

Unidad 3 Tipos de datos en Kotlin. Funciones.

- 1. Tipos de datos
- 2. Define y usa variables

Comparación entre definir y usar una variable

Ejemplo

Declaración de variable

Palabra clave para definir una variable nueva

Nombre de la variable

Tipo de datos variable

Operador de asignación

Valor inicial variable

Usa una variable

Plantilla de strings

Inferencia de tipo

Operaciones matemáticas básicas con números enteros

- 3. Actualiza variables

Operadores de incremento y disminución

- 4. Explora otros tipos de datos

Double

String

- 5. Convenciones de codificación
- 6. Comenta en tu código
- 7. Cómo definir una función y cómo llamarla
- 8. Cómo devolver un valor de una función

El tipo `Unit`

Cómo mostrar `String` de `birthdayGreeting()`

- 9. Cómo agregar un parámetro a la función `birthdayGreeting()`

- 10. Funciones con varios parámetros

Firma de la función

- 11. Argumentos con nombre
- 12. Argumentos predeterminados



Unión Europea

Fondo Social Europeo

El FSE invierte en tu futuro

Fecha	Versión	Descripción
11/10/2024	1.0.0	Versión inicial

Unidad 3 Tipos de datos en Kotlin. Funciones.

1. Tipos de datos

Cuando decides qué aspectos de tu app pueden ser variables, es importante que especifiques *qué tipo de datos* se pueden almacenar en esas variables. En Kotlin, existen algunos tipos de datos básicos comunes. En la siguiente tabla, se muestra un tipo de datos diferente en cada fila. Para cada tipo de datos, se incluye una descripción del tipo de datos que puede contener y valores de ejemplo.

Tipo de datos de Kotlin	Qué tipo de datos puede contener	Ejemplos de valores literales
<code>String</code>	Texto	<code>"Add contact"</code> <code>"Search"</code> <code>"Sign in"</code>
<code>Int</code>	Número entero	<code>32</code> <code>1293490</code> <code>-59281</code>
<code>Double</code>	Número decimal	<code>2.0</code> <code>501.0292</code> <code>-31723.99999</code>
<code>Float</code>	Número decimal (que es menos preciso que un <code>double</code>). Tiene un <code>f</code> o <code>F</code> al final del número.	<code>5.0f</code> <code>-1630.209f</code> <code>1.2940278F</code>
<code>Boolean</code>	<code>true</code> o <code>false</code> . Usa este tipo de datos cuando solo haya dos valores posibles. Ten en cuenta que <code>true</code> y <code>false</code> son palabras clave en Kotlin.	<code>true</code> <code>false</code>

Nota: Para conocer los rangos válidos de los tipos de datos numéricos (`Int`, `Double` y `Float`), consulta [Números](#). Para obtener información específica sobre la diferencia entre `Double` y `Float`, consulta [esta tabla](#) en la que se comparan los dos tipos de datos.

2. Define y usa variables

Comparación entre definir y usar una variable

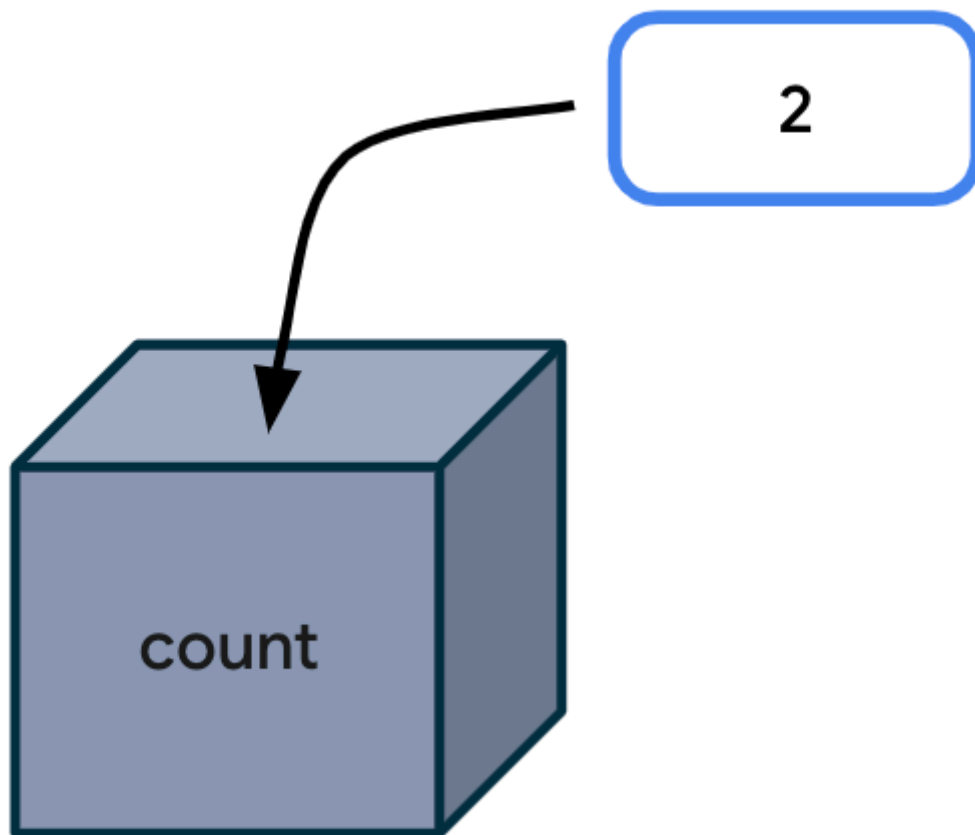
Para poder usar una variable en tu código, primero debes definirla. Esto es similar a lo que aprendiste en el codelab anterior sobre cómo definir funciones antes de llamarlas.

Cuando *defines* una variable, le asignas un nombre para identificarla de manera única. También debes decidir qué tipo de datos puede contener mediante la especificación del tipo de datos. Por último, puedes proporcionar un valor inicial que se almacenará en la variable, pero es opcional.

Nota: Es posible que escuches la frase alternativa "declarar una variable". Las palabras declarar y definir se pueden usar indistintamente y tienen el mismo significado. También puedes escuchar el término "definición de variables" o "declaración de variables", que hace referencia al código que define una variable.

Cuando defines una variable, podrás *usarla* en el programa. Para usar una variable, escribe el nombre de la variable en tu código, lo que le indica al compilador de Kotlin que deseas usar el valor de la variable en ese punto del código.

Por ejemplo, define una variable para la cantidad de mensajes no leídos en la carpeta Recibidos de un usuario. La variable puede tener el nombre `count`. Almacena un valor, como el número `2` dentro de la variable, que represente `2` mensajes no leídos en la bandeja Recibidos del usuario. (Puedes elegir un número diferente para almacenar en la variable, pero a los efectos de este ejemplo, usa el número `2`).



Cada vez que tu código necesite acceder a la cantidad de mensajes no leídos, escribe `count`. Cuando ejecutas tus instrucciones, el compilador de Kotlin ve el nombre de la variable en tu código y usa el valor de la variable en su lugar.

Técnicamente, existen palabras de vocabulario más específicas para describir este proceso:

Una *expresión* es una unidad de código pequeña que tiene un valor. Puede consistir en variables, llamadas a funciones y mucho más. En el siguiente caso, la *expresión* se compone de una variable: la variable `count`. El valor de la expresión es `2`.

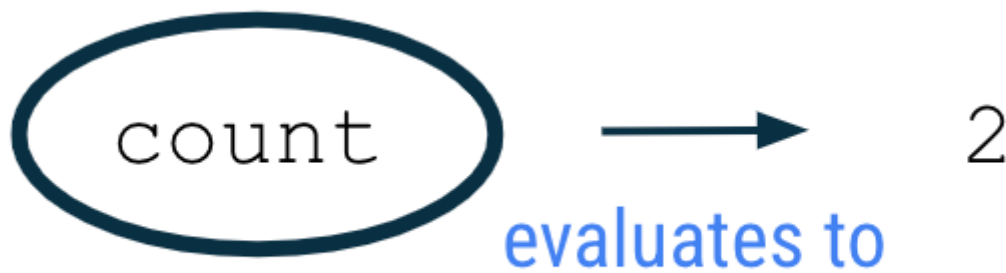
expression

count

value

2

Evaluar significa determinar el valor de una expresión. En este caso, la expresión se evalúa como `2`. El compilador evalúa expresiones en el código y usa esos valores cuando ejecuta las instrucciones en el programa.



Para observar este comportamiento en el Playground de Kotlin, ejecuta el programa en la siguiente sección.

Ejemplo

1. Abre el [Playground de Kotlin](#) en un navegador web.
2. Reemplaza el código existente en el Playground de Kotlin con el siguiente programa.

