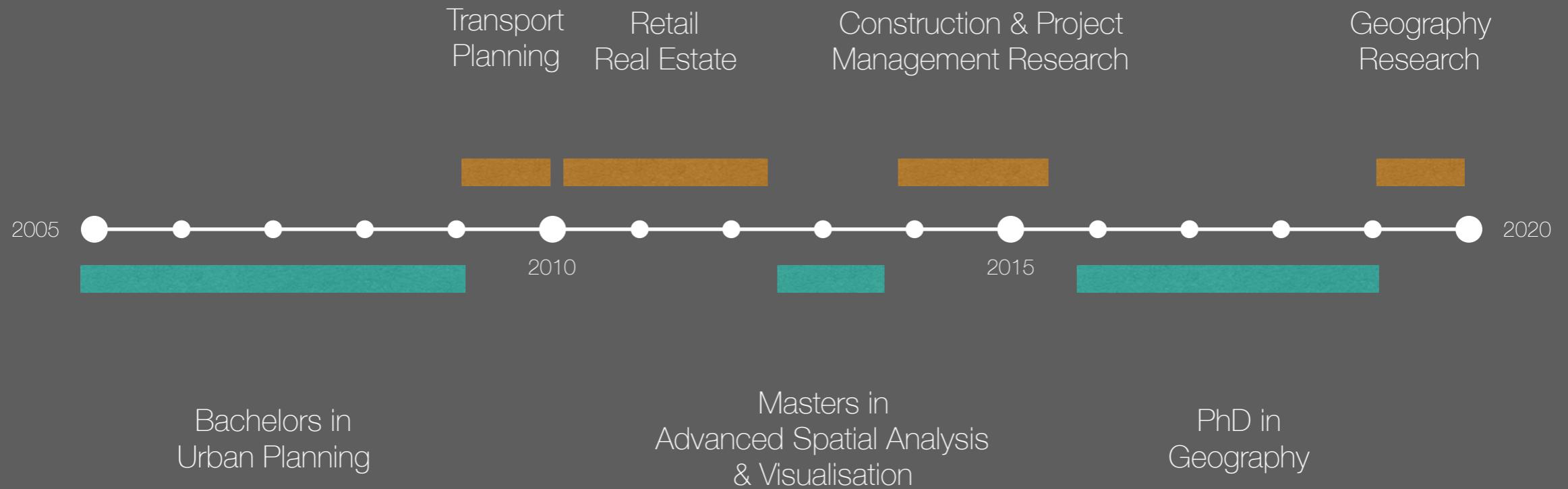


Balamurugan Soundararaj

Research Associate, Dept. Of Geography



Problems

Little or **No** Computing Background

Rediscovering the methods on their own

Work on problems that are **already solved**

Technical Tuesdays

Objectives

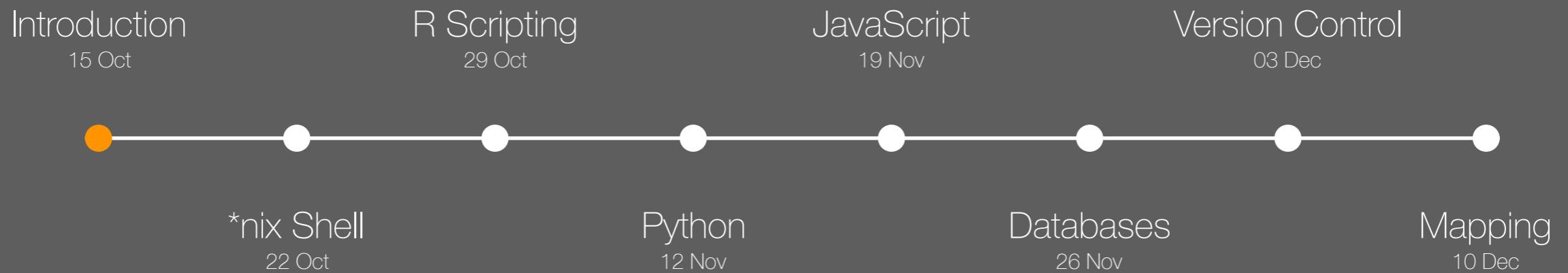
Introduction but **not a tutorial**

Tell people what is already there and **what is possible**

Give some **examples** for inspiration

Provide a **minimum viable environment** for
further learning and exploration

Technical Tuesdays



j.t.vandijk@ucl.ac.uk

Introduction to Computer Science

Identifiers and primitive data types

Assignment, arithmetic, logical and relational operators

Expression and statements, Debugging

Flow of control: selection and repetition

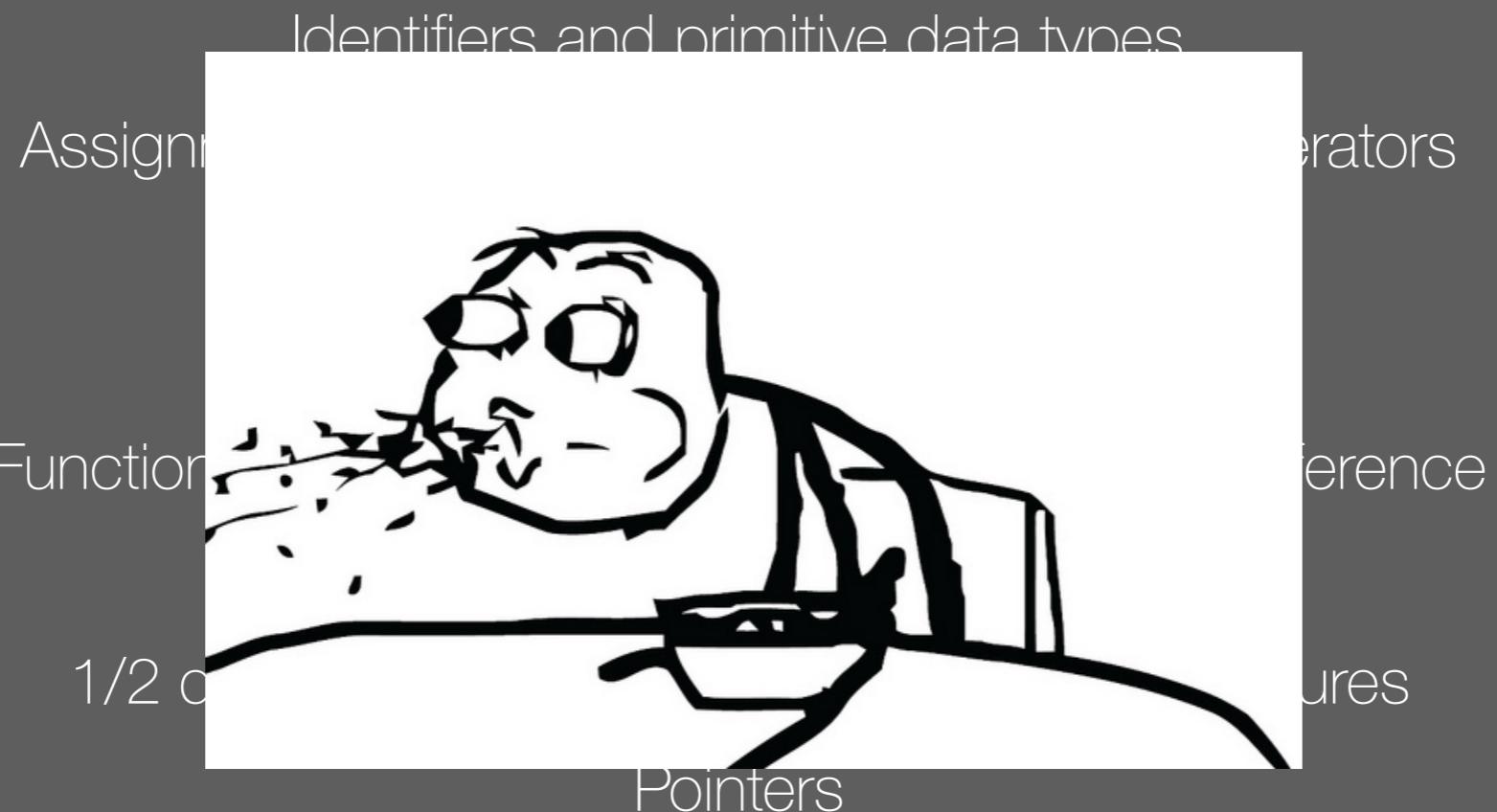
Functions, parameters passing, call by value & reference

