[스파르타코딩클럽] 안드로이드 앱개발 기초반 - 4주차

매 주차 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

[수업 목표]

1. 새로운 액티비티를 시작하고 데이터를 전달한다.
2. 성격 유형 조회 앱을 완성한다.
   (1) 웹페이지를 보여주는 웹뷰를 사용한다.
3. 유형 검사 프로젝트를 시작한다.
   (1) SeekBar를 코드로 다뤄본다.

[목차]

모든 토글을 열고 닫는 단축키
Windows : `Ctrl` + `alt` + `t`
Mac : `⌘` + `⌥` + `t`

---

01. 4주차 오늘 배울 것

▼ 1) 4주차: '성격 유형 조회' 앱 만들기

앱에서 다른 화면으로 이동하는 방법과 앱 내에서 링크를 다루는 WebView를 배우고 '성격 유형 조회' 앱을 마무리할거에요.
+
그동안 배운 것들을 반복하며 '성격 유형 조회' 앱과 유사한 난이도의 '나만의 유형 검사' 앱을 시작해 봅니다.

02. 새로운 액티비티(Activity) 시작하기 (1)

▼ 2) 성격 유형 조회 프로젝트 생성하기 (복습)

> 👉 코틀린을 공부하느냐고 키보드만 사용하시다가 안드로이드 스튜디오 프로젝트 사용법을 잊으신 건 아니겠죠? 다시 한 번 프로젝트를 만들고 버튼에 동작 구현하는 것을 복습할 겸 빠르게 해볼게요.

(1) PersonalityType 이름의 프로젝트 생성

> 👉 1. File > New > New Project
> 2. Empty Activity
> 3. 프로젝트 이름 'PersonalityType' 설정
> 4. 폴더 경로 확인 및 개발언어 Kotlin 확인
> 5. Finish!

(2) 버튼 동작을 구현하기 위해 필요한 레이아웃은 템플릿을 활용할게요.

▼ [코드스니펫] - 성격 유형 조회 레이아웃 템플릿

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity">

  <TextView
    android:id="@+id/introTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    android:gravity="center"
    android:text="성격 유형 조회"
    android:textSize="32sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

  <ScrollView
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_marginTop="32dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/introTextView">

    <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:gravity="center"
      android:orientation="vertical">

      <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:orientation="horizontal">
```

```
        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="ISTJ" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="ISFJ" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="INFJ" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="INTJ" />
      </LinearLayout>

      <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:orientation="horizontal">

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="ISTP" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="ISFP" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="INFP" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="INTP" />
      </LinearLayout>

      <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:orientation="horizontal">

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="ESTP" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="ESFP" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="ENFP" />
```

```
        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="ENTP" />
      </LinearLayout>

      <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:orientation="horizontal">

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="ESTJ" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="ESFJ" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="ENFJ" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:text="ENTJ" />
      </LinearLayout>
    </LinearLayout>
  </ScrollView>

</androidx.constraintlayout.widget.ConstraintLayout>
```

👉 **[코드스니펫] - 성격 유형 조회 레이아웃 템플릿** 코드를 activity_main.xml에 붙여넣어주세요.

(3) 레이아웃에서 onClick 연결하기전에 MainActivity.kt 에 **[코드스니펫] - 성격 유형 조회 clickIstj 버튼** 코드를 위치에 맞춰서 추가해주세요.

▼ **[코드스니펫] - 성격 유형 조회 clickIstj 버튼**

```
fun clickIstj(view: View) {

}
```

```
class MainActivity : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
  }

  fun clickIstj(view: View) {

  }
}
```

👉 View라는 단어가 빨간색으로 나타난다구요?
마우스 커서를 올리고 Alt+Enter(Mac의 경우 Option+Enter)를 누르면 자동으로 추가가 되거나 만약 항목이 나온다면 추천 리스트에 있는 Import 를 선택해주세요.
View에 표시되었던 Error가 사라진 것을 확인할 수 있어요.

(4) 버튼이 동작하는지 확인하기 위해서 Toast 메세지를 추가해볼까요?

▼ **[코드스니펫] - 성격 유형 조회 Toast 메세지 추가하기**

```
Toast.makeText(view.context, "청렴결백한 논리주의자", Toast.LENGTH_LONG).show()
```

```
fun clickIstj(view: View) {
  Toast.makeText(view.context, "청렴결백한 논리주의자", Toast.LENGTH_LONG).show()
}
```

👉 Toast 단어가 빨간색으로 나타나면 Import 해주세요.

👉 그동안 그냥 추가만 해본 한 줄의 코드를 분석해볼까요?
코드를 분석할 때 괄호 단위로 분석하면 코드가 읽기 쉬워요. Toast.makeText(~~~).show() 어떤가요, 의미가 보이시나요?
네 맞아요! 토스트. 글자를 만들고. 보여준다.

그럼 생략했던 makeText 안의 내용을 봐볼까요?
view.context → 뷰의 맥락. 즉, 앱이 어떤 상태에 있는지를 토스트에게 알려주기 위해 전달하는 값이에요.
"청렴결백한 논리주의자" → 토스트에 표시하려는 컨텐츠를 전달하는 값이에요.
Toast.LENGTH_LONG → 길이는 길게. 토스트가 노출되는 시간을 설정해요. LENGTH_SHORT 도 있답니다.

이처럼 makeText는 컨텍스트, 컨텐츠, 노출시간 순서로 값을 받아요.

(5) activity_main.xml 파일로 이동해서 ISTJ가 적힌 버튼을 클릭해주세요. 레이아웃 에디터를 이용하신다면 Attributes 에서 onClick을 찾아 clickIstj를 선택해주세요.

👉 이제 레이아웃 수정은 마우스보다 코드가 편하신가요? 코드에서 ISTJ 버튼을 찾아서 `android:onClick="clickIstj"` 를 추가해주세요.

(6) 제대로 동작하는지 실행해서 확인해볼까요? 프로젝트를 'Run'해서 ISTJ 버튼을 누르니 토스트 메세지가 나오네요!

▼ 3) ISTJ 버튼 클릭하면 새로운 액티비티 띄우기

(1) 다른 액티비티를 띄우기 위해 새로운 액티비티 추가부터 해볼까요?

👉 1. Project 창에서 app 폴더 마우스 오른쪽 클릭
2. New > Activity > Empty Activity
3. 액티비티 이름 'IstjActivity' 설정
4. Finish!

(2) 새로운 액티비티의 레이아웃 'activity_istj'에 간단한 TextView 하나를 추가해주세요.

▼ **[코드스니펫] - 성격 유형 조회 ISTJ 화면 TextView**

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="청렴결백한 논리주의자"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

(3) 새로운 액티비티를 띄우기 위한 준비가 모두 끝났습니다. 그러면 MainActivity의 clickIstj 함수안에 새로운 화면을 시작하는 코드를 추가해볼게요.

▼ **[코드스니펫] - 성격 유형 조회 ISTJ 화면 시작 코드**

```
var intent = Intent(this, IstjActivity::class.java)
startActivity(intent)
```

