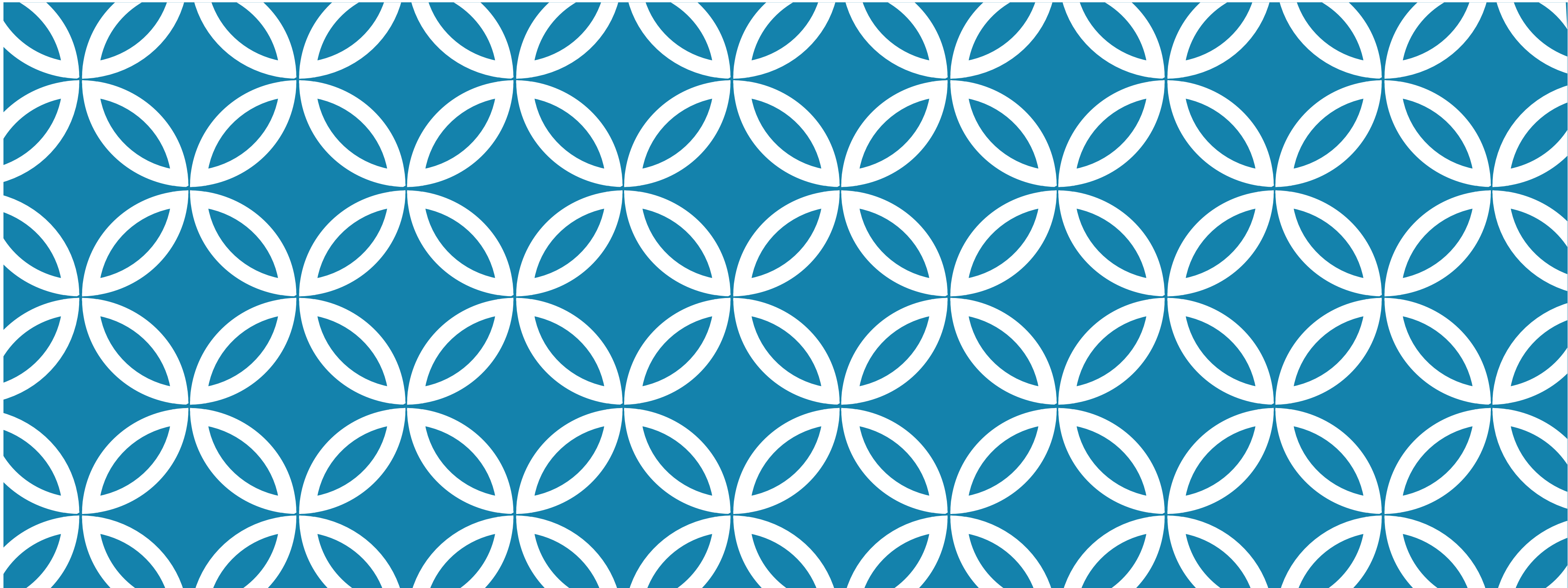


KOTLIN PART 1: BASICS

CS 346: Application Development

WHY KOTLIN?

There are literally hundreds of programming languages to choose from.

How do you pick a programming language?

- Does it offer features and capabilities that you require?
- How mature is the ecosystem around the language — do you have useful libraries and tools support to use it effectively?
- How productive can you be with it - are there books, tutorials, videos etc.?
- Is it suitable for the type of software + environment in which you are working?

> *"Why can't we just build everything in C++?"*
> — every C++ developer, ever

We need to extend our programming language capabilities to address required functionality.

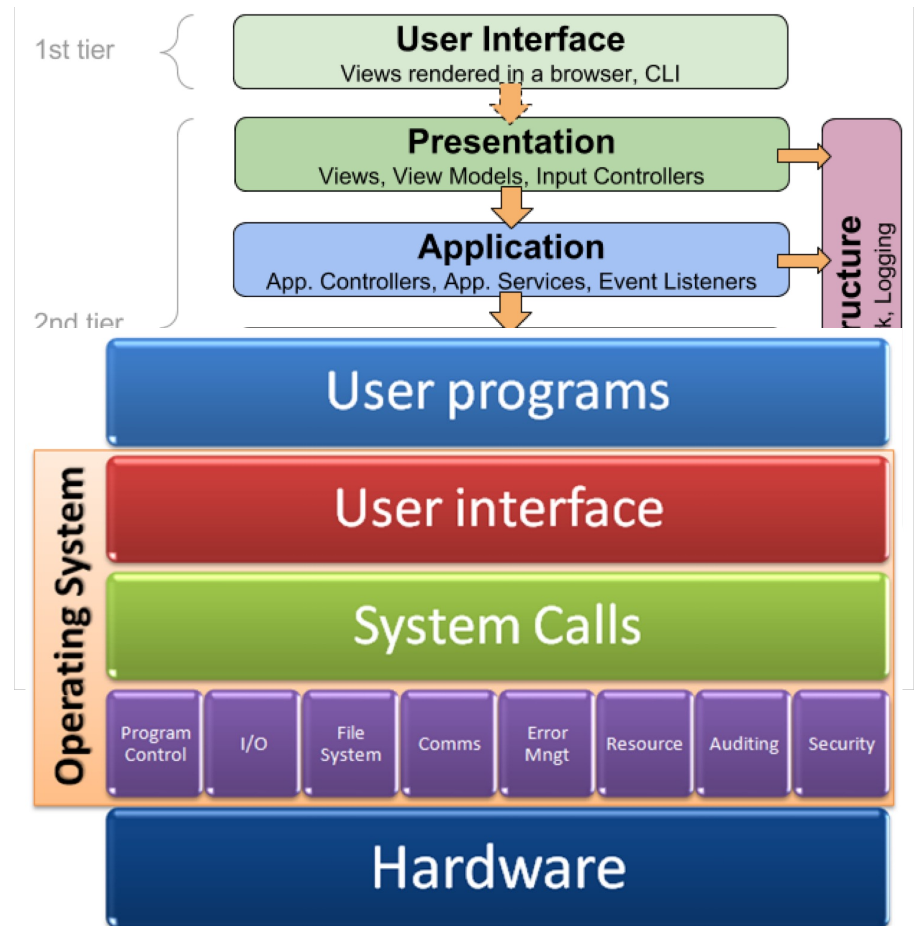
Combination of base language + stdlib + ext-libs + frameworks