Este programa crea una variable llamada `count` con un valor inicial de `2` y la usa para imprimir el valor de la variable `count` en el resultado. No te preocupes si aún no comprendes todos los aspectos de la sintaxis del código. Se explicarán con más detalle en las próximas secciones.

```
fun main() {  
    val count: Int = 2  
    println(count)  
}
```

1. Ejecuta el programa. Debería mostrar el siguiente resultado:

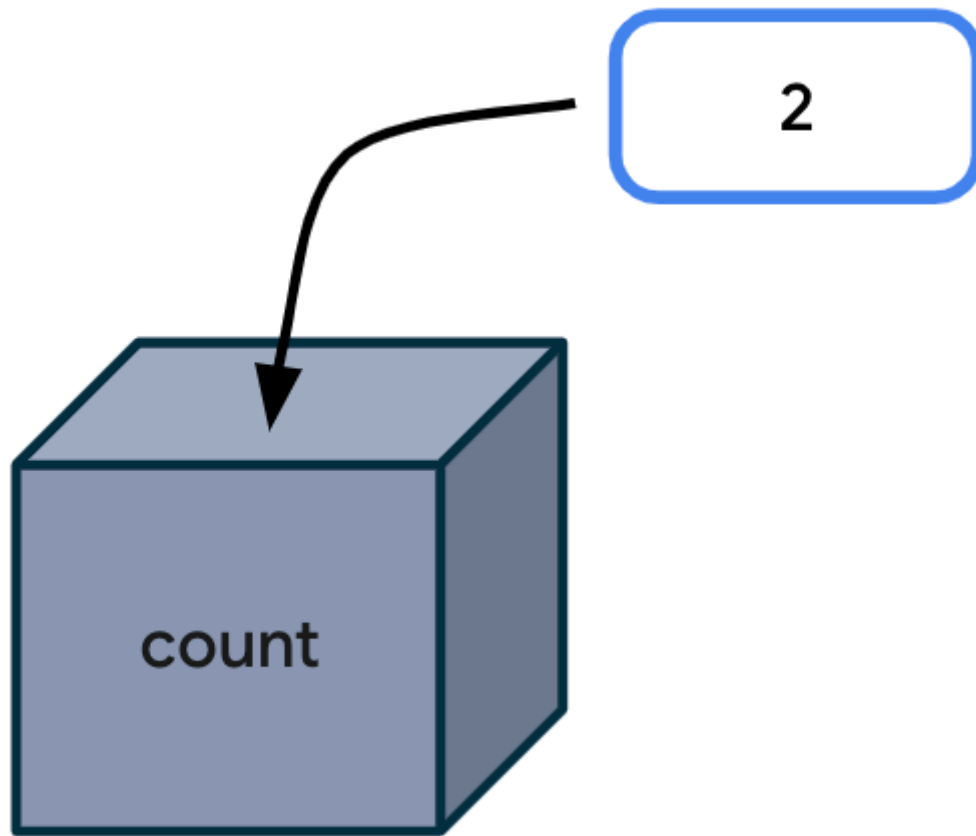
```
2
```

Declaración de variable

En el programa que ejecutaste, la segunda línea del código indicará lo siguiente:

```
val count: Int = 2
```

Con esta declaración, se crea una variable de número entero llamada `count` que contiene el número `2`.



Es posible que tardes un poco en familiarizarte con la lectura de la sintaxis (o el formato) para declarar una variable en Kotlin. En el siguiente diagrama, se muestra dónde se debe ubicar cada detalle de la variable, así como la ubicación de los espacios y símbolos.

`val` **name** `:` **data type** `=` **initial value**

En el contexto de la variable `count`, puedes ver que la declaración de variables comienza con la palabra `val`. El nombre de la variable es `count`. El tipo de datos es `Int` y el valor inicial es `2`.

name data type initial value
↓ ↓ ↙
`val count: Int = 2`

Cada parte de la declaración de variables se explica con más detalle a continuación.

Palabra clave para definir una variable nueva

Para definir una nueva variable, comienza con la palabra clave `val` de Kotlin (que significa valor). Luego, el compilador de Kotlin sabe que en esta declaración hay una declaración de variable.

Nombre de la variable

Del mismo modo en que nombras una función, también debes asignar un nombre a una variable. En la declaración de variables, el nombre de la variable sigue a la palabra clave `val`.

`val` **name** **:** **data type** **=** **initial value**

Puedes elegir el nombre de cualquier variable que desees, pero como práctica recomendada, evita usar [palabras clave](#) de Kotlin como nombre de variable.

Te recomendamos que elijas un nombre que describa los datos que contiene la variable para que sea más fácil de leer.

Los nombres de las variables deben seguir la convención de mayúsculas y minúsculas, como aprendiste con los nombres de las funciones. La primera palabra del nombre de variable se escribe en minúscula. Si el nombre contiene varias palabras, no se escriben espacios entre ellas y todas las demás palabras deben comenzar con mayúscula.

Ejemplos de nombres de variables:

- `numberOfEmails`
- `cityName`
- `bookPublicationDate`

Para el ejemplo de código que se mostró antes, `count` es el nombre de la variable.

```
val count: Int = 2
```

Tipo de datos variable

Después del nombre de la variable, agrega dos puntos, un espacio y, luego, el tipo de datos de la variable. Como se mencionó anteriormente, `String`, `Int`, `Double`, `Float`, y `Boolean` son algunos de los tipos básicos de datos de Kotlin. Aprenderás más tipos de datos más adelante en este curso. Recuerda escribir los tipos de datos exactamente como se muestran y comenzar cada uno con una letra mayúscula.


`val` **name** **:** **data type** **=** **initial value**

En el ejemplo de la variable `count`, `Int` es el tipo de datos de la variable.

```
val count: Int = 2
```

Operador de asignación

En la declaración de variables, el signo igual (`=`) sigue al tipo de datos. El signo igual se denomina *operador de asignación*. El operador de asignación asigna un valor a la variable. En otras palabras, el valor del lado derecho del signo igual se almacena en la variable del lado izquierdo del signo igual.



```
val count: Int = 2
```

Valor inicial variable

El valor de la variable son los datos reales que están almacenados en la variable.

```
val name : data type = initial value
```

Para el ejemplo de la variable `count`, el número `2` es el valor inicial de la variable.

```
val count: Int = 2
```

También puedes escuchar la frase: "la variable `count` se *inicializó* en `2`". Esto significa que `2` es el primer valor almacenado en la variable cuando se declara la variable.

El valor inicial será diferente según el tipo de datos declarado para la variable.

Consulta la siguiente tabla. Tal vez la reconozcas de un momento anterior del codelab. La tercera columna muestra valores de ejemplo que se pueden almacenar en una variable de cada tipo correspondiente. Estos valores se denominan *literales* porque son valores fijos o constantes (el valor es siempre el mismo). Por ejemplo, el número entero `32` siempre tendrá el valor de `32`. En cambio, una variable no es un valor literal porque su valor puede cambiar. Es posible que escuches nombrar a estos valores literales según su tipo: literal de string, literal de número entero, literal booleano, etc.

Tipo de datos de Kotlin	Qué tipo de datos puede contener	Ejemplos de valores literales
<code>String</code>	Texto	<code>"Add contact"</code> <code>"Search"</code> <code>"Sign in"</code>
<code>Int</code>	Número entero	<code>32</code> <code>1293490</code> <code>-59281</code>
<code>Double</code>	Número decimal	<code>2.0</code> <code>501.0292</code> <code>-31723.99999</code>
<code>Float</code>	Número decimal (que es menos preciso que un <code>double</code>). Tiene un <code>f</code> o <code>F</code> al final del número.	<code>5.0f</code> <code>-1630.209f</code> <code>1.2940278F</code>

Tipo de datos de Kotlin	Qué tipo de datos puede contener	Ejemplos de valores literales
<code>Boolean</code>	<code>true</code> o <code>false</code> . Usa este tipo de datos cuando solo haya dos valores posibles. Ten en cuenta que <code>true</code> y <code>false</code> son palabras clave en Kotlin.	<code>true</code> <code>false</code>

Es importante proporcionar un valor apropiado y válido según el tipo de datos de la variable. Por ejemplo, no puedes almacenar un literal de string como `"Hello"` dentro de una variable de tipo `Int` porque el compilador de Kotlin te mostrará un error.

Usa una variable

A continuación, se muestra el programa original que ejecutaste en el Playground de Kotlin. Hasta ahora, aprendiste que la segunda línea de código crea una nueva variable de número entero llamada `count` con un valor de `2`.

```
fun main() {
    val count: Int = 2
    println(count)
}
```

Ahora, veamos la tercera línea de código. Imprime la variable `count` en el resultado:

```
println(count)
```

Observa la palabra `count` no está entre comillas. Es un nombre de variable, no un literal de string. (Si fuera un literal de string, la palabra aparecería entre comillas). Cuando ejecutas el programa, el compilador de Kotlin evalúa la expresión dentro de los paréntesis, que es `count`, para la instrucción `println()`. Dado que la expresión se evalúa como `2`, se llama al método `println()` con `2` como la entrada: `println(2)`.

Por lo tanto, el resultado del programa es el siguiente:

```
2
```

El número por sí solo en el resultado no es muy útil. Sería más útil tener un mensaje más detallado en el resultado para explicar lo que representa el `2`.

