

## Localización -Permisos

- La localización es un recurso con riesgo pues involucra información privada del usuario.
- Dos tipos de localización:
  - Localización fina (GPS) :

```
<uses-permission  
    android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

- Localización Gruesa (triangulación WIFI) :

```
<uses-permission  
    android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

## ¿GPS- Cómo funciona?

- Hay alrededor de 30 satélites que orbitan la tierra. (Circle the earth twice a day)
- En cualquier punto de la tierra se deben ver al menos 4 satélites.
- Con la ubicación de 3 satélites se puede hacer el cálculo de la posición exacta.

## Location

- La aplicación puede consultar la ubicación una sola vez, sin embargo, para aplicaciones que ayudan al usuario a encontrar una ruta mientras se desplazan, la ubicación debe actualizarse en intervalos regulares.
- Se puede obtener información como:
  - Latitud: se expresa en medidas angulares: 90° N a los 90° S
  - Longitud: se basan en el meridiano de Greenwich (Londres), se expresa en medidas angulares: 180° a 180°
  - Altitud
  - Velocidad de desplazamiento
- Esta info. está disponible en la clase [Location](#) que se obtiene del [fused location provider](#).

## Agregar las dependencias

- En el archivo gradle del módulo de la aplicación se debe agregar la dependencia a todos los servicios de Google:
  - Última versión 19.0.1

```
implementation 'com.google.android.gms:play-services-location:19.0.1'
```

## Consultar la localización

- Se utiliza el servicio de localización de Google:

- Atributo de la clase

```
private FusedLocationProviderClient mFusedLocationClient;
```

- En onCreate se inicializa

```
mFusedLocationClient = LocationServices.getFusedLocationProviderClient(this);
```

- Cuando se tengan permisos, se puede acceder a la localización:

```
mFusedLocationClient.getLastLocation().addOnSuccessListener(this,
new OnSuccessListener<Location>() {
    @Override
    public void onSuccess(Location location) {
        Log.i("LOCATION", "onSuccess location");
        if (location != null) {
            Log.i(" LOCATION ", "Longitud: " + location.getLongitude());
            Log.i(" LOCATION ", "Latitud: " + location.getLatitude());
        }
    }
});
```

## Location-aware Apps- Notificación de cambios de posición

- Para especificar mejor las condiciones que se requieren de localización y suscribirse a actualizaciones periódicas es necesario crear un objeto de tipo **RequestLocation**:

```
import com.google.android.gms.location.LocationRequest;
```

//Atributo

```
private LocationRequest mLocationRequest;
```

//en OnCreate

```
mLocationRequest = createLocationRequest();
```

```
private LocationRequest createLocationRequest(){
```

```
    LocationRequest locationRequest = LocationRequest.create()
        .setInterval(10000)
        .setFastestInterval(5000)
        .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
    return locationRequest;
}
```

## Prioridad de la solicitud

### Tipos de prioridad:

- **PRIORITY\_BALANCED\_POWER\_ACCURACY**: Se usa para solicitar la ubicación dentro de un bloque de ciudad: Exactitud de **100 metros**.
- **PRIORITY\_HIGH\_ACCURACY**: Se usa para solicitar la ubicación más precisa posible.
- **PRIORITY\_LOW\_POWER**: Se usa para solicitar precisión a nivel de ciudad: Exactitud de **10 kilómetros**.
- **PRIORITY\_NO\_POWER**: Se usa si necesita un impacto insignificante en el poder consumo, pero desea recibir actualizaciones de ubicación cuando estén disponibles.

## Notificación de cambios de posición-Suscripción a la localización

- Para suscribirse a la localización, se debe definir un objeto callback() cuyos métodos son invocados de acuerdo a la tasa de refresco antes establecida.

```
//Atributo  
private LocationCallback mLocationCallback;
```

- Este atributo se asigna al fuseLocationClient para recibir las actualizaciones.

```
private void startLocationUpdates() {  
    //Verificación de permiso!  
    if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) ==  
        PackageManager.PERMISSION_GRANTED){  
        mFusedLocationClient.requestLocationUpdates(mLocationRequest, mLocationCallback, null);  
    }  
}
```

Petición de Localización      Objeto Callback!

- Finalmente se programa el objeto callback para reaccionar a la actualización de la ubicación:

```
//En OnCreate()  
mLocationCallback = new LocationCallback() {  
    @Override  
    public void onLocationResult(LocationResult locationResult) {  
        Location location = locationResult.getLastLocation();  
        Log.i("LOCATION", "Location update in the callback: " + location);  
        if (location != null) {  
            latitude.setText("Latitude: " + String.valueOf(location.getLatitude()));  
            longitude.setText("Longitude: " + String.valueOf(location.getLongitude()));  
            altitude.setText("Altitude: " + String.valueOf(location.getAltitude()));  
        }  
    }  
};
```

Se asignan los valores de la localización a los TextViews  
Esto va a suceder cada 5 segundos aproximadamente

## Encender la localización de forma programática

- Si el usuario no tiene encendida la localización, la aplicación puede encenderla mostrando un diálogo al usuario. Para eso se debe:
  - Acceder a la configuración del usuario.
  - Si la tarea es exitosa, la localización está encendida y se puede seguir adelante.
  - Sino, pero si existe el hardware en el dispositivo se debe enviar un diálogo para encender la localización de forma programática.
  - Finalmente si no se cuenta con el hardware se debe deshabilitar la funcionalidad asociada a la localización.

```

private void checkLocationSettings(){
    LocationSettingsRequest.Builder builder = new
        LocationSettingsRequest.Builder().addLocationRequest(locationRequest);
    SettingsClient client = LocationServices.getSettingsClient(this);
    Task<LocationSettingsResponse> task = client.checkLocationSettings(builder.build());
    task.addOnSuccessListener(new OnSuccessListener<LocationSettingsResponse>() {
        @Override
        public void onSuccess(LocationSettingsResponse locationSettingsResponse) {
            Log.i(IndexActivity.TAG, "GPS is ON");
            settingsOK = true;
            startLocationUpdates();
        }
    });
    task.addOnFailureListener(new OnFailureListener() {
        @Override
        public void onFailure(@NonNull Exception e) {
            if(((ApiException) e).getStatusCode() == CommonStatusCodes.RESOLUTION_REQUIRED){
                ResolvableApiException resolvable = (ResolvableApiException) e;
                IntentSenderRequest isr = new IntentSenderRequest.Builder(resolvable.getResolution()).build();
                getLocoationSettings.launch(isr);
            }else{
                elevation.setText("No GPS available");
            }
        }
    });
}

```

1. Acceder a la configuración

2. GPS ya está encendido

3. GPS está apagado pero se puede encender

3.1 Lanza actividad con nuevo API

4. Hardware no disponible

## Actividad de solicitud para encender GPS

```

ActivityResultLauncher<IntentSenderRequest> getLocoationSettings =
registerForActivityResult(
    new ActivityResultContracts.StartIntentSenderForResult(),
    new ActivityResultCallback<ActivityResult>() {
        @Override
        public void onActivityResult(ActivityResult result) {
            Log.i(IndexActivity.TAG, "Result from settings: "+result.getResultCode());
            if(result.getResultCode() == RESULT_OK){
                settingsOK = true;
                startLocationUpdates();
            }else{
                elevation.setText("GPS is off");
            }
        }
    });