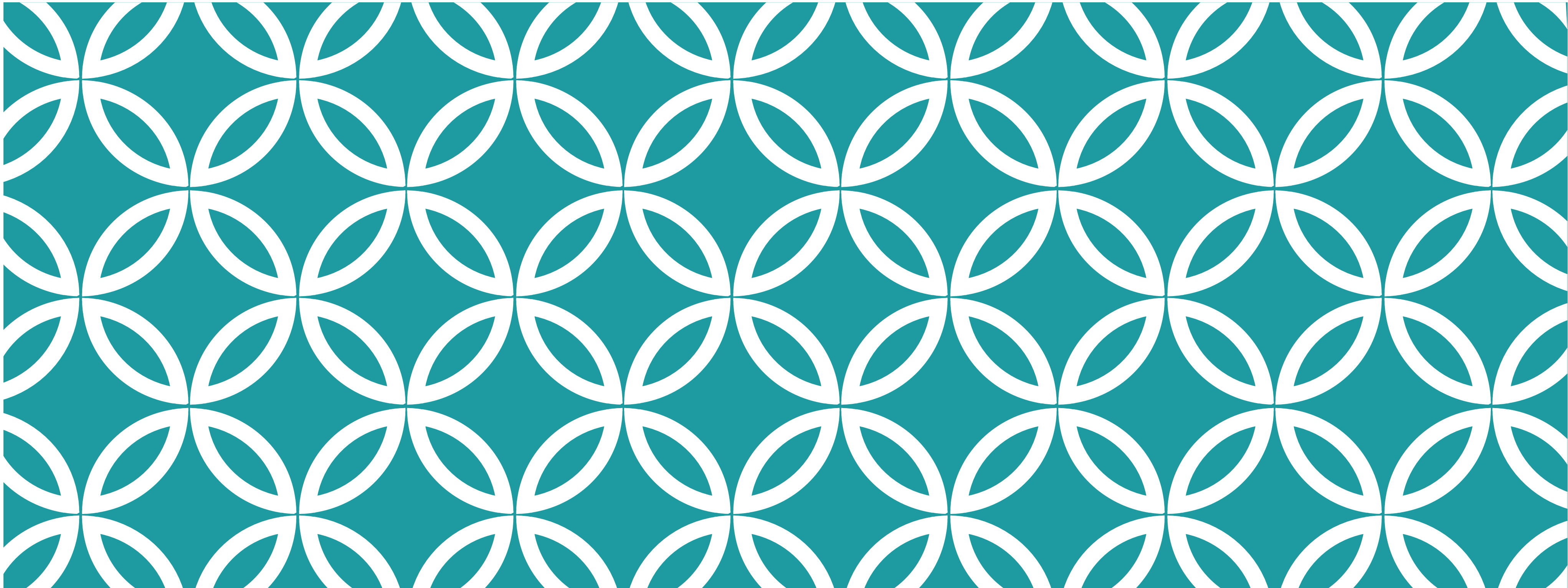
Problematic layers:

- **User interface** (UI, graphics)
- **Presentation** (screens/flow)
- **Data** (data files/databases)
- **Services** (web, bluetooth, camera) - not shown

We need libraries to provide all of this.

- We want to leverage OS capabilities.
- We want all of this to work on our supported platforms.





KOTLIN > INTRODUCTION

CS 346: Application Development

FEATURES

kotlinlang.org

Kotlin is a modern, general-purpose language. Like Swift.

Developed by JetBrains (IDE company), initially as a Java replacement.

- Imperative, object-oriented, functional programming styles (*hybrid language*, like JS, Swift).
- Statically-typed; type inference; NULL safety; automatic memory management (GC).

It's multi-platform.

- Currently: Android (native), Desktop (Windows, Linux, Mac on JVM)
- Future: iOS (*beta*), JS (*beta*), WASM/web (*alpha*), and more.

It has broad library support!

- Compose (UI), Ktor (Networking), Exposed (DB) and others.

HOW TO LEARN KOTLIN

Online (free)

- **Kotlin Documentation.** https://kotlinlang.org/
- **Dave Leeds on Kotlin.** https://typealias.com/start/ ★
- **Kotlin Basics course.** JetBrains Academy. https://www.jetbrains.com/academy

Books (not free)

- Elizarov, Isakova, Aigner, Jemerov. **Kotlin in Action.** 2nd ed. Manning. https://www.manning.com/books/kotlin-in-action-second-edition
- Marcin Moskala. 2022. **Kotlin Essentials.** Packt. ISBN 978-8396684721. https://leanpub.com/kotlin_developers

INSTALLATION

You need the Kotlin compiler and runtime. We'll run on the Java JVM.

