

Elementos básicos



Sentencias, bloques y
comentarios

Sentencias

- En Swift, las sentencias se escriben una en cada línea
- No es necesario incluir un `;` al final

Sentencias y ;

```
let cat = "🐱"; print(cat)
```

Bloques

- Agrupan instrucciones
- Definen el ámbito de las variables
- En Swift se utilizan las llaves { y } para delimitarlos

Comentarios

- De una línea, `//`
- De múltiples líneas, `/* */`
- Se pueden anidar, si están balanceados `/* /* */ */`

Variables y constantes

Declaración

```
let maximumNumberOfLoginAttempts = 10 // Constante  
var currentLoginAttempt = 0 // Variable
```

```
var x = 0.0, y = 0.0, z = 0.0 // Múltiple
```


Anotaciones de tipo

```
var welcomeMessage: String
```

```
welcomeMessage = "Hello"
```

```
var red, green, blue: Double
```

Tipos de datos básicos

Tipo	Descripción
Int	Valor numérico entero
Float	Valor numérico de precisión simple
Double	Valor numérico de precisión doble
Bool	Valor lógico, verdadero o falso
Character	Caracter individual
String	Cadena de texto

Nombres

```
let π = 3.14159  
let 你好 = "你好世界"  
let 🐶🐮 = "dogcow"
```

- No pueden contener espacios ni símbolos de flecha o bloques
- No pueden comenzar por número
- Admiten Unicode

Salida por consola

```
print("This is a string")
```

```
var friendlyWelcome = "Hello!"
```

```
print("The current value of friendlyWelcome is \$(friendlyWelcome)")
```

Valores numéricos y lógicos

Tipos de datos básicos

Tipo	Descripción
Int	Valor numérico entero
Float	Valor numérico de precisión simple
Double	Valor numérico de precisión doble
Bool	Valor lógico, verdadero o falso
Character	Caracter individual
String	Cadena de texto

Enteros

- Enteros con signo: `Int`
- Enteros sin signo: `UInt`
- Existen `Int8`, `Int16`, `Int32` e `Int64` (y las versiones sin signo)
- Al usar `Int`, internamente la longitud cambia a `Int32` o `Int64` dependiendo de la plataforma (32 o 64 bits)

Límites de los enteros

```
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
```


Coma flotante

- Precisión simple: `Float` (32 bits, 6 dígitos decimales de precisión)
- Precisión doble: `Double` (64 bits, 15 dígitos decimales de precisión)

Inferencia de tipos

```
let meaningOfLife = 42
```

```
let pi = 3.14159
```

```
let anotherPi = 3 + 0.14159
```

Literales numéricos

	Literal	Notación	Ejemplo
Enteros	Decimal	Sin prefijo	1_000_000
	Binario	0b	0b1011
	Octal	0o	0o34
	Hexadecimal	0x	0xF3A
Coma flotante	Decimal	e	1.25e4
	Hexadecimal	p	0xAp3

Conversiones de tipo

```
let three = 3
let pointOneFourOneFiveNine = 0.14159
let pi = Double(three) + pointOneFourOneFiveNine

let integerPi = Int(pi) // Se trunca el valor
```

Alias de tipos

```
typealias AudioSample = UInt16
```

```
var maxAmplitudeFound = AudioSample.min
```

Valores lógicos

```
let orangesAreOrange = true
```

```
let turnipsAreDelicious = false
```

Cadenas de texto

Tipos de datos básicos

Tipo	Descripción
Int	Valor numérico entero
Float	Valor numérico de precisión simple
Double	Valor numérico de precisión doble
Bool	Valor lógico, verdadero o falso
Character	Caracter individual
String	Cadena de texto

Cadenas de texto

```
let someString = "Some string literal value"
```

Multiline String

```
let quotation = """  
The White Rabbit put on his spectacles.  "Where shall I begin,  
please your Majesty?" he asked.  
  
"Begin at the beginning," the King said gravely, "and go on  
till you come to the end; then stop."  
"""
```

