

1. W tym ćwiczeniu dowiesz się więcej o głównych składnikach aplikacji na Androida i dodasz prostą interaktywność do aplikacji za pomocą przycisku.

- Przeglądanie pliku Kotlin `MainActivity` i plików activity's layout.
- Edycja activity's layout w XML.
- Dodawanie elementu `Button` do activity's layout.
- Wyodrębnianie hardcoded strings do pliku zasobów string.
- Zaimplementowanie metody obsługi kliknięć, aby wyświetlać komunikaty na ekranie, gdy użytkownik dotknie przycisk `Button`.

2. Opis Zadania

W tym ćwiczeniu tworzymy nowy projekt aplikacji realizującej rzut kostką i dodajesz podstawową interaktywność za pomocą przycisku. Każde kliknięcie przycisku powoduje zmianę wartości wyświetlanego tekstu. Ostateczna aplikacja ma wyglądać następująco:



3. Zadanie1: poznaj pliki aktywności i układu

W tym zadaniu koncentrujesz się na dwóch najważniejszych plikach tworzących Twoją aplikację: pliku `MainActivity` Kotlin i pliku układu `activity_main.xml`.

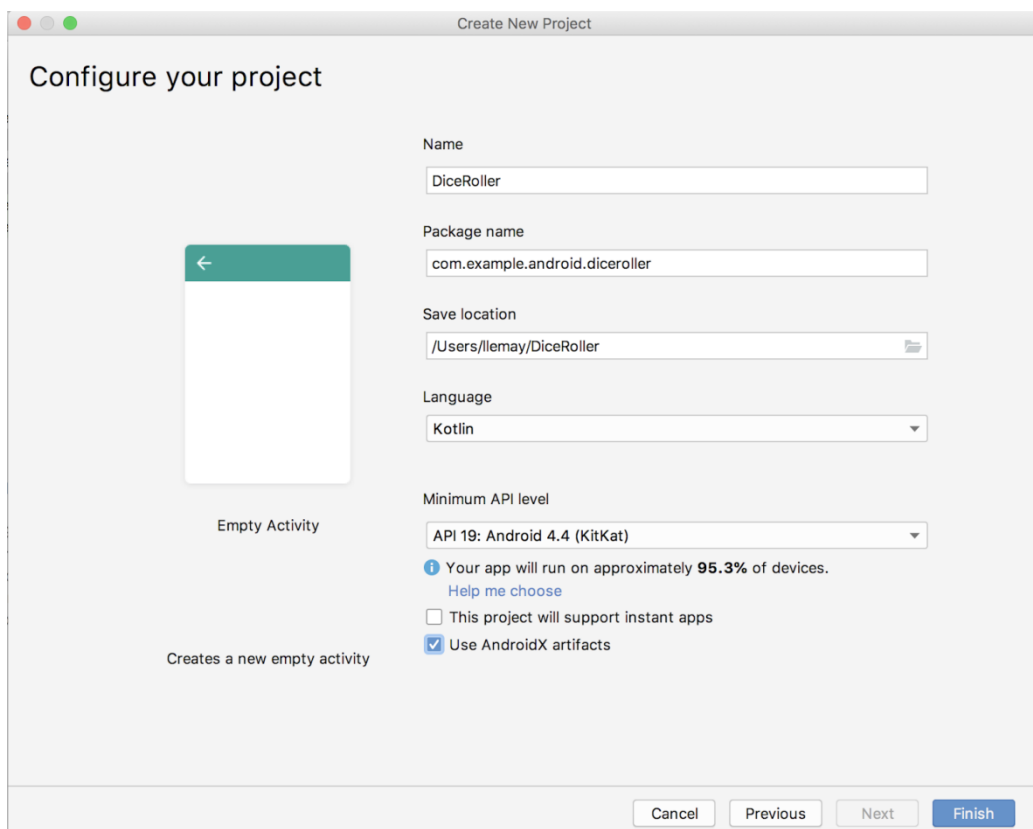
Krok 1: MainActivity

`MainActivity` jest przykładem Aktywności. Aktywność to podstawowa klasa Androida, która rysuje interfejs użytkownika aplikacji na Androida i odbiera zdarzenia wejściowe. Po uruchomieniu aplikacja uruchamia Aktywność określone w pliku `AndroidManifest.xml`.