```
fun clickIstj(view: View) {
  Toast.makeText(view.context, "청렴결백한 논리주의자", Toast.LENGTH_LONG).show()
  var intent = Intent(this, IstjActivity::class.java)
  startActivity(intent)
}
```

👉 Intent가 빨간색으로 Error가 나타나면 뭘 해야할까요?
네, 맞아요! Alt+Enter 를 눌러 Import 해주세요.

👉 추가한 코드를 설명드리면
1. intent라는 변수를 선언해서
2. intent 변수에 Intent(~~)로 만들어진 값을 저장한다.
3. Intent(~~)는 this 와 IstjActivity::class.java 를 이용해 만드네요.
4. 이렇게 만들어진 변수 intent를 이용해 startActivity() 함수를 실행한다.
정도로 이해할 수 있어요.

(4) Intent가 뭔지 파악하기 전에 동작하는지 한번 확인해볼까요? 프로젝트를 실행해서 ISTJ 버튼을 클릭해주세요. 글자가 가운데 적혀있는 화면이 나타나게 됩니다. 뒤로가기 버튼을 눌러보세요. 다시 메인화면이 나오는 것을 확인할 수 있습니다.

👉 Intent는 개별 앱 컴포넌트 간에 런타임 바인딩을 제공하는 객체에요.
풀어서 설명하면 인텐트(Intent)는 어떤 작업을 하겠다라는 '의도'를 표현하는 용도에요. 인텐트는 다양한 작업에서 활용가능하지만 주로 다른 액티비티를 시작할 때 사용해요.

위에서 사용한 코드(`Intent(this, IstjActivity::class.java)`)를 자세히 보면, this는 현재 코드가 적혀있는 곳을 의미합니다. 지금은 MainActivity.kt에 있으니 MainActivity를 의미해요.
즉, 인텐트를 생성할 때 첫번째 값은 출발지 액티비티, 두번째 값은 도착지 액티비티를 적어주면 됩니다.

👉 코틀린은 가독성을 높여주는 언어라고 한 것 기억하시나요? 코틀린에서는 this를 더 명확히 명시하기 위해서 `this@MainActivity` 같은 방식으로도 사용가능해요.
마치 댓글에 이름 언급하는 것과 같이 this뒤에 @를 이용해 명시하는 것을 권장하고 있어요.

👉 IstjActivity::class.java 에서 ::class.java 는 표현 방식에서 오는 타입의 차이라고 생각하시면 됩니다.
코드에서 `0`과 `"0"`을 보면 자연스럽게 앞의 0은 숫자(Int)고 뒤의 0은 글자(String)라고 생각되어지지 않나요? 마찬가지로 IstjActivity 와 IstjActivity::class.java는 같아보이지만 다른 타입이랍니다.
IstjActivity는 액티비티 객체를 ::class.java는 참조를 위한 class 값을 의미해요.

03. 새로운 액티비티(Activity) 시작하기 (2)

▼ 4) 데이터와 함께 액티비티 전환하기

(1) 액티비티를 시작할 때 데이터를 전달해주는 것 부터 시작해볼게요. 기존 MainActivity의 코드에 한줄만 추가하면 돼요.

👉 인텐트는 어떤 작업을 하겠다는 '의도'를 나타낸다고 했죠? 그 의도에 내가 주고 싶은 정보도 담아서 전달할 수 있어요. putExtra(key, value) 를 이용하면 됩니다. putExtra는 key, value 순서로 값을 받아요.
여기서 key는 다른 화면에서 데이터를 확인할 때 쓰는 값이고 value는 전달하려는 값이에요.
마치 우편함에 물건을 넣어두면 주인이 와서 우편함에서 가져갈 수 있는 것과 같아요.

▼ [코드스니펫] - 성격 유형 조회 ISTJ 데이터 전달하기

```
intent.putExtra("PersonalKey", "ISTJ")
```

```
//MainActivity.kt
fun clickIstj(view: View) {
  //Toast.makeText(view.context, "청렴결백한 논리주의자", Toast.LENGTH_LONG).show()
  var intent = Intent(this, IstjActivity::class.java)
  intent.putExtra("PersonalKey", "ISTJ")
  startActivity(intent)
}
```

👉 clickIstj에서 실행되는 Toast 메세지는 잠시 주석처리해둘게요.
// 두개를 쓰게되면 해당 코드 라인의 // 뒤에 부터는 주석처리가 돼요. 주석은 컴퓨터가 읽지 않는 코드라서 임시로 지워두거나 메세지를 기록할 때 유용하답니다.

(2) 데이터를 전달했으니 새로운 액티비티에서 데이터를 받아서 확인해볼게요. IstjActivity.kt 파일로 이동해주세요.

👉 인텐트에 담겨있는 데이터를 확인하는 방법은 getStringExtra(key)를 사용할거에요.
getStringExtra(key)를 써서 key에 담긴 정보를 가져올거에요.
정보가 제대로 가져와졌는지 Toast를 이용해서 확인해볼게요.

▼ [코드스니펫] - 성격 유형 조회 ISTJ 데이터 확인하기

```
var personalKey = intent.getStringExtra("PersonalKey")
Toast.makeText(this, personalKey, Toast.LENGTH_LONG).show()
```

```
//IstjActivity.kt
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setContentView(R.layout.activity_istj)

  var personalKey = intent.getStringExtra("PersonalKey")
  Toast.makeText(this, personalKey, Toast.LENGTH_LONG).show()
}
```

- 프로젝트를 실행해서 ISTJ라는 토스트 메세지가 나오는지 먼저 확인하고 코드를 분석해볼게요.

👉 코드를 분석해볼까요? this가 또 등장했네요!
this 어디서 봤는지 기억하시나요?
네 맞아요! MainActivity에서 인텐트를 만들때 쓰였어요.
여기서의 this는 인텐트와 의미상으로는 같지만 가리키는 것은 달라요. `Intent(this, IstjActivity::class.java)` 에서는 MainActivity.kt에서 this를 썼기때문에 this는 MainActivity를 의미하고, IstjActivity의 `Toast.makeText`에서 사용한 this는 IstjActivity를 의미해요.

Intent 와 동일하게 Toast.makeText도 둘 다 '어디서'가 중요하다보니 첫번째 값으로 this를 전달했어요. 인텐트는 '어디서' 시작하는 건지, 토스트는 '어디서' 보여줘야하는지를 표현하기 위해 this를 넣어준답니다.

👉 엇, 그러고보니 Toast.makeText에 전에는 view.context였는데 이번에는 this가 들어왔어요!
눈썰미가 좋으시네요. Intent와 Toast.makeText의 첫번째 값은 '컨텍스트(context)'를 전달해줘야해요.
컨텍스트는 현재 상태의 맥락을 의미한다했던 것 기억하시나요? 쉽게 표현하면 현재 어디에 있는지, 어느 화면에 있는가에 대한 정보라고 생각하시면 돼요.
이번에는 actvity의 컨텍스트를 쓰겠다고 `this`를 사용한 것이고 이전에는 view에 담겨있는 컨텍스트를 쓰겠다는 의미로 `view.context`를 사용한 것이랍니다.

👉 혹시나 view.context와 this가 같은것인지 궁금하실까봐 추가로 설명드리면 뷰에는 뷰를 생성할 때 사용된 컨텍스트가 들어있어요. 일반적으로 뷰가 속해있는 액티비티의 컨텍스트가 들어가게 됩니다.

(3) 전달 받은 값을 토스트 메세지가 아닌 TextView를 이용해 표시해볼까요? 3주차에 했던 것을 복습해볼게요.

👉 우선 안쓸 코드인 토스트 코드는 주석처리를 하고 시작할게요.