Plantilla de strings

Un mensaje más útil para mostrar en el resultado es el siguiente:

```
You have 2 unread messages.
```

Sigue los pasos que aparecen a continuación para que el programa muestre un mensaje más útil.

1. Actualiza tu programa en el Playground de Kotlin con el siguiente código. Para la llamada `println()`, pasa un literal de string que contenga el nombre de variable `count`. Recuerda encerrar el texto entre comillas. Ten en cuenta que esto no te brindará los resultados que esperas. Solucionarás el problema en un paso posterior.

```
fun main() {  
    val count: Int = 2  
    println("You have count unread messages.")  
}
```

1. Ejecuta el programa. Debería mostrarse el siguiente resultado:

```
You have count unread messages.
```

Esa oración no tiene sentido. Querrás que se muestre el valor de la variable `count` en el mensaje, no el nombre de la variable.

1. Para corregir tu programa, agrega un signo de dólar `$` antes de la variable `count`: `"You have $count unread messages."` Esta es una *plantilla de strings* porque contiene una *expresión de plantillas*, que en este caso es `$count`. Una expresión de plantilla es una expresión que se evalúa como un valor y que luego se sustituye en la string. En este caso, la expresión de plantilla `$count` se evalúa como `2`, y el `2` se sustituye en la string en la que se encontró la expresión.

```
fun main() {  
    val count: Int = 2  
    println("You have $count unread messages.")  
}
```

1. Cuando ejecutas el programa, el resultado coincide con el objetivo deseado:

```
You have 2 unread messages.
```

Esa oración tiene mucho más sentido para el usuario.

1. Ahora, cambia el valor inicial de la variable `count` a un literal de número entero diferente. Por ejemplo, puedes elegir el número `10`. Deja el resto del código del programa sin modificar.

```
fun main() {  
    val count: Int = 10  
    println("You have $count unread messages.")  
}
```

1. Ejecuta el programa. Observa que el resultado cambia en consecuencia y que no necesitas cambiar la declaración `println()` en tu programa.

```
You have 10 unread messages.
```

Puedes ver qué tan útil puede ser una plantilla de strings. Solo escribiste la plantilla de strings una vez en tu código ("You have \$count unread messages."). Si cambias el valor inicial de la variable `count`, la sentencia `println()` seguirá funcionando. Ahora, tu código es más flexible.

Para destacar este punto, compara los dos programas que aparecen a continuación. El primer programa usa un literal de string, con la cantidad exacta de mensajes no leídos directamente en la string. Este programa solo funciona cuando el usuario tiene 10 mensajes no leídos.

```
fun main() {  
    println("You have 10 unread messages.")  
}
```

Si usas una variable y una plantilla de strings en el segundo programa, tu código podrá adaptarse a más situaciones. Este segundo programa es más flexible.

```
fun main() {  
    val count: Int = 10  
    println("You have $count unread messages.")  
}
```

Inferencia de tipo

A continuación, se muestra una sugerencia que te permite escribir menos código cuando declaras variables.

La inferencia de tipo es cuando el compilador de Kotlin puede inferir (o determinar) qué tipo de datos debe ser una variable, sin que el tipo se escriba de manera explícita en el código. Eso significa que puedes omitir el tipo de datos en una declaración de variable si proporcionas un valor inicial para ella. El compilador de Kotlin examina el tipo de datos del valor inicial y supone que quieres que la variable contenga datos de ese tipo.

A continuación, se muestra la sintaxis de una declaración de variable que usa inferencia de tipo:

`val` **name** = **initial value**

Volviendo al ejemplo de conteo, el programa contenía inicialmente esta línea de código:

```
val count: Int = 2
```

Sin embargo, esta línea de código también se puede escribir de la siguiente manera: Observa que se omiten el símbolo de dos puntos (`:`) y el tipo de datos `Int`. La sintaxis actualizada tiene menos palabras para escribir y logra el mismo resultado de la creación de una variable `Int` llamada `count` con un valor de `2`.

```
val count = 2
```

El compilador de Kotlin sabe que deseas almacenar `2` (un número entero) en la variable `count`, de modo que pueda inferir que la variable `count` es del tipo `Int`. Es práctico, ¿no? Este es un ejemplo de cómo escribir código de Kotlin es más conciso.

Nota: Si no proporcionas un valor inicial cuando declaras una variable, debes especificar el tipo.

En esta línea de código, no se proporciona un valor inicial, por lo que debes especificar el tipo de datos:

```
val count: Int
```

En esta línea de código, se proporciona un valor asignado para que puedas omitir el tipo de datos:

```
val count = 2
```

Aunque en este ejemplo solo se analiza una variable de tipo `Int`, el concepto de inferencia de tipos se aplica a todos los tipos de datos en Kotlin.

Operaciones matemáticas básicas con números enteros

¿Cuál es la diferencia entre una variable `Int` con el valor `2` y una variable `String` con el valor `"2"`? Cuando ambos se imprimen en el resultado, se ven iguales.

La ventaja de almacenar números enteros como `Int` (en lugar de `String`) es que puedes realizar operaciones matemáticas con variables `Int`, como la suma, la resta, la división y la multiplicación (consulta otras [operaciones](#)). Por ejemplo, se pueden sumar dos variables de número entero para obtener el resultado. Hay casos en los que es razonable tener números enteros almacenados como strings, pero el propósito de esta sección es mostrarte qué puedes hacer con las variables `Int`.

1. Regresa al Playground de Kotlin y quita todo el código del editor de código.
2. Crea un programa nuevo en el que se defina una variable de número entero para la cantidad de correos electrónicos no leídos de la carpeta de Recibidos y, luego, inicialízalo en un valor como `5`. Puedes elegir otro número si lo deseas. Define una segunda variable de número entero para la cantidad de correos electrónicos leídos en una carpeta de Recibidos. Inicialízalo a un valor como `100`. Puedes elegir otro número si lo deseas. Luego, imprime la cantidad total de mensajes en Recibidos, suma los dos números enteros.

```
fun main() {  
    val unreadCount = 5  
    val readCount = 100  
    println("You have ${unreadCount + readCount} total messages in your inbox.")  
}
```

1. Ejecuta el programa. Debería mostrar la cantidad total de mensajes en Recibidos:

```
You have 105 total messages in your inbox.
```

Para una plantilla de strings, aprendiste que puedes colocar el símbolo `$` antes de un solo nombre de variable. Sin embargo, si tienes una expresión más compleja, debes encerrar la expresión entre llaves con el símbolo `$` antes de las llaves: `${unreadCount + readCount}`. La expresión dentro de las llaves, `unreadCount + readCount`, se evalúa como `105`. Luego, se sustituye el valor `105` dentro del literal de string.

expression	value
<code>unreadCount + readCount</code>	<code>105</code>

Advertencia: Si olvidas las llaves de la expresión de la plantilla, obtendrás resultados inesperados. Puedes probar esto el Playground de Kotlin. Para ello, cambia la sentencia `println()` a `println("You have $unreadCount + readCount total messages in your inbox.")` y observa el resultado.

1. Para profundizar más este tema, crea variables con diferentes nombres y valores iniciales, y usa expresiones de plantillas para imprimir mensajes en el resultado.

Por ejemplo, modifica tu programa para imprimir lo siguiente:

```
100 photos
10 photos deleted
90 photos left
```

Esta es una forma de escribir el programa, aunque también hay otras formas correctas de escribirlo.

```
fun main() {
    val numberOfPhotos = 100
    val photosDeleted = 10
    println("$numberOfPhotos photos")
    println("$photosDeleted photos deleted")
    println("${numberOfPhotos - photosDeleted} photos left")
}
```

3. Actualiza variables

Cuando una app se ejecuta, es posible que sea necesario actualizar el valor de una variable. Por ejemplo, en una app de compras, a medida que el usuario agrega artículos al carrito, el total del carrito aumenta.

Simplificamos el caso de uso de esa compra en un programa simple. La lógica está escrita a continuación con lenguaje humano, no en Kotlin. Esta función se denomina *pseudocódigo* porque describe los puntos clave en los que se escribirá el código, pero no contiene todos los detalles del código.

Nota: El pseudocódigo no debe ser un código de trabajo que pueda compilarse, por eso se denomina pseudocódigo.

En la función principal de un programa, haz lo siguiente:

- Crea una variable de número entero `cartTotal` que comience en el valor `0`.

- El usuario agrega un suéter que cuesta USD 20 a su carrito de compras.
- Actualiza la variable `cartTotal` a `20`, que es el costo actual de los artículos de su carrito de compras.
- Imprime el costo total de los artículos de su carrito, que es la variable `cartTotal`, en el resultado.

Para simplificar aún más el código, no es necesario que escribas el código cuando el usuario agrega artículos al carrito de compras. (Todavía no aprendiste cómo un programa puede responder a las entradas del usuario. Eso se abordará en una unidad posterior). Por lo tanto, enfócate en las partes en las que creas, actualizas y también imprimes la variable `cartTotal`.