Multiline String

```
let lineBreaks = """
```

```
This string starts with a line break.  
It also ends with a line break.
```

```
"""
```

Multiline String

```
let codigoFuenteIndentado = """  
    func greet(person: String) -> String {  
        let greeting = "Hello, " + person + "!"  
        return greeting  
    }  
    """ // Se ignoran los primeros cuatro espacios
```

Caracteres especiales

```
let wiseWords = "\"Imagination is more important than knowledge\" – Einstein"  
// "Imagination is more important than knowledge" – Einstein
```

```
let dollarSign = "\u{24}"           // $,   Unicode scalar U+0024  
let blackHeart = "\u{2665}"         // ♥,   Unicode scalar U+2665  
let sparklingHeart = "\u{1F496}"    // 💖,   Unicode scalar U+1F496
```

Cadena vacía

```
var emptyString = ""           // empty string literal
var anotherEmptyString = String() // initializer syntax

if emptyString.isEmpty {
    print("Nothing to see here")
}

var noEstaVacía: String // No está inicializada
```

Características de los String

- Si se declaran con `var` son mutables, si se declaran con `let`, no
- Se pueden concatenar con `+` y `+=`
- Se pueden recorrer los caracteres individuales con un `for-in`
- Son tipos por valor, se copian al pasarlos a funciones o asignarlos a otras variables
- Son compatibles al 100% con el `NSString` de Foundation

Recorrer los caracteres de un String

```
for character in "Dog!🐶" {  
    print(character)  
}
```


Concatenación de Strings

```
let string1 = "hello"  
let string2 = " there"  
var welcome = string1 + string2
```

```
var instruction = "look over"  
instruction += string2
```

```
let exclamationMark: Character = "!"  
welcome.append(exclamationMark)
```

Interpolación de Strings

```
let multiplier = 3  
let message = "\(multiplier) times 2.5 is \((Double(multiplier) * 2.5))"
```

Recuento de caracteres

```
let unusualMenagerie = "Koala 🐨, Snail 🐌, Penguin 🐧, Dromedary 🐪"  
print("unusualMenagerie has \(unusualMenagerie.count) characters")
```

Acceso a los caracteres individuales

```
let greeting = "Guten Tag!"  
greeting[greeting.startIndex] // G  
  
greeting[greeting.index(before: greeting endIndex)] // !  
  
greeting[greeting.index(after: greeting.startIndex)] // u  
  
let index = greeting.index(greeting.startIndex, offsetBy: 7)  
greeting[index] // a
```

Acceso a los caracteres individuales

```
greeting[greeting.endsWith] // Error  
greeting.index(after: greeting.endsWith) // Error
```

```
for index in greeting.indices {  
    print("\(greeting[index]) ", terminator: " ")  
}
```

Insertar caracteres

```
var welcome = "hello"  
welcome.insert("!", at: welcome endIndex)  
// welcome now equals "hello!"
```

```
welcome.insert(contentsOf: " there", at:  
welcome.index(before: welcome endIndex))  
// welcome now equals "hello there!"
```

Eliminar caracteres

```
welcome.remove(at: welcome.index(before: welcome.endIndex))  
// welcome now equals "hello there"
```