```

## Encender y apagar la suscripción

- Suscribirse a la actualización de cambios en la posición es costoso en términos de procesamiento y consumo de energía.
- Sólo se debe hacer cuando la aplicación necesite mostrar actualizaciones de forma constante como parte de su funcionamiento.
- En una aplicación con navegación (Waze, google maps, etc.) sólo se debe escuchar la ubicación si el usuario está viendo algo como un mapa y es consciente de su desplazamiento.
- Si no se están mostrando los datos la suscripción debería cancelarse:

```

@Override
protected void onPause() {
    super.onPause();
    stopLocationUpdates();
}

private void startLocationUpdates(){
    if(ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) ==
        PackageManager.PERMISSION_GRANTED){
        if(settingsOK){
            fusedLocationProvider.requestLocationUpdates(locationRequest, locationCallback, null);
        }
    }
}

private void stopLocationUpdates(){
    fusedLocationProvider.removeLocationUpdates(locationCallback);
}

```

La actividad pasa a un segundo plano, se cancela la suscripción

## Distancia entre dos puntos

- Es posible calcular la distancia entre dos puntos a partir de la latitud y la longitud de cada punto:

```

public double distance(double lat1, double long1, double lat2, double long2) {
    double latDistance = Math.toRadians(lat1 - lat2);
    double lngDistance = Math.toRadians(long1 - long2);

    double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2)
              + Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2))
              * Math.sin(lngDistance / 2) * Math.sin(lngDistance / 2);

    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    double result = RADIUS_OF_EARTH_KM * c;
    return Math.round(result*100.0)/100.0;
}

```

## Mapas

### Actividad para mapas

- Se crean tres archivos:
- La actividad y el layout del mapa
- Un nuevo tag en el manifest con instrucciones para obtener un API KEY

Niveles de zoom:

1: Mundo	<del>mMap.moveCamera(CameraUpdateFactory.zoomTo(10));</del>
5: Tierra firme y continente	
10: Ciudad	
15: Calles	
20: Edificios	

## Interfaz de usuario para el mapa

- Controles del mapa:

- Habilitar los “gestures” como “pinch to zoom”

```
mMap.getUiSettings().setZoomGesturesEnabled(true);
```

- Habilitar los botones de zoom

```
mMap.getUiSettings().setZoomControlsEnabled(true);
```

- Inclinación, rotación, desplazamiento etc.

## Fragments

- Representa una parte de la interfaz o del comportamiento dentro de una actividad.
- Se pueden usar varios fragmentos en una actividad y reutilizarlos en varias actividades.
- Siempre debe estar integrado en una actividad y su ciclo de vida es propio pero depende del ciclo de vida de la actividad.
- Si la actividad está en onPause() o onDestroy() los fragmentos igual, sin embargo cuando la actividad está en onResume() cada fragmento se manejó de forma independiente.
- El fragmento puede estar contenido dentro de la actividad o puede estar en un archivo diferente.

## Rounded Corners

- En drawable, se puede crear un archivo “roundcorner.xml” (todo en minúsculas):

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"  
    android:shape="rectangle" android:padding="10dp">  
    <solid android:color="#bb0099cc"/> Fondo semitransparente  
    <corners  
        android:bottomRightRadius="10dp"  
        android:bottomLeftRadius="10dp"  
        android:topLeftRadius="10dp"  
        android:topRightRadius="10dp"/>  
    </shape>
```

- En el layout:

```
<EditText  
    ...  
    android:background="@drawable/roundcorner"  
    ...  
/>
```

## Marcadores

- Son indicadores de una posición en el mapa. Se pueden personalizar de forma simple (colores) o incluso colocando un ícono personalizado.

```
LatLng bogota = new LatLng(4.65, -74.05);
mMap.addMarker(new MarkerOptions().position(bogota).title("Marcador en Bogotá"));
```

- Las ventanas de información asociadas proporcionan información adicional.
- Agregando un snippet se agrega información que se despliega cuando se hace click en el marcador.

```
mMap.addMarker(new MarkerOptions().position(bogota)
    .title("BOGOTÁ")
    .snippet("Población: 8.081.000") //Texto de información
    .alpha(0.5f)); //Transparencia
```

## Marcadores Personalizados

- Se puede cambiar el color del marcador

```
Marker bogotaAzul = mMap.addMarker(new MarkerOptions()
    .position(bogota)
    .icon(BitmapDescriptorFactory
        .defaultMarker(BitmapDescriptorFactory.HUE_BLUE)));
```

- O utilizar una imagen personalizada

```
Marker bogotaBike = mMap.addMarker(new MarkerOptions()
    .position(bogota)
    .icon(BitmapDescriptorFactory
        .fromResource(R.drawable.bike)));
```

- Se pueden mostrar y ocultar los marcadores

```
bogotaAzul.setVisible(true);
bogotaAzul.setVisible(false);
```

## Vector asset

- Android incluye una galería de íconos a los que se pueden agregar los recursos de la aplicación (New -> Vector Asset).

## Quitar marcadores

- Cuando se hace seguimiento o hay mucha información, se hace necesario eliminar los marcadores.
  - Eliminar un sólo marcador:

```
Marker marker = mMMap.addMarker(new MarkerOptions().position(latLng).title("Marcador"));
marker.remove();
```

Para poder eliminarlo se debe guardar en una variable

- Limpiar el mapa(borra todos los marcadores):

```
mMMap.clear();
```

## Sensores y Hardware

- Android cuenta con un API que permite la consulta de los diferentes sensores del dispositivo. Estos sensores incluyen:
  - Temperatura
  - Presión Atmosférica
  - Luminosidad
  - Acelerómetro
  - Giroscopio
  - Proximidad

## Uso del sensor de luminosidad

- Se necesitan tres atributos nuevos para la actividad disponibles en el paquete android.hardware:

```
SensorManager sensorManager;
Sensor lightSensor;
SensorEventListener lightSensorListener;
```
- Inicializar el administrador de los sensores:

```
//onCreate
sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
lightSensor = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
```
- El atributo light Sensor será nulo si el dispositivo no cuenta con este sensor.
- Definir las acciones del listener. El sensor de luminosidad recibe valores entre 0 y 40000 lx.

```
lightSensorListener = new SensorEventListener() {
    @Override
    public void onSensorChanged(SensorEvent event) {
        if (mMMap != null) {
            if (event.values[0] < 5000) {
                Log.i("MAPS", "DARK MAP " + event.values[0]);
                mMMap.setStyle(MapStyleOptions.loadRawResourceStyle(MapsActivity.this, R.raw.dark_style_map));
            } else {
                Log.i("MAPS", "LIGHT MAP " + event.values[0]);
                mMMap.setStyle(MapStyleOptions.loadRawResourceStyle(MapsActivity.this, R.raw.light_style_map));
            }
        }
    }
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {}
};
```

## Registrarse a los eventos del sensor

- Al igual que con la localización, es importante suscribirse y desuscribirse de los cambios en el sensor usando los métodos adecuados del ciclo de vida de la actividad.