1. Reemplaza el código existente en el Playground de Kotlin con el programa que aparece a continuación. En la línea 2 del programa, inicializa la variable `cartTotal` en el valor `0`. Ya que proporcionas un valor inicial, no es necesario especificar el tipo de datos `Int` debido a la inferencia de tipo. En la línea 3 del programa, intenta actualizar la variable `cartTotal` a `20` con el operador de asignación (`=`). En la línea 4 del programa, imprime la variable `cartTotal` con una plantilla de strings.

```
fun main() {
    val cartTotal = 0
    cartTotal = 20
    println("Total: $cartTotal")
}
```

1. Ejecuta el programa. Se mostrará un error de compilación.
2. Observa que el error indica que no se puede reasignar el `val`. El error se encuentra en la tercera línea del programa, la que intenta cambiar el valor de `cartTotal` variable a `20`. `val cartTotal` no se puede reasignar a otro valor (`20`) después de que se le asigne un valor inicial (`0`).

`val cannot be reassigned`

Si necesitas actualizar el valor de una variable, declara la variable con la palabra clave de Kotlin `var`, en lugar de `val`.

- Palabra clave `val`: Úsala cuando esperes que el valor de la variable no cambie.
- Palabra clave `var`: Úsala cuando esperes que el valor de la variable pueda cambiar.

Con `val`, la variable es de *solo lectura*, lo que significa que solo puedes leer el valor de la variable o acceder a él. Una vez que se establece el valor, no puedes editarlo ni modificarlo. Con `var`, la variable es *mutable*, lo que significa que el valor se puede cambiar o modificar. El valor se puede mutar.

Para recordar la diferencia, piensa en `val` como un *valor* fijo y en `var` como una *variable*. En Kotlin, se recomienda usar la palabra clave `val` en lugar de la palabra clave `var` cuando sea posible.

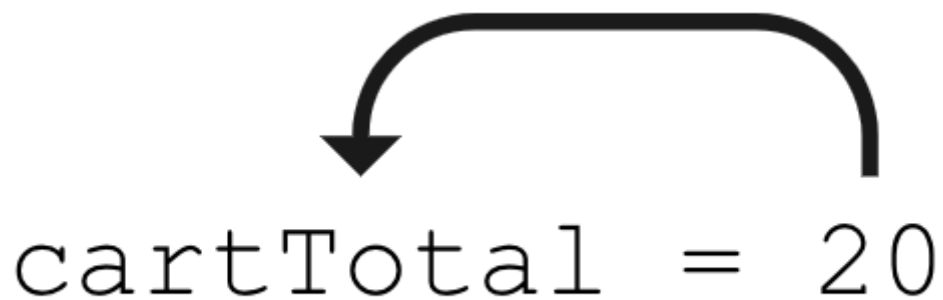
1. Actualiza la declaración de variable de `cartTotal` en la línea 2 del programa a fin de usar `var` en lugar de `val`. El código debería verse así:

```
fun main() {  
    var cartTotal = 0  
    cartTotal = 20  
    println("Total: $cartTotal")  
}
```

1. Observa la sintaxis del código en la línea 3 del programa que actualiza la variable.

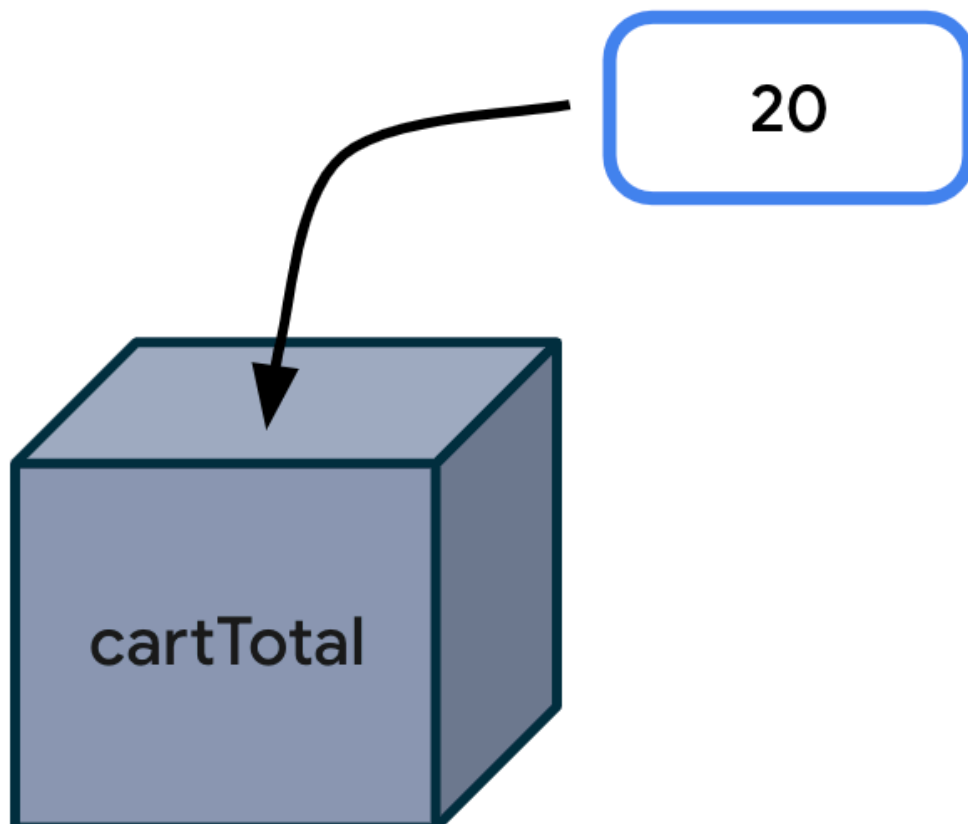
```
cartTotal = 20
```

Usa el operador de asignación (=) para asignar un valor nuevo (20) a la variable existente (cartTotal). No necesitas utilizar la palabra clave `var` nuevamente porque la variable ya está definida.



cartTotal = 20

Con la analogía de la caja, imagina que el valor 20 se almacena en la caja etiquetada `cartTotal`.



Aquí se muestra un diagrama de la sintaxis general para actualizar una variable, que ya se declaró en una línea de código anterior. Inicia la sentencia con el nombre de la variable que deseas actualizar. Agrega un espacio y el signo igual, seguido de otro espacio. Luego, escribe el valor actualizado de la variable.



1. Ejecuta el programa. El código debería compilarse correctamente. Se debería imprimir este resultado:

```
Total: 20
```

1. Para ver cómo cambia el valor de la variable mientras se ejecuta el programa, imprime la variable `cartTotal` en el resultado después de declarar la variable inicialmente. Consulta los cambios en el código a continuación. La línea 3 tiene una declaración `println()` nueva. También se agrega una línea en blanco en la línea 4 del código. Las líneas en blanco no afectan la manera en que el compilador comprende el código. Agrega una línea en blanco donde sea más fácil leer tu código porque separa los bloques de código relacionados.

```
fun main() {  
    var cartTotal = 0  
    println("Total: $cartTotal")  
  
    cartTotal = 20  
    println("Total: $cartTotal")  
}
```

1. Vuelve a ejecutar el programa. El resultado debería ser el siguiente:

```
Total: 0  
Total: 20
```

Podrás ver que, en un principio, el total del carrito de compras es de `0`. Luego, se actualiza a `20`. Actualizaste correctamente una variable. Esto fue posible porque cambiaste `cartTotal` de una variable de solo lectura (con `val`) a una variable mutable (con `var`).

Recuerda que solo debes usar `var` para declarar una variable si esperas que cambie el valor. De lo contrario, deberías usar `val` de forma predeterminada para declarar una variable. Esta práctica hace que tu código sea más seguro. El uso de `val` garantiza que las variables no se actualicen en tu programa si no esperas que lo hagan. Una vez que se asigna un valor a `val`, siempre se mantiene ese valor.

Nota: Si estás familiarizado con otros lenguajes de programación, declarar un `val` es como declarar un valor constante porque es una variable de solo lectura. Hay convenciones adicionales que se deben seguir cuando se declaran constantes en Kotlin, lo cual es más avanzado para este codelab, pero puedes encontrarlas en la sección [Constantes](#) de la guía de estilo.

Operadores de incremento y disminución

Ahora, sabes que se debe declarar una variable como `var` para actualizar su valor. Aplica este conocimiento al ejemplo de mensaje de correo electrónico que debería aparecer a continuación.

