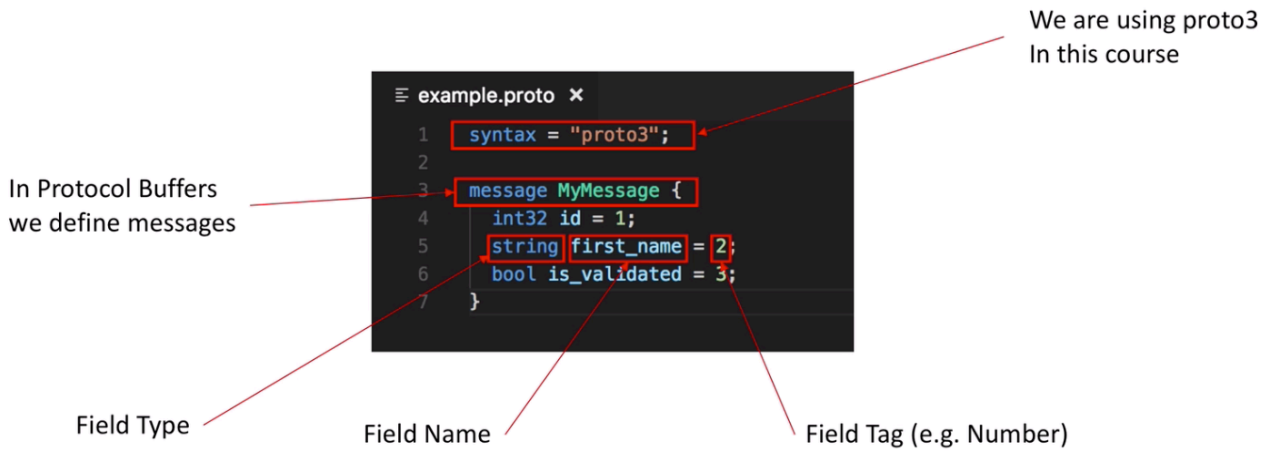
2 - Protocol Buffers Basics I

First Message

First message - Introduction

- Here's our first message:



Scalar Types

Scalar Types Number

- Numbers can take various forms based on what values you expect them to have: double, float, int32, int64, uint32, uint64, sint32, sint64, fixed32, fixed64, sfixed32, sfixed64
- Integer: For now, let's use int32 (There's a discussion in the advanced section of advantages of each specific type)
- Floating point numbers:
  - float (32 bits)
  - double (64 bits) - for more precision (if you really need it)

Scalar Types Boolean

- Boolean can hold the value True of False
- It is represented as bool in protobuf

Scalar Types String

- String represents an arbitrary length of text
- It is represented as string in Protobuf
- A string must always contain UTF-8 encoded or 7-bit ASCII text.

Scalar Types Bytes

- Bytes represents any sequence of byte array.
- It is represented as bytes in Protobuf
- It will be up to you to interpret what these bytes mean
- For example you could use these bytes to include a small image

Scalar Types Summary

- Let's create a message Person that has
  - int 32 (Age)
  - string (first name)
  - string (last name)
  - bytes (small picture)
  - bool (profile verified)
  - float (height)

/src/basics/scalar-types.proto

```
syntax = "proto3";

message Person {
    int32 age = 1;
    string first_name = 2;
    string last_name = 3;
    bytes small_picture = 4;
    bool is_profile_verified = 5;
    float height = 6;
}
```

Tags

- In Protocol Buffers, field names are not important! (but when programming the fields are important)
- For protobuf the important element is the tag
- Smallest tag: 1
- Largest tag:

$2^{29} - 1$, or 536,870,911

- You also cannot use the numbers 19000 through 19999
- Tags numbered from 1 to 15 use 1 byte in space, so use them for frequently populated fields

- Tags numbered from 16 to 2047 use 2 bytes
- There's a concept of reserved tag that we'll see in the advanced lectures

Repeated Fields

- To make a "list" or an "array", you can use the concept of repeated fields
- The list can take any number (0 or more) of elements you want
- The opposite of repeated is "singular" (we don't write it)
- Let's add a list of phone numbers to our Person example!

/src/basics/repeated-fields.proto

```
syntax = "proto3";

message Person {
    int32 age = 1;
    string first_name = 2;
    string last_name = 3;
    bytes small_picture = 4;
    bool is_profile_verified = 5;
    float height = 6;

    repeated string phone_numbers = 7;
}
```

Comments

- It is possible to embed comments in your .proto file
- It is acutally recommended to use comments as a form of documentation for your schemas.
- Comments can be of these two forms:
    - // this is a comment
    - /* this is a
    - * multiline comment */
- Let's add comments to our Person!

Default Values for Fields

- All fields, if not specified or unknown, will take a default value

- bool:false
- number (int32, etc...): 0
- string: empty string
- bytes: empty bytes
- enum:first value

- repeated:empty list

Enumerations (Enums)

- If you know all the values a field can take in advance, you can leverage the Enum type
- **The first value of an Enum is the default value**
- Enum must start by the tag 0 (which is the default value)
- Let's add an Enum to our Person for the field Eye Color

/src/basics/enums.proto

```
// The syntax for this file is proto3
syntax = "proto3";

/* Person is used to identify users
* across our system */
message Person {
    // the age as of the person's creation
    int32 age = 1;
    // the first name as documented in the signup form
    string first_name = 2;
    string last_name = 3; // last name as documented in the signup form
    // small_picture represents a small .jpg file
    bytes small_picture = 4;
    bool is_profile_verified = 5;
    // height of the person in cms
    float height = 6;

    // a list of phone numbers that is optional to provide at signup
    repeated string phone_numbers = 7;

    // we currently consider only 3 eye colors
    enum EyeColor {
        UNKNOWN_EYE_COLOR = 0;
        EYE_GREEN = 1;
        EYE_BROWN = 2;
        EYE_BLUE = 3;
    }

    // it's an enum as defined above
    EyeColor eye_color = 8;
}
```

Solutions to Practice Exercise I