```
@Override  
protected void onResume() {  
    super.onResume();  
    sensorManager.registerListener(lightSensorListener, lightSensor,  
        SensorManager.SENSOR_DELAY_NORMAL);  
}  
  
@Override  
protected void onPause() {  
    super.onPause();  
    sensorManager.unregisterListener(lightSensorListener);  
}
```

## API de Google para buscar direcciones

- Google provee un API para buscar direcciones de forma textual y obtener como resultado localizaciones que se pueden usar en un mapa.
- Usar el objeto Geocoder.

### Limitar búsqueda a una región

- Si se quiere limitar geográficamente la búsqueda a sólo una zona del planeta, se define una región a través de 4 puntos:

```
// Limits for the geocoder search (Colombia)  
public static final double lowerLeftLatitude = 1.396967;  
public static final double lowerLeftLongitude=-78.903968;  
public static final double upperRightLatitude= 11.983639;  
public static final double upperRigthLongitude=-71.869905;
```

- Luego se invoca a geocoder incluyendo a los parámetros:

```
List<Address> addresses = mGeocoder.getFromLocationName( addressString, 2,  
    lowerLeftLatitude, lowerLeftLongitude,  
    upperRightLatitude, upperRigthLongitude);
```

## Listener sobre un mapa

- Es posible programar dos eventos en el mapa, Click o LongClick.
- El evento generado retorna las coordenadas (objeto LatLng) donde el usuario hace click.
- El siguiente fragmento de código genera un marcador con el nombre de la dirección encontrada donde la persona hace un click largo.

```

mMap.setOnMapLongClickListener(new GoogleMap.OnMapLongClickListener() {
    @Override
    public void onMapLongClick(LatLng latLng) {
        mMap.clear();
        mMap.addMarker(new MarkerOptions().position(latLng).title(geoCoderSearchLatLang(latLng)));
    }
});

```

## Otros proveedores

- Open Street Maps
- Here
- Bing
- Apple maps

## Firebase

### Backend

- Para la creación de aplicaciones móviles existen dos formas de trabajar el backend:
  - Utilizar un backend genérico provisto por un tercero, por ejemplo: google **Firebase**, Amazon amplify, Parse Platform.
  - Construir un backend a la medida, típicamente exponer servicios REST a través de :
    - JAVA (JAX-RS)
    - Spring
    - Django
    - .Net
    - Etcl

### Firebase

- Backend para aplicaciones:
  - Base de datos para tiempo real: ios, android, unity, c++
  - Autenticación: ios, android, c++
  - Almacenamiento
  - cloud messaging
  - Analitica

### Pre-requisitos:

- Mínimo android 4.0 y Google play services 11.4.0

### Manejo de autenticación

- Definir los atributos de la actividad necesarios para la autenticación

- En onStart se verifica el estado actual de la autenticación del usuario, en este caso, si el usuario está autenticado, se lanza la siguiente actividad:

```

@Override
protected void onStart() {
    super.onStart();
    FirebaseUser currentUser = mAuth.getCurrentUser();
    updateUI(currentUser);
}

private void updateUI(FirebaseUser currentUser) {
    if(currentUser!=null){
        Intent intent = new Intent(getApplicationContext(), HomeActivity.class);
        intent.putExtra("user", currentUser.getEmail());
        startActivity(intent);
    } else {
        email.setText("");
        password.setText("");
    }
}

```

---

### Validación de campos

- Antes de enviar la petición a Firebase, es importante validar los campos ingresados por el usuario.
- Esto se hace en la aplicación para no gastar “consultas” en firebase en caso de que el usuario meta datos sin sentido.

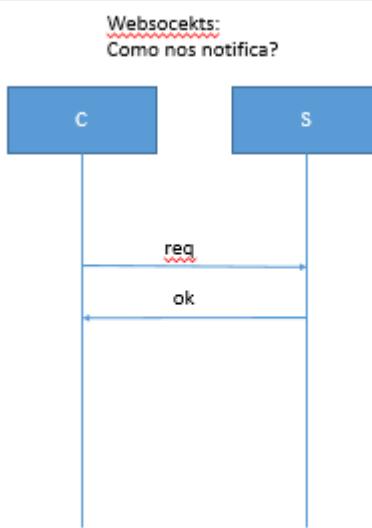
```

private boolean validateForm() {
    boolean valid = true;
    String email = mUser.getText().toString();
    if (TextUtils.isEmpty(email)) {
        mUser.setError("Required.");
        valid = false;
    } else {
        mUser.setError(null);
    }
    String password = mPassword.getText().toString();
    if (TextUtils.isEmpty(password)) {
        mPassword.setError("Required.");
        valid = false;
    } else {
        mPassword.setError(null);
    }
    return valid;
}

```

---

## Realtime Database



- Realtime Database: es una base de datos original de Firebase. Es una solución eficiente y de baja latencia para aplicaciones móviles .
- Es una base de datos documental (basada en JSON) dinámica.
- Por defecto, sólo usuarios autenticados pueden leer y escribir datos. Se pueden modificar las reglas para permitir hacer cambios a cualquier usuario, pero no se recomienda dejar sin protección de los datos.

Fecha: 20-04-22

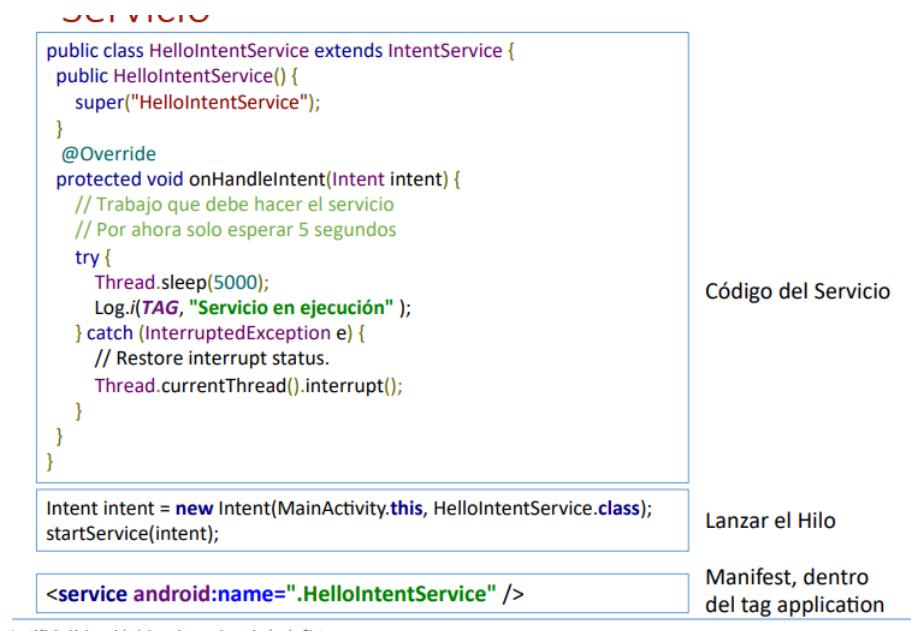
## Servicios

- Un servicio es otro componente de Android que no tiene asociada una interfaz gráfica y se encarga de hacer tareas en primero y segundo plano.
- Se puede usar para por ejemplo, recibir notificaciones de cambios en el Backend y notificar al usuario.
- Un servicio puede seguir activo a pesar de que la aplicación se haya cerrado.
- Foreground: El usuario es consciente de que se están haciendo las operaciones
  - A foreground service performs some operation that is noticeable to the user. For example, an audio app would use a foreground service to play an audio track. Foreground services continue running even when the user isn't interacting with the app.
- Background: El usuario no es consciente
  - A background service performs an operation that isn't directly noticed by the user. For example, if an app used a service to compact its storage, that would usually be a background service.

## Tipos de servicio

- Service:
  - Se usa cuando se deben atender muchas peticiones al tiempo
- Service Intent:

- Se usa cuando no es necesario atender múltiples peticiones al tiempo, más simple de implementar, sólo se sobrescriben los métodos necesarios.



## Cambios desde Android 0

Desde android 8 se limitan los tiempos de ejecución en background.

- **Service** – Tarea que se ejecuta en background. Corre en el mismo hilo de la actividad que lo invoca. Si es muy demorado debería crearse un hilo internamente en el servicio para no afectar el rendimiento de quien lo llama.
  - **Desventaja:** Corre en el mismo hilo
- **IntentService** – Tarea que se ejecuta en background pero crea su propio hilo.
  - Desventaja : El trabajo que se le asigne se pierde si quien lo llama se cierra.  
Deprecated since Android 11!
- **JobIntentService** – Similar al IntentService, pero quien lo ejecuta puede terminarlo en cualquier momento y puede reiniciarlo cuando él mismo reinicie.

## JobIntentService