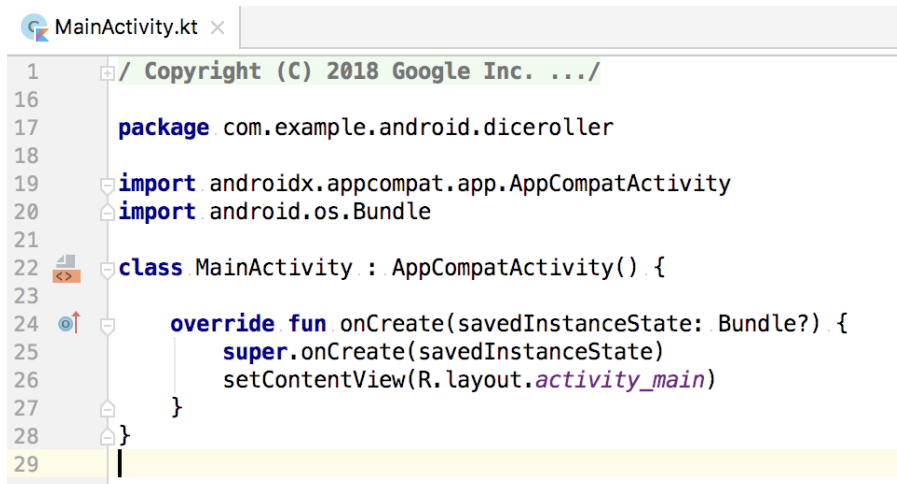
Wiele języków programowania definiuje metodę `main` do uruchamiania programu. Aplikacje na Androida nie mają metody `main`. Zamiast tego plik `AndroidManifest.xml` wskazuje, że `MainActivity` należy uruchomić, gdy użytkownik stuknie ikonę uruchamiania aplikacji. Aby uruchomić działanie, system operacyjny Android wykorzystuje informacje zawarte w manifestcie, aby skonfigurować środowisko dla aplikacji i zbudować `MainActivity`. Następnie `MainActivity` wykonuje kilka ustawień po kolei.

Każde `activity` (Aktywność) ma powiązany plik układu `Layout`. Aktywność i układ są połączone procesem zwanym inflacją układu. Po uruchomieniu `activity` widoki zdefiniowane w plikach układu XML są przekształcane w (lub „nadmuchiwane”) w obiekty Kotlin w pamięci. Gdy to nastąpi, aktywność może rysować te obiekty na ekranie i dynamicznie je modyfikować.

1. W Android Studio wybierz **File> New> New project**, aby utworzyć nowy projekt. Użyj szablonu Empty activity i kliknij **Next**.
2. Nazwij projekt **DiceRoller**, sprawdź wszystkie pozostałe wartości dla lokalizacji projektu nazwy projektu. Upewnij się, że "Use AndroidX Artifacts" jest zaznaczone. Kliknij przycisk Zakończ..



3. W okienku **Project> Android** rozwiń `java> com.example.android.diceroller`. Kliknij dwukrotnie `MainActivity`. Edytor kodu pokazuje kod `MainActivity`.



4. Poniżej nazwy pakietu i instrukcji importu znajduje się deklaracja klasy dla MainActivity. Klasa MainActivity rozszerza AppCompatActivity.

```
class MainActivity : AppCompatActivity() { ...
```

`AppCompatActivity` to podklasa `Activity`, która obsługuje wszystkie nowoczesne funkcje Androida, zapewniając jednocześnie kompatybilność wsteczną ze starszymi wersjami Androida. Aby Twoja aplikacja była dostępna dla jak największej liczby urządzeń i użytkowników, zawsze używaj `AppCompatActivity`.

5. Zwróć uwagę na metodę `onCreate()`. Aktywności nie używają konstruktora do inicjowania obiektu. Zamiast tego wywoływana jest seria predefiniowanych metod (zwanych „metodami cyklu życia”) w ramach konfiguracji działania. Jedną z tych metod cyklu życia jest `onCreate()`, którą zawsze zastępujesz we własnej aplikacji. Dowiesz się więcej o metodach cyklu życia w późniejszym kodzie.

W `onCreate()` określasz, który układ jest powiązany z działaniem, i nadmuchujesz układ. Metoda `setContentView()` wykonuje obie te rzeczy.

```

override fun onCreate(savedInstanceState: Bundle?) {

    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
}

```

Metoda `setContentView()` odwołuje się do układu za pomocą `R.layout.activity_main`, która w rzeczywistości jest odwołaniem do liczby całkowitej. Klasa `R` jest generowana podczas tworzenia aplikacji. Klasa `R` obejmuje wszystkie zasoby aplikacji, w tym zawartość katalogu `res`.

W tym przypadku `R.layout.activity_main` odnosi się do wygenerowanej klasy `R`, folderu układu i pliku układu `activity_main.xml`. (Zasoby nie zawierają rozszerzeń plików.) Będziesz odwoływał się do wielu zasobów aplikacji (w tym obrazów, stringów i elementów w pliku układu) przy użyciu podobnych odniesień w klasie `R`.