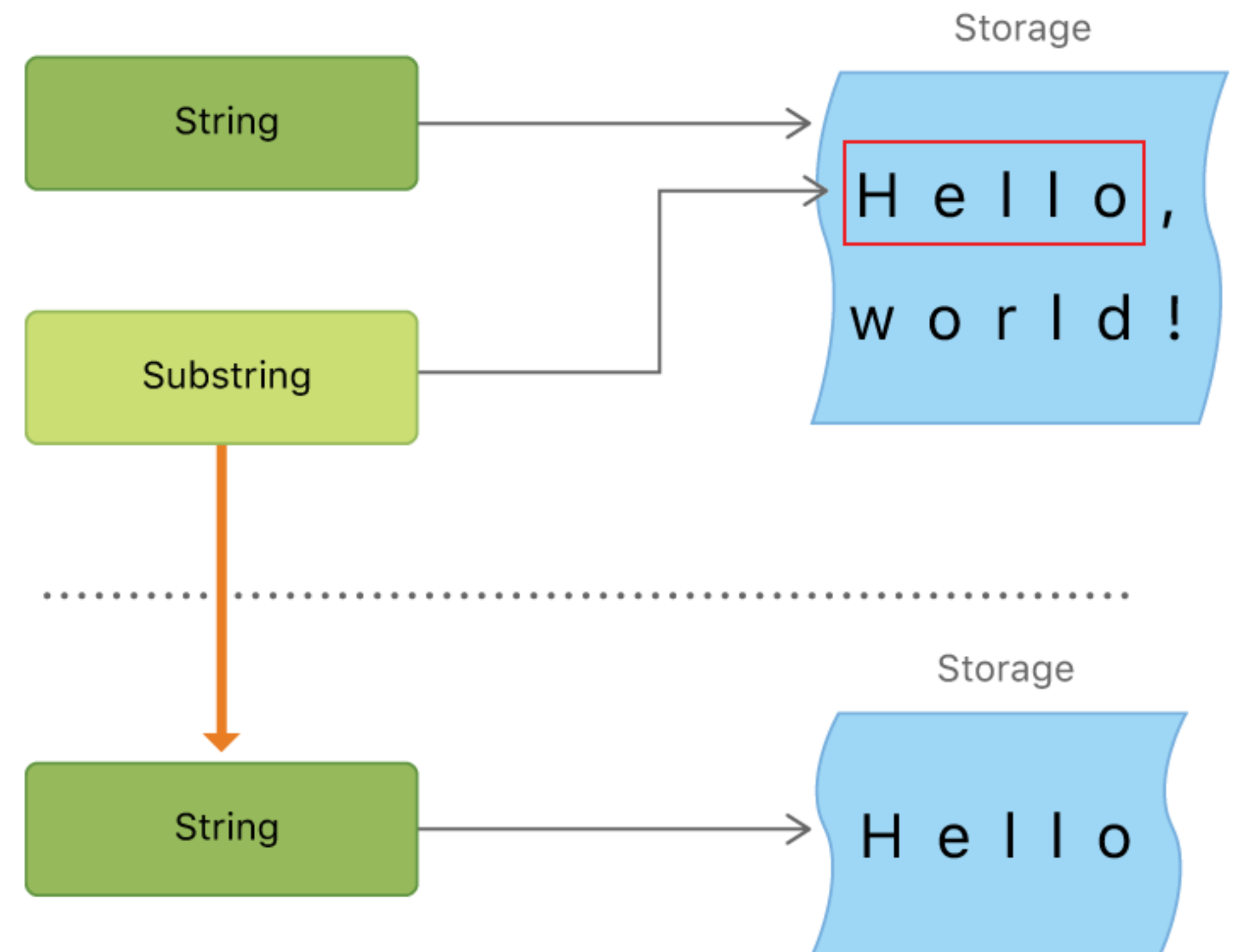
```
let range = welcome.index(welcome.endIndex, offsetBy: -6)..<welcome.endIndex  
welcome.removeSubrange(range)  
// welcome now equals "hello"
```

Subcadenas

```
let greeting = "Hello, world!"  
let index = greeting.index(of: ",") ?? greeting endIndex  
let beginning = greeting[..<index]  
// beginning is "Hello"  
  
// Convert the result to a String for long-term storage.  
let newString = String(beginning)
```


Subcadenas

```
let greeting = "Hello, world!"  
let index = greeting.index(of: ",") ?? greeting.endIndex  
let beginning = greeting[..<index]  
// beginning is "Hello"  
  
// Convert the result to a String for long-term storage.  
let newString = String(beginning)
```



Comparar Strings

- Se pueden comparar directamente con el operador `==`
- Disponen de `hasPrefix(_ :)` y `hasSuffix(_ :)` para comparar el principio o el final de la cadena

Igualdad de Strings

```
let quotation = "We're a lot alike, you and I."  
let sameQuotation = "We're a lot alike, you and I."  
  
if quotation == sameQuotation {  
    print("These two strings are considered equal")  
}
```