```
public class HelloJobIntentService extends JobIntentService {  
    private static final int JOB_ID = 12;  
    public static void enqueueWork(Context context, Intent intent) {  
        enqueueWork(context, HelloJobIntentService.class, JOB_ID, intent);  
    }  
  
    @Override  
    protected void onHandleWork(@NonNull Intent intent){  
        int miliSeconds = intent.getIntExtra("miliSeconds", 10000);  
        try {  
            Thread.sleep(miliSeconds);  
            Log.i(HomeActivity.TAG, "Service Finished Waiting" );  
        } catch (InterruptedException e) {  
            Thread.currentThread().interrupt();  
        }  
    }  
}
```

- Hay que utilizar el permiso WAKE\_LOCK para versiones inferiores a Oreo.

```
<uses-permission android:name="android.permission.WAKE_LOCK"/>
```

- Hay que definir el permiso BIND\_JOB\_SERVICE dentro del tag de servicio para versiones Android Oreo y posterior

```
<application>  
    <!-- ... -->  
    <service  
        android:name=".services.HelloJobIntentService"  
        android:permission="android.permission.BIND_JOB_SERVICE" />  
    </application>
```

- Para invocarlo:

```
Intent intent = new Intent(HomeActivity.this, HelloJobIntentService.class);  
intent.putExtra("miliSeconds", 5000);  
HelloJobIntentService.enqueueWork(HomeActivity.this, intent);  
Log.i(TAG, "After the call to the service");
```

## Notificaciones PUSH

Aunque se puede usar de la forma que se presentó, hay varios inconvenientes:

- Cada aplicación que requiera notificaciones tendrá un servicio en background lo que iría en detrimento del rendimiento del dispositivo.
- Si se quieren escuchar los cambios en el backend sin que la aplicación este corriendo, se debe usar un ForeGround service(desde android 8), lo que implica que siempre existirá una notificación que no se puede quitar de la barra de notificaciones, a menos que se detenga el servicio.
- No es fácil hacer este proceso desde plataformas híbridas y algunos fabricantes pueden impedir la ejecución de estos servicios.

¿Qué alternativas hay?

- En Android y Apple
    - Firebase Cloud Messaging (FCM), incluido en la capa gratis de Firebase
  - Sólo en Apple
    - Apple Push Notifications Service (APNs) -> FCM lo usa
- Ventajas
    - Se utilizan los procesos que ya corren en el dispositivo para recibir notificaciones de distintas aplicaciones y backends (Google APIs)
    - Implementación mucho más simple, no es necesario correr procesos en background ni crear servicios en el arranque del dispositivo.
  - Desventajas
    - Se crea una dependencia fuerte con el proveedor del servicio de notificaciones.
    - Costos y limitaciones pueden cambiar a futuro

## Servicios REST

Volley es una librería construida por Google para consumir servicios REST •

- Para usarla se debe incluir la dependencia en gradle:  
implementation 'com.android.volley:volley:1.1.1'
- Más información de documentación y versiones:  
<https://github.com/google/volley/releases>

```
public void consumeRESTVolley(){  
    RequestQueue queue = Volley.newRequestQueue(this);  
    String url = "https://restcountries.eu/rest/v2/";  
    String path = "currency/cop";  
    String query = "?fields=name;capital";  
    StringRequest req = new StringRequest(Request.Method.GET, url+path+query,  
        new Response.Listener() {  
            @Override  
            public void onResponse(Object response) {  
                String data = (String)response;  
                restResponse.setText(data);  
            }  
            new Response.ErrorListener() {  
                @Override  
                public void onErrorResponse(VolleyError error) {  
                    Log.i("TAG", "Error handling rest invocation"+error.getCause());  
                }  
            };  
        queue.add(req);  
    }  
}
```

Requiere el permiso de internet en el manifest!!!!

Requiere el permiso de internet en el manifest!!

## AsyncTask

The three types used by an asynchronous task are the following:

- Params, the type of the parameters sent to the task upon execution.
- Progress, the type of the progress units published during the background computation.
- Result, the type of the result of the background computation.
-

Not all types are always used by an asynchronous task. To mark a type as unused, simply use the type Void:

```
private class MyTask extends AsyncTask<Void, Void, Void> { ... }
```

- `onPreExecute()`, invoked on the UI thread before the task is executed. This step is normally used to set up the task, for instance by showing a progress bar in the user interface.
- `doInBackground(Params...)`, invoked on the background thread immediately after `onPreExecute()` finishes executing. This step is used to perform background computation that can take a long time.
- `onProgressUpdate(Progress...)`, invoked on the UI thread after a call to `publishProgress(Progress...)`. This method is used to display any form of progress in the user interface while the background computation is still executing.
- `onPostExecute(Result)`, invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

## Flutter

- Flutter es un framework de desarrollo móvil, web y de escritorio (windows) creado por Google.
- Presentado en 2015. La versión 1 fue lanzada en diciembre de 2018.
- Basado en Material Design.
- Se utiliza el lenguaje de programación Dart de Google
- Una base de código para las dos plataformas Android y iOS
- Generación de código **nativo!**
  - No se depende de un browser
  - No está basado en JavaScript ni HTML 5
- Se puede desarrollar con Linux, Windows o Mac usando Android Studio o Visual Studio Code.
- Se pueden usar los mismos emuladores creados con Android Studio para Android, y los de XCode para iOS
  - Si se desea probar la aplicación en iOS es necesario usar un computador con MacOS y XCode

## Flutter vs otros frameworks

	Flutter	React Native	Ionic	Xamarin	PhoneGap
Desarrollado por:	Google	Facebook	Ionic	Microsoft	Apache - Adobe
Native	Si	Si (UI + javascript)	No	No	No
Usa un Browser	No	No, Motor de JS	Si	No	Si
Single-Code Base	Si	Si	Si	No (hasta 75% del código común)	Si
Lenguajes	Dart	JavaScript/TypeScript	JavaScript / TypeScript	C#	JavaScript - Html CSS
Año	2018	2015	2013	2011	2009

## Instalación

- Descargar Flutter SDK
- Extraer el contenido del SDK a una carpeta específica.
  - Alternativamente puede bajar la última versión estable de `C:\src>git clone https://github.com/flutter/flutter.git -b stable`
- Crear una entrada en el Path a `flutter/bin`
- Correr: `flutter doctor`

```
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, v1.12.13+hotfix.8, on Microsoft Windows [Version 10.0.18362.720], locale en-US)
[✓] Android toolchain - develop for Android devices (Android SDK version 29.0.2)
[✓] Android Studio (version 3.5)
[✓] VS Code (version 1.43.2)
[✓] Connected device (1 available)

• No issues found!