Object-oriented programming

1/2 dimensional arrays, strings and data structures

Pointers

Introduction to Computer Science



A non computer
scientist's

Introduction to Computer Science

A non computer
scientist's

Introduction to Computer Science

Context - Where does all of this come from?

Utility - For what these things are used?

Relevance - How can I use these for my purposes

Resources - Where can I learn more?

History of Computing

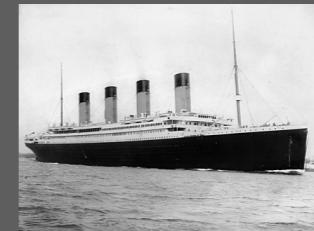
Computer



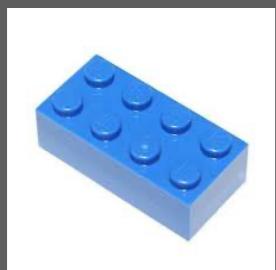
Computer

History of Computer Science

Technological **Advancements**



Layers of **Abstraction**



Introduction to Computer Science

Folding or
extending fingers

Processing
Add or subtract



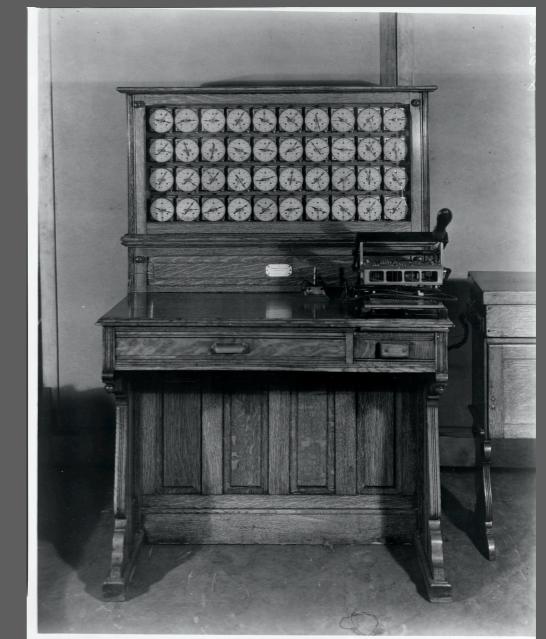
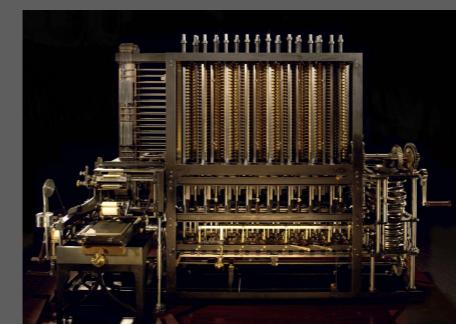
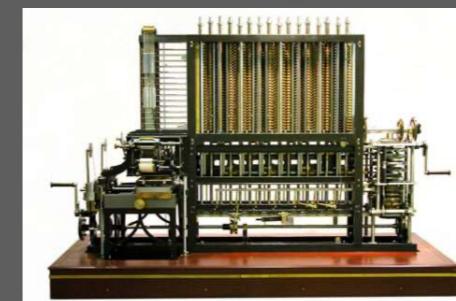
Mind

Memory
Registers A and B

Fingers

Even the most powerful complex computers
of the world are literally doing this at their core

Numbers based computing



Marks (m)
Process of marking (p)

Position of Beads (m)
Moving the beads (p)

Position of wheels (m)
Rotating the wheels (p)

Position of wheels (m)
Rotating the wheels (p)

Punched Card(m)
Electromechanical wheels (p)

All of them count from 0-9 (decimal) system

They have difference in storage and processing

Hypothetical

Boolean Arithmetic

Every number can be represented by true (1) and false (0)

Binary System!