1. Install Java 17.0.5 *or later* from azul.com or another site of your choice.
2. Install Kotlin from kotlinlang.org
3. Check installation from shell:

```
❯ java -version
openjdk version "18.0.2.1" 2022-08-18
OpenJDK Runtime Environment Zulu18.32+13-CA (build 18.0.2.1+1)
OpenJDK 64-Bit Server VM Zulu18.32+13-CA (build 18.0.2.1+1, mixed mode, sharing)

❯ kotlinc -version
info: kotlinc-jvm 1.9.10 (JRE 18.0.2.1+1)
```

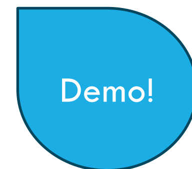
IDE INSTALLATION (SEE "GETTING STARTED")

We highly recommend installing and using **IntelliJ IDEA** in this course.

IDEs offer advanced features: debugging, profiling, code-completion, refactoring.

Install from: IntelliJ Downloads

- Community Edition will work fine.
- You can get a free Student license for Ultimate Edition which provides extra features — *highly recommended*
- Runs on macOS (Intel or Apple), Windows, Linux.
- Requires you to setup projects (which we will demonstrate/discuss soon).

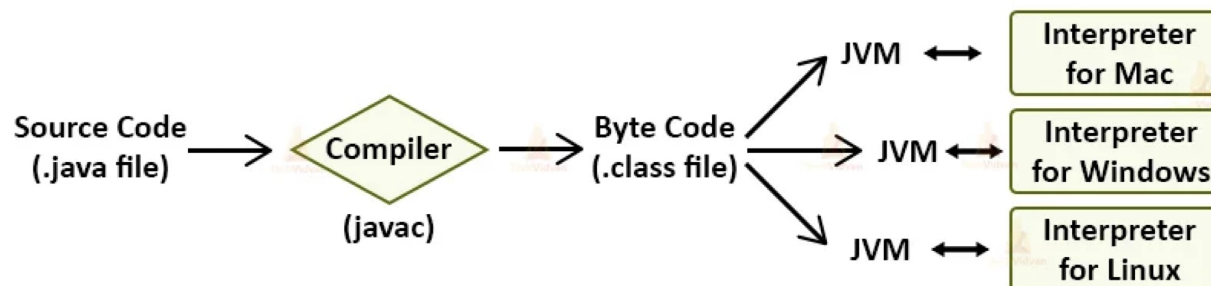


CS 346: Application Development

COMPILING CODE

The kotlinc compiler consists of multiple backend compilers:

- Kotlin/Native compiles Kotlin code to native binaries. It includes an LLVM based backend for the Kotlin compiler and native implementation of the Kotlin standard library. Works for Windows, macOS, Linux (iOS, JS).
- Kotlin/Android compiles Kotlin code to native Android binaries.
- Kotlin/JVM compiles Kotlin code to JVM bytecode. This is common used for desktop/server support, so that we can leverage existing JVM libraries!



Kotlin/JVM compiler generates bytecode that executes in a native JVM.

COMPILING “HELLO WORLD”

It’s tradition to write ”Hello World” when learning a new programming language.

Here’s the Kotlin version!

```
fun main() {
  println("Hello World")
}
```

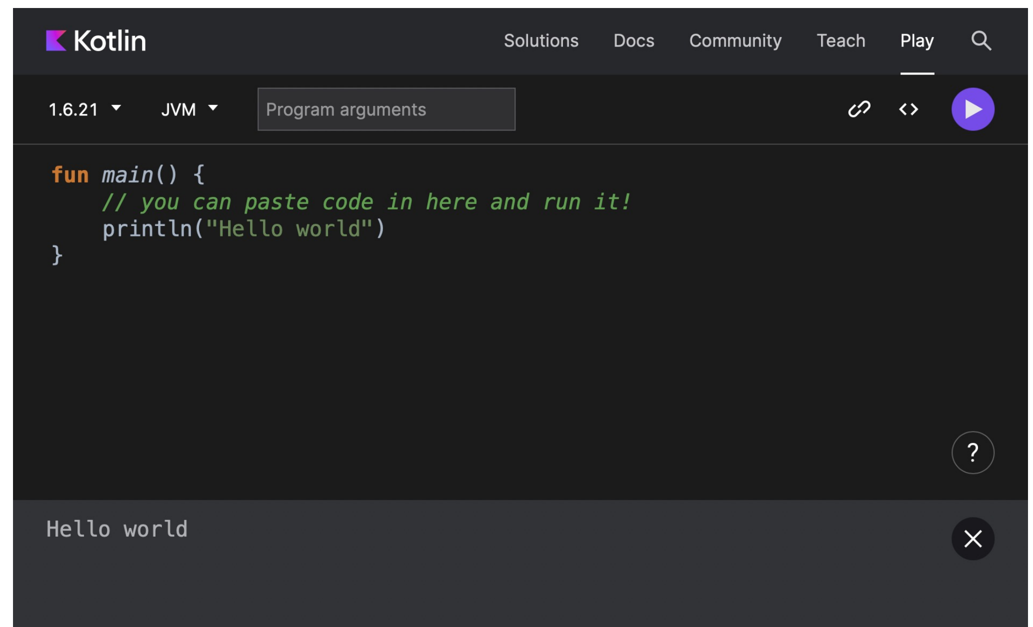
So how do we run it? We have multiple options.

1. KOTLIN PLAYGROUND

https://play.kotlinlang.org/

Works great for simple code. Fine for demoing basic language features.