! Doctor found issues in 4 categories.
```

<https://flutter.dev/docs/get-started/install/windows>

## Estructura del proyecto

- Lib: carpeta con el código fuente.
- Test: carpeta con el código fuente de las pruebas
- pubspec.yaml: archivo de configuración del proyecto

## Dart-Conceptos básicos

```
int a=6; //declaración normal  
var b=5; //inferencia del tipo  
final c=7; //no modificable pero asignable en ejecución  
const d=1; //no modificable y sólo asignado al inicio del programa  
print(a+b); // impresión normal  
print("$a - $b - $c - $d"); // impresión con uso de cadena
```

```
11  
6 - 5 - 7 - 1 - 1
```

- Existe un tipo de datos dinámico que puede cambiar en ejecución. Para esto es necesario usar la palabra reservada dynamic.

```
dynamic dinamico = 8;  
print(dinamico);  
dinamico = "hola";  
print (dinamico);
```

```
8  
hola
```

- Constructor

- Al igual que en java el constructor de la clase lleva el mismo nombre de la clase y los parámetros a inicializar. Sin embargo, se puede abbreviar:

```
class Dog {  
  String name;  
  int age;  
  void Dog(String name, int age) {  
    this.name = name;  
    this.age = age;  
  }  
}
```

```
class Dog {  
  String name;  
  int age;  
  Dog({this.name, this.age});  
}
```

- Las funciones en Dart pueden tener parámetros con nombre y opcionales:

```
void bark(String language, int times){  
  int counter = 0;  
  print(language);  
  while (counter < times){  
    if(language == "English"){  
      print("wofff");  
    } else {  
      print("guau");  
    }  
    counter++;  
  }  
}
```

```
void barkTwo({String language, int times})){  
  int counter = 0;  
  print(language);  
  while (counter < times){  
    if(language == "English"){  
      print("wofff");  
    } else {  
      print("guau");  
    }  
    counter++;  
  }  
}
```

- Invocar la función

```
dog.bark("English", 5);
```

```
dog.barkTwo(language: "English", times: 5);  
dog.barkTwo(times: 5, language: "English");  
dog.barkTwo(times: 5,);
```

- Referencias a funciones. Dart es un lenguaje funcional y permite tener referencias a funciones y pasárselas como parámetro:

```
floatingActionButton: FloatingActionButton(  
    onPressed: buttonPressed, //referencia a la función  
    tooltip: 'Increment',  
    child: Icon(Icons.add),  
,
```

```
void buttonPressed() {  
    _counter++;  
}
```

- **Funciones anónimas** - Son funciones con un bloque de Código que no tienen un nombre. Si en el ejemplo anterior no se quisiera definir una función, el atributo onPressed se podría definir así:

```
floatingActionButton: FloatingActionButton(  
    onPressed: () {  
        _counter++;  
    }, //fin función anónima  
    tooltip: 'Increment',  
    child: Icon(Icons.add),  
,
```

- O usando el operador => para funciones que sólo tienen una línea:

```
onPressed: () => _counter++,
```

- List, es un conjunto de elementos

```
List<String> cadenas = ['hola', 'mundo', 'hola', 'mundo'];  
print(cadenas[0]);  
cadenas.add('valor');  
print(cadenas);
```

```
hola  
[hola, mundo, hola, mundo, valor]
```

- Map, conjunto llave valor

```
Map mapa = {'hola': 'mundo', 'entero': 5, 'double': 56.4};  
print(mapa['double']);  
print(mapa);  
Map<String, String> mapaParametrizado = {'hola': 'mundo', 'hello': 'world'};  
print(mapaParametrizado['hello']);  
print(mapaParametrizado);
```

```
56.4  
{hola: mundo, entero: 5, double: 56.4}  
world  
{hola: mundo, hello: world}
```

## Más de Flutter

- Todo es un widget!! - Cada Widget tiene un método build asociado • Los widgets pueden ser:
  - **Stateless**
    - No tienen estado, solo se corre el método **build** una vez y si tienen datos estos no pueden variar durante la ejecución (**final**)

- **Statefull**
  - Tiene un estado asociado.
  - Se corre el método **build** cada vez que cambia el estado .
  - Se debe usar el método **setState** para actualizar el estado.
- Flutter funciona de forma declarativa.
- Se define un árbol o jerarquía de widgets que deben aparecer en la pantalla.
- El método **build** se encarga de recorrer el árbol y dibujar los elementos que encuentre.
  - Si es un stateless widget, se hace una vez.
  - Si es un stateful widget, se hace cada vez que cambie el estado

## Flutter - Widgets

Flutter - *Android* - *iOS*

- Text ([TextView](#), [Label](#))
- TextField ([EditText](#), [TextField](#))
- Buttons
  - RaisedButton, FlatButton, FloatingActionButton
- Scaffold
- AppBar
- Column ([LinearLayout vertical](#), [StackView](#))
- Row ([LinearLayout horizontal](#), [StackView](#))
- ListView
- TileList
- Image

<https://flutter.dev/docs/reference/widgets>

## Imágenes

- Para almacenar las imágenes en la aplicación, se agregan a una carpeta dentro del proyecto y se referencian desde el archivo `pubspec.yaml`.

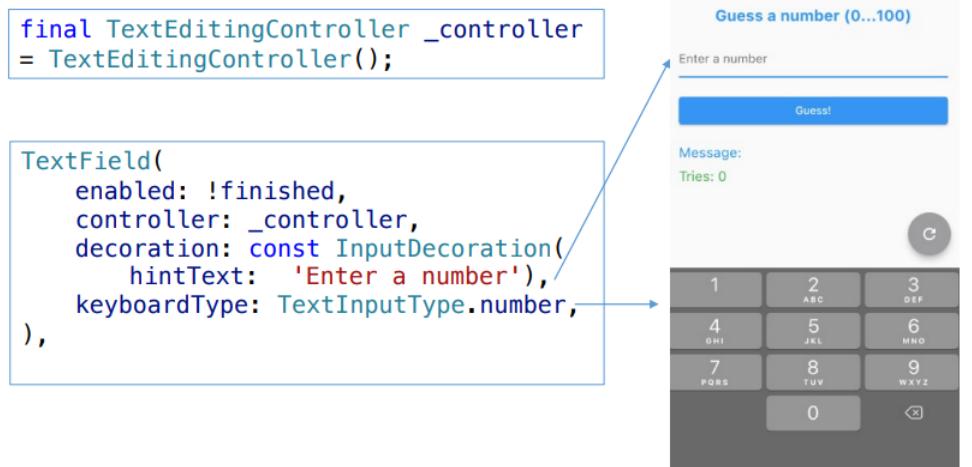
```
# To add assets to your
application, add an assets
section, like this:
assets:
- assets/images/question.png
# - images/a_dot_ham.jpeg
```