1

1

2

10

25

11001

1356

10101001100

65,535

1111111111111111

Boolean Arithmetic

Even arithmetic can be reduced to the basic logic of

TRUE

FALSE

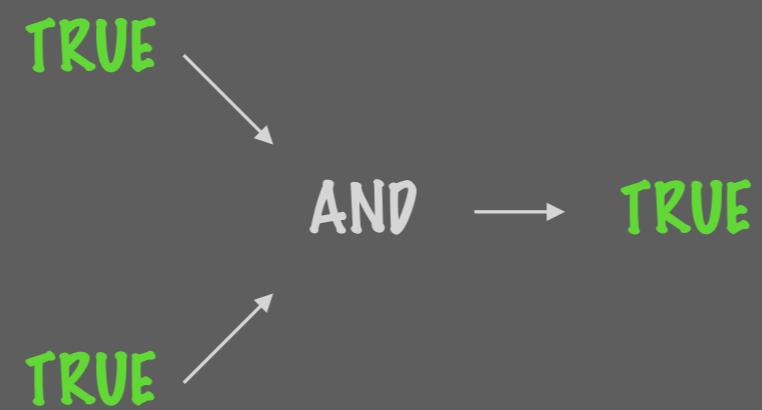
AND



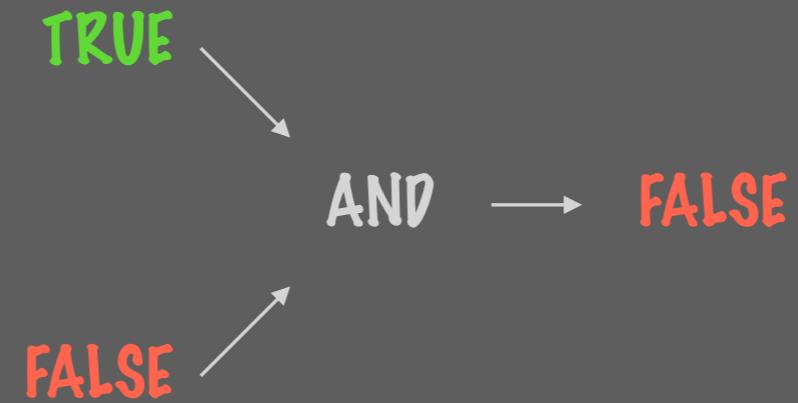
AND operator

TRUE/FALSE → AND → TRUE/FALSE
TRUE/FALSE

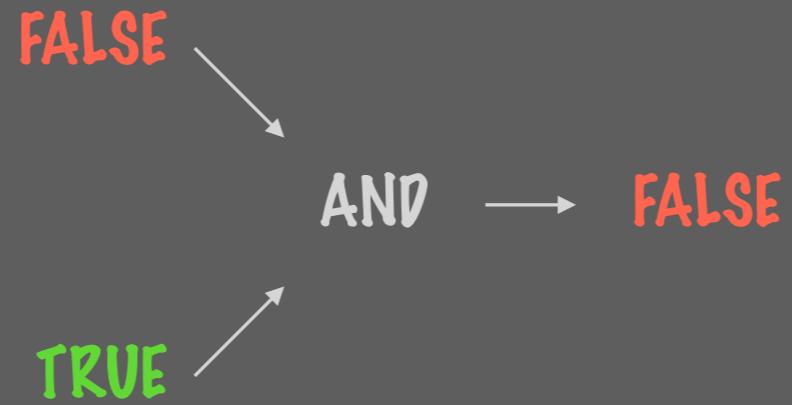
Boolean Arithmetic



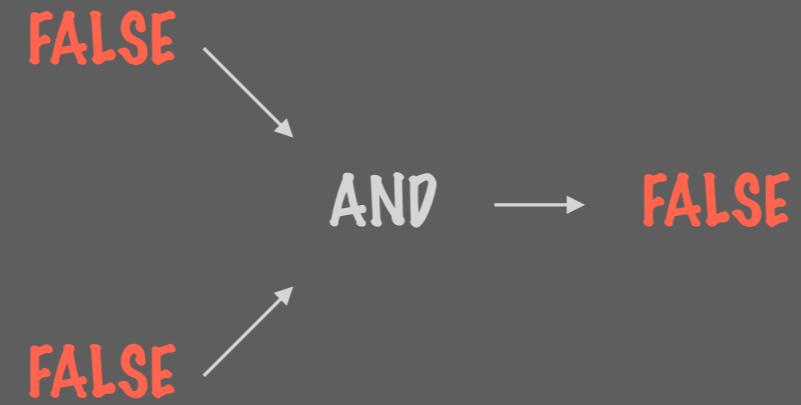
Boolean Arithmetic



Boolean Arithmetic

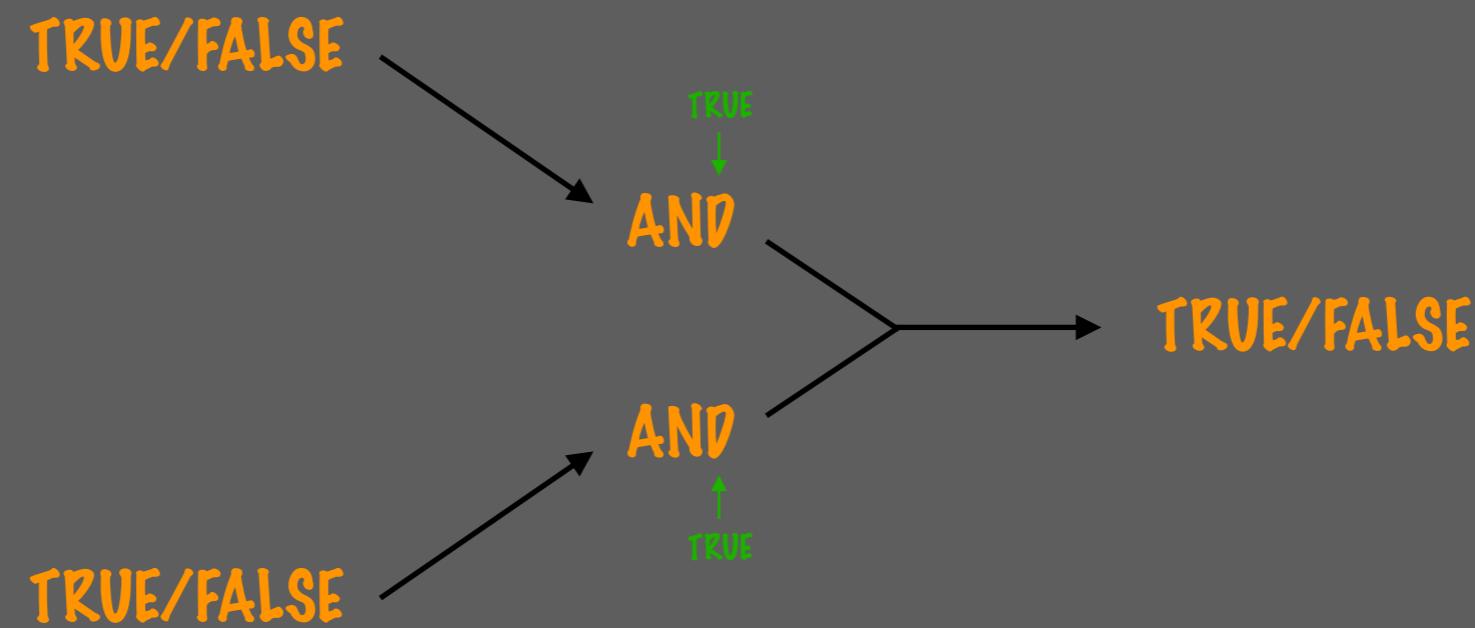


Boolean Arithmetic



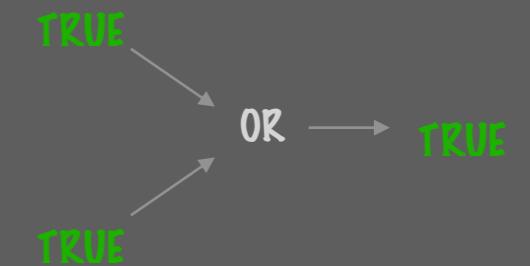
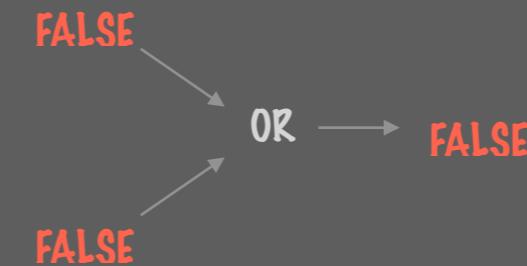
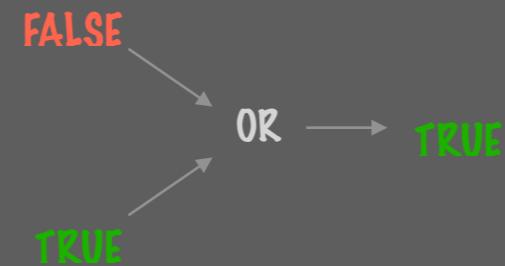
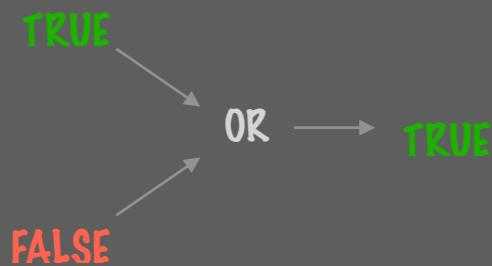
OR operator

