Variables

Swift uses variables to *store* and *refer to* values by an identifying name.

Swift also makes extensive use of variables whose values cannot be changed. These are known as *constants*.

Swift is a type-safe language, which means the language helps you to be clear about the types of values your code can work with.

Variables

Constants and variables associate a name with a value of a particular type.

```
var angle = 0.0

let iconWidth = 100
```

The value of a constant cannot be changed once it is set, whereas a variable can be set to a different value in the future.

```
var friendlyWelcome = “hello!”

friendlyWelcome = “bonjour!”
```

Variables

Constants and variables must be declared before they are used. You declare constants with the let keyword and variables with the var keyword.

```
var angle = 0.0

let iconWidth = 100
```

Variables

You can provide a type annotation when you declare a constant or variable, to be clear about the kind of values the constant or variable can store. Write a type annotation by placing a colon after the constant or variable name, followed by a space, followed by the name of the type to use.

```
var welcomeMessage: String

var playerScore: Int
```

If you don’t specify the type of value you need, Swift uses type inference to work out the appropriate type. Type inference enables a compiler to deduce the type of a particular expression automatically when it compiles your code, simply by examining the values you provide.

```
var number = 7
```

Variables

Constant and variable names cannot contain whitespace characters, mathematical symbols, arrows, private-use (or invalid) Unicode code points, or line- and box-drawing characters. Nor can they begin with a number, although numbers may be included elsewhere within the name.

```
var welcomeMessage: String

var playerScore: Int
```

Types

`Int`

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

Int is the standard integer type that is recommended for most situations. Int provides a signed value. UInt is the standard unsigned integer type.

`UInt`
`UInt8`
`UInt16`
…

Types

```
Double   (64-bit floating point number)
Float    (32-bit floating point number)
```

Floating-point numbers are numbers with a fractional component, such as 3.14159, 0.1, and -273.15.

Floating-point types can represent a much wider range of values than integer types, and can store numbers that are much larger or smaller than can be stored in an Int.

Type Safety

Because Swift is type safe, it performs type checks when compiling your code and flags any mismatched types as errors. This enables you to catch and fix errors as early as possible in the development process.


```
let alpha = 3
let beta = 0.14159
let pi = alpha + beta
```

This example produces an error. **“Binary operator ‘+’ cannot be applied to operands of type ‘Int’ and ‘Double’”**

The solution is to turn the Int into a Double, using one of the Double type’s initializer functions.

```
let pi = Double(alpha) + beta
```

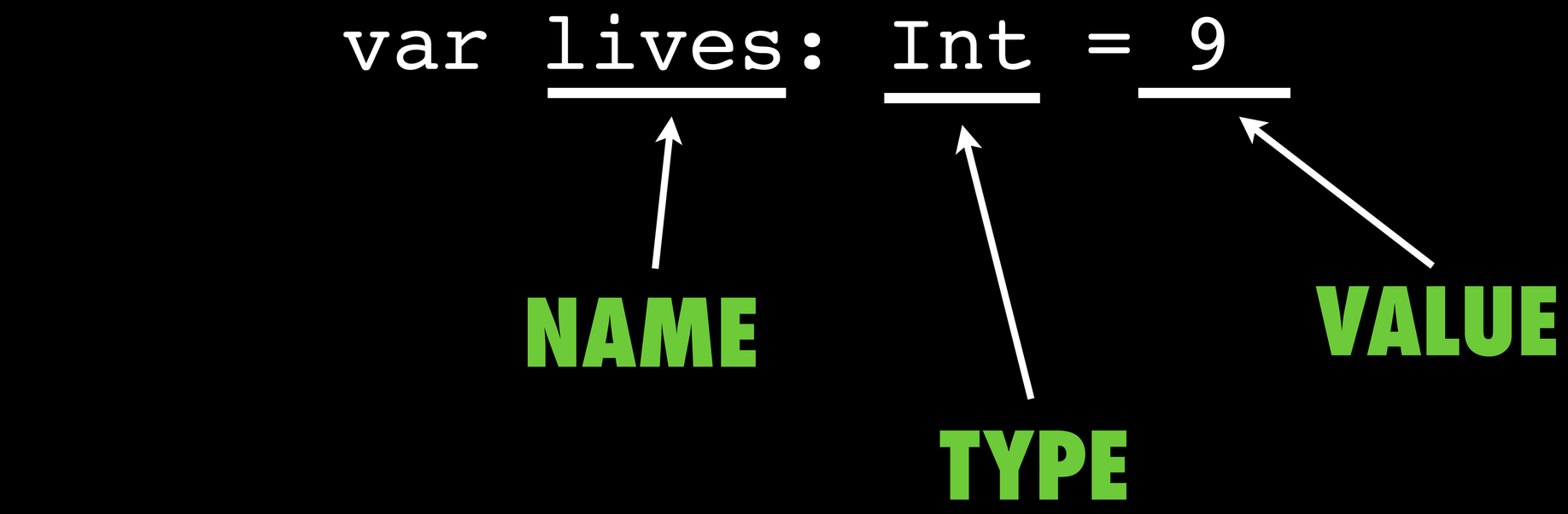
Types

`Bool`

Swift has a basic Boolean type, called Bool. Boolean values are referred to as logical, because they can only ever be true or false.

```
let orangesAreOrange = true
let turnipsAreDelicious = false
```

3 aspects of a variable



Names are important

You must avoid key words already used by Swift.
For example, “var” is a special word used by the language.

Be thoughtful about your names, and choose ones that have meaning.

Choose a convention for compound names, and be consistent.

```
highScore or high_score
```

Begin with a lower case letter. Starting with an upper case letter is reserved for class names.

Names are important

Creating good names will make your program easier to understand.

For example, you need a variable for the horizontal position of the front wheel in a drawing of a vehicle.

x, circleX, or frontWheelX

are possible names you might come up with.
Is one of them better than the others?