1. Reemplaza el código en el Playground de Kotlin con este código:

```
fun main() {  
    val count: Int = 10  
    println("You have $count unread messages.")  
}
```

1. Ejecuta el programa. Se debería imprimir lo siguiente:

```
You have 10 unread messages.
```

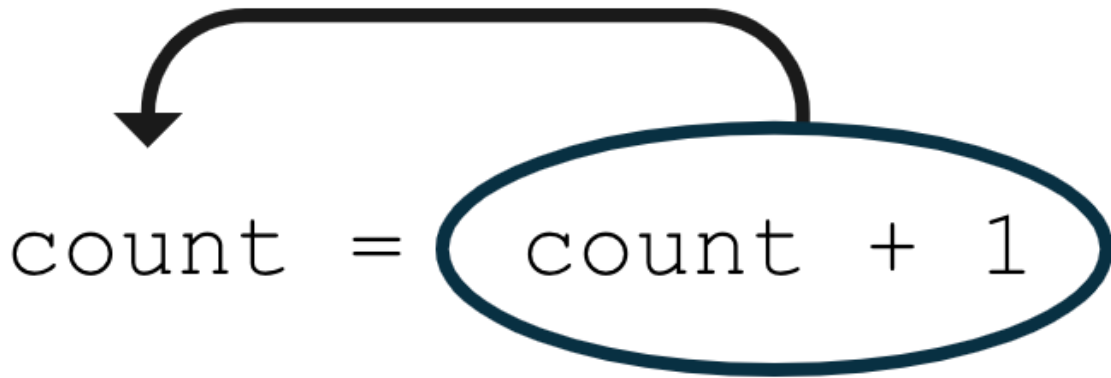
1. Reemplaza la palabra clave `val` por la palabra clave `var` a fin de hacer que la variable `count` sea una variable mutable. No debería haber cambios en el resultado cuando ejecutes el programa.

```
fun main() {  
    var count: Int = 10  
    println("You have $count unread messages.")  
}
```

1. Sin embargo, ahora puedes actualizar `count` a un valor diferente. Por ejemplo, cuando llega un correo electrónico nuevo a la carpeta Recibidos del usuario, puedes aumentar `count` una vez. (No es necesario que escribas el código cuando llegue un correo electrónico. Obtener datos de Internet es un tema más avanzado que se detallará en una unidad posterior). Por ahora, enfócate en la variable `count` que aumenta a `1` con esta línea de código:

```
count = count + 1
```

La expresión a la derecha del signo igual es `count + 1` y se evalúa como `11`. Esto se debe a que el valor actual de `count` es `10` (que se encuentra en la línea 2 del programa) y `10 + 1` es igual a `11`. Luego, con el operador de asignaciones, se asigna o se almacena el valor `11` en la variable `count`.



count = count + 1

Agrega esta línea de código a tu programa en la parte inferior de la función `main()`. El código debería verse como este fragmento:

```
fun main() {  
    var count = 10  
    println("You have $count unread messages.")  
    count = count + 1  
}
```

Si ejecutas el programa ahora, el resultado será el mismo que antes porque no agregaste ningún código para usar la variable `count` después de actualizarla.

1. Agrega otra sentencia de impresión que imprima la cantidad de mensajes no leídos después de que se haya actualizado la variable.

```
fun main() {  
    var count = 10  
    println("You have $count unread messages.")  
    count = count + 1  
    println("You have $count unread messages.")  
}
```

1. Ejecuta el programa. El segundo mensaje debería mostrar el valor `count` actualizado de `11` mensajes.

```
You have 10 unread messages.  
You have 11 unread messages.
```

1. En resumen, si deseas aumentar una variable en `1`, puedes usar el *operador de incremento* (`++`), que se compone de dos símbolos de suma. Si usas estos símbolos inmediatamente después de un nombre de variable, le indicas al compilador que deseas agregar 1 al valor actual de la variable y, luego, almacenar el valor nuevo en la variable. Las dos líneas de código siguientes son equivalentes, pero el uso del operador de incremento `++` implica menos escritura.

```
count = count + 1  
count++
```

Realiza esta modificación en tu código y, luego, ejecuta el programa. No debe haber espacios entre el nombre de la variable y el operador de incremento.

```
fun main() {  
    var count = 10  
    println("You have $count unread messages.")  
    count++  
    println("You have $count unread messages.")  
}
```

1. Ejecuta el programa. El resultado es el mismo, pero ahora aprendiste sobre un nuevo operador.

```
You have 10 unread messages.  
You have 11 unread messages.
```

1. Ahora, modifica la línea 4 de tu programa para usar el *operador de disminución* (`--`) después del nombre de la variable `count`. El operador de disminución está compuesto por dos símbolos menos. Cuando colocas el operador de disminución después del nombre de la variable, le indicas al compilador que deseas disminuir el valor de la variable en `1` y almacenar el valor nuevo en ella.

```
fun main() {  
    var count = 10  
    println("You have $count unread messages.")  
    count--  
    println("You have $count unread messages.")  
}
```

1. Ejecuta el programa. Se debería imprimir este resultado:

```
You have 10 unread messages.  
You have 9 unread messages.
```

En esta sección, aprendiste a actualizar una variable mutable mediante el operador de incremento (`++`) y el operador de disminución (`--`). Más específicamente, `count++` es igual a `count = count + 1` y `count--` es igual a `count = count - 1`.

4. Explora otros tipos de datos

Anteriormente en el codelab, aprendiste algunos tipos de datos básicos comunes: `String`, `Int`, `Double` y `Boolean`. Acabas de usar el tipo de datos `Int`, ahora explorarás otros tipos de datos.

Tipo de datos de Kotlin	Qué tipo de datos puede contener
<code>String</code>	Texto
<code>Int</code>	Número entero
<code>Double</code>	Número decimal

Tipo de datos de Kotlin	Qué tipo de datos puede contener
<code>Boolean</code>	<code>true</code> o <code>false</code> (solo dos valores posibles)

Prueba estos programas en el Playground de Kotlin para ver cuál es el resultado.

Double

Cuando necesites una variable con un valor decimal, usa una variable `Double`. Para obtener información sobre su rango válido, consulta [esta tabla](#) y observa las cifras decimales que puede almacenar, por ejemplo.

Nota: El nombre del tipo de datos `Double` proviene del tipo de datos que tiene una precisión doble en comparación con el tipo de datos `Float`, que tiene una precisión única. La precisión es la cantidad de dígitos decimales que puede contener. Por lo tanto, una variable `Double` puede almacenar un valor más preciso. [En esta tabla](#), se muestran más detalles sobre las diferencias específicas entre los tipos `Double` y `Float`, si te interesa. Esta sección del codelab se enfoca en el uso de `Double` para trabajar con números decimales.

Imagina que estás navegando a un destino y tu viaje se divide en tres partes distintas porque debes hacer paradas durante el trayecto. Este programa muestra la distancia total que queda para llegar a tu destino.

1. Ingresa este código en el Playground de Kotlin. ¿Puedes entender lo que sucede en cada línea de código?


```
fun main() {  
    val trip1: Double = 3.20  
    val trip2: Double = 4.10  
    val trip3: Double = 1.72  
    val totalTripLength: Double = 0.0  
    println("$totalTripLength miles left to destination")  
}
```

Se declaran tres variables, llamadas `trip1`, `trip2` y `trip3`, para representar la distancia de cada parte del viaje. Todas son variables `Double` porque almacenan valores decimales. Usa `val` para declarar cada variable porque sus valores no cambian en el transcurso del programa. El programa también crea una cuarta variable llamada `totalTripLength`, que actualmente se inicializa en `0.0`. En la última línea del programa, se imprime un mensaje con el valor de `totalTripLength`.

1. Corrige el código para que la variable `totalTripLength` sea la suma de las tres duraciones de viaje.

```
val totalTripLength: Double = trip1 + trip2 + trip3
```

La expresión de la derecha del signo igual se evalúa como `9.02` porque `3.20 + 4.10 + 1.72` es igual a `9.02`. El valor de `9.02` se almacena en la variable `totalTripLength`.



```
val totalTripLength: Double = trip1 + trip2 + trip3
```

Todo tu programa debería verse como el siguiente código:

```
fun main() {  
    val trip1: Double = 3.20  
    val trip2: Double = 4.10  
    val trip3: Double = 1.72  
    val totalTripLength: Double = trip1 + trip2 + trip3  
    println("$totalTripLength miles left to destination")  
}
```

1. Ejecuta el programa. Se debería imprimir lo siguiente:

```
9.02 miles left to destination
```

1. Corrige tu código para quitar el tipo de datos `Double` innecesario de las declaraciones de variables debido a la inferencia de tipo. El compilador de Kotlin puede inferir que estas variables son tipos de datos `Double` según los números decimales proporcionados como valores iniciales.

```
fun main() {  
    val trip1 = 3.20  
    val trip2 = 4.10  
    val trip3 = 1.72  
    val totalTripLength = trip1 + trip2 + trip3  
    println("$totalTripLength miles left to destination")  
}
```

1. Vuelve a ejecutar el código para asegurarte de que siga compilando. El resultado debería ser el mismo, pero ahora tu código es más simple.

