
  - 따라서 8비트 문자형들은 unsinged가 좀 더 합리적이라고 판단
* **언제 사용하는가?**
* 부호 있는 변수와 부호 없는 변수는 굉장히 중요함  
  - **프로그래머의 명백한 의도**를 보여주기 때문
* int age = 17;  
  - 예를 들어 ‘나이’와 같은 int형 코드는 음수가 되면 안 됨  
  - 이를 방지하기 위함
* **오버플로우(overflow) / 언더플로우(underflow)**
* 연산의 결과로 자료형보다 큰/작은 수가 나오는 경우를 말함
* 자료형의 크기는 변하지 않기 때문에 넘어가버린 비트는 버림
* 오버/언더플로우가 발생할 경우 원치 않는 값이 나올 수 있음  
  - 오버플로우는 최대값에서 최소값으로,  
  - 언더플로우는 최소값에서 최대값으로
* **변수의 자료형을 선택할 때 주의할 점?**
* 변수의 역할에 알맞은 자료형을 선택할 것  
  - 사용할 자료형이 충분히 데이터를 담을 수 있을지
* 오버/언더플로우를 고려해서 자료형을 선택할 것

**5. 상수의 접두사(prefix)와 접미사(postfix)**

* **접두사와 접미사**
* 접두사와 접미사는 상수의 앞이나 뒤에 상수(리터럴)를 꾸미는 기호  
  - 이를 통해 리터럴에 대한 추가적인 정보를 제공
* **상수 접두사**
* 상수 앞에 붙는 기호로, 진법을 나타냄  
  - 2진수  
   - 0B-, **0b-** ex) int num1 = **0b**10;  
  - 16진수  
   - 0X-, **0x-** ex) int num1 = **0x**10;
* 일반적으로 소문자를 더 많이 사용함
* 일반적으로 접두사는 진법을 나타냄
* **상수 접미사**
* 상수 뒤에 붙는 기호로, 상수의 자료형을 나타냄  
  - 부호 여부  
   - **-u**  
   ex) uint num1 = 10**u**;  
  - long형  
   - **-l**  
   ex) long num1 = 10**u**;  
   ex) ulong num2 = 10**lu**;  
   ex) ulong num2 = 10**ul**; // 조금 더 일반적인 형태  
  - float형  
   - **-f**/-F  
   ex) float num1 = 10.0**f**;  
   ex) float num2 = 10.0**F**;

- double형  
 - **없음** 혹은 -d/-D  
 ex) double num2 = 10.0; // double의 경우 안 쓰는 경우가 더 많음  
 ex) double num2 = 10.0**d**;  
 ex) double num3 = 10.0**D**;