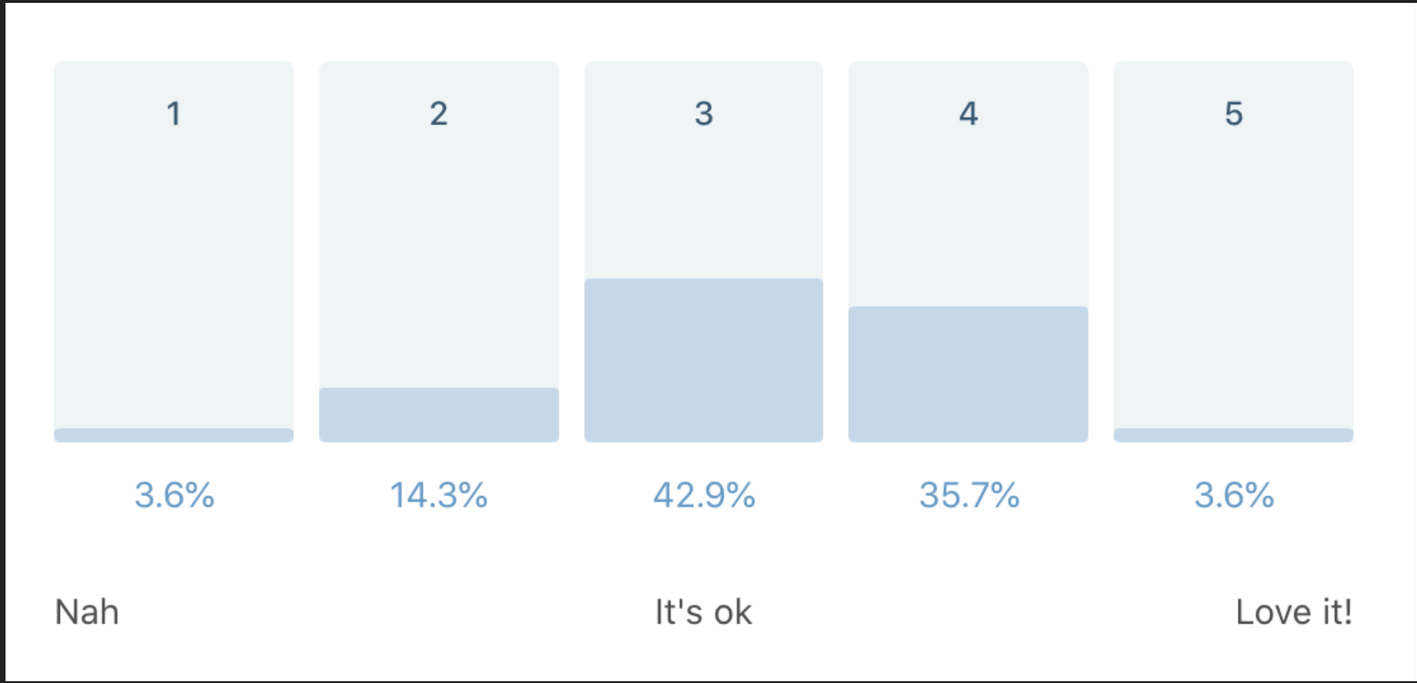
SEPTEMBER 9TH 2020

ELEMENTARY PROGRAMMING

SOME COVID BEST PRACTICES BEFORE WE START

- If you fill ill, go home
- Keep your distance to others
- Wash or sanitise your hands
- Disinfect table and chair
- Respect guidelines and restrictions

FEEDBACK CHECK



Average 3.2

FEEDBACK IN WORDS

What was good

"Good engagement"

"You seem to know your stuff"

"You didn't forget us on Zoom"

To improve

"Talk too fast"