We can combine two AND to make an OR operator



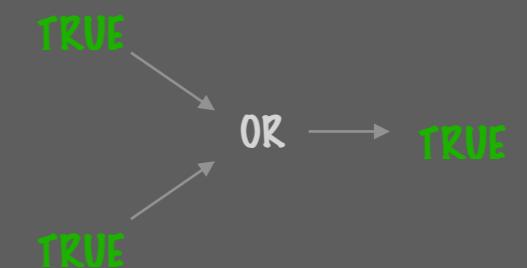
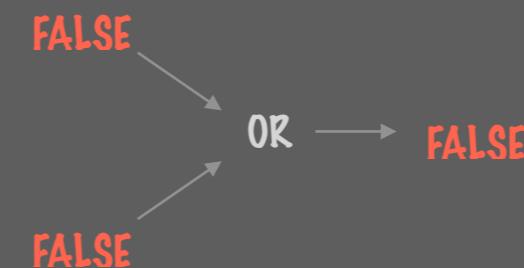
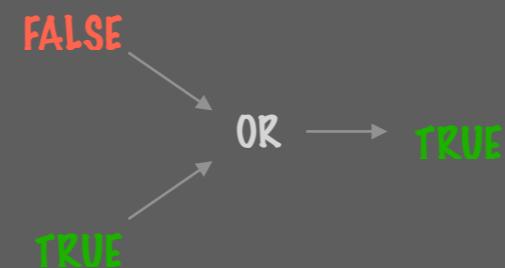
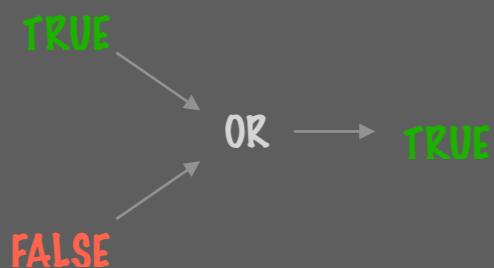
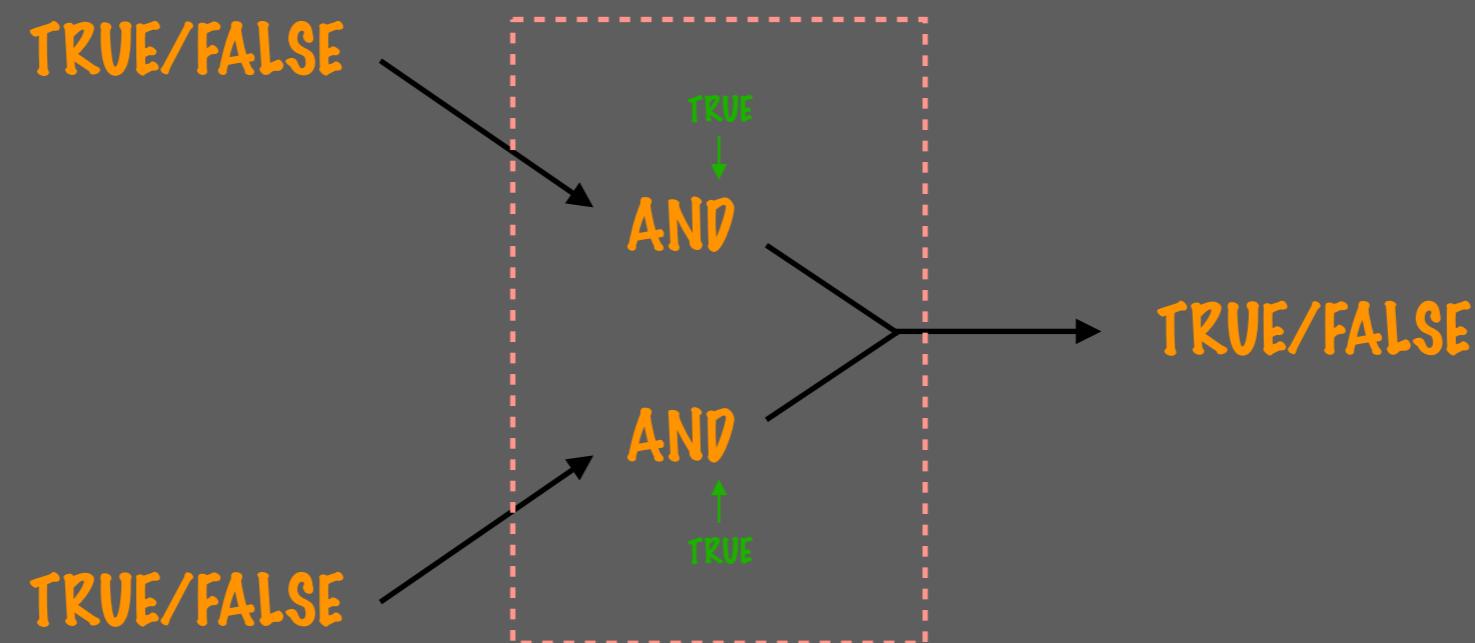
OR operator

We can combine two AND to make an OR operator



OR operator

We can combine two AND to make an OR operator

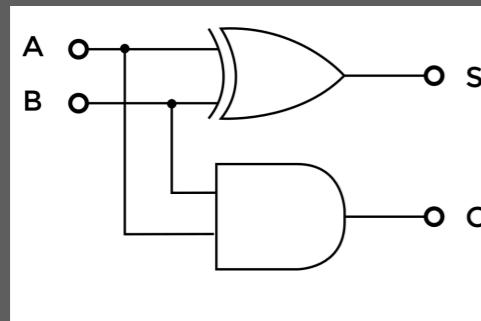


Logic gates

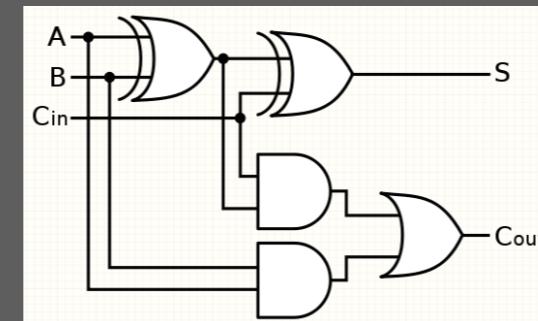
Name	NOT	AND	NAND	OR	NOR	XOR	XNOR																																																																																																
Alg. Expr.	\bar{A}	AB	\overline{AB}	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$																																																																																																
Symbol																																																																																																							
Truth Table	<table border="1"> <tr> <th>A</th> <th>X</th> </tr> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </table>	A	X	0	1	1	0	<table border="1"> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </table>	B	A	X	0	0	0	0	1	0	1	0	0	1	1	1	<table border="1"> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </table>	B	A	X	0	0	1	0	1	1	1	0	1	1	1	0	<table border="1"> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	0	<table border="1"> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	0	<table border="1"> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	1	<table border="1"> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	1
A	X																																																																																																						
0	1																																																																																																						
1	0																																																																																																						
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	1																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					