Tuplas

Tuplas

- Agrupan múltiples valores en uno
- Los valores pueden ser de cualquier tipo
- No tienen que ser del mismo tipo
- Permiten que una función devuelva varios valores agrupados
- Para agrupaciones complejas, hay que usar estructuras o clases, no tuplas

Tuplas

```
let http404Error = (404, "Not Found")
```

```
let (statusCode, statusMessage) = http404Error
```

```
print("The status code is \(statusCode)")
```

```
print("The status message is \(statusMessage)")
```

Tuplas

```
let (justTheStatusCode, _) = http404Error
print("The status code is \ (justTheStatusCode) ")

print("The status code is \ (http404Error.0) ")
print("The status message is \ (http404Error.1) ")
```

Tuplas

```
let http200Status = (statusCode: 200, description: "OK")  
  
print("The status code is \(http200Status.statusCode)")  
print("The status message is \(http200Status.description)")
```


Variables opcionales

Optionals

- Permiten definir variables que pueden o no tener valor
- Se crean añadiendo ? al tipo de dato de la variable

Optionals

```
let possibleNumber = "123"
```

```
let convertedNumber = Int(possibleNumber)
```

Valor nil

```
var serverResponseCode: Int? = 404  
serverResponseCode = nil
```

```
var surveyAnswer: String? // nil
```

Extraer el valor de un opcional

- Imprescindible: `Int` no es lo mismo que `Int?`
- Forced unwrapping: usando `!`
- Optional binding: para extraer el valor en un `if` o `while`
- Optional chaining: usando `?`, cuando trabajemos con propiedades de estructuras o clases

Forced unwrapping

```
let possibleNumber = "123"
let convertedNumber = Int(possibleNumber)

if convertedNumber != nil {
    print("\(possibleNumber) has an integer value of \(convertedNumber!)" )
} else {
    print("\(possibleNumber) could not be converted to an integer")
}
```

Optional binding

```
let possibleNumber = "123"

if let actualNumber = Int(possibleNumber) {
    print("\(possibleNumber) has an integer value of \(actualNumber)")
} else {
    print("\(possibleNumber) could not be converted to an integer")
}
```

Opcionales implícitos

- Se declaran con ! en vez de ? en el tipo de dato
- No necesitan de ! para acceder al valor, pero si no tienen valor disparar un error en tiempo de ejecución
- Se usan en la inicialización de clases con referencias **unowned**

Opcionales implícitos

```
let possibleString: String? = "An optional string."  
let forcedString: String = possibleString!  
  
let assumedString: String! = "An implicitly unwrapped optional string."  
let implicitString: String = assumedString
```

Operador de coalescencia nil

- Se utiliza con opcionales mediante ??
- Permite extraer el valor del opcional o si vale `nil`, un valor por defecto
- `a ?? b` es una abreviatura de `a != nil ? a! : b`

Operador de coalescencia nil

```
let defaultColorName = "red"  
var userDefinedColorName: String? // nil  
  
var colorNameToUse = userDefinedColorName ?? defaultColorName
```

Operador de coalescencia nil

```
userDefinedColorName = "green"
```

```
colorNameToUse = userDefinedColorName ?? defaultColorName
```

Operadores: asignación y aritméticos

Operador de asignación

- Copia el contenido de la parte derecha en la parte izquierda
- Descompone los valores de las tuplas en variables individuales
- No devuelve valor
- Hay versiones compuestas, como +=

Operador de asignación

```
let b = 10  
var a = 5  
a = b
```

```
let (x, y) = (1, 2)
```

Operador de asignación

```
var i = 6  
var j = i = 5 // Error  
  
if j=5 { // Error  
  
}
```