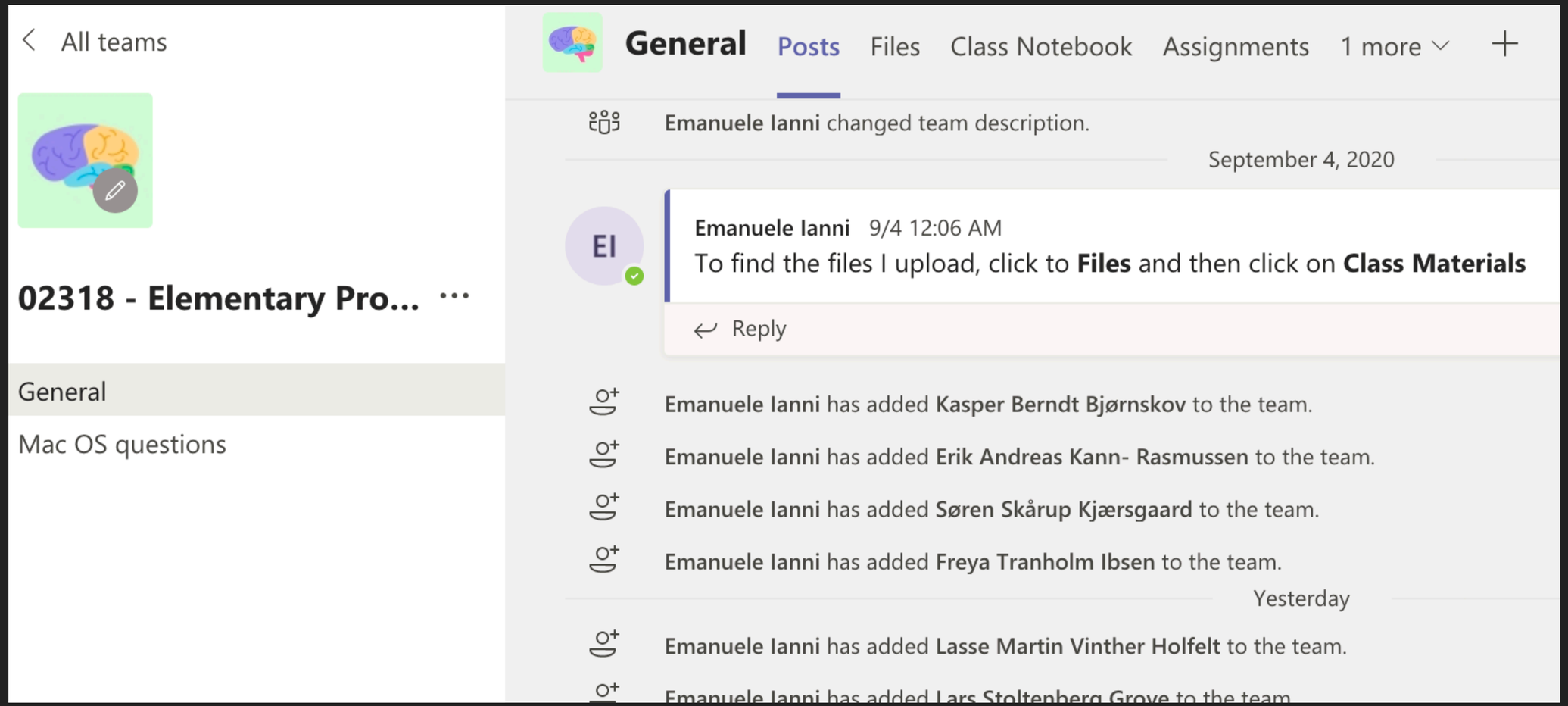
"Your accent"

"Give us more C!"

NEW FEEDBACK

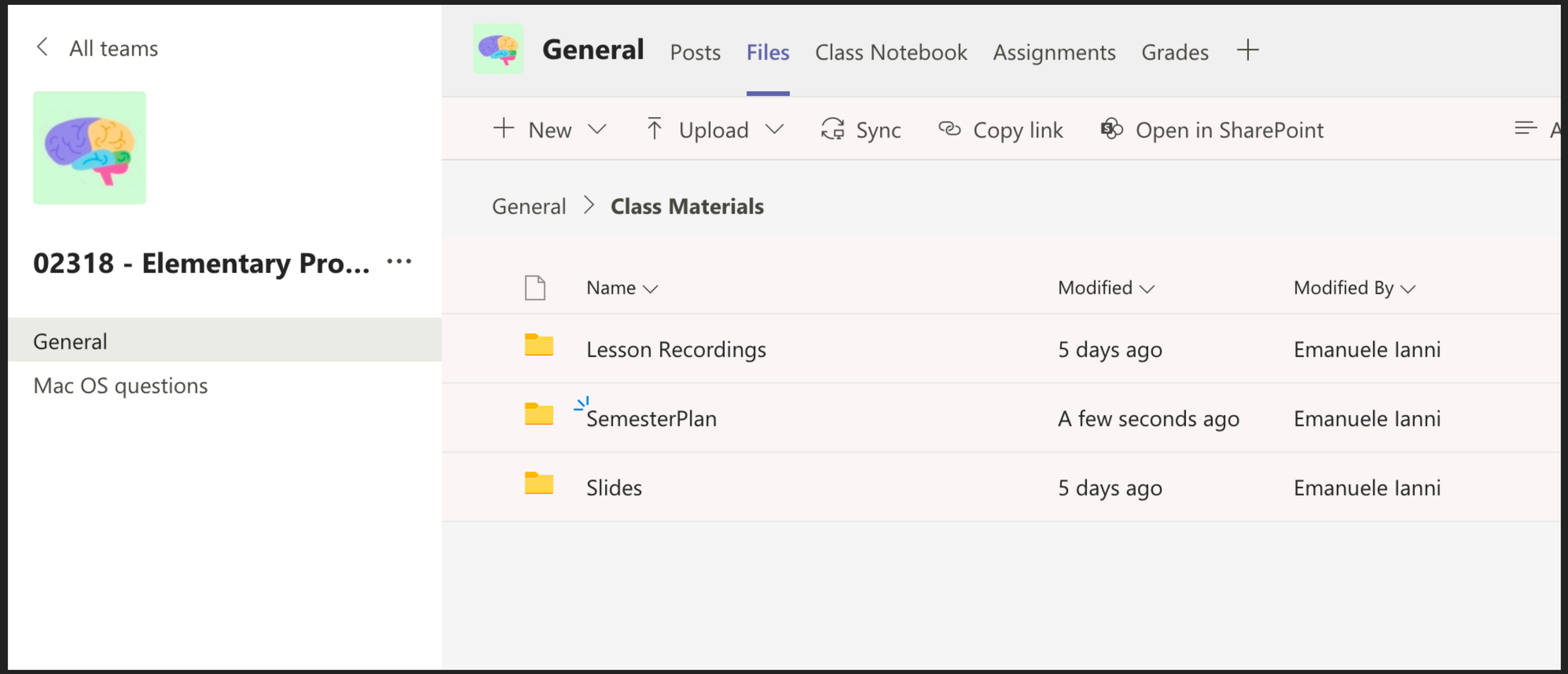
- I would really like for you to take a survey at the end of the session
- Feedback is important, please take the time to do it
- Pretty please <3
- Type this in your browser http://bit.ly/elemprog2