Arithmetic

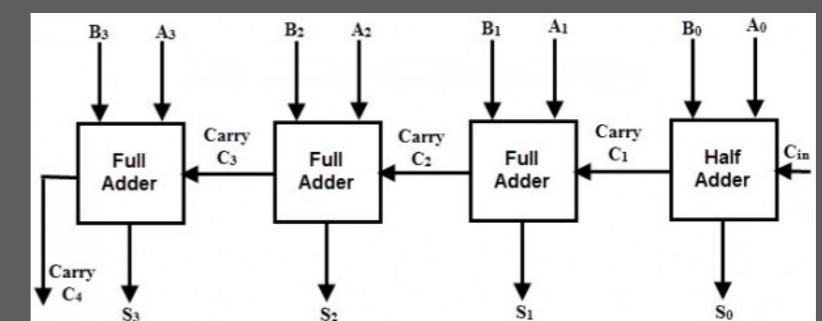
Half Adder



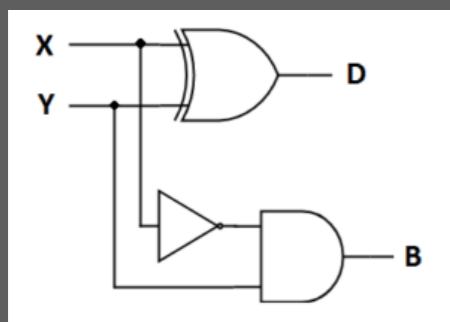
Full Adder



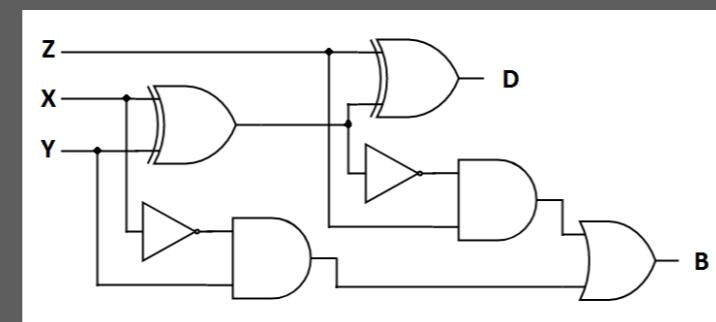
4 bit Adder



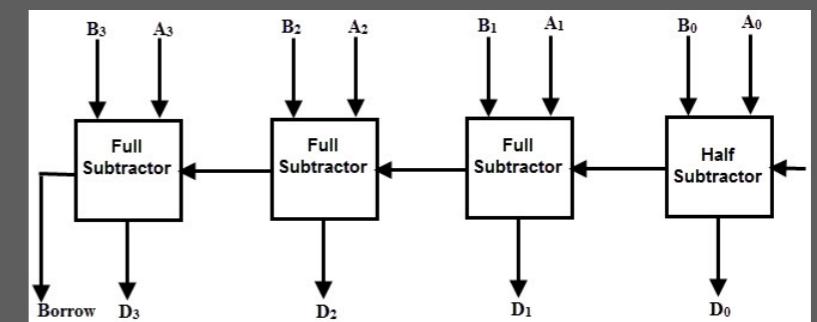
Half Subtractor

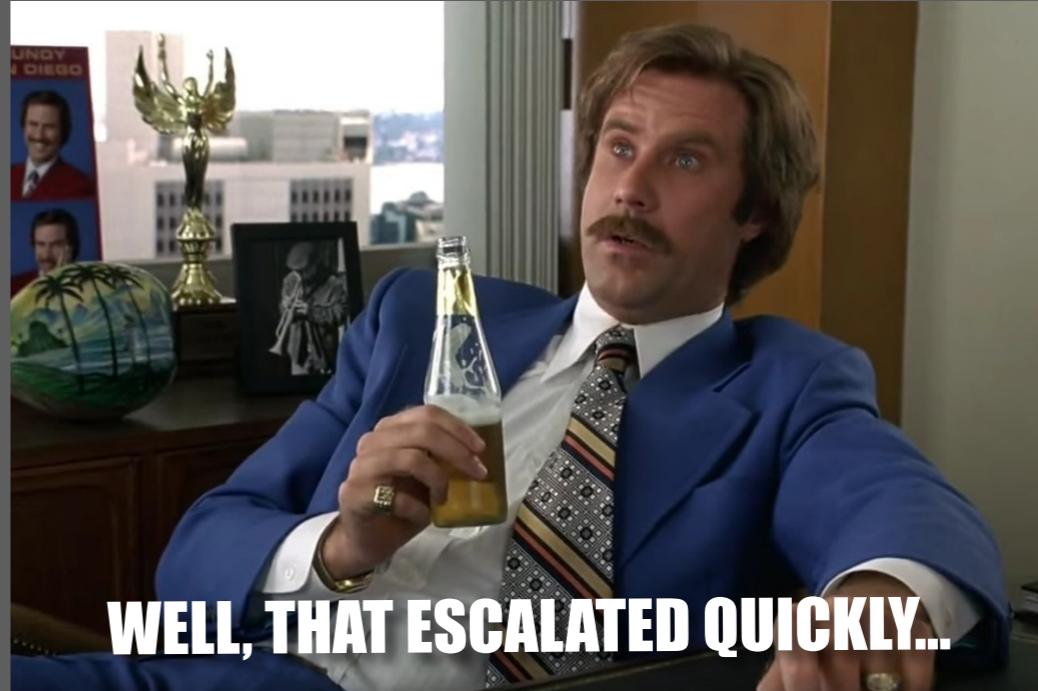
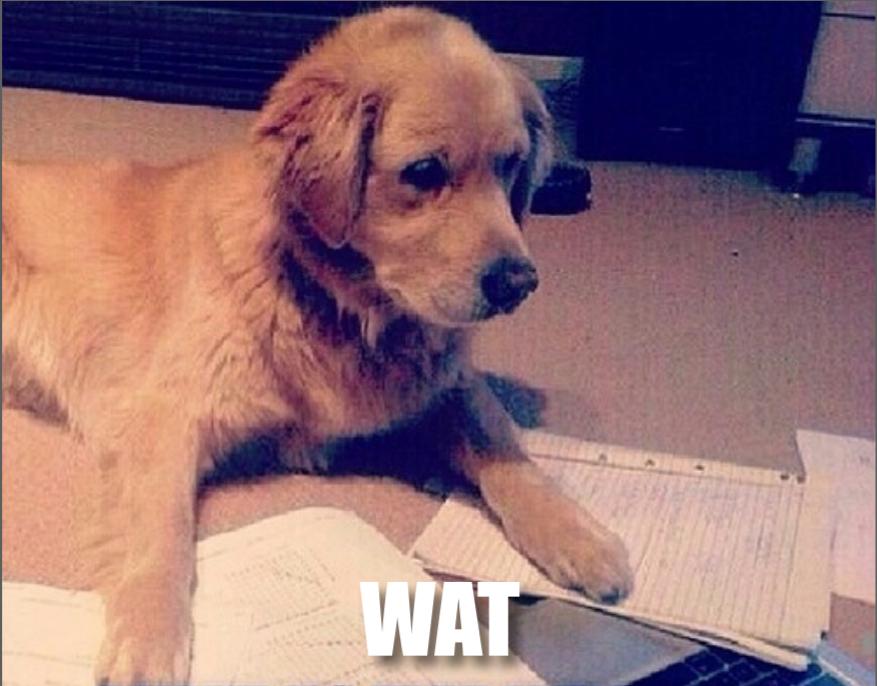


