What is “Programming Languages”? as a course — at least as far as I am teaching it
   Study of the features of programming languages: why and how those features exist and
   how to make the best use of those features.
   NOT emphasizing how those features can be implemented, but how those features are
   best used

   Objectives:
   give background for choosing appropriate language for programming problem
   increase ability to learn a new programming language
   increase ways in which you can express and implement programming
   concepts (know how and why to choose)
   understand obscure language features

All the languages are turning complete
In computability theory, a system of data-manipulation rules (such as a
computer’s instruction set, a programming language, or a cellular
automaton) is said to be Turing-complete or computationally universal if
it can be used to simulate any Turing machine. This means that this system
is able to recognize or decide other data-manipulation rule sets. Turing
completeness is used as a way to express the power of such a data-
manipulation rule set. Virtually all programming languages today are
Turing-complete. The concept is named after English mathematician and
computer scientist Alan Turing.

A related concept is that of Turing equivalence – two computers P and Q
are called equivalent if P can simulate Q and Q can simulate P.
The Church–Turing thesis conjectures that any function whose values can
be computed by an algorithm can be computed by a Turing machine, and
therefore that if any real-world computer can simulate a Turing machine, it
is Turing equivalent to a Turing machine. A universal Turing machine can
be used to simulate any Turing machine and by extension the
computational aspects of any possible real-world computer.[NB 1]

To show that something is Turing-complete, it is enough to show that it can
be used to simulate some Turing-complete system. For example,
an imperative language is Turing-complete if it has conditional
branching (e.g., "if" and "goto" statements, or a "branch if zero" instruction; see one-instruction set computer) and the ability to change an arbitrary
amount of memory (e.g., the ability to maintain an arbitrary number of data
items). Of course, no physical system can have infinite memory; but if the
limitation of finite memory is ignored, most programming languages are otherwise Turing-complete.

Go through course web page

No a programming heavy course … again aim is to think

Survey of Programming languages people have used or simply have heard of
Language family tree show where Java, C, Go and Kotlin all fit

That said, writing programs in two languages — Go and Kotlin.

Why go and Kotlin:
  stackoverflow survey

Go
  imperative programming
    procedures are often executed for side effects
    Lots of variables that are set and values changes frequently.
    what is a side effect?
    how many Java methods have you written that return void?

Kotlin
  OO designed as a clean, powerful successor to Java
  Actually built on top of Java
  Kotlin “compiler” actually translates to Java and can translate to Javascript
  We will be using Kotlin to do functional programming.
  Can do imperative in Kotlin, we will not
  NO variables!

GO:

```go
package main
import "fmt"
func main() {
    fmt.Println("hello geoff!")
}
```
Kotlin:

```kotlin
fun main(args: Array<String>) {
    println("hello world");
}
```