```
//IstjActivity.kt
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setContentView(R.layout.activity_istj)

  var personalKey = intent.getStringExtra("PersonalKey")
  //Toast.makeText(this, personalKey, Toast.LENGTH_LONG).show()
}
```

▼ **[코드스니펫] - 성격 유형 조회 TextView에 표시하기 TextView 선언**

```
private lateinit var textView: TextView
```

▼ **[코드스니펫] - 성격 유형 조회 TextView에 표시하기 onCreate에서 적용**

```
textView = findViewById(R.id.textView)
textView.text = personalKey
```

```
//IstjActivity.kt
class IstjActivity : AppCompatActivity() {
  private lateinit var textView: TextView

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_istj)

    var personalKey = intent.getStringExtra("PersonalKey")
    //Toast.makeText(this, personalKey, Toast.LENGTH_LONG).show()
    textView = findViewById(R.id.textView)
    textView.text = personalKey
  }
}
```

- 프로젝트를 실행하면 새로운 화면에서 TextView가 ISTJ로 바뀌어서 표현되네요!

👉 글자인 String 형태말고 숫자인 Int나 Boolean 값을 전달하고 싶으신가요?
값을 건네주는 putExtra(key, value)에서는 동일한 형태로 사용하면 돼요. 값을 받을 때는 getStringExtra 외에도 getIntExtra, getBooleanExtra 등이 있어서 값의 타입에 맞춰서 사용하면 된답니다.

04. 모든 성격 유형 버튼에 동작 구현하기 (1)

▼ 5) (복습) ISFJ 버튼 클릭하면 ISFJ 보여주기

👉 앞에서 진행한 내용을 빠르게 순서를 조금 바꿔서 복습해볼게요.

(1) 액티비티 추가하기

👉 1. Project 창에서 app 폴더 마우스 오른쪽 클릭
2. New > Activity > Empty Activity
3. 액티비티 이름 'IsfjActivity' 설정
4. Finish!

(2) 새로만들어진 activity_isfj.xml에 TextView 추가하기

▼ **[코드스니펫] - 성격 유형 조회 ISFJ 화면 TextView**

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="용감한 수호자"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

(3) ISFJ 버튼 동작 구현하기

👉 잠깐! clickIstj를 그대로 복사하는게 편하고 빠른 방법인건 맞아요. 개발자들도 종종 이전에 구현했던 코드를 복사해서 사용하기도 한답니다.
다만 지금은 학습의 과정이니 순서와 개념을 복습한다는 느낌으로 자료의 순서에 맞춰서 한스텝 한스텝 복습할게요.

▼ **[코드스니펫] - 성격 유형 조회 clickIsfj 버튼**

```
fun clickIsfj(view: View) {
  Toast.makeText(view.context, "용감한 수호자", Toast.LENGTH_LONG).show()
}
```

```
class MainActivity : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
  }

  fun clickIstj(view: View) {
    //Toast.makeText(view.context, "청렴결백한 논리주의자", Toast.LENGTH_LONG).show()
    var intent = Intent(this, IstjActivity::class.java)
    intent.putExtra("PersonalKey", "ISTJ")
    startActivity(intent)
  }

  fun clickIsfj(view: View) {
    Toast.makeText(view.context, "용감한 수호자", Toast.LENGTH_LONG).show()
  }
}
```

(4) MainActivity에 추가한 clickIsfj 함수를 ISFJ 버튼에 연결하기

👉 코드에서 ISFJ 버튼을 찾아서 `android:onClick="clickIsfj"` 를 추가해주세요.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="100dp"
    android:layout_margin="4dp"
    android:onClick="clickIsfj"
    android:text="ISFJ" />
```

👉 프로젝트를 실행해서 버튼이 잘 연결되었는지 확인해보세요. 개발하는 중간 중간 프로젝트를 실행해서 문제가 없는지 확인하는 것도 좋은 방법이에요. 다 만들어놓고 나중에 문제점을 찾는 것보다 좋은 방법이랍니다.
ISFJ 버튼을 누르면 '용감한 수호자'라는 메세지가 나오셨나요? 안나왔다면 (3)과 (4)에서 작업한 코드중 오타나 다른 부분이 있는지 확인해보세요.
토스트 메세지가 잘 나오셨다면 (5)을 진행해주세요.

(5) ISFJ 클릭하면 데이터와 함께 IsfjActivity 실행하기

▼ **[코드스니펫] - 성격 유형 조회 ISFJ 화면 시작 코드**

```
var intent = Intent(this, IsfjActivity::class.java)
intent.putExtra("PersonalKey", "ISFJ")
startActivity(intent)
```

```
//MainActivity.kt
fun clickIsfj(view: View) {
  //Toast.makeText(view.context, "용감한 수호자", Toast.LENGTH_LONG).show()
  var intent = Intent(this, IsfjActivity::class.java)
  intent.putExtra("PersonalKey", "ISFJ")
  startActivity(intent)
}
```

👉 프로젝트를 실행해서 '용감한 수호자' 글자가 적힌 화면이 나오는지 확인해보는 것도 좋아요.

(6) 액티비티에 전달 받은 데이터 토스트로 보여주기

▼ **[코드스니펫] - 성격 유형 조회 ISFJ 데이터 확인하기**

```
var personalKey = intent.getStringExtra("PersonalKey")
Toast.makeText(this, personalKey, Toast.LENGTH_LONG).show()
```

```
//IsfjActivity.kt
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setContentView(R.layout.activity_isfj)

  var personalKey = intent.getStringExtra("PersonalKey")
```

```
  Toast.makeText(this, personalKey, Toast.LENGTH_LONG).show()
}
```

👉 프로젝트를 실행해서 'ISFJ' 토스트가 나오는지 확인해보는 것도 좋아요.

👉 인텐트를 이용해 값을 전달할 때는 putExtra()로 전달하고 가져올 때는 intent.getStringExtra() 로 전달한다는 것 잊지마세요.
한개의 인텐트에 여러개의 extra를 추가(putExtra)하고 extra를 확인(getStringExtra)할 수 있어요. 그래서 key값이 필요하답니다.

(7) 마지막 단계! IsfjActivity의 TextView를 바꿔볼게요

▼ **[코드스니펫] - 성격 유형 조회 TextView에 표시하기 TextView 선언**

```
private lateinit var textView: TextView
```

▼ **[코드스니펫] - 성격 유형 조회 TextView에 표시하기 onCreate에서 적용**

```
textView = findViewById(R.id.textView)
textView.text = personalKey
```

```
//IsfjActivity.kt
class IsfjActivity : AppCompatActivity() {
  private lateinit var textView: TextView

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_isfj)

    var personalKey = intent.getStringExtra("PersonalKey")
    //Toast.makeText(this, personalKey, Toast.LENGTH_LONG).show()
    textView = findViewById(R.id.textView)
    textView.text = personalKey
  }
}
```

👉 완성! 2개의 액티비티를 만들었어요. 그런데 버튼은 총 16개네요. 14번을 더 이렇게 작업해야할까요?
아니에요. 반복을 줄여줄 좋은 방법이 있답니다. 6)에서 다뤄볼게요.