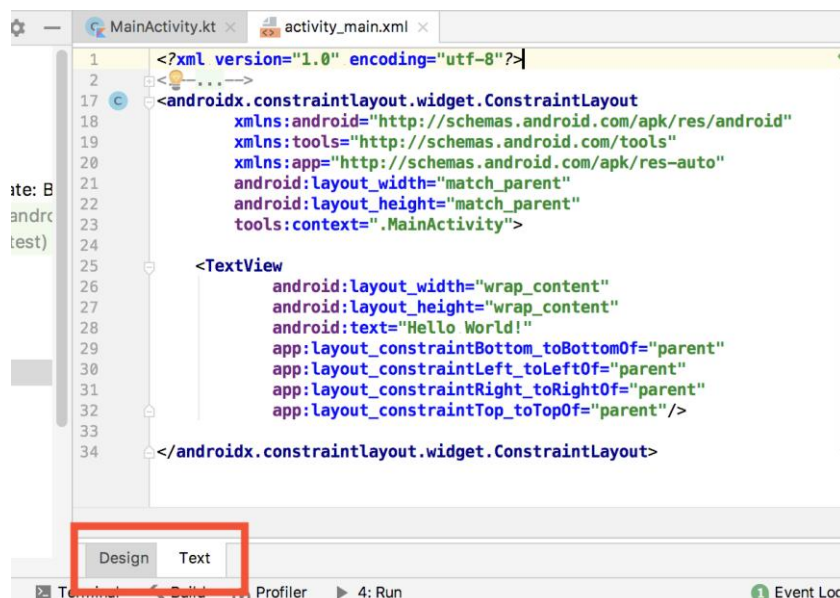
Krok 2: Plik layout aplikacji

Wszystkie działania w aplikacji mają powiązany plik układu w katalogu res / layout aplikacji. Plik układu to plik XML, który wyraża, jak faktycznie wygląda aktywność. Plik układu robi to, definiując widoki i określając, gdzie widoki pojawiają się na ekranie.

Views to takie elementy, jak tekst, obrazy i przyciski, które rozszerzają klasę View. Istnieje wiele rodzajów widoków, w tym TextView, Button, ImageView i CheckBox.

W tym zadaniu zbadasz i zmodyfikujesz plik układu aplikacji.

1. W okienku **Projekt** > **Android** rozwiń **res** > układ i kliknij dwukrotnie `activity_main.xml`. Zostanie otwarty edytor projektu układu. Android Studio udostępnia edytor, który pozwala budować Layout aplikacji w sposób wizualny i wyświetlać podgląd układu Layoutu.
2. Aby wyświetlić plik układu jako XML, kliknij kartę Tekst u dołu okna.



3. Usuń cały istniejący kod XML w edytorze układu. Domyślny układ otrzymany w nowym projekcie jest dobrym punktem wyjścia, jeśli pracujesz z edytorem projektu Android Studio. W tej lekcji będziesz pracować z bazowym XML, aby zbudować nowy układ od podstaw.
4. Skopiuj i wklej ten kod do układu:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    tools:context=".MainActivity" >
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
```

</LinearLayout>

Teraz sprawdź kod:

1. Elementem najwyższego poziomu lub głównym układem jest element <LinearLayout>. Widok LinearLayout to ViewGroup. Grupy widoków to kontenery, które zawierają inne widoki i pomagają określić pozycje widoków na ekranie.

Wszystkie widoki i grupy widoków, które dodajesz do swojego układu, są zorganizowane w hierarchię widoków, przy czym najwyższy element XML stanowi rdzeń tej hierarchii. Widok główny może zawierać inne widoki i grupy widoków, a zawarte grupy widoków mogą zawierać inne widoki i grupy widoków. Gdy aplikacja się uruchomi, hierarchia widoków w pliku układu XML staje się hierarchią obiektów, gdy Layout jest „napompowany”. W tym przypadku grupa widoków głównych jest Layoutem liniowym, który porządkuje swoje widoki potomne liniowo, jeden po drugim (pionowo lub poziomo).

Domyślnym korzeniem nowego projektu dla systemu Android jest ConstraintLayout, który działa dobrze we współpracy z edytorem projektu. W tej aplikacji korzystasz z grupy widoków LinearLayout, która jest prostsza niż ConstraintLayout. Dowiesz się więcej o grupach widoków (view groups) i ConstraintLayout w następnej lekcji.

2. Wewnątrz tagu LinearLayout zwróć uwagę na atrybut android: layout_width. Szerokość tego LinearLayout jest ustawiona tak, aby pasowała do elementu nadrzędnego, co czyni go taką samą, jak jego element nadrzędny. Ponieważ jest to widok główny, układ rozwija się do pełnej szerokości ekranu.
3. Zwróć uwagę na atrybut android: layout_height, który jest ustawiony na wrap_content. Ten atrybut powoduje, że wysokość LinearLayout jest zgodna z połączoną wysokością wszystkich zawartych w nim widoków, która na razie jest tylko TextView.
4. Sprawdź element <TextView>. TextView, który wyświetla tekst, jest jedynym elementem wizualnym w Twojej aplikacji DiceRoller. Atrybut android: text zawiera ciąg znaków do wyświetlenia, w tym przypadku ciąg "Hello World!"
5. Zwróć uwagę na atrybuty android: layout_width i android: layout_height w elemencie <TextView>, które są ustawione na wrap_content. Treść widoku tekstu jest samym tekstem, więc widok zajmie tylko miejsce wymagane dla tekstu.

4. Zadanie2: Dodaj przycisk

Aplikacja do rzucania kostkami nie jest zbyt przydatna bez sposobu, w jaki użytkownik może rzucić kostką i zobaczyć, co rzucił. Aby rozpocząć, dodaj przycisk do układu, aby rzucić kostką, i dodaj tekst pokazujący wartość kości, którą rzucił użytkownik.

Krok 1: Dodaj przycisk do layout

1. Dodaj element Button do układu pod widokiem tekstu, wprowadzając <Button, a następnie naciśnij Return. Pojawia się blok przycisku, który kończy się na /> i zawiera atrybuty layout_width i layout_height.

<Button

