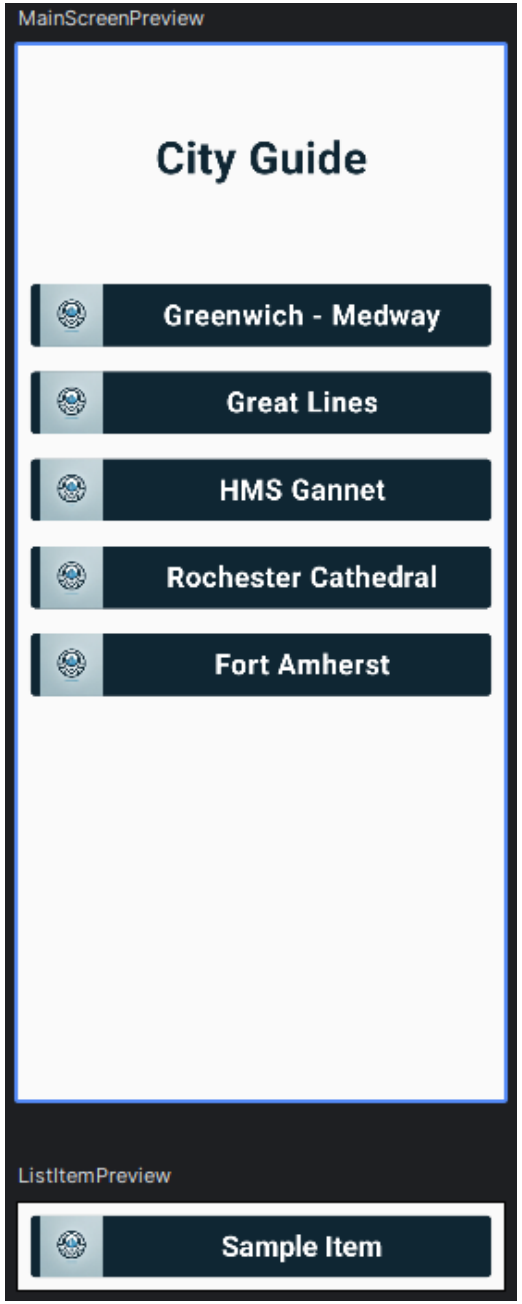
Android Lists Arrays and Web Browsers

```
Course Code: ELEE1146

Course Name: Mobile Applications for Engineers

Credits: 15

Module Leader: Seb Blair BEng(H) PGCAP MIET MIHEEM FHEA
```

Download as a PDF

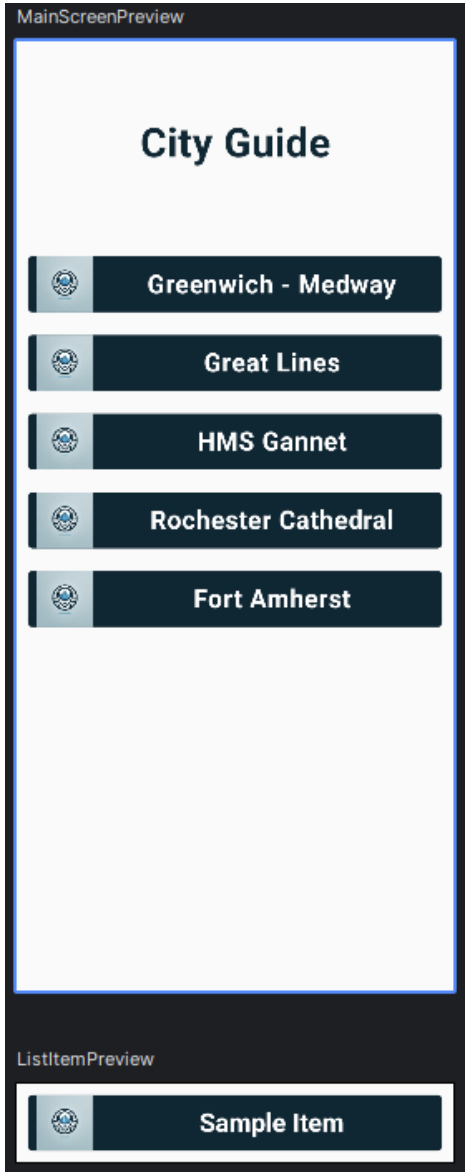
Lazy lists

- `column` and `row` are good for a known length of items, generally of size less than 10
    - Performance issues aas all items are composed and laid out despite not being in view!
- LazyColumn & LazyRow
    - Dynamic or larger number of items, as only items in components viewport are composed.