05. 모든 성격 유형 버튼에 동작 구현하기 (2)

▼ 6) 한 개의 액티비티로 16개 버튼 연결하기

👉 남은 14개를 반복하려니 막막하신가요? 물론, 새로운 액티비티를 14번 생성해서 연결하는 작업을 진행해도 괜찮아요. 간단하면서 시간만 쓰면 충분한 작업이에요.
그런데 단순반복하기엔 너무 힘들 것 같은데 반복문처럼 단순화할 수 없을까요?
가능해요! 이제부터는 패턴과 규칙찾는게 중요해져요.

👉 ISTJ와 ISFJ의 액티비티와 click 함수를 비교해보면 규칙성과 반복되는 부분이 보이지 않나요? 이 부분을 이용하면 한 개의 액티비티와 한개의 클릭함수로 16개 버튼을 다룰 수 있어요.

(1) 공통적인 패턴을 찾기 위해 코드 분석해볼게요. MainActivity.kt의 clickIstj 함수와 clickIsfj 함수를 비교해보세요.

```
//MainActivity.kt
...
  fun clickIstj(view: View) {
    //Toast.makeText(view.context, "청렴결백한 논리주의자", Toast.LENGTH_LONG).show()
    var intent = Intent(this, IstjActivity::class.java)
    intent.putExtra("PersonalKey", "ISTJ")
    startActivity(intent)
  }

  fun clickIsfj(view: View) {
    //Toast.makeText(view.context, "용감한 수호자", Toast.LENGTH_LONG).show()
    var intent = Intent(this, IsfjActivity::class.java)
    intent.putExtra("PersonalKey", "ISFJ")
    startActivity(intent)
  }
}
```

👉 공통점과 차이점이 보이시나요?
네 맞아요. 시작하는 Activity 이름과 전달하는 "PersonalKey"의 값이 달라요. 이것을 활용해서 한 개의 액티비티로 데이터를 띄워볼게요.

(2) "PersonalKey"에 데이터를 넣어서 IstjActivity를 실행하면 IstjActivity에서 "PersonalKey"에 있는 값을 활용해서 화면에 표시하게 만들어뒀어요. 그럼 한번 clickIstj 값을 일부 수정해서 확인해볼까요?

👉 clickIstj 함수에서 "ISFJ" 값을 "패턴이 보인다"로 변경해주세요.

```
//MainActivity.kt
...
  fun clickIstj(view: View) {
    //Toast.makeText(view.context, "청렴결백한 논리주의자", Toast.LENGTH_LONG).show()
    var intent = Intent(this, IstjActivity::class.java)
    intent.putExtra("PersonalKey", "패턴이 보인다")
    startActivity(intent)
  }
...
```

- 프로젝트를 실행해 ISTJ 버튼을 누르면 '패턴이 보인다'라는 글자가 적힌 화면이 나오네요.

👉 어떻게하면 버튼을 클릭하면 나오는 액티비티에서 클릭한 성격유형이 나오게 할 수 있을까요?
너무 복잡하고 어렵게 생각하지마세요. 단순합니다.
버튼에 있는 글자를 전달해서 액티비티의 TextView에 그려주면 되겠죠!

(3) MainActivity에서 "PersonalKey"에 전달해주는 값을 "ISTJ"를 명시해주지 않고 버튼에 있는 글을 이용해서 전달해볼게요.

👉 clickIstj 함수가 xml에서 `android:onClick="clickIstj"`으로 연결되어 클릭하면 동작하는 것은 알겠는데, clickIstj 함수에 적혀있는 view: View는 뭔지 궁금하지 않으신가요?
바로 onClick을 발생시킨 view의 정보를 담고 있어요. 그렇다면 clickIstj는 Button에 연결되어 있으니 view는 Button 정보가 담겨있겠네요. 그럼 이것을 활용해볼까요?
우선 view를 Button으로 바꿔서 변수에 저장해볼게요.

▼ [코드스니펫] - 성격 유형 조회 view를 Button으로 바꿔 관리하기

```
var button = view as Button
```

```
//MainActivity.kt
fun clickIstj(view: View) {
  var button = view as Button
  //Toast.makeText(view.context, "청렴결백한 논리주의자", Toast.LENGTH_LONG).show()
  var intent = Intent(this, IstjActivity::class.java)
  intent.putExtra("PersonalKey", "ISTJ")
  startActivity(intent)
}
```

👉 `as` 는 타입을 변경해서 사용하겠다라는 코틀린 문법이에요. 개발용어로는 타입 캐스팅 이라고 표현한답니다. 아무거나 원하는 타입으로 타입 캐스팅이 가능하냐구요?
가능한 경우에만 가능하답니다. View와 Button 이 가능한 이유는 Button이 View의 하위 개념이라서 가능해요. 전자제품과 노트북의 관계라고 생각하시면 돼요.

- 이제 view를 Button 형태로 코드에서 사용할 수 있게 되었어요. 버튼에 있는 text를 가져와서 "ISTJ"를 대체해볼게요.
  버튼의 텍스트를 가져오는 방식은 3주차때 한 텍스트뷰의 텍스트 가져오는 방식과 동일해요. `button.text`를 사용하시면 돼요.

```
//MainActivity.kt
fun clickIstj(view: View) {
  var button = view as Button
  //Toast.makeText(view.context, "청렴결백한 논리주의자", Toast.LENGTH_LONG).show()
  var intent = Intent(this, IstjActivity::class.java)
  intent.putExtra("PersonalKey", button.text)
  startActivity(intent)
}
```

- 프로젝트를 실행하면 button.text로 변경했음에도 이전과 똑같이 "ISTJ"가 나오네요!

(4) 거의 다왔어요! 버튼의 글자를 이용해서 다른 화면의 TextView에 나타나게 했어요. 이제 모든 버튼에 `android:onClick="clickIstj"`를 추가해주면 끝난답니다.

> 👉 단순 반복을 대체할 방법을 만들려면 만들 수 있지만 현재 이상의 작업은 단순 반복을 하는 시간보다 대체할 방법을 고민하는데 시간을 더 사용하게 될 수도 있어요. 시간뿐만 아니라 코드의 복잡도를 높여서 가독성을 해칠 수도 있어서 나중에 다시보면 무슨 코드인지 이해하는데 어려울 수 있어요.