```
android:layout_width=""  
android:layout_height="" />
```

2. Ustaw atrybuty `layout_width` i `layout_height` na „`wrap_content`”. Przy tych wartościach przycisk ma tę samą szerokość i wysokość co etykieta tekstowa, którą zawiera.
3. Dodaj atrybut `android:text` do przycisku i nadaj mu wartość „Roll”. Element Button wygląda teraz następująco:

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Roll" />
```

W widokach przycisku atrybut tekstowy jest etykietą przycisku. W edytorze układu atrybut jest podświetlony na żółto, co oznacza wskazówkę lub ostrzeżenie. W tym przypadku żółte podświetlenie jest spowodowane tym, że ciąg „Roll” jest zakodowany na stałe w etykiecie przycisku, ale ciąg powinien być zasobem. Dowiesz się o zasobach ciągów w następnej sekcji.

Krok 2: Wyodrębnij zasoby ciągów

Zamiast na stałe wpisywać ciągi w układzie lub plikach kodu, najlepszą praktyką jest umieszczenie wszystkich ciągów aplikacji w osobnym pliku. Ten plik nazywa się `strings.xml` i znajduje się w zasobach aplikacji, w katalogu `res / values /`.

Posiadanie ciągów w osobnym pliku ułatwia zarządzanie nimi, zwłaszcza jeśli używasz tych ciągów więcej niż raz. Ponadto zasoby ciągów są obowiązkowe do tłumaczenia i lokalizacji aplikacji, ponieważ musisz utworzyć plik zasobów ciągów dla każdego języka.

Android Studio pomaga pamiętać, aby umieścić ciągi w pliku zasobów z podpowiedziami i ostrzeżeniami.

1. Kliknij raz ciąg „Roll” w atrybucie `android:text` znacznika `<Button>`.
2. Naciśnij `Alt + Enter` i wybierz Wyodrębnij zasób ciągu z menu podręcznego.
3. Wpisz `roll_label` jako nazwę zasobu.
4. Kliknij OK. Zasób ciągu jest tworzony w pliku `res/values/string.xml` a ciąg w elemencie Button jest zastępowany przez odwołanie do tego zasobu:
`android:text="@string/roll_label"`
5. W panelu Projekt> Android rozwiń `res`> wartości, a następnie kliknij dwukrotnie `strings.xml`, aby zobaczyć zasoby ciągów w pliku `strings.xml`:

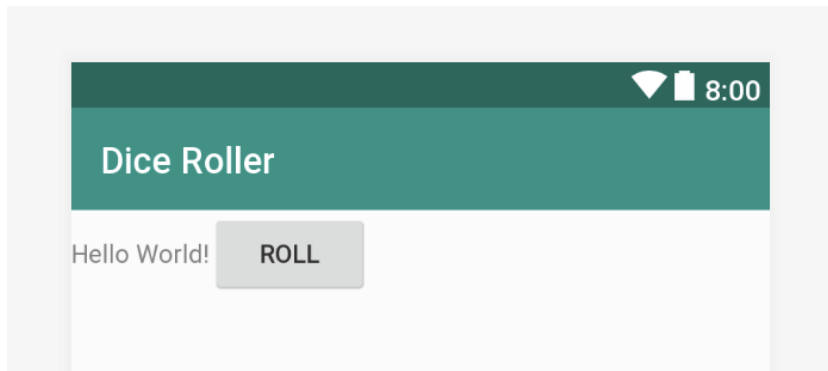
```
<resources>  
    <string name="app_name">DiceRoller</string>  
    <string name="roll_label">Roll</string>  
</resources>
```

Tip: Oprócz dodanego właśnie ciągu plik `strings.xml` zawiera także nazwę aplikacji. Nazwa aplikacji pojawi się na pasku aplikacji u góry ekranu, jeśli rozpoczniesz projekt aplikacji za pomocą pustego szablonu. Możesz zmienić nazwę aplikacji, edytując zasób `app_name`.