SOME INFO ABOUT THE COURSE - TEAMS



http://bit.ly/elemprogteams

SOME INFO ABOUT THE COURSE – MATERIAL



http://bit.ly/elemprogteams

SOME INFO ABOUT THE COURSE - MATERIAL

- There is a repository on GitHub here: https://github.com/invasionofsmallcubes/elementary-programming-dtu
- You will find here all the code I use for the lesson or when I try something to prepare the lesson

SOME INFO ABOUT THE COURSE - OFFICE HOURS

- You can write me on Teams
- You can send me an email at emia@dtu.dk
- If you need a call we agree on it, it will be online

http://bit.ly/elemprogteams

SOME INFO ABOUT THE COURSE - GRADE

- Two assignments during the class
- They will count 30% each for the final grade
- One final test that will account for 40%
- I will double check this with DTU Compute

http://bit.ly/elemprogteams

ABOUT C

- C is a **general-purpose**, **procedural** computer programming language supporting structured programming, lexical variable scope, and recursion, with a static type system.
- By design, **C provides constructs that map efficiently to typical machine instructions**. It has found lasting use in applications **previously coded in assembly language**.
- Such applications include operating systems and various application software for computer architectures that range from supercomputers to PLCs and embedded systems.

From *Wikipe*

ABOUT C

- C is an **imperative procedural language**.
- It was designed to be **compiled** to provide low-level access to memory and language constructs that map efficiently to machine instructions, all with minimal runtime support.
- Despite its low-level capabilities, the language was designed to **encourage cross-platform programming**. A standards-compliant C program written with portability in mind can be compiled for a wide variety of computer platforms and operating systems with few changes to its source code.

From *Wikipe*

COMPILERS

- In computing, a compiler is a **computer program** that **translates** computer code written in one programming language (the source language) into another language (the target language).
- A compiler is likely to perform many or all of the following operations: **preprocessing**, **lexical analysis**, **parsing**, s**emantic analysis**, conversion of input programs to an intermediate representation, code optimization and code generation.

From *Wikipe*

BACK TO BASICS

Variable Declarations

Variable Initilization

```
1  #include <stdio.h>
2
3  int main() {
4      int j;
5      int i;
6      printf("Tell i\n");
7      scanf("%d", &i);
8      printf("Tell j\n");
9      scanf("%d", &j);
10     int sum = i + j;
11     printf("Sum is %d", sum);
12 }
```

Body of the pr

VARIABLES DECLARATION AND INITIALIZATION

```
int main(void) {
    int count;
    count = 0;
}
```

data type

variable name

assignment

DATA TYPES

- In compiled languages data types exists to tell to the compiler how we want to use data
- Explicitly defining data types allows us to tell in advance how much memory we want to reserve for a variable
- It also helps us while programming because a compiler will be able to recognise a wrong assignment

