Swift

Tema 2. Variables y tipos de datos I

Objetivos

Luego de navegar esta cápsula, deberás poder hacer lo siguiente:

- definir y diferenciar variables y constantes;
- reconocer los tipos de datos básicos que soporta Swift;
- convertir tipos de datos;
- realizar operaciones aritméticas;
- realizar operaciones booleanas; y
- crear, modificar y acceder a valores opcionales.

Constantes y variables

Durante el desarrollo de las aplicaciones, es necesario almacenar valores que permiten, por ejemplo, mantener la información del usuario, realizar operaciones de lógica de negocio, mantener un contador, etcétera. Para esto, contamos con dos formas de hacerlo: constantes y variables.

Las constantes asocian un valor con un nombre y no se pueden cambiar durante la ejecución del programa. Se declaran usando la palabra clave `let`.

```
let maximaCantidadDeIntentos: Int = 10
```

Las variables, por otro lado, son aquellas cuyo valor puede ser modificado por otro en el futuro. Se declaran usando la palabra clave `var`.

```
var resultado: Int = 0
```

Tipos de datos básicos

Cada variable o constante se declara con un tipo de dato específico, el cual permite indicar qué posibles valores puede almacenar; por ejemplo, si se indica que es un número, no se podrán almacenar caracteres.

Los tipos de datos básicos más usados son los siguientes:

Enteros

Los números enteros se definen con el tipo `Int`. Se pueden utilizar valores positivos, negativos o cero.

```
let numeroEntero: Int = 10
```

Fraccionarios (Float y Double)

Para definir números con componente fraccionario, se utilizan estos dos tipos de datos. La diferencia entre ambos reside en la cantidad de valores que pueden representar: un `Double` representa un valor de 64 bits (precisión de 15 dígitos decimales), mientras que un `Float` representa un valor de 32 bits (precisión de 6 dígitos decimales).

```
let variableFraccionaria1: Float = 15.20

let variableFraccionaria2: Double = 15.20
```

En este ejemplo particular, ambas variables tienen el mismo valor, pero, como mencionamos anteriormente, `variableFraccionaria2` permitiría almacenar mayor precisión (si se necesitara).

Booleanos (verdadero o falso)

El tipo `Booleano` en Swift se define como `Bool`, y sus valores puede ser `true` o `false`:

```
let usuarioRegistrado: Bool = true
```

Strings

Los `strings` se definen como el tipo de dato `String`. Se pueden declarar como literales (secuencia de caracteres entre comillas `“”`):

```
let unMensaje: String = "Hola, soy un mensaje"
```

o como la concatenación de diferentes `strings`, usando el operador `+` :

```
let string1: String = "Hola,"

let string2: String = " soy un mensaje"

saludo = string1 + string2

// saludo tiene el valor "Hola, soy un mensaje"
```

En ocasiones, se requiere que los strings tengan varias líneas. Si bien esto se puede lograr agregando “\n”, también puede definirse usando tres comillas al inicio y otras tres en el final, teniendo en cuenta que tanto las comillas iniciales como las finales deben estar en su propia línea.

```
let unMensajeEnVariasLineas: String = """

Swift es un lenguaje de programación multiparadigma

creado por “Apple” enfocado en el desarrollo de

aplicaciones para iOS y macOS.

"""
/* unMensajeEnVariasLineas tiene el valor

"Swift es un lenguaje de programación multiparadigma

creado por “Apple” enfocado en el desarrollo de

aplicaciones para iOS y macOS." */
```

Si lo que se desea es tener varias líneas para organizar el código, pero dichos saltos de línea no deben incluirse en el string propiante, entonces debe agregarse una barra invertida (“\”), para señalar el salto de línea.

```
let unMensajeDividido: String = """

Swift es un lenguaje de programación multiparadigma \

creado por “Apple” enfocado en el desarrollo de \

aplicaciones para iOS y macOS.
```

"""

// unMensajeDividido tiene el valor "Swift es un lenguaje de programación multiparadigma creado por “Apple” enfocado en el desarrollo de aplicaciones para iOS y macOS."

String interpolation
Para poder utilizar el valor de variables dentro de un string (para mostrar un mensaje, por ejemplo), se utiliza una técnica conocida como *string interpolation*. Esta implica agregar la variable dentro de la siguiente sintaxis: “\(variable)”.

```
let nombre: String = "Matias"

let edad: Int = 30

let mensaje: String = "Mi nombre es \(nombre) y tengo \(edad) años."

// mensaje tiene el valor "Mi nombre es Matias y tengo 30 años."
```

Conversiones entre tipos de datos

En ocasiones, se puede tener un tipo de dato y requerir que se convierta a otro tipo de dato específico.

En el caso de los numéricos, podemos pasar de un entero a uno fraccionario o viceversa. Por ejemplo, para convertir un número entero a fraccionario, podemos hacer lo siguiente:

```
let parteEntera: Int = 3

let parteFraccionaria: Double = 0.14159

let numeroPi: Double = Double(parteEntera) + parteFraccionaria

// numeroPi tiene el valor 3.14159 y es de tipo Double
```

En este caso, la variable parte Entera es usada para crear el numeroPi, y, para ello, ambas partes de la suma deben ser del mismo tipo, por lo que es necesario convertirlo a Double. Si esta conversión no estuviera hecha, se obtendría un error en tiempo de compilación.