Krok 3: Widoki stylu i pozycji

Twój układ zawiera teraz jeden widok TextView i jeden widok przycisku. W tym zadaniu układasz widoki w grupie widoków, aby wyglądały bardziej atrakcyjnie..

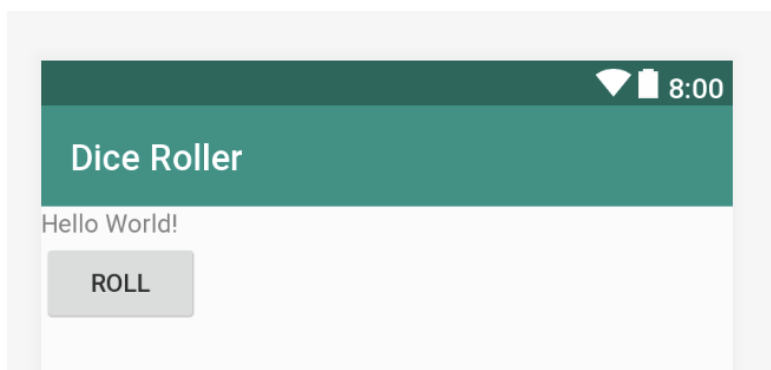
1. Kliknij kartę **Design** aby wyświetlić podgląd układu. W tej chwili oba widoki są obok siebie i przesunięte w górę ekranu.



2. Kliknij kartę **Tekst**, aby powrócić do edytora XML. Dodaj atrybut `android:orientation` do tagu `LinearLayout` i nadaj mu wartość „vertical”. Element `<LinearLayout>` powinien teraz wyglądać następująco::

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    tools:context=".MainActivity">
```

Grupa widoków `LinearLayout` umieszcza zawarte w niej widoki jeden po drugim w linii, poziomo w rzędzie lub pionowo w stosie. Pozioma jest domyślna. Ponieważ chcesz, aby `TextView` było ułożone na przycisku, ustaw orientację na pionową. Wygląd wygląda teraz tak: przycisk pod tekstem:



3. Dodaj atrybut `android:layout_gravity` zarówno do `TextView`, jak i przycisku, i nadaj mu wartość „center_horizontal”. To wyrównuje oba widoki wzdłuż środka osi poziomej. Elementy `TextView` i `Button` powinny teraz wyglądać następująco:

```
<TextView
    android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
android:layout_gravity="center_horizontal"
android:text="Hello World!" />
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:text="@string/roll_label" />
```

4. Add the `android:layout_gravity` attribute to the linear layout, and give it the value of `"center_vertical"`. Your `LinearLayout` element should now look like this:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_gravity="center_vertical"
    tools:context=".MainActivity">
```

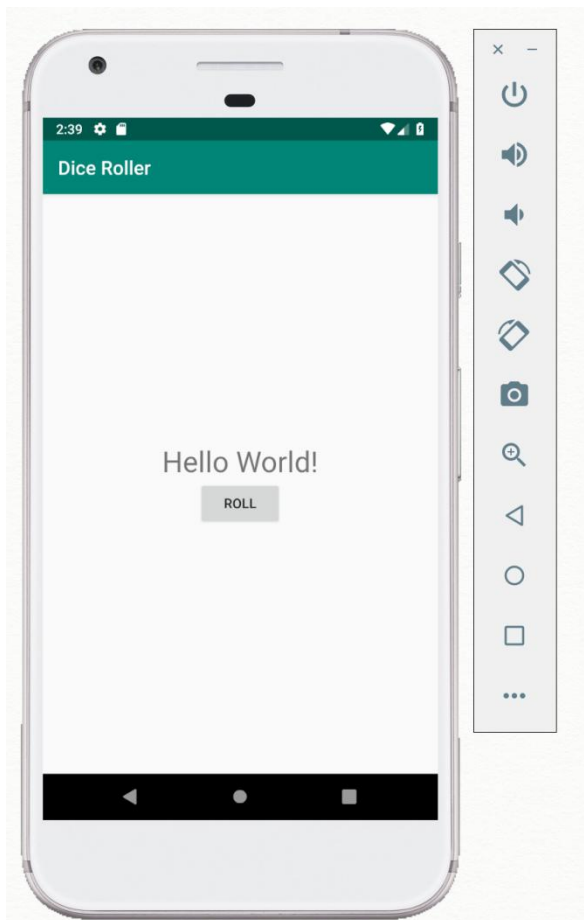
Note: Uwaga: Jeśli dodasz `center_vertical` gravity do widoku przycisku i tekstu (zamiast `center_horizontal`), widoki będą wyśrodkowane zarówno poziomo, jak i pionowo na środku układu. Oznacza to, że widoki będą jeden na drugim.