INT

```
int count = 0;
```

- you can store integers values
- takes up 4 bytes of memory (32 bit)
- can contain a value from **-2^{31}** to **$2^{31}-1$**

CHAR

```
char val = 'a';
```

- you can store single characters
- always take 1 byte of memory (8 bit)
- can contain a value from **-128** to **127**

FLOAT

```
float val = 10.4f;
```

- you can store decimal values
- they take up to 4 bytes of memory (32 bit)
- can contain a value from **1.2E-38** to **3.4E+381.2E-38 to 3.4E+38**
- 6 decimal precision

DOUBLE

```
double val = 10.4;
```

- you can store decimal values
- they take up to 8 bytes of memory (64 bit)
- can contain a value from **2.3E-308** to **1.7E+308**
- 15 decimal precision

VOID

```
void my_function(){}
```

- it’s not a data type but just a type
- a function that return a void type, doesn't return anything. void can be seen as a placeholder for "nothing" (it's more complex than that but for now we are fine with this definition)

BACK TO BASICS

function definition

function call

arithmetic operator

assignment operator

```
1  #include <stdio.h>
2
3  int main() {
4     int j;
5     int i;
6     printf("Tell i\n");
7     scanf("%d", &i);
8     printf("Tell j\n");
9     scanf("%d", &j);
10    int sum = i + j;
11    printf("Sum is %d", sum);
12 }
```

ASSIGNMENT OPERATOR

```
data_type variable_name = value;
```

- whenever we define a variable the first thing we do is assign a value
- = is the assignment operator

ARITHMETIC OPERATORS

```
1 #include <stdio.h>
2 int main(void) {
3   int y = 5;
4   int x = y + 1;
5   x = x * 11;
6   printf("x is %d\n", x); // => x is 66
7 }
```

- We can do arithmetic operations, we can add (+), subtract (-), multiplay(*) and divide (/)

ARITHMETIC OPERATORS

```
int m = 13 % 4;   // m is 1
```

- We also have % which is the modulus operator which gives us the reminder of a/b

FUNCTIONS

```
return_type my_function(type1 arg1, type1 arg2, …, type argN){
    return_type something;
    //I do something here
    …
    return something;
}
```

- a function is a block of code between { and } with a name
- the extensibility of the language is given by functions

CONVERSION SPECIFICATION

%m.pX or %-m.pX

- ***m*** is called **minimum field width**, which means will always displays at least m numbers (or spaces if it's smaller)
- ***p*** is the the **precision**
- ***X*** is called **conversion specifier**
- The **-** sign tells you to align left instead of right

EXAMPLES OF CONVERSION SPECIFIER

- ***d*** is an integer in base 10
- ***e*** express floating point numbers in exponential format
- ***f*** express floating point numbers in fixed decimal format
- ***g*** express floating point number as either ***f*** or ***e*** depending on the number size

LETS RUN THE PRINTF

```c
#include <stdio.h>

int main() {
  int i = 40;
  float x = 839.21;
  printf("|%d|%5d|%-5d|%5.3d|\n", i, i, i, i);
  printf("|%10.3f|%10.3e|%-10g|\n", x, x, x);
}
```

```
➜  lesson02 git:(master) ✗ ./show-formatting.out
|40|   40|40   |  040|
|   839.210| 8.392e+02|839.21    |
```

THE SCANF FUNCTION

- The **scanf** function is very powerful
- It uses pattern matching to recognise a sequence of chars to be converted using conversion specifiers
- It ignores spaces

SCANF EXAMPLE

```
1  #include <stdio.h>
2
3  int main(void) {
4    printf("Write two decimals and two floats,
5           separated by spaces: \n");
6    int x, y;
7    float z, w;
8    scanf("%d%d%f%f", &x, &y, &z, &w);
9    printf("You wrote %d %d %f %f\n", x, y, z, w);
10 }
```

COMPUTING THE DIMENSIONAL WEIGHT OF A BOX

- A shipping company doesn't like if the box are large but very light
- They take space but don't produce the same amount of value of a heavy box
- For this reason shipping companies don't charge only by weight but also by volume

COMPUTING THE DIMENSIONAL WEIGHT OF A BOX

- if the box's "dimensional" or "volumetric" weight exceeds the actual weight, the shipping fee is based on dimensional weight
- dimensional weight = volume / 166
- the dimensional weight needs to be round up

NEW FEEDBACK

- I would really like for you to take a survey at the end of the session
- Feedback is important, please take the time to do it
- Pretty please <3
- Type this in your browser http://bit.ly/elemprog2

SOME COVID BEST PRACTICES BEFORE WE LEAVE

- Disinfect table and chair
- Maintain your distance to others
- Wash or sanitise your hands
- Respect guidelines and restrictions outside