Esto también podría hacerse de la forma inversa. Sigamos con el ejemplo anterior:

```
let numeroPiEntero: Int = Int(numeroPi)

// numeroPiEntero tiene el valor 3 y es del tipo Int
```

Es importante resaltar que las conversiones no siempre se realizarán de forma satisfactoria, debido a que los valores pueden no ser "compatibles"; en esos casos, la conversión entre tipos devuelve un valor opcional, y en el caso de que dicha conversión falle, se tendrá un valor nulo (nil).

Veremos el concepto de variables opcionales al final de esta cápsula, en el apartado "Opcionales".

```
let mensajeNumerico: String = "1234"

let mensajeDeTexto: String = "Hola mundo"

let numeroEntero1: Int? = Int(mensajeNumerico)

let numeroEntero2: Int? = Int(mensajeDeTexto)

// numeroEntero1 es un entero opcional con el valor 1234

// numeroEntero2 es un entero opcional sin valor (nil)
```

Seguridad e inferencia de tipos

Swift verifica en tiempo de compilación el contenido de las variables. Esto se conoce como "seguridad de tipo" (*type-safety*); si se declara una variable como un entero Int, no será posible posteriormente asignarle un valor booleano Bool, ya que arrojará un error.

La inferencia, por su parte, implica que Swift puede determinar el tipo de dato sin necesidad de que sea definido explícitamente; simplemente, tomará el valor inicial de la variable y lo determinará a partir de ahí:

```
let hola = "Hola"
```

Swift interpreta

```
let hola: String = "Hola"
```

En el caso de los números, si el valor es entero, se tomará como un Int, y para el caso de los fraccionarios, se tomará por defecto Double.

```
let maximaCantidadDeIntentos = 10

// maximaCantidadDeIntentos es definido como Int


let pi = 3.14159

// pi is definido como Double
```

En el caso particular en que se tenga una combinación de valores enteros y fraccionarios, se inferirá `Double` como tipo de dato.

```
let otraFormaDeEscribirPi = 3 + 0.14159

// otraFormaDeEscribirPi es definido como Double
```

Para verificar el tipo de dato de una variable en Xcode, puedes mantener presionada la tecla «Option» y hacer clic sobre la variable que deseas consultar (⌥ + clic).

Actividad de repaso en un Playground de Xcode

1. Crea una variable de tipo String llamada `parcial1` y asígnale un texto con una nota entre 1 y 10 (sin decimales).

2. Crea una variable llamada `parcial2` y asígnale un texto con una nota entre 1 y 10, incluyendo un valor decimal.

3. Convierte las variables del punto 1 y 2 en enteros. ¿Qué sucede con la nota de `parcial2`?

4. Convierte las variables del punto 1 y 2 en otro tipo de dato de forma que no se pierda la precisión de las notas. ¿Hace alguna diferencia que el fraccional de `parcial2` se defina usando punto o coma?

5. Utiliza `print` para mostrar en consola un texto indicando las notas obtenidas; por ejemplo, si en el primer parcial se obtuvo 5 y en el segundo 8.3, el texto debe consistir en “Los resultados del parcial fueron 5 en el primero y 8.3 en el segundo”.

Operadores:
Operaciones aritméticas

Asignación

Para inicializar o actualizar el valor de una variable o constante, se usa el operador `=`.

```
let b = 10

var a = 5

a = b
```

Operaciones estándar (suma, resta, multiplicación, división)

Swift admite los cuatro operadores aritméticos estándares para todos los tipos de números:

```
1 + 2 // es igual a 3

5 - 3 // es igual a 2

2 * 3 // es igual a 6

10.0 / 2.5 // es igual a 4.0
```

Módulo (%)

El módulo es un operador binario `a % b` que define cuántos múltiplos de *b* entrarán en *a* y devuelve el valor que queda fuera.

```
let resto = 17 % 5

// resto es igual a 2 (en 17 entraran 3 veces 5 y quedan afuera 2)
```

Operadores de comparación

- Igual a (a == b)
- Distinto de (a != b)
- Mayor a (a > b)
- Menor a (a < b)
- Mayor o igual a (a >= b)
- Menor o igual a (a <= b)

El resultado de cada comparación es un Bool (true o false). Ejemplos:

```
1 == 1 // true, 1 es igual a 1

2 != 1 // true, 2 es distinto a 1

2 > 1 // true, 2 es mayor a 1

1 < 2 // true, 1 es menor a 2

1 >= 1 // true, 1 es mayor o igual a 1

2 <= 1 // false, 2 NO es menor o igual a 1
```

Operadores lógicos

NOT (!a)

El operador lógico NOT invierte el valor de un Bool. Es un prefijo (!) y suele usarse para definir cuando algo “no es”.

```
let usuarioRegistrado = true

!usuarioRegistrado // false
```