Aby wyśrodkować wszystkie elementy potomne na raz, użyj `center_vertical` na obiekcie nadrzędnym (element `LinearLayout`), jak pokazano powyżej.

5. Aby zwiększyć rozmiar tekstu w widoku tekstowym, dodaj atrybut `android:textSize` do elementu `<TextView>` o wartości „30sp”. Skrót `sp` oznacza skalowalne piksele, które są miarą rozmiaru tekstu niezależnie od jakości wyświetlania urządzenia. Element `TextView` powinien teraz wyglądać następująco::

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:textSize="30sp"
    android:text="Hello World!" />
```


6. Skompiluj i uruchom swoją aplikację.



Teraz zarówno tekst, jak i przycisk są ładnie umieszczone, a w widoku tekstu jest większy tekst. Przycisk nie ma jeszcze żadnej funkcji, więc po kliknięciu nic się nie dzieje.

Krok 4: Tworzenie odnośnika do przycisku w kodzie

Kod Kotlin w MainActivity jest odpowiedzialny za definiowanie interaktywnych części aplikacji, takich jak to, co dzieje się po naciśnięciu przycisku. Aby napisać funkcję, która będzie wykonywana po kliknięciu przycisku, musisz uzyskać odniesienie do obiektu Button w nadmuchanym układzie w MainActivity. Aby uzyskać odniesienie do przycisku:

- Przypisz przyciskowi identyfikator ID w pliku XML.
- Użyj metody `findViewById()` w kodzie, aby uzyskać odwołanie do widoku o określonym identyfikatorze. ID.

Po uzyskaniu odwołania do widoku przycisku możesz wywoływać metody w tym widoku, aby dynamicznie go zmieniać w miarę działania aplikacji. Na przykład można dodać moduł obsługi kliknięć, który wykonuje kod po naciśnięciu przycisku.

1. Otwórz plik układu `activity_main.xml`, jeśli nie jest jeszcze otwarty, i kliknij kartę **Text**.
2. Dodaj atrybut `android:id` do przycisku i nadaj mu nazwę (w tym przypadku, `"@+id/roll_button"`). `<Button>` wygląda teraz tak::

```
<Button
    android:id="@+id/roll_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:text="@string/roll_label" />
```

Podczas tworzenia identyfikatora widoku w pliku układu XML Android Studio tworzy stałą całkowitą o nazwie tego identyfikatora w wygenerowanej klasie R. Więc jeśli nazwiesz widok `roll_button`, Android Studio wygeneruje i utworzy stałą całkowitą o nazwie `roll_button` w klasie R. Prefiks „@ + id” dla nazwy identyfikatora informuje kompilator o dodaniu stałej identyfikatora do klasy R. Wszystkie identyfikatory widoków w pliku XML muszą mieć ten prefiks.

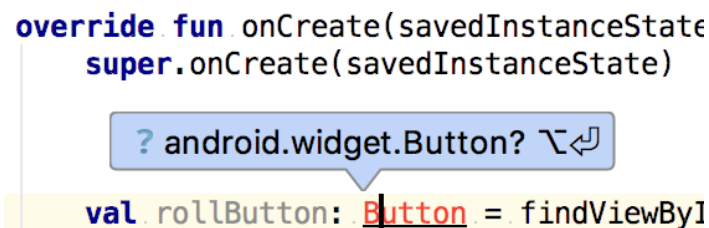
3. Otwórz plik MainActivity Kotlin. Wewnątrz `onCreate ()`, po `setContentView ()`, dodaj ten wiersz:

```
val rollButton: Button = findViewById(R.id.roll_button)
```

Użyj metody `findViewById ()`, aby uzyskać odwołanie do widoku dla widoku zdefiniowanego w klasie XML. W takim przypadku otrzymujesz odwołanie `Button` z klasy R i identyfikator `roll_button` i przypisujesz to odwołanie do zmiennej `rollButton`.

Note: Uwaga: jeśli wpiszesz wiersz zamiast go skopiować i wkleić, zauważysz, że Android Studio zapewnia autouzupełnianie podpowiedzi dla nazwy identyfikatora po rozpoczęciu pisania..

4. Zauważ, że Android Studio wyróżnia klasę przycisku na czerwono i podkreśla ją, aby wskazać, że jest to nierozwiązane odwołanie i że musisz zaimportować tę klasę, zanim będziesz mógł z niej korzystać. Może również pojawić się etykieta wskazująca pełną nazwę klasy:



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

    val rollButton: Button = findViewById(R.id.roll_button)
```

5. Naciśnij klawisze `Alt + Enter`, aby zaakceptować w pełni kwalifikowaną nazwę klasy.

Krok 5: Dodaj moduł obsługi kliknięć, aby wyświetlić toast

Moduł obsługi kliknięć to metoda wywoływana za każdym razem, gdy użytkownik kliknie lub stuknie element interfejsu użytkownika, który można kliknąć, na przykład przycisk. Aby utworzyć moduł obsługi kliknięć, potrzebujesz:

- Metode, która wykonuje pewne operacje.
- Metode `setOnClickListener ()`, która łączy `Button` z metodą handlera.