▼ [코드스니펫] - 성격 유형 조회 레이아웃에 clickIstj 적용

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity">

  <TextView
    android:id="@+id/introTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    android:gravity="center"
    android:text="성격 유형 조회"
    android:textSize="32sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

  <ScrollView
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_marginTop="32dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/introTextView">

    <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:gravity="center"
      android:orientation="vertical">

      <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:orientation="horizontal">

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:onClick="clickIstj"
          android:text="ISTJ" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:onClick="clickIstj"
          android:text="ISFJ" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:onClick="clickIstj"
          android:text="INFJ" />

        <Button
```

```
        android:layout_width="wrap_content"
        android:layout_height="100dp"
        android:layout_margin="4dp"
        android:onClick="clickIstj"
        android:text="INTJ" />
    </LinearLayout>

    <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:gravity="center"
      android:orientation="horizontal">

      <Button
        android:layout_width="wrap_content"
        android:layout_height="100dp"
        android:layout_margin="4dp"
        android:onClick="clickIstj"
        android:text="ISTP" />

      <Button
        android:layout_width="wrap_content"
        android:layout_height="100dp"
        android:layout_margin="4dp"
        android:onClick="clickIstj"
        android:text="ISFP" />

      <Button
        android:layout_width="wrap_content"
        android:layout_height="100dp"
        android:layout_margin="4dp"
        android:onClick="clickIstj"
        android:text="INFP" />

      <Button
        android:layout_width="wrap_content"
        android:layout_height="100dp"
        android:layout_margin="4dp"
        android:onClick="clickIstj"
        android:text="INTP" />
    </LinearLayout>

    <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:gravity="center"
      android:orientation="horizontal">

      <Button
        android:layout_width="wrap_content"
        android:layout_height="100dp"
        android:layout_margin="4dp"
        android:onClick="clickIstj"
        android:text="ESTP" />

      <Button
        android:layout_width="wrap_content"
        android:layout_height="100dp"
        android:layout_margin="4dp"
        android:onClick="clickIstj"
        android:text="ESFP" />

      <Button
        android:layout_width="wrap_content"
        android:layout_height="100dp"
        android:layout_margin="4dp"
        android:onClick="clickIstj"
        android:text="ENFP" />

      <Button
        android:layout_width="wrap_content"
        android:layout_height="100dp"
        android:layout_margin="4dp"
        android:onClick="clickIstj"
        android:text="ENTP" />
    </LinearLayout>

    <LinearLayout
      android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:gravity="center"
        android:orientation="horizontal">

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:onClick="clickIstj"
          android:text="ESTJ" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:onClick="clickIstj"
          android:text="ESFJ" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:onClick="clickIstj"
          android:text="ENFJ" />

        <Button
          android:layout_width="wrap_content"
          android:layout_height="100dp"
          android:layout_margin="4dp"
          android:onClick="clickIstj"
          android:text="ENTJ" />
      </LinearLayout>
    </LinearLayout>
  </ScrollView>

</androidx.constraintlayout.widget.ConstraintLayout>
```

👉 여러분의 시간은 소중하니까 단순 작업은 코드 스니펫을 활용할게요. **[코드스니펫] - 성격 유형 조회 레이아웃에 clickIstj 적용** 코드를 activity_main.xml에 붙여넣어주세요.

- 프로젝트를 실행해서 버튼을 눌러보세요. 버튼에 맞춰서 글자가 나타네요!

▼ 7) 불필요한 코드 정리하기

- 동작이 구현되서 끝난 것과 다름이 없지만 불필요한 코드를 정리해볼게요.

👉 코드정리는 나중에 코드를 파악하거나 다른 사람과 협업할 때 매우 중요해요. 요리를 처음할 때 이것저것 꺼내고 어질러진 상태로 식사를 마치면 다음 요리할 때 굉장히 힘들어지겠죠? 코딩도 마찬가지라서 정리가 코딩하는 것 만큼 코드를 정리하는 것도 중요해요!

(1) 우선 사용하지 않는 clickIsfj를 과감하게 지울게요.

```
//MainActivity.kt
...
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
  }

  fun clickIstj(view: View) {
    var button = view as Button
```

```
    //Toast.makeText(view.context, "청렴결백한 논리주의자", Toast.LENGTH_LONG).show()
    var intent = Intent(this, IstjActivity::class.java)
    intent.putExtra("PersonalKey", button.text)
    startActivity(intent)
  }

}
```

(2) 주석도 불필요하니까 지워버립시다.

```
//MainActivity.kt
...
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
  }

  fun clickIstj(view: View) {
    var button = view as Button

    var intent = Intent(this, IstjActivity::class.java)
    intent.putExtra("PersonalKey", button.text)
    startActivity(intent)
  }
}
```

(3) clickIstj라는 이름이 애매하지 않나요? clickTypeButton으로 바꿔볼게요.

> 👉 힘들게 추가했는데 하나씩 다시 수정해야하냐구요?
> 아니에요! 안드로이드 스튜디오에서 좋은 기능을 제공해준답니다.
>
> clickIstj에 커서를 올리고 우클릭해주세요. 메뉴에 Refactor > Rename 을 눌러 이름을 변경할 수 있어요. 단축키는 **Shift + F6** 이랍니다.

```
//MainActivity.kt
...
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main3)
  }

  fun clickTypeButton(view: View) {
    var button = view as Button
    var intent = Intent(this, IstjActivity::class.java)
    intent.putExtra("PersonalKey", button.text)
    startActivity(intent)
  }
}
```

(4) 이번에는 IstjActivity의 이름을 DetailActivity로 바꿔볼게요. 탐색창의 IstjActivity.kt 파일에 커서를 올리고 우클릭해주세요. 메뉴에 Refactor > Rename 을 누르면 Rename 창이 나옵니다. DetailActivity로 변경하고 'Refactor' 버튼을 눌러주세요.

> 👉 activity_istj.xml 파일도 동일하게 activity_detail.xml 로 변경하면 좋겠죠?

👉 이렇게 코드를 정리하는 과정을 '리팩토링'이라고 불러요. 사실 정말정말 사소한 작업이다보니 리팩토링이라고 부르긴 애매하긴 하지만 그래도 불필요한 코드를 제거하고 코드의 가독성을 높이는 작업도 리팩토링의 일부긴 합니다.

리팩토링을 꼭 해야하냐구요?
꼭은 아니지만 하면 좋아요. 마치 '짐 정리를 해야하나요?'와 비슷한 질문이랍니다. 혼자쓰는 방에서 물건이 어디있는지 찾아쓸 수 있고 생활에 불편함이 없다면 정리를 할 필요성이 크지 않지만, 여러명이쓰는 공용 공간에서 물건을 제자리에 두지 않으면 다른 사람들이 사용하기 어려운 것과 같아요. 혼자 개발할 땐 알아두면 좋은 정도지만 여러 명이 개발을 할 땐 리팩토링하지 않으면 힘들답니다.

- 단순 반복으로 16개 유형별 액티비티 파일과 레이아웃 파일이 생성되고 16개 유형별 함수가 생성될 뻔 했는데, 패턴을 찾아서 적용해서 한 개로 성격 유형을 조회할 수 있게 되었어요! 이제 실제로 성격을 조회할 수 있게 해볼까요?

06. 앱에서 웹페이지를 보여주는 웹뷰(WebView)다뤄보기

▼ 8) 웹뷰를 이용해 웹페이지 띄우기

👉 성격 유형에 대한 정보를 글과 이미지를 만들어서 직접 레이아웃을 구성하면 사용자들이 성격 유형에 대한 정보를 조회할 수 있어요.
하지만 레이아웃을 구성하는 것은 그동안 많이 연습해왔으니 이번에는 새로운 뷰인 웹뷰를 사용해볼게요.
웹뷰는 앱 내에서 웹페이지를 조회할 수 있게 해주는 뷰랍니다. 앱 내에서 성격 유형을 구글 검색 결과를 이용해 보여주도록 할게요.

(1) 성격 유형 상세 액티비티(DetailActivity)의 레이아웃에 웹뷰를 넣기 위한 공간을 만들기 위해 텍스트뷰의 위치를 조절해볼게요. id/textView를 찾아서 `app:layout_constraintBottom_toBottomOf="parent"` 속성을 지워주세요.