AND (a && b)

Crea una expresión lógica donde ambos valores deben ser verdaderos para que la expresión completa sea verdadera. Utiliza el símbolo &&.

```
let usuarioValido = true

let claveValida = true

let usuarioRegistrado = false
```

usuarioValido && claveValida // true

usuarioValido && usuarioRegistrado // false

OR (a || b)

Crea una expresión lógica donde al menos uno de los dos valores debe ser `true` para que la expresión completa sea `true`. Utiliza el símbolo `||`.

let usuarioValido = true

let claveValida = false

let usuarioRegistrado = false

usuarioValido || claveValida // true

claveValida || usuarioRegistrado // false

Los operadores lógicos se pueden combinar (`a && b || c && d`), recordando siempre que son asociativos a izquierda. Es decir, que Swift evaluará de a pares, empezando por la expresión que más a la izquierda esté primero. Para definir la asociatividad, se pueden utilizar paréntesis.

Ejercitación 2 en el Playground de Xcode

El estado de una materia depende del valor del promedio:

- mayor o igual a 7 ➡ Aprobada;
- mayor o igual a 4 y menor que 7 ➡ Debe presentarse al final;
- menor a 4 ➡ Reprobada;

Teniendo en cuenta estos datos, lleva a cabo las siguientes tareas:

1. Crea tres variables que corresponden a las notas de los parciales de una materia y en una cuarta variable calcula la nota final de la materia promediando dichos parciales.

2. Crea una variable `materiaAprobada` que indique si la materia se aprobó.

3. Crea una variable `requiereFinal` que indique si es necesario presentar el examen final.

4. Utilizando las variables del punto 2 y 3, crea una variable materiaReprobada que indique si la materia fue reprobada.

Opcionales

Una variable puede definirse como opcional cuando puede no tener valor en algún momento; es decir, que el valor puede ser nil (nulo). Un valor se puede definir como opcional agregando un signo de pregunta (?) al tipo.

```
var codigoDeError: Int? = 500

// codigoDeError es un Int opcional, que tiene el valor 500



codigoDeError = nil

// codigoDeError es un Int opcional, que no tiene valor (nil)
```

Si se define una variable opcional y no se le “setea” un valor inicial, el valor será nil por defecto.

```
var respuesta: String?

// respuesta es automáticamente definido como nil
```

Para utilizar el valor de una variable opcional, primero hay que validar si dicho valor existe. Esto se conoce como *unwrap*. Hay diferentes formas de hacerlo:

Force unwrapping (obtener el valor a la fuerza)

Una forma de obtener el valor de un opcional es agregando un signo de admiración (!) al final de este, lo que se conoce como *force unwrapping*.

Supongamos que tenemos dos variables, una definida como opcional y la otra no:

```
var resultadoParcial: Int? = 50
```

```
var resultadoTotal: Int = 0
```

Si quisiéramos obtener el valor utilizando *force unwrapping*, deberíamos hacer lo siguiente:

```
resultadoTotal = resultadoParcial! + 10
```

Esto permitiría acceder al valor de `resultadoParcial` y utilizarlo en la operación.

¿Y si la variable opcional no tiene valor?

Si la variable opcional no tiene valor (es `nil`) y se intenta acceder a ella utilizando la técnica del *force unwrapping*, se producirá un error en tiempo de ejecución, y la aplicación se detendrá. Es por esto que el *force unwrapping* solo debería usarse cuando se está seguro de que la variable en efecto tiene un valor.

⚠ Dado el riesgo que genera el *force unwrapping*, se recomienda utilizar otras alternativas más seguras para obtener el valor de un opcional.

Nil coalescing

Esta técnica es utilizada cuando queremos trabajar con una variable opcional y tener un valor por defecto en caso que el valor sea `nil`. Se define con el operador `??` y se utiliza de la siguiente manera:

```
var resultadoParcial: Int? = 50

var resultadoParcialConValor: Int = resultadoParcial ?? 0

// resultadoParcialConValor tiene el valor 50
```

En este caso, como la variable opcional `resultadoParcial` tiene un valor 50, utilizando la técnica *nil coalescing,* `resultadoParcialConValor` se asigna con dicho valor.

```
var resultadoParcial: Int?
```

```
var resultadoParcialConValor: Int = resultadoParcial ?? 0

// resultadoParcialConValor tiene el valor 0
```

Por otro lado, en este segundo caso, `resultadoParcial` es `nil`, por lo que `resultadoParcialConValor` se asigna con el valor por defecto, que en este ejemplo es 0.

Ejercitación 3 en el Playground de Xcode

1. Crea una variable del tipo `string` llamada `miEdadEnTexto` y asígnale un texto con tu edad en números.

2. Crea una variable opcional del tipo `entero` llamada `miEdad` y asígnale el resultado de convertir `miEdadEnTexto` a `entero`.

3. Toma la variable `miEdad` creada anteriormente y muestra en consola un mensaje indicando tu edad, por ejemplo “Mi edad es 30 años”. Utiliza *nil coalescing* para mostrar 0 como edad por defecto en caso de que el valor sea `nil`.