String

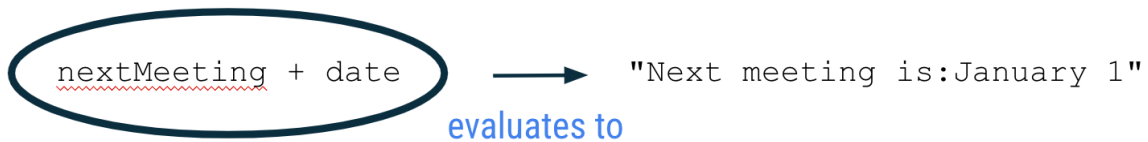
Cuando necesites una variable que pueda almacenar texto, usa una variable `String`. Recuerda usar comillas alrededor de los valores literales de string, como `"Hello kotlin"`, mientras que los valores literales `Int` y `Double` no tienen comillas.

1. Copia y pega este programa en el Playground de Kotlin.

```
fun main() {  
    val nextMeeting = "Next meeting is:"  
    val date = "January 1"  
    val reminder = nextMeeting + date  
    println(reminder)  
}
```

Observa que hay dos variables `String` declaradas, una variable `nextMeeting` y una variable `date`. Luego, se declara una tercera variable `String` llamada `reminder`, que se establece igual a la variable `nextMeeting` más la variable `date`.

Con el símbolo `+`, puedes agregar dos strings juntas, que se llama *concatenación*. Las dos strings se combinan, una tras otra. El resultado de la expresión, `nextMeeting + date`, es `"Next meeting is:January 1"`, como se muestra en el siguiente diagrama.



Luego, el valor `"Next meeting is:January 1"` se almacena en la variable `reminder` con el operador de asignación en la línea 4 del programa.

1. Ejecuta el programa. Se debería imprimir lo siguiente:

```
Next meeting is:January 1
```

Cuando concatenas dos strings, no se agregan espacios entre ellas. Si quieres un espacio después de los dos puntos en la string resultante, debes agregar el espacio en una string o en la otra.

1. Actualiza la variable `nextMeeting` para que tenga un espacio adicional al final de la string antes de las comillas. (De manera alternativa, podrías haber agregado un espacio adicional al principio de la variable `date`). Tu programa debería verse de la siguiente manera:

```
fun main() {  
    val nextMeeting = "Next meeting is: "  
    val date = "January 1"  
    val reminder = nextMeeting + date  
    println(reminder)  
}
```

5. Convenciones de codificación

En el codelab anterior, se presentó la guía de estilo de Kotlin para escribir código de Android de manera coherente según lo recomendado por Google y seguido por otros desarrolladores profesionales.

Estas son otras convenciones de formato y codificación que puedes seguir en función de los nuevos temas que aprendiste:

- Los nombres de las variables deben seguir la convención de mayúsculas y minúsculas, y comenzar con una letra minúscula.
- En una declaración de variable, debes especificar un espacio después de los dos puntos cuando se especifica el tipo de datos.

space



```
val discount: Double = .20
```

- Debe haber un espacio antes y después de un operador como la asignación (=), la suma (+), la resta (-), la multiplicación (*) la división (/) y muchos más.

space



```
var pet = "bird"
```

space



```
val sum = 1 + 2
```

- A medida que escribes programas más complejos, se recomienda usar un límite de [100 caracteres por línea](#). De esta manera, te aseguras de poder leer todo el código de un programa fácilmente en la pantalla de la computadora, sin necesidad de desplazarse horizontalmente cuando lees código.

6. Comenta en tu código

Durante la codificación, otra práctica recomendada es agregar *comentarios* que describan para qué sirve el código. Los comentarios pueden ayudar a las personas que leen tu código a seguirlo con más facilidad. Dos símbolos de barra diagonal, o `//`, indican que el texto que sigue en el resto de la línea se considera un comentario, por lo que no se interpreta como código. Es común agregar un espacio después de los dos símbolos de barra diagonal.

```
// This is a comment.
```

Un comentario también puede comenzar en el medio de una línea de código. En este ejemplo, `height = 1` es una instrucción de codificación normal. Todo lo que aparece después de `//` o `Assume the height is 1 to start with`, se interpreta como un comentario y no se considera parte del código.

```
height = 1 // Assume the height is 1 to start with.
```

Si quieres describir el código con más detalle junto con un comentario largo que supere los 100 caracteres en una línea, usa un comentario de varias líneas. Inicia el comentario de varias líneas con una barra diagonal (`/`) y un asterisco (`*`) como `/*`. Agrega un asterisco al principio de cada nueva línea del comentario. Por último, termina el comentario con un asterisco y un símbolo de barra diagonal `*/`.

```
/*
 * This is a very long comment that can
 * take up multiple lines.
 */
```

Este programa contiene comentarios de una o varias líneas que describen lo que sucede:

```
/**
 * This program displays the number of messages
 * in the user's inbox.
 */
fun main() {
    // Create a variable for the number of unread messages.
    var count = 10
    println("You have $count unread messages.")

    // Decrease the number of messages by 1.
    count--
    println("You have $count unread messages.")
}
```

Como se mencionó anteriormente, puedes agregar líneas vacías en blanco a tu código para agrupar declaraciones relacionadas y facilitar la lectura del código.

1. Agrega algunos comentarios a un fragmento de código anterior que hayas usado.
2. Ejecuta el programa para asegurarte de que no haya cambiado el comportamiento porque los comentarios no deberían afectar el resultado.

7. Cómo definir una función y cómo llamarla

Antes de explorar las funciones en profundidad, revisemos la terminología básica.

- Declarar (o definir) una función usa la palabra clave `fun` e incluye código dentro de las llaves que contiene las instrucciones necesarias para ejecutar una tarea.
- Llamar a una función causa que se ejecute todo el código dentro esta.

Hasta ahora, escribiste todo el código en la función `main()`. En realidad, en ninguna parte del código se llama a la función `main()`; el compilador de Kotlin la usa como punto de partida. La función `main()` está diseñada para incluir solo otro código que desees ejecutar, como llamadas a la función `println()`.

La función `println()` es parte del lenguaje Kotlin. Sin embargo, puedes definir tus propias funciones. De esta manera, se permite reutilizar tu código si necesitas llamarlo más de una vez. Considera el siguiente programa como ejemplo.

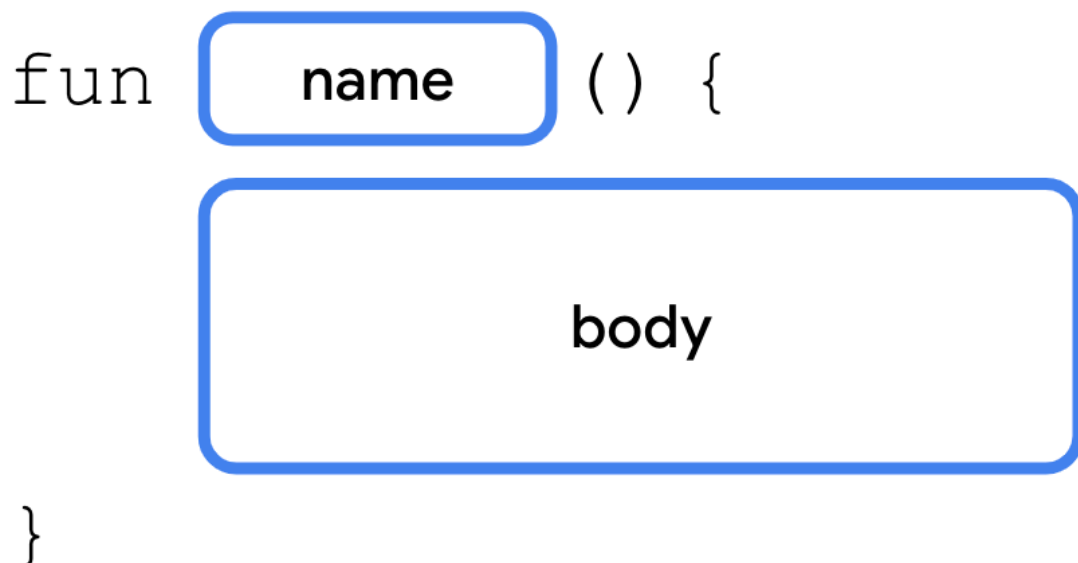
```
fun main() {  
    println("Happy Birthday, Rover!")  
    println("You are now 5 years old!")  
}
```

La función `main()` consta de dos declaraciones `println()`: una para desearle un feliz cumpleaños a Rover y otra que indica su edad.

Si bien Kotlin te permite colocar todo el código en la función `main()`, es posible que no siempre desees hacerlo. Por ejemplo, si también quieres que tu programa contenga un saludo de Año Nuevo, la función `main` también deberá incluir esas llamadas a `println()`. O bien, es posible que quieras saludar a Rover varias veces. Simplemente, puedes copiar y pegar el código o crear una función diferente para el saludo de cumpleaños. Harás lo último. Crear funciones diferentes para tareas específicas tiene una serie de beneficios.

- Código reutilizable: En lugar de copiar y pegar el código que necesitas usar más de una vez, simplemente puedes llamar a una función cuando sea necesario.
- Legibilidad: Garantizar que las funciones realicen una única tarea específica permite que otros desarrolladores y compañeros de equipo, así como tú mismo en el futuro, sepan con exactitud qué hace un fragmento de código.