Full Subtractor



4 bit Subtractor





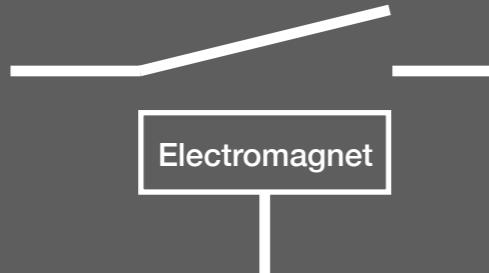
Don't worry. Nobody gets all the details...

Basic units are true, false and some logic

Numbers can be represented by combining true or false.

Arithmetic can be represented by combining logic.

Implementing all the above



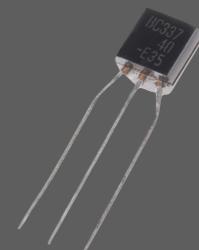
Mechanical Relay



Thermionic Valve



Vacuum Tube



Transistors

All of them are essentially AND gates

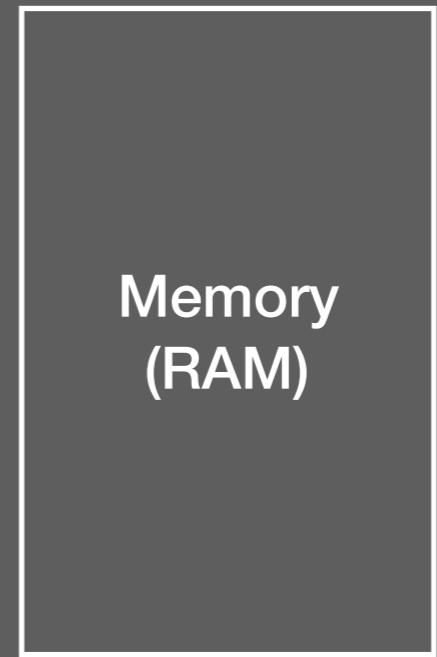
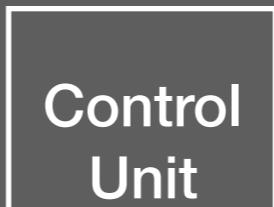
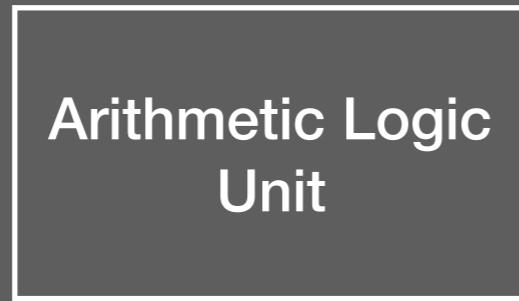
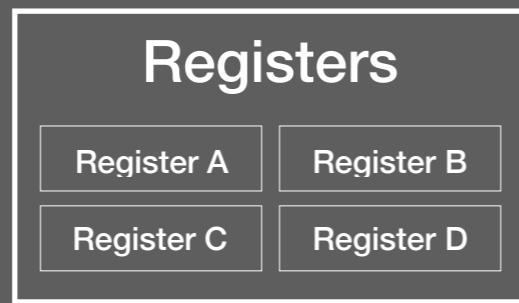
They can be combined for constructing Processing and Memory

CPU

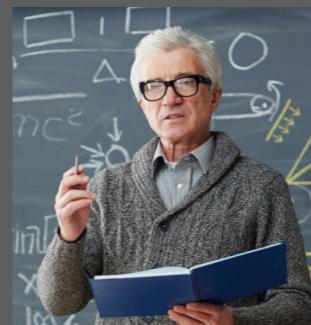
RAM

A very very
Inaccurate

Anatomy of a CPU



1000 0111
1100 1000
0011 1010
1000 1001
1100 1001
0011 1011
0101 0001
1101 0010
0000 0000
0000 0000



Adding with No Regrouping (A)
Find each sum.

$$\begin{array}{cccccccccc} 6 & 2 & 4 & 3 & 3 & 2 & 1 & 2 \\ +2 & +1 & +2 & +5 & +2 & +5 & +7 & +3 \\ \hline 4 & 4 & 2 & 6 & 8 & 5 & 7 & 4 \\ +5 & +3 & +5 & +2 & +1 & +1 & +2 & +1 \\ \hline 1 & 6 & 5 & 3 & 5 & 4 & 3 \\ +8 & +1 & +2 & +4 & +5 & +1 & +4 \\ \hline 1 & 3 & 2 & 7 & 4 & 6 & 2 \\ +6 & +1 & +3 & +2 & +4 & +2 & +5 \\ \hline 5 & 6 & 4 & 8 & 2 & 3 & 3 \\ +4 & +3 & +3 & +1 & +1 & +4 & +3 \\ \hline 6 & 1 & 2 & 5 & 1 & 1 & 4 \\ +2 & +5 & +4 & +3 & +2 & +4 & +2 \\ \hline 5 & 5 & 1 & 3 & 2 & 2 & 3 \\ +4 & +3 & +5 & +1 & +7 & +5 & +2 \\ \hline 2 & 2 & 3 & 4 & 6 & 3 & 8 \\ +5 & +4 & +4 & +2 & +3 & +1 & +1 \end{array}$$

Computer Program!!

1000 0111
1100 1000
0011 1010
1000 1001
1100 1001
0011 1011
0101 0001
1101 0010
0000 0000
0000 0000

Every program ever written is read by the CPU in this format

MACHINE CODE!