Example LazyColumn

```
@Composable
fun ItemList(items: List<String>, onItemClick: (Int) -> Unit){
    LazyColumn {
        itemsIndexed(items){index, item ->
            ListItem(item = item, onClick = {onItemClick(index)})
        }
    }
}

@Composable
fun ListItem(item: String, onClick: () -> Unit){

    Row (...){
        Spacer(modifier = Modifier.width(8.dp))
        Image( ... )
        Spacer(modifier = Modifier.width(8.dp))
        Text( ... )
    }
}
```



Creating an Array

- Array variables can store more than one value
- Different from other data types that can hold only one value
- Each individual item in an array is called an element
- Refer to each element using an index in the array

Element	Value
attaction[0]	Greenwich - Medway
attaction[1]	Great Lines
attaction[2]	HMS Gannet
attaction[3]	Rochester Cathedral
attaction[4]	Fort Amherst

Declaring an Array

- `arrayOf(value,value2,value3)` implictly sets the array data type to that supplied
- `Array<T>(size){value;value2;value3}` explicit size and Type
- Attribute

  `arrayName.size` total number of elements

Code Syntax:

```
val attractions = arrayOf("Greenwich - Medway","Great Lines",
        "HMS Gannet","Rochester Cathedral","Fort Amherst")
```

or

```
val attractions = Array<String?>(5){"Greenwich - Medway";"Great Lines",
        "HMS Gannet";"Rochester Cathedral";"Fort Amherst"}
```

Arrays [1]

- Definition
  - Arrays are data structures consisting of data items of the same type packaged together under one name.
- An array has:
  - elements have:
    - positions (indices) in the array

27	**-7**	**0**	**16**	**38**	**40**	**16**	**77**	**16**
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]

`c[]` array elements, `c` array name

Array [2]

Declaration

- `var arrayName: Array<arrayType>`
- **Allocation - as an object**
  - `arrayName = Array(arraySize) { value } (arraySize is a positive number)`
- **Initialization**
  - `var arrayName = arrayOf(value1, value2, … , valueN)`

Arrays [3] - Initialisation

- In the declaration

```
var a = arrayOf(0, 0, 0, 0, 0, 0)
val length = a.size // length is equal to 6
```

- one by one

```
var a = Array(6) { 0 }
a[0] = 0; a[1] = 0; a[2] = 0; [3] = 0; a[4] = 0; a[5] = 0
```

- Using a for loop

```
var a = Array(6) { 0 }
for (i in a.indices) { a[i] = 0 }
```

Arrays [4] - Examples

```
fun main() {
    var day = 5
    var dayName: String

    val dayOfTheWeek = arrayOf("Monday", "Tuesday", "Wednesday", "Thursday",
    "Friday", "Saturday", "Sunday")

    if (day > 7 || day < 1) {
        dayName = "That is not a valid day of the week"
    } else {
        dayName = dayOfTheWeek[day - 1] // Saturday or Friday?
    }
}
```

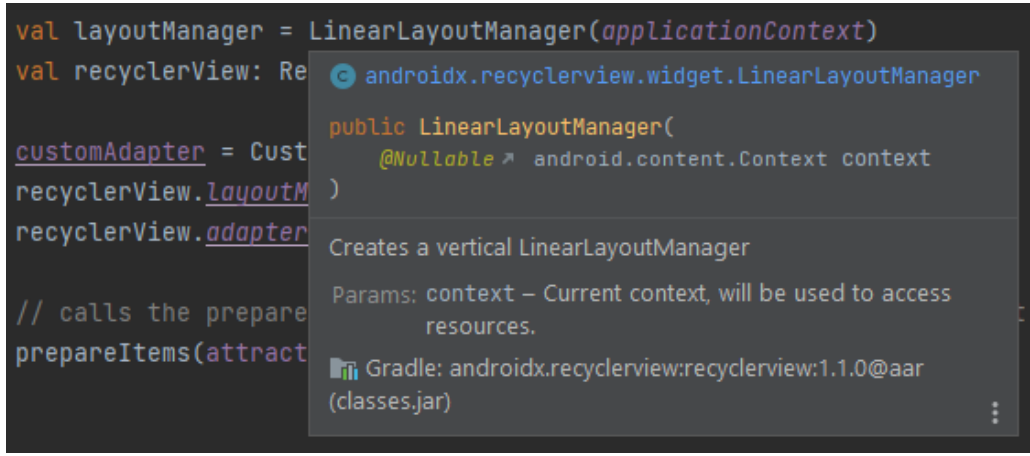
Lists

- The default implementation (List) is **immutable**
- Commonly used for collections of items where the number of elements does not change frequently

```
val immutableList = listOf(1, 2, 3)
val mutableList = mutableListOf("mutable", "list", "of")
mutableList.add(4) // Mutable list can be modified

// Sacrifice type saftey
val mixedList: List<Any> = listOf(1, "Hello", 3.14, true)
val firstElement = mixedList[0] as Int
val secondElement = mixedList[1] as String

fun ItemList(items: List<String>, onItemClick: (Int) -> Unit){ ...}
```