W tym zadaniu stworzysz metodę obsługi kliknięć w celu wyświetlenia Toast. (Toast to komunikat, który pojawia się na krótko na ekranie.) Połączysz metodę obsługi kliknięć z przyciskiem. Button.

1. W MainActivity class po onCreate(), utwórz funkcję prywatną o nazwie rollDice():

```
private fun rollDice() {  
}
```

2. Dodaj ten wiersz do metody rollDice (), aby wyświetlić Toast po wywołaniu rollDice ():

```
Toast.makeText(this, "button clicked",  
    Toast.LENGTH_SHORT).show()
```

Aby utworzyć toast, wywołaj metodę Toast.makeText (). Ta metoda wymaga trzech rzeczy:

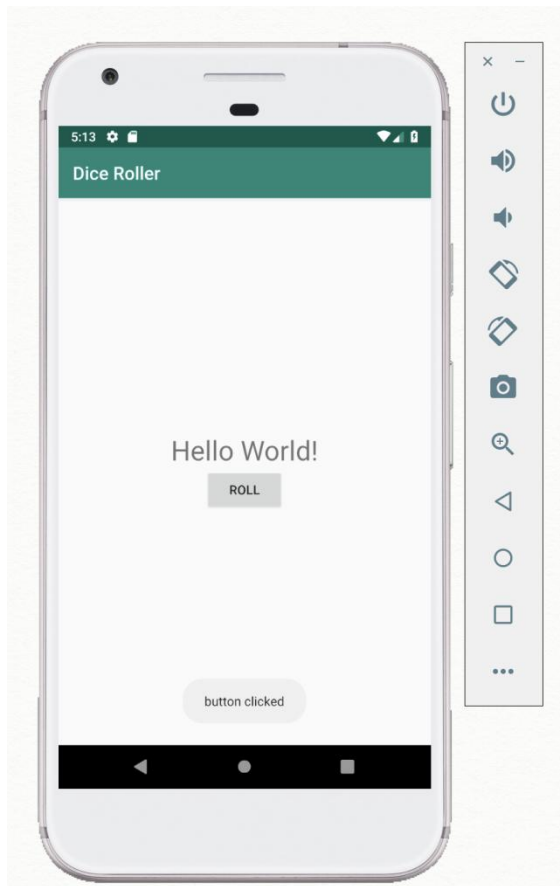
- Obiekt kontekstu. Obiekt Context pozwala komunikować się i uzyskiwać informacje o bieżącym stanie systemu operacyjnego Android. Potrzebujesz tutaj kontekstu, aby obiekt Toast mógł powiedzieć systemowi operacyjnemu, aby wyświetlił toast. Ponieważ AppCompatActivity jest podklasą kontekstu, możesz po prostu użyć słowa kluczowego this dla kontekstu.
 - Komunikat do pokazania, tutaj „przycisk kliknięty”.
 - czas wyświetlania wiadomości. Metoda show () na końcu wyświetla toast.
3. W onCreate () po wywołaniu findViewById () dodaj ten wiersz, aby przypisać funkcję rollDice () jako procedurę obsługi kliknięć do obiektu rollButton:

```
rollButton.setOnClickListener { rollDice() }
```

Pełna definicja klasy MainActivity wygląda teraz następująco:

```
class MainActivity : AppCompatActivity() {  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        val rollButton: Button = findViewById(R.id.roll_button)  
        rollButton.setOnClickListener { rollDice() }  
    }  
  
    private fun rollDice() {  
        Toast.makeText(this, "button clicked",  
            Toast.LENGTH_SHORT).show()  
    }  
}
```

4. Skompiluj i uruchom swoją aplikację. Za każdym naciśnięciem przycisku powinien pojawić się toast.



5. Zadanie3: zmień tekst

W tym zadaniu modyfikujesz metodę `rollDice()`, aby zmienić tekst w `TextView`. Pierwszym krokiem jest zmiana tekstu z „Hello World!” do string `"Dice Rolled!"`. W drugim kroku wyświetlana jest losowa liczba od jednego do sześciu.