Layers of Abstraction

```
1000 0111  
1100 1000  
0011 1010  
1000 1001  
1100 1001  
0011 1011  
0101 0001  
1101 0010  
0000 0000  
0000 0000
```

ASSEMBLER

```
global _start  
section .text  
_start: mov rax, 1  
        mov rdi, 1  
        mov rsi, message  
        mov rdx, 13  
        syscall  
        mov rax, 60  
        xor rdi, rdi  
        syscall  
section .data  
message:db "Hello, World", 10
```

COMPILER

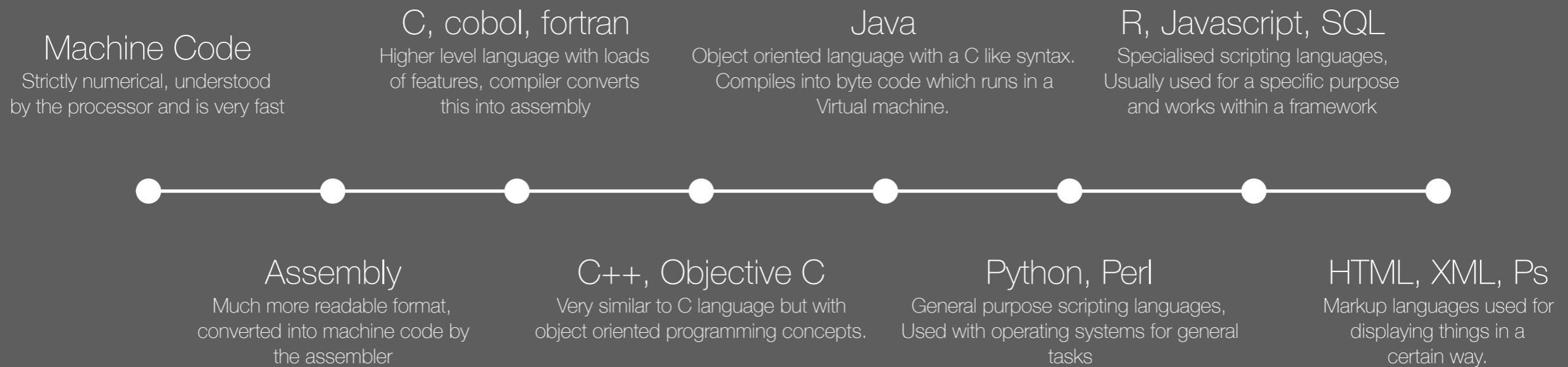
```
# include <stdio.h>  
  
int main()  
{  
    printf("Hello, World!");  
    return 0;  
}
```

Machine Code

Assembly

Higher-level Language

Layers of Abstraction (Processing)



Which is the **best** language?

readability, efficiency, data structure, available ecosystem, base use case, paradigm

Programming Languages

Language	Most Probable Scenario
Machine code	You are a robot
Assembly	Either you are developing embedded systems or writing a high performance software on a browser.
C	Developing an operating system or writing software for embedded systems/ IoT hardware or writing a package for python or R
C++	You are developing a desktop application
C#	You are developing a windows desktop application
Objective C	You are developing an iOS app
Java	You are developing an Android app or your company has already paid a tonne of money to Oracle
R	You are a data scientist with statistics background, You deal with maps and geo data and you cannot replace ggplot.

Language	Most Probable Scenario
Python	You are a data scientist with computer science background, working fields related to machine learning and AI
SQL	You are using a relational database
Perl	You are working with NLP or an old fashioned sysadmin
Bash	You are working on a linux/unix system/ server or you are copying instructions from web
Javascript	You are developing websites and other interactive stuff.
Scala	You are dealing with a LOT of data.
Haskell	You want to do proper functional programming and you like math a lot
COBOL	You are maintaining a very old mainframe or you are a bored retiree with loads of money :D
Fortran	You are a physicist doing simulations in old software or developing packages for R/Python.

Layers of Abstraction (Storage)

Raw data
Series of 0s and 1s on
a storage medium

File System
The disk is partitioned and in
Each partition data is written in a
Certain method

Databases
Adds a layer of strict structure to
the filesystem. Improves the efficiency of the
Storage and retrieval but sacrifices flexibility

Distributed File System
These are filesystems which reside
In multiple systems but behaves as one.

Which is the **best** way to store data?

Interesting new developments



Rust - a multi-paradigm system programming language focused on safety, especially safe concurrency. Rust is syntactically similar to C++, but is designed to provide better memory safety while maintaining high performance.

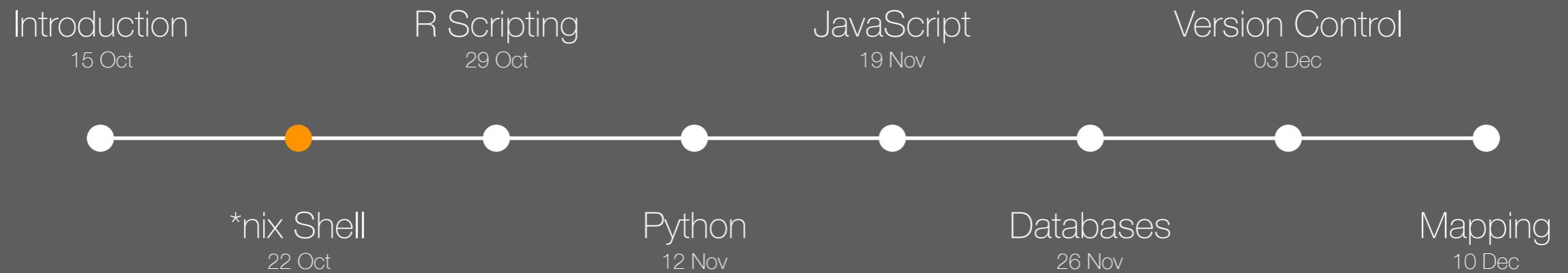


Go - a statically typed, compiled programming language designed at Google. It is syntactically similar to C, but with memory safety, garbage collection, structural typing, and CSP-style concurrency.



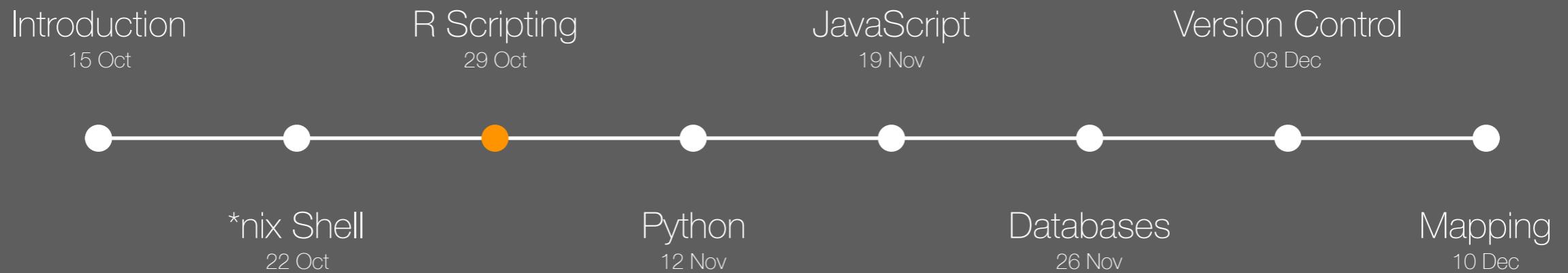
Julia - a high-level, high-performance, dynamic programming language. While it is a general purpose language and can be used to write any application, many of its features are well-suited for high-performance numerical analysis and computational science.