```
<!-- activity_detail.xml (이름 변경 안한 경우 activity_istj.xml) -->
...
  <TextView
      android:id="@+id/textView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="청렴결백한 논리주의자"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintLeft_toLeftOf="parent"
      app:layout_constraintRight_toRightOf="parent"
      app:layout_constraintTop_toTopOf="parent" />
...
```

(2) 텍스트뷰 아래에 웹뷰를 추가해볼게요. 다른 텍스트뷰, 이미지뷰와 동일하게 웹뷰가 미리 준비되어 있어요. 동일하게 추가를 해주시면 됩니다. 텍스트뷰 아래에 들어갈 수 있도록 위치를 설정하겠습니다.

▼ [코드스니펫] - 레이아웃에 웹뷰 추가하기

```
<WebView
    android:id="@+id/webview"
    android:layout_width="match_parent"
    android:layout_height="0dp"
```

```
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@id/textView" />
```

👉 실행해보면 텍스트뷰의 위치가 바뀐 것 말고는 아무런 차이가 없네요. 웹뷰에게 어떤 웹페이지를 보여줄지 아직 알려주지 않아서 그래요!

(3) URL을 이용하여 웹뷰에서 웹페이지를 띄워볼게요.

👉 구글 페이지(https://www.google.com/)를 이용해서 웹뷰가 잘 동작하는지 확인해볼게요.
웹뷰도 텍스트뷰처럼 비슷한 방법으로 코틀린에서 활용할 수 있어요.

▼ [코드스니펫] - 웹뷰에 안드로이드 개발자 페이지 띄우기

```
var myWebView: WebView = findViewById(R.id.webview)
myWebView.webViewClient = WebViewClient()
myWebView.loadUrl("https://www.google.com/")
```

```
//DetailActivity.kt
class DetailActivity : AppCompatActivity() {
  private lateinit var textView: TextView

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_detail)

    var personalKey = intent.getStringExtra("PersonalKey")
    //Toast.makeText(this, personalKey, Toast.LENGTH_LONG).show()
    textView = findViewById(R.id.textView)
    textView.text = personalKey

    var myWebView: WebView = findViewById(R.id.webview)
    myWebView.webViewClient = WebViewClient()
    myWebView.loadUrl("https://www.google.com/")
  }
}
```

👉 WebView와 WebViewClient가 빨간색일 땐 Alt+Enter 를 눌러 Import 진행하는 것 잊지마세요!

👉 앗, 실행을 해보니 안드로이드 개발자 페이지는 나오지 않고 'Webpage not available'이라고 나오네요.
당황하지말고 메세지를 조금 더 읽어볼까요?
'net::ERR_CACHE_MISS'가 발생했다는데 뭔지 모르겠네요. 구글에 '안드로이드 웹뷰 net::ERR_CACHE_MISS'로 검색해봅시다.

(4) 웹뷰를 표시하기 위해서는 앱이 인터넷 사용 권한이 있어야해요. 앱에 인터넷 사용 권한을 추가해볼게요.

👉 웹뷰 작업은 인터넷을 사용할 수 있어야하는데 앱에 인터넷 사용권한이 없어서 웹페이지를있어야만 가능해요.

▼ **[코드스니펫] - 앱에 인터넷 권한 요청 추가하기**

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

▼ **[코드스니펫] - 앱에 일반 텍스트 네트워크 트래픽 설정하기**

```
android:usesCleartextTraffic="true"
```

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

  <application
    ...
    android:usesCleartextTraffic="true"
    ...
  >
    <activity ...>
    ...
  </application>
</manifest>
```

👉 AndroidManifest.xml 매니페스트 파일에 인터넷 권한을 요청하겠다고 명시를 할게요. 웹뷰 뿐만 아니라 앱에서 인터넷을 사용하기 위해서는 매니페스트 파일에 꼭 권한을 추가해야한답니다.

- 프로젝트를 실행시켜서 성격 유형 버튼을 누르면 웹뷰에 구글페이지가 나오는 것을 확인할 수 있어요!

!! 앗, 여전히 'Webpage not available' 에러가 발생하시나요? 설치되어 있는 앱을 한번 삭제하시고 다시 프로젝트를 실행시켜주세요!

▼ 9) (함께하기) 구글 검색 결과를 활용해 조회 앱 완성하기

👉 loadUrl을 이용하면 원하는 웹페이지를 띄울 수 있네요! 그럼 이제 조건문을 활용해 성격 유형별로 다른 웹페이지가 보이게 만들어볼게요.

▼ **[코드스니펫] - 성격 유형별 검색 결과 URL**

```
ISTJ > https://www.google.com/search?q=istj
ISFJ > https://www.google.com/search?q=isfj
INFJ > https://www.google.com/search?q=infj
INTJ > https://www.google.com/search?q=intj
ISTP > https://www.google.com/search?q=istp
ISFP > https://www.google.com/search?q=isfp
INFP > https://www.google.com/search?q=infp
INTP > https://www.google.com/search?q=intp
ESTP > https://www.google.com/search?q=estp
ESFP > https://www.google.com/search?q=esfp
ENFP > https://www.google.com/search?q=enfp
```

```
ENTP > https://www.google.com/search?q=entp
ESTJ > https://www.google.com/search?q=estj
ESFJ > https://www.google.com/search?q=esfj
ENFJ > https://www.google.com/search?q=enfj
ENTJ > https://www.google.com/search?q=entj
```

(1) 만약 성격값이 ISTJ이면 구글에서 ISTJ 검색 결과페이지를 보여주고 아니면 구글 홈을 보여주도록 구현해볼게요.

```
//DetailActivity.kt
...
    var myWebView: WebView = findViewById(R.id.webview)
    myWebView.webViewClient = WebViewClient()
    myWebView.loadUrl("https://www.google.com/")
    if(personalKey == "ISTJ") {
      myWebView.loadUrl("https://www.google.com/search?q=istj")
    } else {
      myWebView.loadUrl("https://www.google.com/")
    }
...
```

(2) ISTJ 버튼은 바로 구글에서 ISTJ 검색 결과가 나오나요? 그렇다면 ISFJ도 추가해볼게요.

```
//DetailActivity.kt
...
    var myWebView: WebView = findViewById(R.id.webview)
    myWebView.webViewClient = WebViewClient()
    if(personalKey == "ISTJ") {
      myWebView.loadUrl("https://www.google.com/search?q=istj")
    } else if (personalKey == "ISFJ") {
      myWebView.loadUrl("https://www.google.com/search?q=isfj")
    } else {
      myWebView.loadUrl("https://www.google.com/")
    }
...
```

(3) 이제 남은 14개 성격 유형들도 동일한 작업을 진행해 봅시다.