Auto-imports the Kotlin standard library (stdlib) but nothing more; it has limited usefulness.



https://pl.kotl.in/tZ2NfMKUv

2. REPL (READ-EVALUATE-PRINT LOOP)

REPL is a paradigm where you type and submit expressions to the compiler one line-at-a-time. It's commonly used with dynamic languages for debugging or checking short expressions.

```
❯ kotlinc
Welcome to Kotlin version 1.9.10 (JRE 18.0.2.1+1)
Type :help for help, :quit for quit
>>> println("Hello world")
Hello world
```

3. COMPILE TO AN APPLICATION

Let’s look at the application code again.

```
Hello.kt
```

// Kotlin filename ends in .kt

```
/*
 * Insert your design recipe :)
 */
```

// Standard C++ comment style

```
fun main(args: Array<String>) {
   println(“Hello Kotlin”)
}
```

// Notice the function signature.
// No top-level class is required - take that Java!
// Semi-colons are optional

https://pl.kotl.in/tZ2NfMKUv

COMPILING COMMAND-LINE

```
$ kotlinc Hello.kt
$ ls Hello*
Hello.kt HelloKt.class

$ kotlin HelloKt
Hello Kotlin

$ javap HelloKt
Compiled from "Hello.kt"
public final class HelloKt {
  public static final void main();
  public static void main(java.lang.String[]);
}
```

Sidebar: The JVM expects every file to contain a top-level class (since that's how it was originally designed).

Kotlin needs to mimic this structure. If you have a top-level function, Kotlin will create a "wrapper class" for our file.

This is why our compiled class has a different class name!

EXECUTING CODE

Each source file (.kt) will get compiled into a separate class file (.class). This can expand out to a large number of files very quickly!

The best-practice is to store compiled classes in a **JAR file** (basically a ZIP file with some extra data included).

We can distribute this single JAR file to users, and then run the program directly from that jar file.

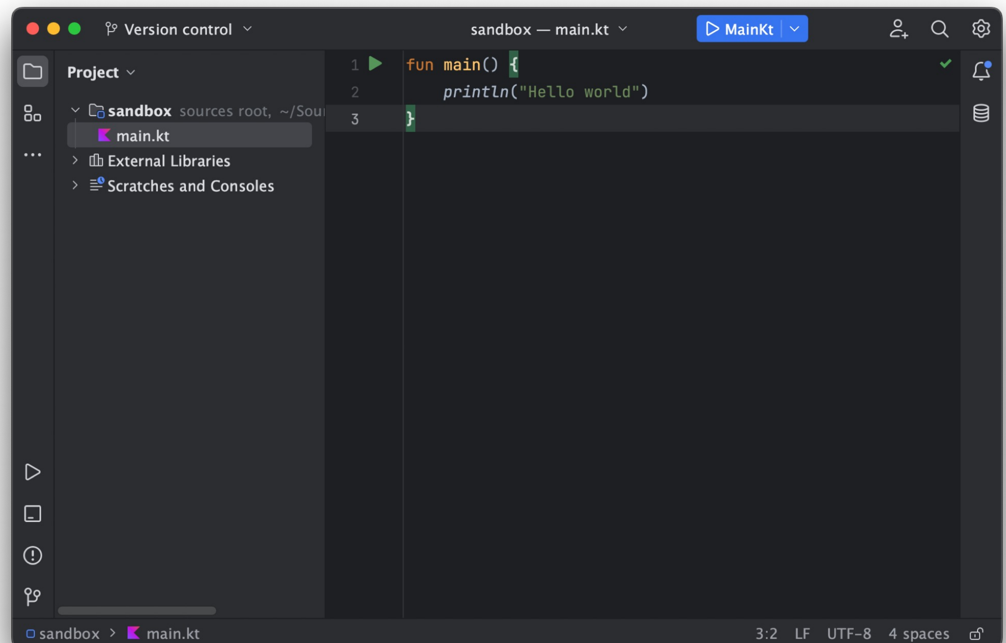
- **-d** means create a jar file
- **-include-runtime** means include Kotlin runtime classes (not in the JRE)

```
$ kotlinc Hello.kt -include-runtime -d Hello.jar
$ java -jar Hello.jar
Hello Kotlin!
```

EXECUTING IN INTELLIJ IDEA

IntelliJ IDEA can do this silently in the background for you.

1. Create a new empty project.
2. Create a file named main.kt.
3. Add a main method.
4. Press the run arrow.
5. Victory!



SIDEBAR: EXE FILES

Outside of command-line applications, we almost *never* encounter standalone executables. Why? Shared libraries!

Operating systems have mechanisms in place to allow your application to share common libraries with other applications. You will probably need to distribute multiple files with your application, and they need to be installed properly.

When you install an application, you typically are running an installer that:

- Installs libraries in a common location (e.g., /Library, or Windows/System32).
- Installs your application in a desired location (e.g., /Applications or Program Files).
- Installs supporting files e.g., preferences, sound clips, icons, README file.
- Registers libraries and your application with the system if needed (i.e., registry).

KOTLIN > PROGRAM STRUCTURE

CS 346: Application Development

MAIN METHOD

Kotlin borrows familiar syntax from C++, Java and similar C-style languages.

Programs require a single main method to serve as the entry point:

```
fun main() {
  println("Hello World")
}
```

Optionally, you can pass command-line arguments into main.

```
fun main(args: Array<String>) {
  println("Hello World")
}
```

Arrays have methods, so you can check `args.size()` for instance.

FILE STRUCTURE

Kotlin source files must end with a .kt extension.

A source file can contain:

- top-level functions
- class definitions
- global variables
- enums

You can have as many (or as few) source files as you wish.