Operadores aritméticos

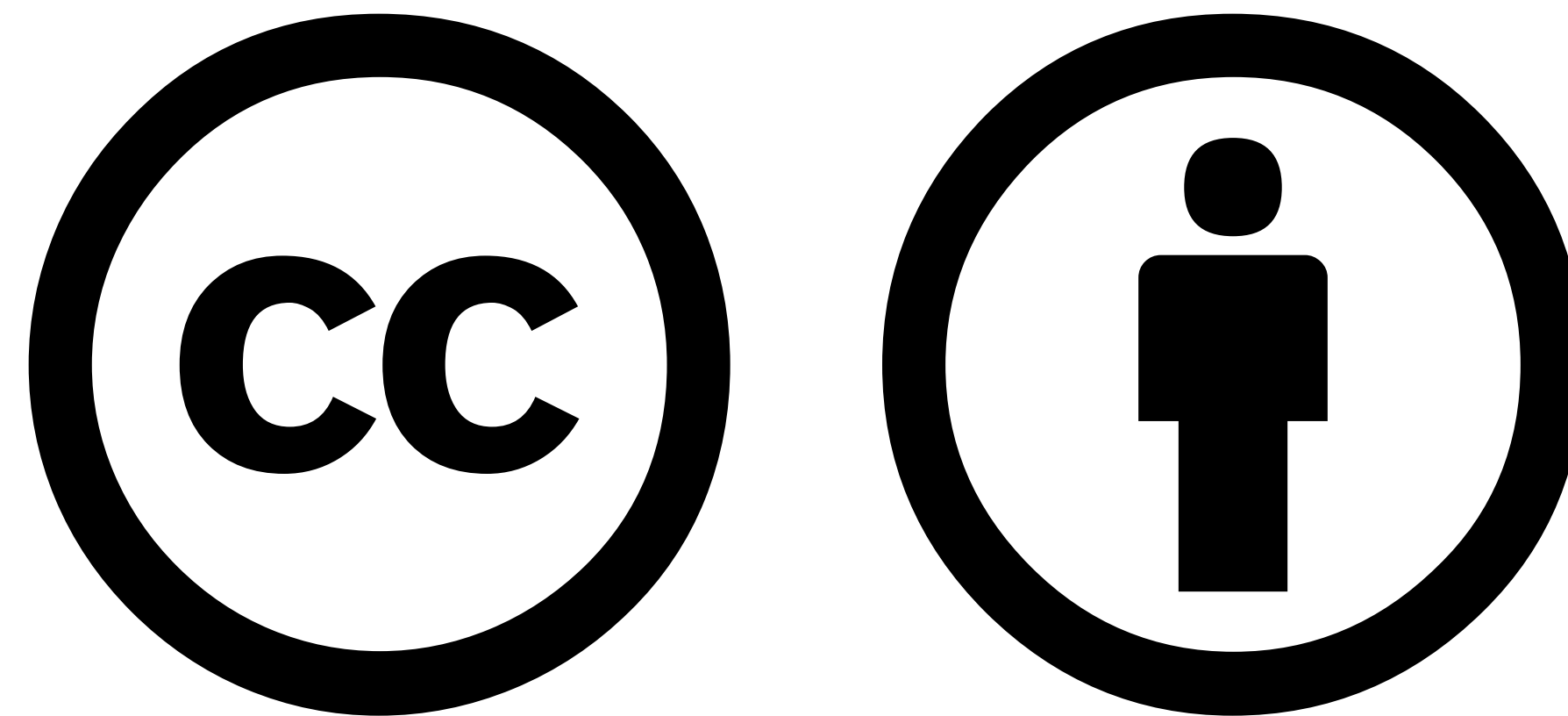
Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto de la división
-i	Menos unario (cambio de signo)
+i	Más unario (no afecta al valor)

Operadores aritméticos

- No soportan overflow o underflow, se produce un error de tiempo de ejecución
- Hay versiones con overflow, como &+
- La división y el resto entre 0 también provocan un error

Operadores aritméticos (con overflow)

Operador	Operación
&+	Suma
&-	Resta
&*	Multiplicación



Excepto si se especifica lo contrario, esta presentación está bajo licencia

<https://creativecommons.org/licenses/by/4.0/>

© 2017 Ion Jaureguialzo Sarasola. Algunos derechos reservados.