La sintaxis para definir una función se muestra en el siguiente diagrama.



La definición de una función comienza con la palabra clave `fun`, seguida del nombre de la función, un conjunto de paréntesis de apertura y cierre, y un conjunto de llaves de apertura y cierre. Entre llaves, se muestra el código que se ejecutará cuando se llame a la función.

Crearás una función nueva para quitar las dos declaraciones `println()` de la función `main()`.

8. Cómo devolver un valor de una función

En apps más complejas, las funciones no se limitan a imprimir texto.

Las funciones de Kotlin pueden generar datos que se denominan *valor de muestra* que se almacenan en una variable que puedes usar en cualquier otra parte del código.

Cuando defines una función, puedes especificar el tipo de datos del valor que deseas que se muestre. Para especificar el tipo de datos que se devuelve, debes colocar dos puntos (:) después de los paréntesis, un solo espacio en blanco y, luego, el nombre del tipo (`Int`, `String`, etc.). A continuación, se coloca un solo espacio entre el tipo de datos que se devuelve y la llave de apertura. Dentro del cuerpo de la función, después de todas las sentencias, usa una sentencia `return` para especificar el valor que desea que muestre la función. Una sentencia `return` consiste en la palabra clave `return`, seguida del valor, como una variable, que quieres que muestre la función como resultado.

La sintaxis para declarar una función con un tipo de datos que se devuelve es la siguiente.

```
fun name ( ) : return type {  
    body  
    return statement  
}
```

El tipo `Unit`

De forma predeterminada, si no se especifica un tipo de datos que se muestra, el predeterminado es `Unit`. `Unit` significa que la función no muestra ningún valor. `Unit` es equivalente a los tipos nulos de datos que se muestran en otros lenguajes (`void` en Java y C, tupla `void`/vacía `()` en Swift; `None` en Python, etc.). Cualquier función que no muestre un valor muestra `Unit` de forma implícita. Si deseas observar este comportamiento, modifica tu código para que muestre `Unit`.

1. En la declaración de la función para `birthdayGreeting()`, agrega dos puntos después del paréntesis de cierre y especifica el tipo de datos que se muestra como `Unit`.

```
fun main() {
    birthdayGreeting()
}

fun birthdayGreeting(): Unit {
    println("Happy Birthday, Rover!")
    println("You are now 5 years old!")
}
```

1. Ejecuta el código y observa que todo continúa funcionando.

```
Happy Birthday, Rover!
You are now 5 years old!
```

Es opcional especificar el tipo de datos que se muestra de `Unit` en Kotlin. Para las funciones que no muestran nada o que muestran `Unit`, no necesitas una sentencia `return`.

Nota: Volverás a ver el tipo `Unit` cuando aprendas sobre una función de Kotlin que se denomina lambdas en un codelab posterior.

Cómo mostrar `String` de `birthdayGreeting()`

A fin de demostrar cómo una función puede mostrar un valor, deberás modificar la función `birthdayGreeting()` para que muestre una string, en lugar de simplemente imprimir el resultado.

1. Reemplaza el tipo de datos que se muestra `Unit` por `String`.

```
fun birthdayGreeting(): String {
    println("Happy Birthday, Rover!")
    println("You are now 5 years old!")
}
```

1. Ejecuta tu código. Se mostrará un error. Si declaras un tipo de datos que se muestra para una función (p. ej., `String`), esa función debe incluir una sentencia `return`.

A 'return' expression required in a function with a block body ('{...}')

1. Solo puedes mostrar una cadena de una función, no dos. Reemplaza las sentencias `println()` por dos variables, `nameGreeting` y `ageGreeting`, con la palabra clave `val`. Como quitaste las llamadas a `println()` de `birthdayGreeting()`, llamar a `birthdayGreeting()` no imprimirá nada.

```
fun birthdayGreeting(): String {
    val nameGreeting = "Happy Birthday, Rover!"
    val ageGreeting = "You are now 5 years old!"
}
```

1. Con la string que le da formato a la sintaxis que aprendiste en un codelab anterior, agrega una sentencia `return` para mostrar una string de la función que incluye ambos saludos.

Para dar formato a los saludos en una línea separada, también debes usar el carácter de escape `\n`. Es similar al carácter de escape `\` que aprendiste en un codelab anterior. El carácter `\n` se reemplaza por una línea nueva, de modo que los dos saludos estén en una línea separada.

```
fun birthdayGreeting(): String {  
    val nameGreeting = "Happy Birthday, Rover!"  
    val ageGreeting = "You are now 5 years old!"  
    return "$nameGreeting\n$ageGreeting"  
}
```

1. En `main()`, dado que `birthdayGreeting()` muestra un valor, puedes almacenar el resultado en una variable de string. Declara una variable `greeting` mediante `val` para almacenar el resultado de la llamada a `birthdayGreeting()`.

```
fun main() {  
    val greeting = birthdayGreeting()  
}
```

1. En `main()`, llama a `println()` para imprimir la string `greeting`. La función `main()` ahora debería verse de la siguiente manera:

```
fun main() {  
    val greeting = birthdayGreeting()  
    println(greeting)  
}
```

1. Ejecuta tu código y, luego, observa que el resultado sea el mismo que antes: Mostrar un valor te permite almacenar el resultado en una variable, pero ¿qué crees que sucede si llamas a la función `birthdayGreeting()` dentro de la función `println()`?

```
Happy Birthday, Rover!  
You are now 5 years old!
```

1. Quita la variable y, luego, pasa el resultado de la llamada a la función `birthdayGreeting()` a la función `println()`:

```
fun main() {  
    println(birthdayGreeting())  
}
```

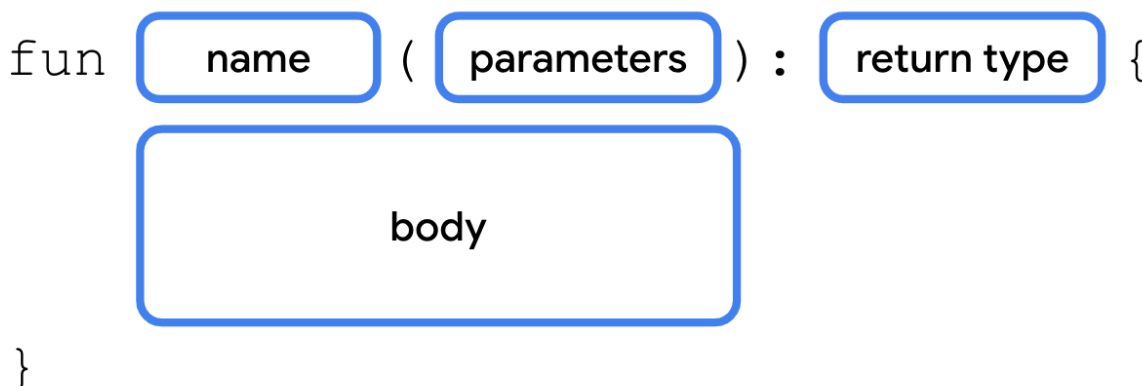
1. Ejecuta tu código y observa el resultado. El valor de muestra de la llamada a `birthdayGreeting()` se pasa directamente a `println()`.

```
Happy Birthday, Rover!  
You are now 5 years old!
```

9. Cómo agregar un parámetro a la función birthdayGreeting()

Como viste, cuando llamas a la función `println()`, puedes incluir una string entre paréntesis o *pasar un valor* a la función. Puedes hacer lo mismo con la función `birthdayGreeting()`. Sin embargo, primero debes agregar un *parámetro* a `birthdayGreeting()`.

Un parámetro especifica el nombre de una variable y un tipo de datos que puedes pasar a una función como datos para acceder dentro de esta. Los parámetros se declaran entre paréntesis después del nombre de la función.



Cada parámetro consiste en un nombre de variable y un tipo de datos, separados por dos puntos y un espacio. Si hay varios parámetros, se separan con una coma.

Por el momento, la función `birthdayGreeting()` solo se puede usar para saludar a Rover. Agregarás un parámetro a la función `birthdayGreeting()` para que puedas saludar con cualquier nombre que pases a la función.

1. Dentro de los paréntesis de la función `birthdayGreeting()`, agrega un parámetro `name` de tipo `String` con la sintaxis `name: String`.

```
fun birthdayGreeting(name: String): String {  
    val nameGreeting = "Happy Birthday, Rover!"  
    val ageGreeting = "You are now 5 years old!"  
    return "$nameGreeting\n$ageGreeting"  
}
```

El parámetro definido en el paso anterior funciona como una variable declarada con la palabra clave `val`. Su valor se puede usar en cualquier parte de la función `birthdayGreeting()`. En un codelab anterior, aprendiste cómo insertar el valor de una variable en una string.