However, you will likely want to introduce modularity (later!)

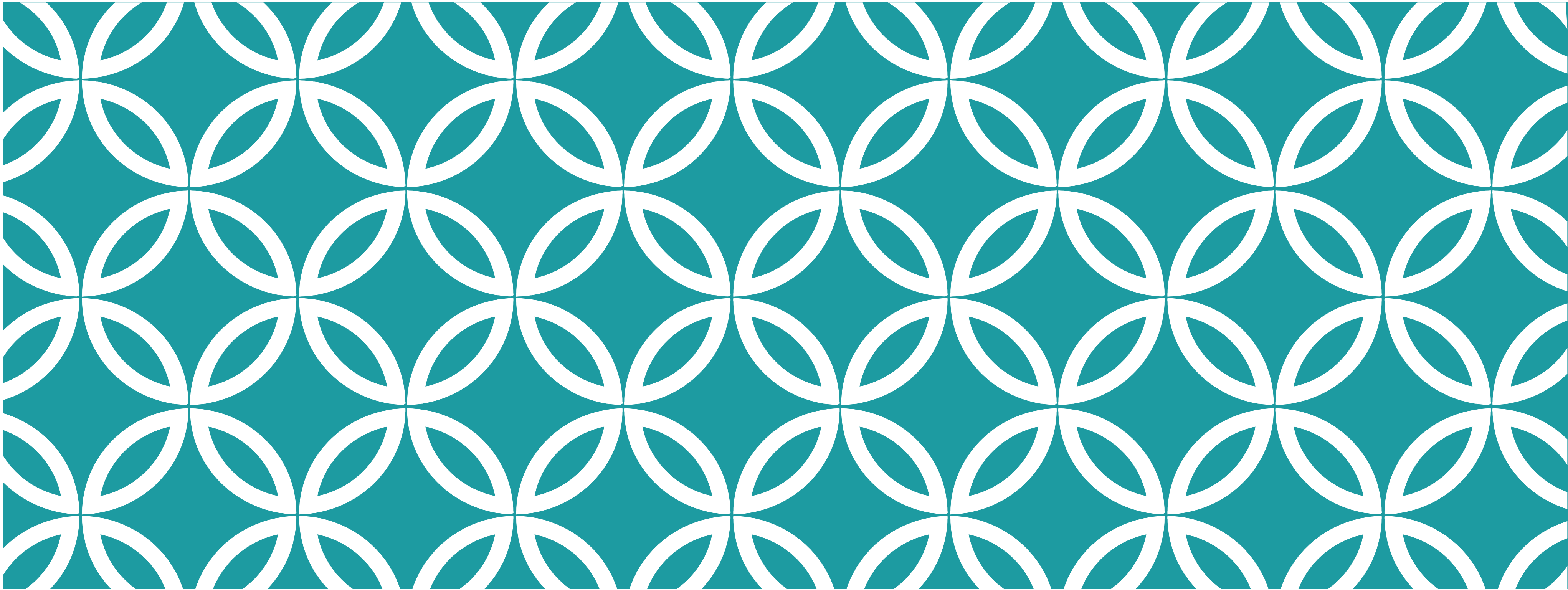
Main.kt

```
/*
 * Main entry point for mm
 * (c) 2024 Jeff Avery
 */

private val HOMEDIR = System.getProperty(“user.home”)
private val FILENAME = “${HOMEDIR}/mm.json"

fun main(args: Array<String>) {
    val list = mutableListOf<Note>()
    list.load(FILENAME)
    list.process(CommandFactory.createFromArgs(args))
    list.save(FILENAME)
}

data class Note(
    val id: String = UUID.randomUUID().toString(),
    var index: Int = 0,
    val title: String? = null,
    val content: String? = null
)
```



KOTLIN > TYPE SYSTEM

CS 346: Application Development

TYPE SYSTEMS

Programming languages can take different approaches to handling types:

- **Dynamic typing**: type is inferred at runtime. e.g. Python, JS.
- **Static typing**: variable types need to be declared before use. e.g. C++, Java, Kotlin, TS. This means that types are verified at compile time! This eliminates runtime type errors.

Type systems are often referred to as strong or weakly typed.

- **Strong typed**: stricter typing rules enforced at compile-time. e.g. Java, C++, Kotlin.
- **Weak typed**: looser typing rules and may allow automatic coercing of variables to different types. Errors deferred to runtime. e.g. JS.

Kotlin is a statically typed language. Kotlin has type inference (like C++ `auto`).

- Types: `Short`, `Int`, `Long`, `Float`, `Double`, `Char`, `String`, `Byte` ← as you might expect!

TYPE DECLARATION

The `var` keyword is used to declare variables. The type of the variable is listed after the name.

e.g.,

```
var pi: Float = 3.14
var e: Float = 2.718
```

If you don't provide a type, the compiler will infer a type **if** it's unambiguous.

e.g.,

```
var name = "Jeff" // string
```

MUTABILITY

Declaration keywords are used to indicate *mutability*.

- **var**: the value of the variable can be changed (it's mutable).
- **val**: the variable cannot be changed after initialization (it's immutable)

Examples:

```
var a: Int = 0 // regular declaration
a = 10 // reassignment is allowed
val b = 3.14
b = 3.14159 // compilation error
```

This is a common feature of newer languages, to encourage safer programming practices. Use `val` as much as you can!

NULL SAFETY

Recall: NULL is the absence of a value.

NULLs typically have special rules governing how they can be used (e.g., you cannot operate on a NULL value).