```
SizedBox(
  width: 200,
  height: 200,
  child: Image.asset(
    "assets/images/question.png"
),
```

`pubspec.yaml`

## TextFields and controllers

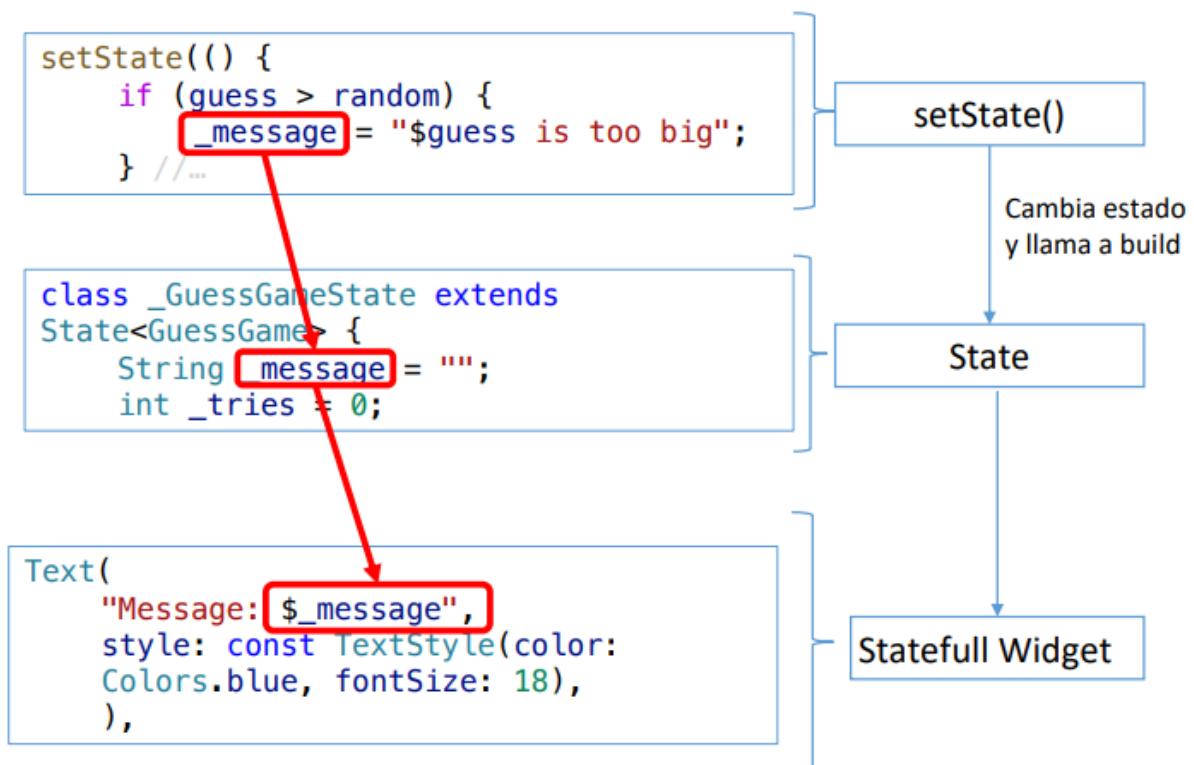
- Para acceder al valor de un campo de texto en flutter, es necesario definir un controlador, y asignarlo al campo



- Para saber lo que el usuario ingresó en el campo se accede a la propiedad `text` del controlador.

```
if (_controller.text.isNotEmpty) {  _tries++;  int guess = int.parse(_controller.text);  //...
```

## Manejo del Estado



## Acceso a elementos de la interfaz

- Dado que flutter se encarga del método `build` para construir el árbol de elementos, éstos no se pueden modificar de forma programática desde otras funciones de la clase.
- En este caso para modificar un valor de la interfaz, por ejemplo deshabilitar un botón, hay que hacerlo a través de la modificación del estado del widget.
- Primero hay que definir que el botón depende del estado y luego modificar el estado con `setState` cuando sea necesario para lograr el efecto deseado

## Deshabilitar un elemento de la GUI

- Estado:

```
class _GuessGameState extends State<GuessGame> {  
    bool finished = false;  
    //...
```

- Widgets dentro de `build`:

```
TextField(  
    enabled: !finished,  
    controller: _controller,  
    keyboardType: TextInputType.number,  
) ,
```

```
SizedBox(  
    width: double.infinity,  
    child: ElevatedButton(  
        onPressed: finished ? null : _play,  
        child: const Text("Guess!"))),
```

- Modificar el estado para que flutter pinte de nuevo los elementos:

```
setState(() {  
    //...  
    finished = true;  
}
```

## Navigator

- Para hacer una transición entre dos pantallas se usa el objeto Navigator de Flutter.

## Navigator, Transición entre pantallas

En el origen se puede usar el método push para iniciar la transición hacia la pantalla (Widget) de destino.

```
import 'package:testapp/guess_game.dart';  
  
//...  
onPressed: () {  
    Navigator.push(context,  
        MaterialPageRoute(builder: (context) => const GuessGame()));  
},
```

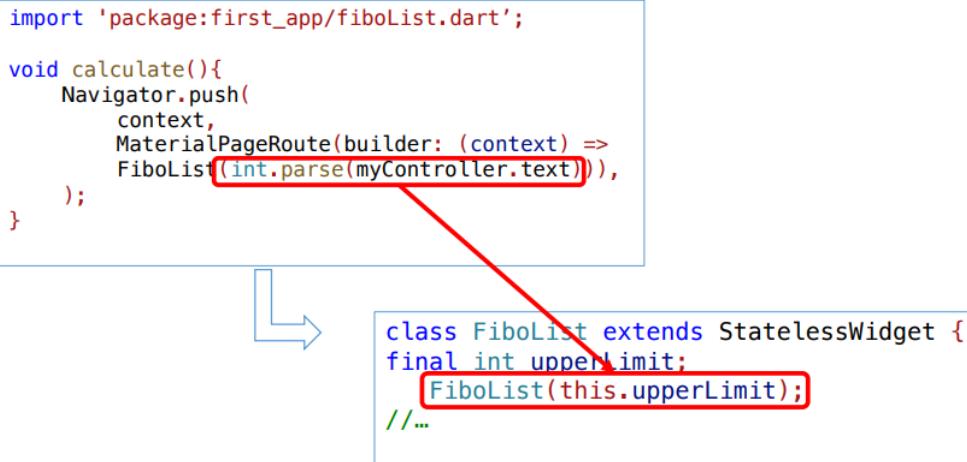
Origen

```
class GuessGame extends StatefulWidget {  
    const GuessGame({Key? key}) : super(key: key);  
  
    @override  
    _GuessGameState createState() => _GuessGameState();  
} //...
```

Destino

## Transición entre pantallas

- También se pueden enviar datos entre las pantallas. En este caso los datos llegan como parámetros al constructor del widget de destino.



## Named Navigator

- Si en la aplicación se tienen muchas pantallas, es ideal definir todas las rutas posibles, asignarles un nombre y luego a través del Navigator, ir a una pantalla en específico.
- Para esto es necesario primero definir las rutas.
- Cada ruta debe aparecer identificada, típicamente en el primer widget de la aplicación: Material App
- En el widget de la aplicación se pueden definir las rutas y la ruta inicial a cargar. Cada ruta puede estar en un archivo diferente pero deben importarse.

```
import 'login.dart';
import 'friends.dart';
import 'map.dart';
void main() {
  runApp(MaterialApp(
    title: "MyApp",
    debugShowCheckedModeBanner: false,
    //home: LoginWidget(),
    initialRoute: '/login',
    routes: {
      '/login': (context) => LoginWidget(),
      '/friends': (context) => FriendsList(),
      '/map': (context) => MapView(),
    },
  ));
}
```

- Una vez se tengan las rutas, se puede utilizar el método pushedNamed( ) del Navigator:

```

void updateUI() {
    if (validateLogin()) {
        _userEmail = FirebaseAuth.instance.currentUser?.email;
        Navigator.of(context).pushNamed('/map');
    }
}

```

## Manejo del estado

- Si bien se puede usar setState como se vio en la sesión inicial, esta práctica no es recomendable para aplicaciones más complejas que manejan diferentes estados en varias pantallas con muchos widgets.
- Para esto se puede usar una librería como Provider para separar la responsabilidad del manejo del estado de cada widget y definir un funcionamiento basado en eventos.
- Provider se puede entender como la implementación de un patrón Observer para el estado de la aplicación.

Pasos para usar Provider:

1. Agregar dependencias.
2. Definir el modelo para el estado
3. Registrarlo en el contexto
4. Operaciones sobre el estado
  - a. Watch
  - b. Read
  - c. Select

## Agregar dependencias

- Dentro del archivo pubspec.yaml, es necesario agregar la dependencia de Provider:

```

dev_dependencies:
  flutter_test:
    sdk: flutter
  provider: ^5.0.0

```

- Para actualizar las dependencias (si no se hace automáticamente), se puede correr el comando: - flutter clean - flutter pub get

## Definir el modelo para el estado

El modelo corresponde a una o varias clases que extienden de ChangeNotifier. En cada modificación del estado, se debe llamar a notifyListeners()

```

class GuessState extends ChangeNotifier {

    String _message = "";

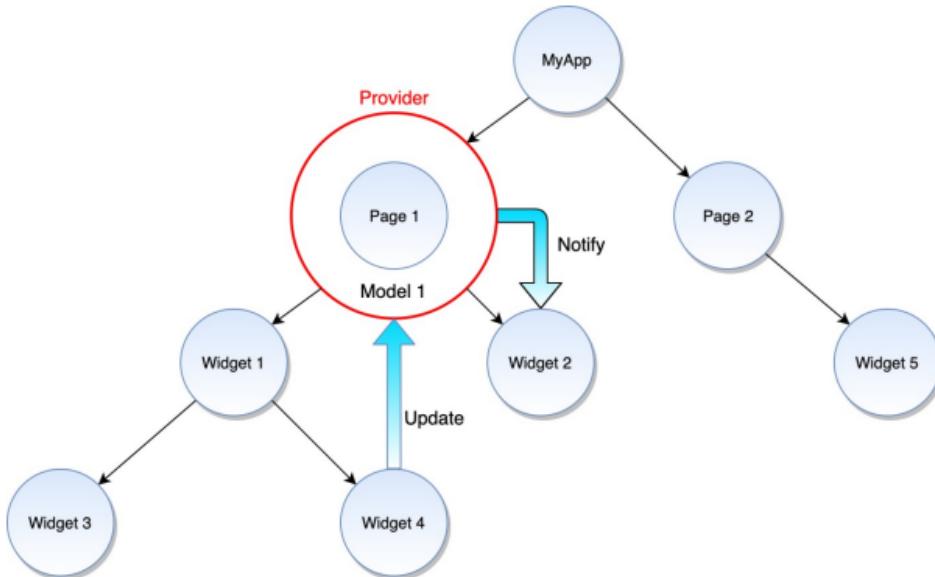
    String get message => _message;

    set message(String message) {
        _message = message;
        notifyListeners();
    }
}

```

### Registrarlo estado- Contexto

- Hay que identificar un ancestro común a todos los widgets que dependen del estado qué se está definiendo. Si toda la aplicación depende del estado, este Widget puede ser MaterialApp



En este caso se envuelve al widget de MaterialApp con ChangeNotifierProvider. Este recibe dos parámetros, el constructor del estado y la clase original que se quiere envolver.

```

void main() => runApp(ChangeNotifierProvider(
    create: (context) => GuessState(),
    child: const MaterialApp(
        title: "FlutterApp",
        home: HomeWidget(),
    )));

```

### Acceder al estado

Hay tres métodos para acceder al estado en Provider:

- watch: se usa cuando se quiere acceder al modelo y además se quiere reconstruir el widget cuando el modelo cambie
- read: se usa cuando se quiere acceder al modelo, pero no se quiere reconstruir el widget cuando el modelo cambie
- select: igual que watch, pero sólo para una parte del modelo
- watch y select se pueden usar sólo dentro del método build de un widget
- read no se puede usar en el método build, se usa típicamente en los métodos callback referenciados en el onPressed de un botón por ejemplo •
- Cuando se llama a notifyListeners() en un modelo, todos los métodos build que hayan accedido al modelo usando watch o select se vuelven a invocar.
- Dentro de un método build:

```
@override
Widget build(BuildContext context) {
var gameState = context.watch<GuessState>();
return Scaffold(
  appBar: AppBar(
  //...
  ...
  Text("Message: " + gameState.message,
    style: const TextStyle(
      color: Colors.blue,
      fontSize: 18),
  ),
  //...
}
```

- Dentro de un método callback:

```
void _newGame(BuildContext context) {
  context.read<GuessState>().finished = false;
  context.read<GuessState>().newRandom();
  context.read<GuessState>().message = "";
  context.read<GuessState>().counter = 0;
}
```

## SWIFT

- Lenguaje orientado a objetos y funcional
- Reemplazo de ObjectiveC
- Se usa en el desarrollo de aplicaciones para Apple.
- Lanzado por Apple en 2014, aún tiene elementos de compatibilidad con ObjectiveC
- Fuertemente tipado, con inferencia de tipos
- Versión actual: 5.5

## Acceder al estado

Una variable puede cambiar su valor a lo largo de la ejecución, una constante no. (Mutable vs Inmutable).

```
var variable = 50
let constante = 50

variable = 100 //ok
constante = 20 //ko
```

- No se define el tipo de dato, tenemos inferencia de tipos.

➤ Tipos implícitos:

```
let enteroImplicito = 70
let dobleImplicito = 70.0
```

➤ Tipos explícitos:

```
let enteroExplicito : Int = 70
```

➤ Para convertir un tipo, se debe hacer una instancia del nuevo tipo:

```
let etiqueta = "el precio es: " //Definición implicita de String
let valor = 90.0 //Definición implicita de Double
let todo = etiqueta + String(valor) // Cast de Double a String
print(todo)
```

el precio es: 90.0

- Nota: ser súper detallista con los espacios: dejarlo a ambos lados o no dejarlos.
- Para interpolar datos: \()

```
let manzanas = 5
let peras = 7
let frase = "Tengo \u2028(manzanas) manzanas y \u2028(peras) peras"
print(frase)
```

Tengo 5 manzanas y 7 peras

➤ Texto multi-línea con 3 comillas dobles

```
let textoMultiLinea = """
Este es un texto multilinea
"Tengo \u2028(manzanas) manzanas y \u2028(peras) peras"
"""
print(textoMultiLinea)
```

Este es un texto multilinea
"\"Tengo 5 manzanas y 7 peras"

## Arreglo y Diccionarios

- Un arreglo es un conjunto de valores ordenados, un diccionario es un conjunto de parejas en cualquier orden como un map.

```
//Array
var myArray = ["Hola", "Mundo", "hello", "world"]
print(myArray)
myArray[0] = "Bonjour"
print(myArray)

//Dictionary
var myDictionary = ["iMac":1500, "iPhone":1000, "AirTag":29]
print(myDictionary)
myDictionary["iPhone"] = 1100 //Nuevos precios
print(myDictionary)

//Sin inicialización, se deben definir los tipos
var myArrayNoData = [String]()
var myDictionaryNoData = [String:Int]()
```

## Ciclos y condicionales

- Para hacer una suma no hay `++`, se hace con dato `+=1`.