👉 어쩔 수 없지만 때로는 개발하다보면 단순작업도 필요하답니다.

▼ [코드스니펫] - 성격 유형 조회 구글 검색 결과 페이지 활용하기 (조건문 완성)

```
if (personalKey == "ISTJ") {
  myWebView.loadUrl("https://www.google.com/search?q=istj")
} else if (personalKey == "ISFJ") {
  myWebView.loadUrl("https://www.google.com/search?q=isfj")
} else if (personalKey == "INFJ") {
  myWebView.loadUrl("https://www.google.com/search?q=infj")
} else if (personalKey == "INTJ") {
  myWebView.loadUrl("https://www.google.com/search?q=intj")
} else if (personalKey == "ISTP") {
  myWebView.loadUrl("https://www.google.com/search?q=istp")
} else if (personalKey == "ISFP") {
  myWebView.loadUrl("https://www.google.com/search?q=isfp")
} else if (personalKey == "INFP") {
  myWebView.loadUrl("https://www.google.com/search?q=infp")
} else if (personalKey == "INTP") {
  myWebView.loadUrl("https://www.google.com/search?q=intp")
} else if (personalKey == "ESTP") {
  myWebView.loadUrl("https://www.google.com/search?q=estp")
} else if (personalKey == "ESFP") {
  myWebView.loadUrl("https://www.google.com/search?q=esfp")
```

```
} else if (personalKey == "ENFP") {
  myWebView.loadUrl("https://www.google.com/search?q=enfp")
} else if (personalKey == "ENTP") {
  myWebView.loadUrl("https://www.google.com/search?q=entp")
} else if (personalKey == "ESTJ") {
  myWebView.loadUrl("https://www.google.com/search?q=estj")
} else if (personalKey == "ESFJ") {
  myWebView.loadUrl("https://www.google.com/search?q=esfj")
} else if (personalKey == "ENFJ") {
  myWebView.loadUrl("https://www.google.com/search?q=enfj")
} else if (personalKey == "ENTJ") {
  myWebView.loadUrl("https://www.google.com/search?q=entj")
} else {
  myWebView.loadUrl("https://www.google.com/")
}
```

- 프로젝트를 실행해서 버튼 별로 검색페이지가 다르게 동작하는지 확인해주세요. 잘 동작하나요? 이런식으로 조건문을 활용하면 같은 액티비티에서도 동작을 다르게 처리할 수 있어요.

> 👉 조건문에서 loadUrl("...")안에 들어가는 링크를 변경하면 사용자에게 더 편리한 성격 유형 조회 앱을 구성할 수 있어요.
> 웹뷰가 아니라 TextView, ImageView를 이용해 레이아웃을 구성하고 조건문 안에서 myWebView.loadUrl 하듯이 텍스트뷰나 이미지뷰를 관리한다면 나만의 개성있는 성격 유형 조회앱을 구성할 수 있겠죠?

> 🔥 간단한 성격 유형 조회 앱을 완성했어요!

07. 유형 검사 앱 검사 화면 만들기

▼ 10) 유형 검사 앱 기본 설정하기

> 👉 마지막으로 그동안 배운 내용들을 복습하고 활용하면서 5주차 때 배포할 유형 검사 앱을 만들어볼게요!

(1) TypeTest 이름의 프로젝트 생성

> 👉 배포를 위한 앱을 만들기 위해서 프로젝트 생성할 때 Package name 을 신경써서 만들어볼게요.
> Package name은 id처럼 사용되기에 중복이되면 안돼요.
> package이름을 작명해주세요. 소문자 영어만 가능하다는 점 잊지마세요!
> 일반 적인 규칙은 com.(중복이 어려울 이름).(프로젝트이름)으로 패키지명을 만든답니다.
> 기본 값인 `com.example.typetest`나 다른 사람들이 사용할만한 `com.app.typetest` 같은 패키지명을 사용하시게되면 앱 배포가 어려워져요.

> 👉 1. File > New > New Project
> 2. Empty Activity
> 3. 프로젝트 이름 'TypeTest' 설정
> 4. (중요) 패키지이름 변경
> 5. 폴더 경로 확인 및 개발언어 Kotlin 확인
> 6. Finish!

(2) 표시되는 앱 이름을 변경해볼게요.

> 👉 앱 이름은 'app > res > values'에 위치한 strings.xml 파일을 열어서 'app_name' 값을 변경하면 돼요.
> 원하시는 이름으로 변경해주세요.

```
<resources>
  <string name="app_name">나만의 유형 검사</string>
</resources>
```

(3) 앱 아이콘을 변경해볼게요.

> !! 사용하고 싶은 이미지가 없으시거나 기존 아이콘도 충분히 괜찮으신가요?
> 그렇다면 앱 아이콘 변경은 따라하지 않으셔도 괜찮아요! 아이콘 변경 방법만 눈으로 확인하고 갈게요.

> 👉 'app > res > mipmap'에 위치한 ic_launcher_round 디렉토리를 우클릭해서 Delete...를 눌러주세요.
> Delete Anyway를 누르면 기존 아이콘이 삭제돼요.
> 동일한 방법으로 'app > res > mipmap'의 ic_launcher도 삭제해주세요.

> 👉 기존 파일을 모두 삭제하셨나요? 그러면 앱 아이콘으로 사용하고 싶은 이미지를 mipmap에 마우스로 끌어서 넣어주세요. 추가한 이미지 파일을 우클릭해서 'Refactor > Rename'을 이용해 이름을 'ic_launcher'로 변경해주세요.
> 'ic_launcher' 파일을 복사해서 'ic_launcher_round'라는 이름으로 추가해주세요.

- 프로젝트 기본 설정이 끝났어요. 프로젝트 생성, 앱 이름 변경, 앱 아이콘 변경은 한번 진행하면 변경할 일이 거의 없는 작업다보니 방법을 외우실 필요는 없어요. 느낌만 기억해두셨다가 필요할 때 검색하면 된답니다.

▼ 11) SeekBar 값 다뤄보기

(1) 유형 검사 레이아웃은 템플릿을 활용할게요.