- If a type system allows NULL in place of a regular value, then you need to be very careful to ensure that you are not accidentally passing a NULL value where a specific value is expected.

Risk: NULL values being mishandled can cause runtime exceptions and crash your application.

Tony Hoare: NULL was my "billion dollar mistake".

NULL SAFETY

Kotlin has special semantics for dealing with nulls that avoids the need to explicitly check for them. By default, Kotlin regular types are not nullable. If you want a nullable type, it needs to be declared as-such.

By default, a variable cannot be assigned a NULL value. A ? suffix indicates a NULLABLE type that can be NULL.

```
var length: Int = null // ERROR, cannot be null
var nlength: Int? = null // OK, can be null
```

If you have a nullable type, then evaluations of that variable must handle nulls.

```
var name: String? = "…" // nullable, so need null checks
if (name != null) println(name)
```

NULL SYNTAX

We have special syntax to make dealing with NULL values a little easier.

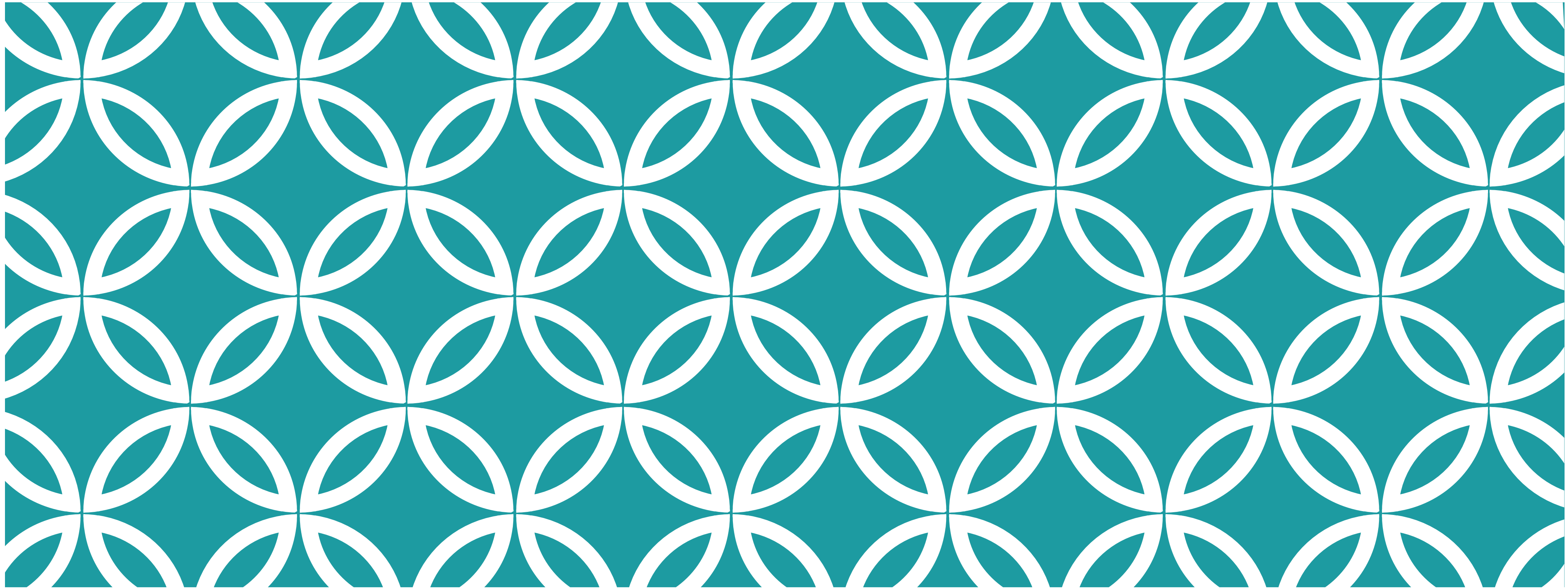
`?.` is the "safe call operator". Method can only be invoked if the object is not null.

```
var name: String? = null
val len = name?.length // len == null
val len = if (name != null) name.length else 0 // could do this
```

`?:` is a ternary operator for NULL data ("elvis operator")

```
val len = name?.length ?: 0 // much nicer syntax
```

https://pl.kotl.in/Fd8L89CPb



KOTLIN > FUNCTIONS

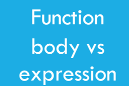
FUNCTION SYNTAX

```kotlin
// No arguments
fun hello() {
   println("Hello World")
}

// Arguments require type annotations!
fun sum(a: Int, b: Int): Int {
   return a + b
}
```

```kotlin
// Return type specified
fun sum(a: Int, b: Int):Int {
   return a + b
}

// Return type inferred
fun sum(a: Int, b: Int) = a + b
```



DEFAULT ARGUMENTS

We can supply default values for parameters.