Overview of Sessions



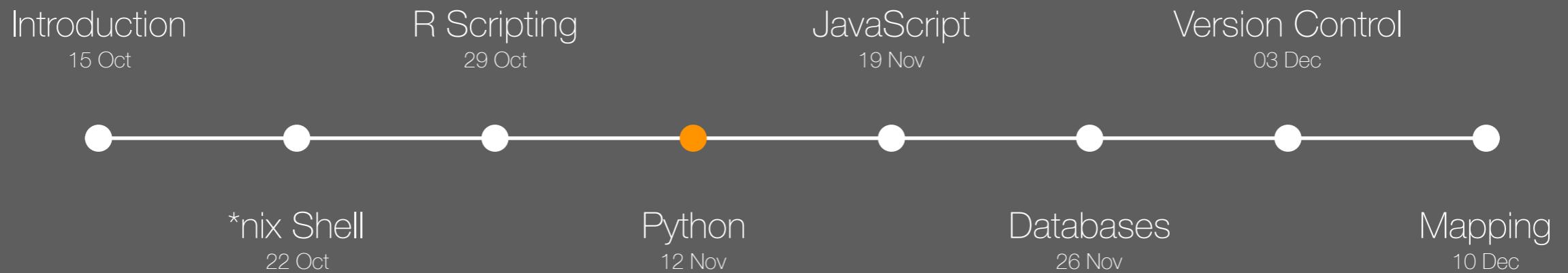
Readability	Low
Efficiency	High
Data structure	Text
Ecosystem	Linux and Unix
Base use case	SysAdmin
Main paradigm	Piping

Overview of Sessions



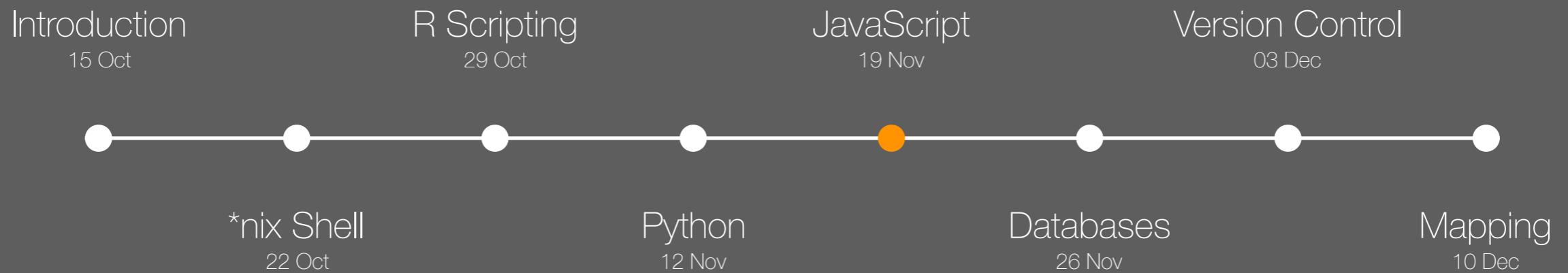
Readability	Fair
Efficiency	Fair
Data structure	Vector
Ecosystem	CRAN
Base use case	Statistics
Main paradigm	Functional

Overview of Sessions



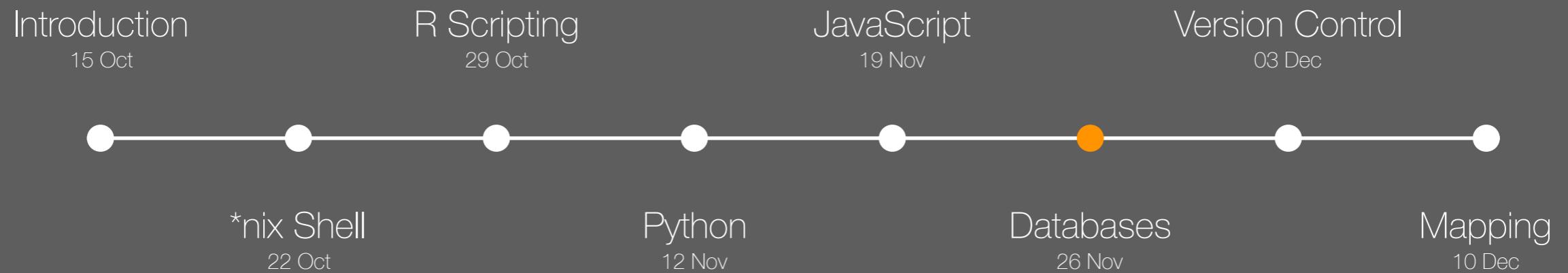
Readability	High
Efficiency	Fair
Data structure	Lists
Ecosystem	**multiple**
Base use case	Scientific Comp.
Main paradigm	Imperative

Overview of Sessions

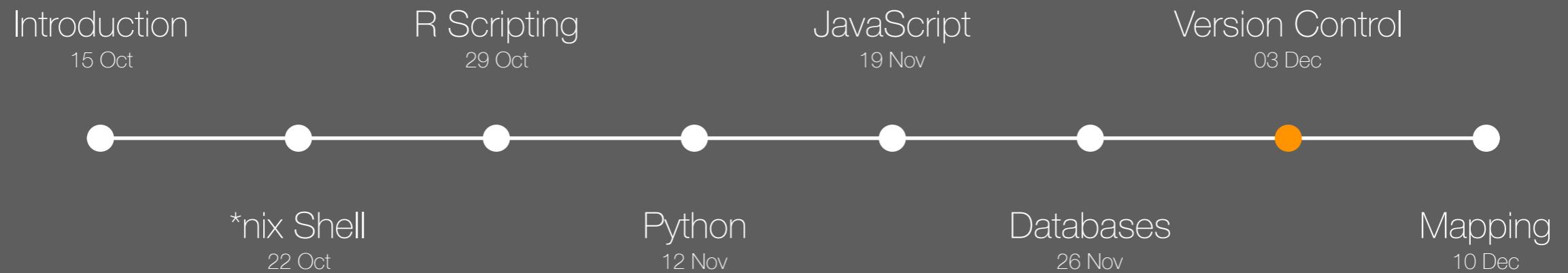


Readability	Fair
Efficiency	Low
Data structure	Object
Ecosystem	npm
Base use case	Web dev.
Main paradigm	Asynchronous

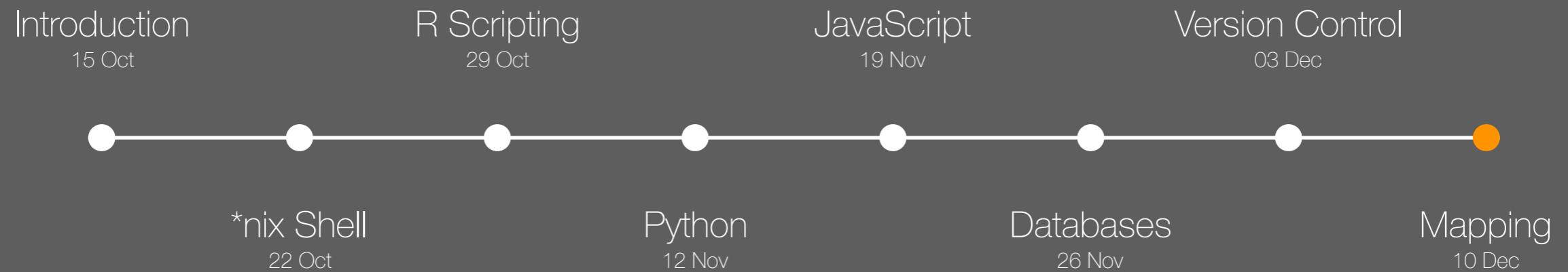
Overview of Sessions



Overview of Sessions



Overview of Sessions



Questions