▼ [코드스니펫] - 유형 검사 레이아웃 템플릿

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MainActivity">

  <androidx.core.widget.NestedScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical"
      android:padding="32dp">

      <TextView
        android:id="@+id/title"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:gravity="center"
        android:padding="8dp"
        android:text="나는 어떤 유형일까?"
        android:textSize="20sp" />

      <EditText
        android:id="@+id/nameEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:hint="이름을 입력해주세요." />

      <androidx.constraintlayout.widget.ConstraintLayout
        android:id="@+id/layoutQ1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:paddingTop="16dp"
        android:paddingBottom="16dp">

        <TextView
          android:id="@+id/titleQ1"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="나는 고민보다 행동부터 한다."
          android:textSize="16sp"
          app:layout_constraintEnd_toEndOf="parent"
          app:layout_constraintStart_toStartOf="parent"
          app:layout_constraintTop_toTopOf="parent" />

        <TextView
          android:id="@+id/disagreeQ1"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_marginTop="16dp"
          android:text="그렇지 않다"
          app:layout_constraintBottom_toBottomOf="parent"
          app:layout_constraintStart_toStartOf="parent"
          app:layout_constraintTop_toBottomOf="@id/titleQ1" />

        <SeekBar
          android:id="@+id/barQ1"
          android:layout_width="0dp"
          android:layout_height="wrap_content"
          android:layout_margin="4dp"
          android:max="5"
          android:min="1"
          android:progress="3"
          app:layout_constraintBottom_toBottomOf="@id/disagreeQ1"
          app:layout_constraintEnd_toStartOf="@id/agreeQ1"
          app:layout_constraintStart_toEndOf="@id/disagreeQ1"
          app:layout_constraintTop_toTopOf="@id/disagreeQ1" />

        <TextView
          android:id="@+id/agreeQ1"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="그렇다"
          app:layout_constraintBottom_toBottomOf="@id/disagreeQ1"
          app:layout_constraintEnd_toEndOf="parent"
          app:layout_constraintTop_toTopOf="@id/disagreeQ1" />

      </androidx.constraintlayout.widget.ConstraintLayout>

      <Button
        android:id="@+id/submitType"
        android:layout_width="match_parent"
        android:layout_height="64dp"
        android:layout_marginTop="16dp"
        android:text="나의 유형 확인하기" />
    </LinearLayout>
  </androidx.core.widget.NestedScrollView>

</androidx.constraintlayout.widget.ConstraintLayout>
```

👉 **[코드스니펫] - 유형 검사 레이아웃 템플릿** 코드를 activity_main.xml에 붙여넣어주세요.

(2) '나의 유형 확인하기' 버튼 동작 관리를 위한 clickSubmit 함수를 생성할게요.

▼ **[코드스니펫] - 유형 검사 clickSubmit 함수**

```
fun clickSubmit(view: View) {
  Toast.makeText(view.context, "실행력 넘치는 개발자", Toast.LENGTH_SHORT).show()
}
```

```
class MainActivity : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
  }

  fun clickSubmit(view: View) {
    Toast.makeText(view.context, "실행력 넘치는 개발자", Toast.LENGTH_SHORT).show()
  }
}
```

(3) xml의 submitType의 id를 갖고 있는 버튼에 onClick 함수를 연결해주세요.

```
<Button
  android:id="@+id/submitType"
  android:layout_width="match_parent"
  android:layout_height="64dp"
  android:layout_marginTop="16dp"
  android:onClick="clickSubmit"
  android:text="나의 유형 확인하기" />
```

👉 프로젝트를 실행해서 버튼이 잘 연동되었는지 한번 확인해주세요.

(4) 현재 선택된 SeekBar 값을 가져와서 토스트로 보여줄게요.

👉 SeekBar에는 progress 값을 이용하면 현재 선택된 값을 확인할 수 있어요.
레이아웃 템플릿을 확인해보면 현재 min은 1, max는 5로 설정을 해둬서 1~5만 나오게 됩니다.

```
class MainActivity : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
  }

  fun clickSubmit(view: View) {
    var barQ1: SeekBar = findViewById(R.id.barQ1)
    var answer1 = barQ1.progress
    Toast.makeText(view.context, "Q1: ${answer1}", Toast.LENGTH_SHORT).show()
  }
}
```

▼ **[코드스니펫] - 유형 검사 SeekBar 값 보여주기 (완성)**

```
var barQ1: SeekBar = findViewById(R.id.barQ1)
var answer1 = barQ1.progress
Toast.makeText(view.context, "Q1: ${answer1}", Toast.LENGTH_SHORT).show()
```

👉 "Q1: ${answer1}" 형태는 코틀린 문법으로 문자열 템플릿(String Templates)라고 불리는 문법이에요.
String을 변수를 이용해서 표현하고 싶을 때 사용하면 된답니다.
사용법은 String을 만드실 때 `${}` 안에 변수를 넣으면 돼요.

(5) 조건문을 이용해서 숫자가 아닌 문장으로 표현해볼게요.

```
class MainActivity : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
  }

  fun clickSubmit(view: View) {
    var barQ1: SeekBar = findViewById(R.id.barQ1)
    var answer1 = barQ1.progress
    if (answer1 == 1) {
      Toast.makeText(view.context, "앱을 설계하는 개발자", Toast.LENGTH_SHORT).show()
    } else if (answer1 == 2) {
      Toast.makeText(view.context, "로직을 고민하는 개발자", Toast.LENGTH_SHORT).show()
    } else if (answer1 == 3) {
      Toast.makeText(view.context, "밸런스형 개발자", Toast.LENGTH_SHORT).show()
    } else if (answer1 == 4) {
      Toast.makeText(view.context, "만드는게 좋은 개발자", Toast.LENGTH_SHORT).show()
    } else if (answer1 == 5) {
      Toast.makeText(view.context, "실행력 넘치는 개발자", Toast.LENGTH_SHORT).show()
    }
  }
}
```

▼ **[코드스니펫] - 유형 검사 수치 활용하기 (완성)**

```
if (answer1 == 1) {
  Toast.makeText(view.context, "앱을 설계하는 개발자", Toast.LENGTH_SHORT).show()
} else if (answer1 == 2) {
  Toast.makeText(view.context, "로직을 고민하는 개발자", Toast.LENGTH_SHORT).show()
} else if (answer1 == 3) {
  Toast.makeText(view.context, "밸런스형 개발자", Toast.LENGTH_SHORT).show()
} else if (answer1 == 4) {
  Toast.makeText(view.context, "만드는게 좋은 개발자", Toast.LENGTH_SHORT).show()
} else if (answer1 == 5) {
  Toast.makeText(view.context, "실행력 넘치는 개발자", Toast.LENGTH_SHORT).show()
}
```

👉 프로젝트를 실행해서 SeekBar를 움직이면서 버튼을 눌러보세요.
값이 다르게 나오나요? 마지막 5주차에는 질문들을 추가하고 값을 계산해서 유형을 구분하는 것을 배워볼거에요. 그리고 앱에 광고와 데이터분석 기능을 넣어 유형 검사 앱을 완성하면 기다리던 앱 배포까지 진행하게 됩니다!

08. 4주차 끝 & 숙제 설명

자기소개 앱의 버튼을 누르면 새로운 액티비티를 띄워볼게요.

기능: 생각확인하기 버튼을 눌렀을 때 '안드로이드 앱 개발은 즐거워' 글자가 적힌 새로운 액티비티가 나타나게 해주세요.
* 3주차 과제를 못하신 분들은 빈 프로젝트에 MainActivity에 Button을 추가해서 진행해주세요.

▼ 예시 화면

- '생각확인하기' 버튼을 누르면 이미지처럼 새로운 액티비티가 나오면 돼요!

HW. 4주차 숙제 답안

▼ **[코드스니펫] 4주차 숙제 코드**

```
var intent = Intent(this, ThinkActivity::class.java)
startActivity(intent)
```