A parameter with a default value is optional for the caller, since the default will be used if not provided.

```
fun mult(a:Int, n:Int = 1): Int {
    return a * n
}
fun main() {
    println(mult(1))    // 1 since a=1, b defaults to 1
    println(mult(5,2))  // 10 since a=5, b=2 positionally
}
```

https://pl.kotl.in/GW424Hz-q

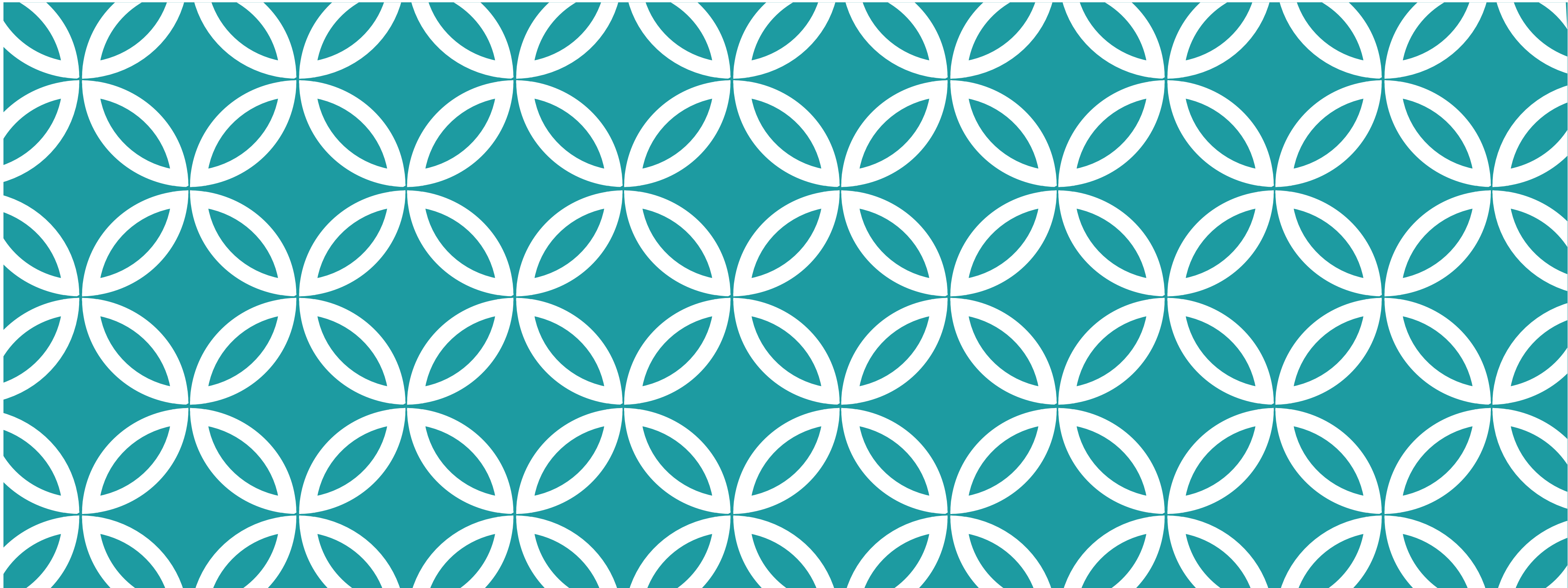
NAMED ARGUMENTS

You can (optionally) provide the argument names when you call a function.

If you do this, you can change the calling order!

```
fun repeat(s:String="*", n:Int=1):String {
    return s.repeat(n)
}
fun main() {
    println(repeat())              // prints '*' using both defaults
    println(repeat("#"))           // prints '#' using default n==1
    println(repeat("=", 3))        // prints '===' positional assignment
    println(repeat(n=5, s="#"))    // prints '#####' using named arguments
}
```

https://pl.kotl.in/vBjdzjDmf



KOTLIN > CONTROL FLOW

CS 346: Application Development

STANDARD CONTROL FLOW

https://kotlinlang.org/docs/reference/control-flow.html

Traditional control flow is supported

```
if... then.. else
```

```
while, do... while
```

```
break, continue
```

New constructs!

```
when                            // replaces switch
for (s in collection)           // iteration
for (a in 1.. 5)                // iteration up through range
for (a in 5 downTo 1)           // iteration down through range
```

IF... THEN

if... then has both statement and expression forms.

This is why Kotlin doesn't have a ternary operator: 'if' as an expression serves the same purpose.

```
// statement
if (a > b) {
    println(a)
} else {
    println(b)
}
// expression
val max = if (a > b) a else b
```

FOR...

A `for` loop iterates through anything that provides an iterator (e.g., the built-in collection classes).

```
val items1 = listOf("apple", "banana", "kiwifruit")
for (item in items1) {
  println(item)
}
```

```
val items2 = listOf("apple", "banana", "kiwifruit")
for (index in items2.indices) {
  println("item at $index is ${items2[index]}")
}
```

https://pl.kotl.in/D8boDJXak

RANGES

```
for(i in 15..18) {
  println(i) // 15 16 17 18
}
for (i in 5 downTo 1 step 2) {
  println(i) // 5 3 1
}
val low=1
val high=10
val num=7
if (num in low..high) {
  println("The number ${num} is between ${low} and ${high}")
}
```

https://pl.kotl.in/vyJZoPZAV

WHEN

`when` is an improved switch statement.

```
val x = 13
val validNumbers = listOf(11,13,17,19)
when (x) {
   0, 1 -> print("x == 0 or x == 1")
   in 2..10 -> print("x is in the range")
   in validNumbers -> print("x is valid")
   !in 10..20 -> print("x is outside the range")
   else -> print("none of the above")
}
```

