Integers

Integers are whole numbers with no 분수 fractional component, such as 42 and -23. Integers are either 부호 *signed* (positive, zero, or negative) or *unsigned* (positive or zero).

Swift provides signed and unsigned integers in 8, 16, 32, and 64 bit forms. These integers follow a naming convention similar to C, in that an 8-bit unsigned integer is of type `UInt8`, and a 32-bit signed integer is of type `Int32`. Like all types in Swift, these integer types have 대문자 capitalized names.

Integer Bounds

You can access the minimum and maximum values of each integer type with its `min` and `max` properties:

```swift
let minValue = UInt8.min  // minValue is equal to 0, and is of type UInt8
let maxValue = UInt8.max  // maxValue is equal to 255, and is of type UInt8
```

The values of these properties are of the appropriate-sized number type (such as `UInt8` in the example above) and can therefore be used in expressions 나란히 alongside other values of the same type.

Int

In most cases, you don’t need to pick a specific size of integer to use in your code. Swift provides an additional integer type, `Int`, which has the same size as the current platform’s native word size:

- On a 32-bit platform, `Int` is the same size as `Int32`.
- On a 64-bit platform, `Int` is the same size as `Int64`.

Unless you need to work with a specific size of integer, always use `Int` for integer values in your code. This aids code consistency and interoperability. Even on 32-bit platforms, `Int` can store any value between `-2,147,483,648` and `2,147,483,647`, and is large enough for many integer ranges.

UInt

Swift also provides an unsigned integer type, `UInt`, which has the same size as the current platform’s native word size:

- On a 32-bit platform, `UInt` is the same size as `UInt32`.
- On a 64-bit platform, `UInt` is the same size as `UInt64`.

> NOTE
>
> Use `UInt` only when you specifically need an unsigned integer type with the same size as the platform’s native word size. 그렇지 않은 경우 If this isn’t the case, `Int` is preferred, even when the values to be stored are known to be nonnegative. A consistent use of `Int` for integer values aids code interoperability, avoids the need to convert between different number types, and matches integer type inference, as described in Type Safety and Type Inference.