KDoc 1

- Kotlin Documentation (**KDoc**), is a documentation format for adding comments and documentation to your Kotlin code.
- easily consumed by developers and tools.
- These comments are written in a specific format and are used to describe classes, functions, properties, and other code elements.
- KDocs are processed by tools like Dokka to generate documentation in various formats, such as HTML or PDF.



KDoc 2

There are 21 **Tags** that can be used here are a few that are fairly common. Others can be seen in the lab.

- `@param`: Describes a parameter of a function or constructor.
- `@return`: Describes the return value of a function.
- `@throws` or `@exception`: Describes exceptions that a function may throw.
- `@sample`: Provides a usage example for the documented code.
- `@since`: Indicates the version or release when the code was introduced.
- `@author`: Identifies the author or contributor of the code.

KDoc 3

```
/**
  * Divides two numbers.
  * @author Seb Blair
  * @since dd/mm/yyyy
  * @param dividend The number to be divided.
  * @param divisor The divisor.
  * @return The result of the division.
  * @throws ArithmeticException if divisor is 0.
  * @sample com.example.cityguide.MainActivity.divideNumbers
  */
fun divideNumbers(dividend: Int, divisor: Int): Int {
    if (divisor == 0) {
        throw ArithmeticException("Division by zero is not allowed.")
    }
    return dividend / divisor
}
```

KDoc 4

```
public final fun divideNumbers(
    dividend: Int,
    divisor: Int
): Int
```

Divides two numbers.

Params: dividend - The number to be divided.
divisor - The divisor.

Returns: The result of the division.

Throws: ArithmeticException - if divisor is 0.

Authors: Seb Blair

Since: dd/mm/yyyy

Samples: com.example.cityguide.MainActivity.divideNumbers

```
if (divisor == 0) {
    throw ArithmeticException("Division by zero is not allowed.")
}
return dividend / divisor
```

com.example.cityguide.MainActivity

CityGuide.app.main

```
divideNumbers( dividend: 80, divisor: 4)
```

Android Intent

- Is a fundamental concept used for communication between different components of an Android application.
- It represents an abstract description of an operation to be performed, such as starting an activity, broadcasting a message, or delivering a message between components.
- **Explicit Intent**: This type of intent is used to start a specific component within your own application, such as starting a new activity or service.
- **Implicit Intent:** Implicit intents are used to request functionality provided by other Android components, like sending an email or opening a web page, without specifying the exact component to be used. The Android system will determine the appropriate component based on the intent's action and data.

`Intent` continued

One common use of Intent is to pass data between different activities or components of your Android app.

```
val intent = Intent(this, MainActivity::class.java)
// Start the MainActivity when the button is clicked
startActivity(intent)
```

Put Extras: Extras are key-value pairs that can be attached to an `Intent` to carry data from one activity to another. The `putExtra()` method is used to add data to an intent. These extras can be accessed in the receiving activity using `getIntent().getExtras()` or `getIntent().getStringExtra(key)` (or similar methods based on the data type).

`Intent` Extras

```
// Sending data from the sender activity
val intent = Intent(this, ReceiverActivity::class.java)
intent.putExtra("name", "YourName")
intent.putExtra("age", 25)
startActivity(intent)
```

```
// Receiving data in the ReceiverActivity
val extras = intent.extras
if (extras != null) {
    val name = extras.getString("name")
    val age = extras.getInt("age")
    // Use name and age as needed
}
```

Launching the Browser from an Android Device (continued)

Code Syntax

```
val intent = Intent(Intent.ACTION_VIEW,Uri.parse("https://www.gre.ac.uk/about-us/campus/medway"))
context.startActivity(intent)
```

The `startActivity` code launches the University of Greenwich website when the user selects the first item in the list item. Mobile friendly sites may display `m.gre.ac.uk`. Where the letter m denotes a mobile site that was launched automatically due to the platform of a mobile device.

Activity

Life

Cycle