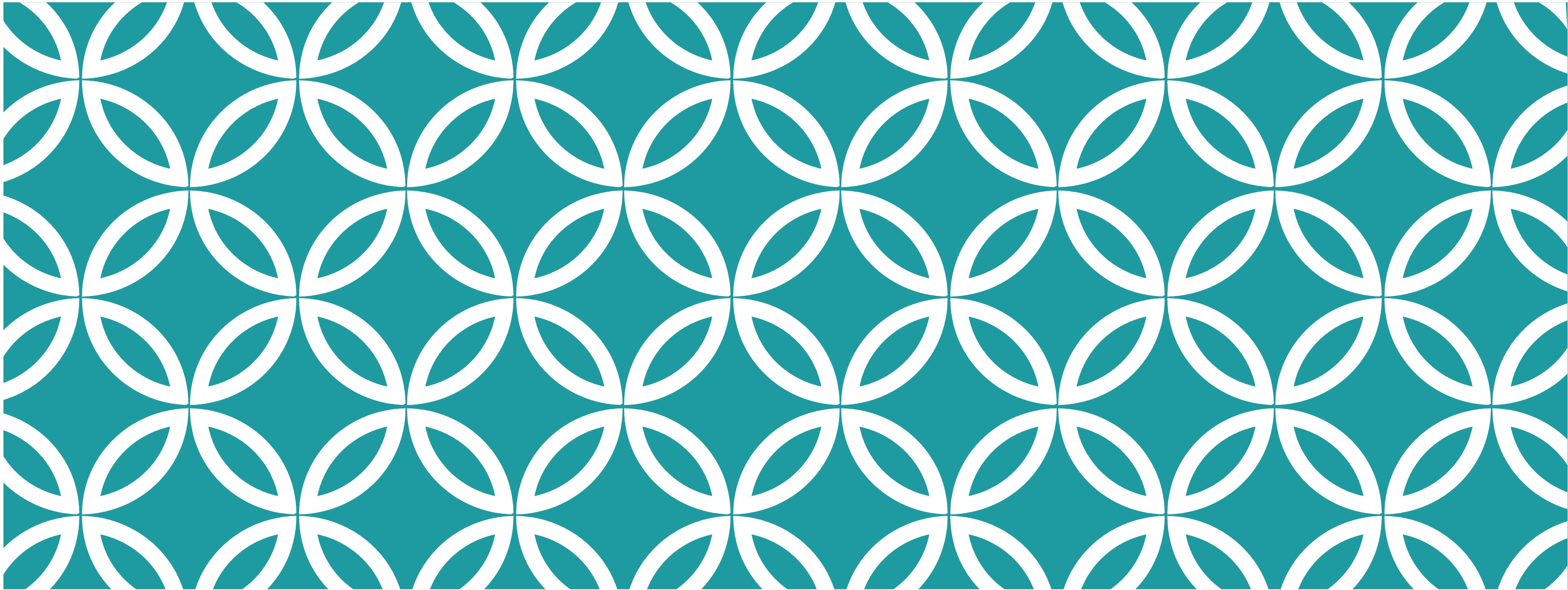
https://pl.kotl.in/CzIL_j4zz

WHEN

```
`when` as an expression

val x = 13
val response = when {
    x < 0 -> "negative"
    x >= 0 && x <= 9 -> "small"
    x >=10 -> "large"
    else -> "how do we get here?"
}
println(response)
```

https://pl.kotl.in/IEf9RSRB0



KOTLIN > COLLECTIONS

CS 346: Application Development

COLLECTIONS

A collection is a group of some variable number of items (possibly zero) of the same type. Objects in a collection are called elements.

Kotlin provides mutable and immutable interfaces to these collections.

List	An ordered collection of objects.
Pair	A tuple of two values.
Triple	A tuple of three values.
Set	An unordered collection of objects.
Map	An associative dictionary of keys and values.
Array	Indexed, fixed-size collection of object or primaries - rarely used

LIST

A list is an ordered collection of objects.

```
// immutable (due to listOf)
var fruits = listOf( "advocado","banana","cantaloupe")
println(fruits.get(0)) // advocado
println(fruits[1]) // banana
// fruits.add("dragon fruit") // unresolved, since immutable

// mutable (due to mutableListOf)
var mutableFruits = mutableListOf("advocado","banana")
mutableFruits.add("cantaloupe") // this works!
println(mutableFruits.last())
```

https://pl.kotl.in/DcUwgxGWx

PAIR

A pair is a tuple of two values.

```
val ns = Pair("Halifax Airport", "YHZ")
println(ns) // (Halifax Airport, YHZ)
```

The contents of Pair are NOT mutable, since this is a data class whose contents aren't expected to change. `copy` to duplicate with a modified value.

```
// characters.second = "Jennifer" // error!!
val characters2 = characters.copy(second = "Jennifer")
println(characters2) // (Tom, Jennifer)
```

https://pl.kotl.in/1uWdBModj

MAP

A map is an associative dictionary of key and value pairs (i.e. it maps one value to another).

```
// immutable (initialize with pairs)
val imap = mapOf(1 to "x", 2 to "y", 3 to "z")
println(imap) // {1=x, 2=y, 3=z}
// imap.put(4, "q") // immutable, so unresolved reference

// mutable
val mmap = mutableMapOf(5 to "x", 6 to "y")
mmap.put(7,"z") // ok
println(mmap) // {5=x, 6=y, 7=z}
```

https://pl.kotl.in/FcT0DJrsP

ACCESSORS

Kotlin has special properties that can be used to access data elements in collections.

```
val list = listOf("one", “two", "three", "four")
list.contains("four")) // true

// slice - extract into a new collection
list.slice(1..2) // [two, three]
list.slice(0..2 step 2) // [one, three]
// take - extract n elements
list.take(3) // [one, two, three]
list.takeLast(2) // [three, four]
```

https://pl.kotl.in/TQL-o3RYI

ACCESSORS

```
// extract using iterators
list.first { it.length > 3 } // [three]
list.last { it.startsWith("o") } // [one]

// iterate over map
for ((k, v) in imap) {
  println("$k = $v")
}
// alternate syntax
imap.forEach { k, v -> println("$k = $v") }
```