1. Reemplaza `Rover` en la string `nameGreeting` por el símbolo `$` seguido del parámetro `name`.

```
fun birthdayGreeting(name: String): String {  
    val nameGreeting = "Happy Birthday, $name!"  
    val ageGreeting = "You are now 5 years old!"  
    return "$nameGreeting\n$ageGreeting"  
}
```

1. Ejecuta tu código y observa el error. Ahora que declaraste el parámetro `name`, debes pasar un `String` cuando llames a `birthdayGreeting()`. Cuando llamas a una función que toma un parámetro, debes pasar un argumento a la función. El argumento es el valor que pasas, como `"Rover"`.

```
No value passed for parameter 'name'
```

1. Pasa `"Rover"` a la llamada `birthdayGreeting()` en `main()`.

```
fun main() {  
    println(birthdayGreeting("Rover"))  
}
```

1. Ejecuta tu código y observa el resultado. El nombre Rover proviene del parámetro `name`.

```
Happy Birthday, Rover!  
You are now 5 years old!
```

1. Como `birthdayGreeting()` toma un parámetro, puedes llamarlo con un nombre que no sea Rover. Agrega otra llamada a `birthdayGreeting()` dentro de la llamada a `println()` y pasa el argumento `"Rex"`.

```
println(birthdayGreeting("Rover"))  
println(birthdayGreeting("Rex"))
```

1. Vuelve a ejecutar el código y, luego, observa que el resultado difiere en función del argumento que se pasó a `birthdayGreeting()`.

```
Happy Birthday, Rover!  
You are now 5 years old!  
Happy Birthday, Rex!  
You are now 5 years old!
```

Nota: Si bien con frecuencia los usos de forma indistinta, un parámetro y un argumento no son lo mismo. Cuando se define una función, se definen los parámetros que se le pasarán cuando se la llame. Cuando llamas a una función, debes pasar los argumentos para los parámetros. Los parámetros son las variables a las que puede acceder la función, como una variable `name`, mientras que los argumentos son los valores reales que pasas, como la string `"Rover"`.

Advertencia: A diferencia de algunos lenguajes, como Java, en los que una función puede cambiar el valor que se pasa a un parámetro, los parámetros en Kotlin son inmutables. No puedes reasignar el valor de un parámetro desde el cuerpo de la función.

10. Funciones con varios parámetros

Anteriormente, agregaste un parámetro para cambiar el saludo según el nombre. Sin embargo, también puedes definir más de un parámetro para una función, incluso parámetros de diferentes tipos de datos. En esta sección, modificarás el saludo de modo que también cambie en función de la edad del perro.

`fun` **Function name** (**First parameter** , **Second parameter** , . . .)

Las definiciones de los parámetros están separadas con comas. De manera similar, cuando llamas a una función con varios parámetros, también separas los argumentos que se pasaron con comas. Veamos esto en acción.

1. Después del parámetro `name`, agrega un parámetro `age` de tipo `Int` a la función `birthdayGreeting()`. La declaración de la función nueva debe tener los dos parámetros, `name` y `age`, separados con comas:

```
fun birthdayGreeting(name: String, age: Int): String {  
    val nameGreeting = "Happy Birthday, $name!"  
    val ageGreeting = "You are now 5 years old!"  
    return "$nameGreeting\n$ageGreeting"  
}
```

1. La nueva string de saludo debe usar el parámetro `age`. Actualiza la función `birthdayGreeting()` para usar el valor del parámetro `age` en la string `ageGreeting`.

```
fun birthdayGreeting(name: String, age: Int): String {  
    val nameGreeting = "Happy Birthday, $name!"  
    val ageGreeting = "You are now $age years old!"  
    return "$nameGreeting\n$ageGreeting"  
}
```

1. Ejecuta la función y, luego, observa los errores en el resultado:

```
No value passed for parameter 'age'  
No value passed for parameter 'age'
```

1. Modifica las dos llamadas a la función `birthdayGreeting()` en `main()` a fin de pasar una edad diferente para cada perro. Pasa `5` para la edad de Rover y `2` para la edad de Rex.

```
fun main() {  
    println(birthdayGreeting("Rover", 5))  
    println(birthdayGreeting("Rex", 2))  
}
```

1. Ejecuta tu código. Ahora que pasaste los valores para ambos parámetros, el resultado debería reflejar el nombre y la edad de cada perro cuando llames a la función.

```
Happy Birthday, Rover!  
You are now 5 years old!  
Happy Birthday, Rex!  
You are now 2 years old!
```

Firma de la función

Hasta ahora, viste cómo definir el nombre de la función, las entradas (parámetros) y los resultados. El nombre de la función y sus entradas (parámetros) se conocen colectivamente como *firma de la función*. La firma de la función consta de todo lo que está antes del tipo de datos que se muestra y se indica en el siguiente fragmento de código.

```
fun birthdayGreeting(name: String, age: Int)
```

A veces, los parámetros separados con comas se denominan lista de parámetros.

Con frecuencia, verás estos términos en la documentación para código que escriben otros desarrolladores. La firma de la función te indica el nombre de la función y los tipos de datos que se pueden pasar.

Ahora aprendiste un montón sobre la sintaxis nueva para definir funciones. Consulta el siguiente diagrama para obtener un resumen de la sintaxis de las funciones.

```
fun birthdayGreeting(name: String, age: Int): String {  
    val nameGreeting = "Happy Birthday, $name!"  
    val ageGreeting = "You are now $age years old!"  
    return "$nameGreeting\n$ageGreeting"  
}
```

11. Argumentos con nombre

En los ejemplos anteriores, no tuviste que especificar los nombres de los parámetros, `name` o `age`, cuando llamaste a una función. Sin embargo, podrás hacerlo si así lo decides. Por ejemplo, puedes llamar a una función con muchos parámetros o pasar los argumentos en un orden diferente; por ejemplo, colocar el parámetro `age` antes del parámetro `name`. Cuando incluyes un nombre de parámetro si llamas a una función, esta se denomina [argumento con nombre](#). Intenta usar un argumento con nombre con la función `birthdayGreeting()`.

1. Modifica la llamada para que Rex use argumentos con nombre como se muestra en este fragmento de código. Para ello, incluye el nombre del parámetro seguido de un signo igual y, luego, el valor (p. ej., `name = "Rex"`).

```
println(birthdayGreeting(name = "Rex", age = 2))
```

1. Ejecuta el código y, luego, observa que el resultado no haya cambiado:

```
Happy Birthday, Rover!  
You are now 5 years old!  
Happy Birthday, Rex!  
You are now 2 years old!
```

1. Reordena los argumentos con nombres. Por ejemplo, coloca el argumento con nombre `age` antes del argumento con nombre `name`.

```
println(birthdayGreeting(age = 2, name = "Rex"))
```

1. Ejecuta el código y observa que el resultado sigue siendo el mismo. Si bien cambiaste el orden de los argumentos, se pasan los mismos valores para los mismos parámetros.

```
Happy Birthday, Rover!  
You are now 5 years old!  
Happy Birthday, Rex!  
You are now 2 years old!
```

12. Argumentos predeterminados

Los parámetros de la función también pueden especificar argumentos predeterminados. Quizás Rover es tu perro favorito, o esperas que se llame a una función con argumentos específicos en la mayoría de los casos. Cuando llamas a una función, puedes decidir omitir los argumentos para los que haya un valor predeterminado, en cuyo caso se usa el predeterminado.

Para agregar un argumento predeterminado, agrega un operador de asignación (`=`) después del tipo de datos para el parámetro y configúralo del mismo modo que a un valor. Modifica tu código para usar un argumento predeterminado.

1. En la función `birthdayGreeting()`, establece el parámetro `name` en el valor predeterminado `"Rover"`.

```
fun birthdayGreeting(name: String = "Rover", age: Int): String {  
    return "Happy Birthday, $name! You are now $age years old!"  
}
```

1. En la primera llamada a `birthdayGreeting()` para Rover en `main()`, establece el argumento con nombre `age` en `5`. Como el parámetro `age` se define después de `name`, debes usar el argumento con nombre `age`. Sin los argumentos con nombre, Kotlin supone que el orden de los argumentos es el mismo en el que se definen los parámetros. El argumento con nombre se usa a fin de garantizar que Kotlin espere un `Int` para el parámetro `age`.

```
println(birthdayGreeting(age = 5))  
println(birthdayGreeting("Rex", 2))
```

1. Ejecuta tu código. La primera llamada a la función `birthdayGreeting()` muestra "Rover" como nombre porque nunca lo especificaste. La segunda llamada a `birthdayGreeting()` todavía usa el valor `Rex`, que pasaste para `name`.

```
Happy Birthday, Rover! You are now 5 years old!  
Happy Birthday, Rex! You are now 2 years old!
```

1. Quita el nombre de la segunda llamada a la función `birthdayGreeting()`. Una vez más, debido a que se omite `name`, debes usar un argumento con nombre para la edad.


```
println(birthdayGreeting(age = 5))  
println(birthdayGreeting(age = 2))
```

1. Ejecuta tu código. Luego, observa que ambas llamadas a `birthdayGreeting()` imprimen "Rover" como nombre porque no se pasa ningún argumento con nombre.

```
Happy Birthday, Rover! You are now 5 years old!  
Happy Birthday, Rover! You are now 2 years old!
```