Krok 1: Wyświetl string

1. Otwórz `activity_main.xml`, i dodaj ID do `TextView`.

```
android:id="@+id/result_text"
```

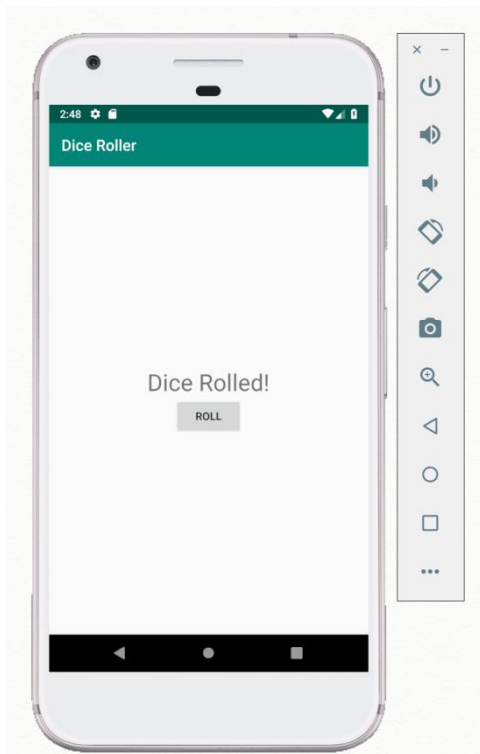
2. Otwórz `MainActivity`. W metodzie `rollDice()` skomentuj wyświetlający `Toast`.
3. Użyj metody `findViewById()` aby uzyskać odwołanie do `TextView` według jego identyfikatora. Przypisz odwołanie do zmiennej wynikowej.

```
val resultText: TextView = findViewById(R.id.result_text)
```

4. Przypisz nowy ciąg do właściwości `resultText.text`, aby zmienić wyświetlany tekst. Możesz zignorować odpowiedź, aby wyodrębnić ten ciąg znaków do zasobu; to tylko tymczasowy ciąg.

```
resultText.text = "Dice Rolled!"
```

5. Skompiluj i uruchom aplikację. Pamiętaj, że dotknięcie przycisku Roll powoduje teraz zaktualizowanie `TextView`.



Krok 2: Wyświetl losową liczbę

Na koniec w tym zadaniu dodajesz losowość do kliknięcia przycisku, aby symulować rzut kostką. Za każdym razem, gdy przycisk zostanie kliknięty lub puknięty, kod wybiera losową liczbę od 1 do 6 i aktualizuje `TextView`. Zadanie generowania liczby losowej nie jest specyficzne dla Androida i używasz do tego klasy `Random`.

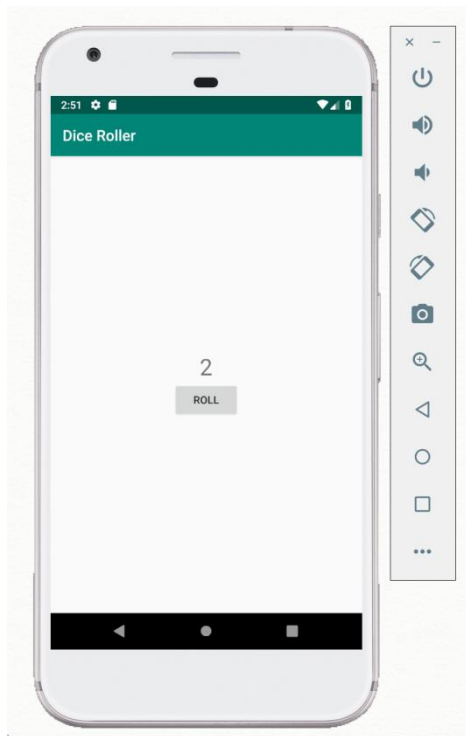
1. U góry metody `rollDice ()` użyj metody `Random.nextInt ()`, aby uzyskać losową liczbę od 1 do 6:

```
val randomInt = Random().nextInt(6) + 1
```

2. Ustaw właściwość `text` na wartość losowej liczby całkowitej, jako ciąg:

```
resultText.text = randomInt.toString()
```

3. Skompiluj i uruchom aplikację. Za każdym naciśnięciem przycisku Roll zmienia się liczba w widoku tekstowym.



7. Zadanie 4

Dodaj drugi przycisk do aplikacji oznaczony „Count Up”, który pojawia się tuż pod przyciskiem Roll. Po dotknięciu przycisk Count Up powinien uzyskać bieżącą wartość widoku tekstu wynikowego, dodać do niego 1 i zaktualizować widok tekstowy. Upewnij się, że prawidłowo obsługujesz przypadki brzegowe:

- Jeśli widok tekstu wynikowego nie zawiera jeszcze liczby (tzn. Jeśli widok tekstu nadal ma domyślny ciąg „Hello World”), ustaw tekst wynikowy na 1.
- Jeśli liczba wynosi już 6, nie rób nic.

Tip: Ciąg znaków w widokach TextView jest instancją klasy CharSequence. Aby przetestować jego wartość, musisz przekonwertować ją na ciąg::

```
resultText.text.toString()
```