```
let notasIndividuales = [2.5, 3.9, 2.0, 4.5, 5]
var pasaron=0
var perdieron = Int(0) //var perdieron:Int=0
for nota in notasIndividuales {
    if nota<3.0 {
        perdieron += 1
    }else {
        pasaron+=1
    }
}
print("\(pasaron) estudiantes pasaron")
print("\(perdieron) estudiantes perdieron")
```

## EJERCICIO 1

- Utilizando la herramienta PlayGround de XCode construya un diccionario que contenga los productos que se muestran a continuación:
  - Apple Watch 6: \$300
  - iPhone 12 = \$749
  - OnePlus9 = \$500
  - iMac24 = \$1400
  - PlayStation 5: \$500
  - Macbook: \$1700
  - FitBit Versa: \$200
- Construya un ciclo que itere sobre los productos y calcule el valor total de los productos que valen menos de \$1000

Respuesta:

```
let MyDictionary=["Apple watch 6":300,"Iphone 12":749,  
"OnePlus9":500,"iMac24":1400,"playStation  
5":500,"Mackbook":1700,"FitBit Versa":200]  
var valorTotal=0  
for i in MyDictionary{  
    if(i.value<1000){  
        valorTotal=valorTotal+i.value  
    }  
}  
print("La suma de los productos con valor menor a $1000 es: \\\n(valorTotal)")
```

## Loops

- Se puede utilizar while para iterar hasta que se cumpla una condición. También se puede dejar la condición al final para que se haga al menos una vez:

- Se puede utilizar while para iterar hasta que se cumpla una condición. También se puede dejar la condición al final para que se haga al menos una vez:

```

var n = 2
while n < 100 {
    n *= 2
}
print(n)

```

→

```

var m = 2
repeat {
    m *= 2
} while m < 100
print(m)

```

→

Se ejecuta al menos una vez

128

- For sobre rangos: se usa ... como equivalente a  $\leq$  y ..< como equivalente a <

```

for i in 0...10{ // <=
    print(i)
}

```

0 1 2 3 4 5 6 7 8 9 10

```

for i in 0..<10{ // <
    print(i)
}

```

0 1 2 3 4 5 6 7 8 9

- El rango puede ser de otro tipo pero hay que indicarle cuanto avanzar en cada iteración

```

for i in stride(from: 5.5, to: 10.2, by:0.4){
    print(i)
}

```

5.5  
5.9  
6.3  
6.7  
7.1  
7.5 ...

## Optionals

- En swift un optional es un valor que puede ser nulo(nil) o tener asociado un contenido de otro tipo. Se marcan en la definición con un interrogante:
- No es otra cosa que un null safety: controlar cuando un objeto puede ser nulo o no.
- Ese string ya es otro tipo de dato. Es un optional que puede obtener dos valores nulo o su valor.
- El greeting es para abrir un optional, siempre hacer eso, si el valor es diferente de nulo.

```

var optionalString: String? = "Pika"
print(optionalString == nil)
print(optionalString)
if let greeting = optionalString {
    print("Hola \(greeting)")
}

```

→

false  
Optional("Pika")  
Hola Pika

```

var optionalString: String? //= "Pika"
print(optionalString == nil)
print(optionalString)
if let greeting = optionalString {
    print("Hola \(greeting)")
}

```

→

true  
nil

## Optionals- GUARD

- Otras formas de trabajar con optionals:
  - Guard: similar a if let, se puede usar para detener la ejecución de un método si no se puede abrir el optional.
  - Si el valor es igual a nulo haga esto:

```

var algoOpcional : String? = "Hola"
func printCadena (cadena:String?) {
    guard let miCadena = cadena else {
        print("La cadena llego nula")
        return
    }
    print(miCadena.count)
}
printCadena(cadena: algoOpcional)

```

**La cadena llego nula**

**4**

## Optionals- FORCED UNWRAPPING

- Force unwrapped se usa para forzar a swift a tomar el valor interno del opcional, si el valor era null, la aplicación va a fallar.
- Sólo se debe usar si se está absolutamente seguro de que el valor no es null.

```

var algoOpcional : String? = "Hola"

//Forced unwrapping
let forced = algoOpcional!
print(forced)

```

**Hola**

**Fatal error: Unexpectedly found nil while unwrapping an Optional value**

- Los switches en Swift soportan diferentes tipos de datos e incluso patrones:

```

let mascota = "gato persa"
switch(mascota) {
    case "perro beagle":
        print("🐶")
    case "perro poodle":
        print("🐩")
    case "gato":
        print("🐱")
    case let x where x.hasSuffix("persa"):
        print("😺")
    default:
        print("✳️")
}

```



- Siempre debe haber un default. No hay necesidad de usar un break entre cases. Sí, los emojis se integran con el código y la consola

## EJERCICIO 2

- Construya un arreglo con 8 emojis de su preferencia.
- Con base en un número aleatorio, escoja una posición del arreglo.

```

(valorTotal)*/
let myArray = ["😊", "😎", "☀️", "👉", "👉", "👉", "👉", "👉"]
let random=Int.random(in: 0..<8)
print(myArray[random])

```

## Funciones y Closures

- Para definir una función, se usa la palabra func, luego se da el nombre a la función, luego entre paréntesis los parámetros separados por comas. El retorno se coloca después con el operador.

```
func saludo(nombre:String, dia:String) -> String {  
    return "Hola \$(nombre), hoy es: \$(dia)"  
}  
  
//Invocacion  
print(saludo(nombre: "Carlos", dia: "Jueves"))
```

**Hola Carlos, hoy es: Jueves**

- Si no se desea tener una etiqueta se usa:

```
func saludoEtiquetaParcial(_ nombre:String, el dia:String) -> String {  
    return "Hola \$(nombre), hoy es: \$(dia)"  
}  
print(saludoEtiquetaParcial("Carlos", el: "Jueves"))
```

## Tuplas

- Es un lenguaje funcional
- Se pueden usar tuplas para devolver varios valores en una función.

```

func estadisticas (entrada_valores: [Int]) -> (min:Int, max:Int, sum:Int){
    var min = valores[0]
    var max = valores[0]
    var sum = 0
    for valor in valores{
        if valor > max{
            max = valor
        } else if valor < min{
            min = valor
        }
        sum += valor
    }
    return (min, max, sum)
}
let estadistica = estadisticas (entrada: [5, 3, 100, 3, 9])
print(estadistica.min)
print(estadistica.max)
print(estadistica.sum)
print(estadistica.2)

```

3  
100  
120  
120

## FUNCIONES COMO TIPOS DE DATOS

- En Swift, las funciones también son tipos de datos, esto quiere decir que una función puede recibir como parámetro una función, o puede devolver una función como retorno

```

func hacerIncremento () -> ((Int) -> Int) {
    let cant = 5
    func sumar (valor:Int)->Int{
        return valor + cant
    }
    return sumar
}
let incremento = hacerIncremento() //valor de tipo función
print(incremento(8))

```

13

Función dentro función:

- Funciones que reciben otras funciones como parámetros

```

func hasAnyMatches(list: [Int], condition : (Int)->Bool)->Bool{
    for item in list {
        if (condition(item)){
            return true;
        }
    }
    return false
}
func lessThanTwenty (value:Int) -> Bool {
    if (value<20){
        return true
    }
    return false
}
let values = [11,43, 56, 12, 5, 6, 74]
print(hasAnyMatches(list: values, condition: lessThanTwenty))

```

true

## EJERCICIO 3

- Modifique el código anterior para que ahora se cree un arreglo donde estén los números que están por debajo de 20. Pruebe el funcionamiento e imprima el resultado en la consola

Respuesta:

```
func hasAnyMatches(list : [Int], condition : (Int)->Bool)->[Int]{  
    var menores=[Int]()  
    for item in list {  
        if(condition(item)){  
            menores.append(item)  
        }  
    }  
    return menores;  
}  
func lessThanTwenty(value:Int)->Bool{  
    if value<20{  
        return true  
    }  
    return false  
}  
let values=[11,43,56,12,5,6,74]  
print(hasAnyMatches(list: values, condition : lessThanTwenty))
```