Basteltruppe 3

Structs

Programming is the Art of using Information

Since the beginning of the computer era, the single most important purpose has ever been and will ever been the usage of information.

Let's quickly recap the milestones

Level 1

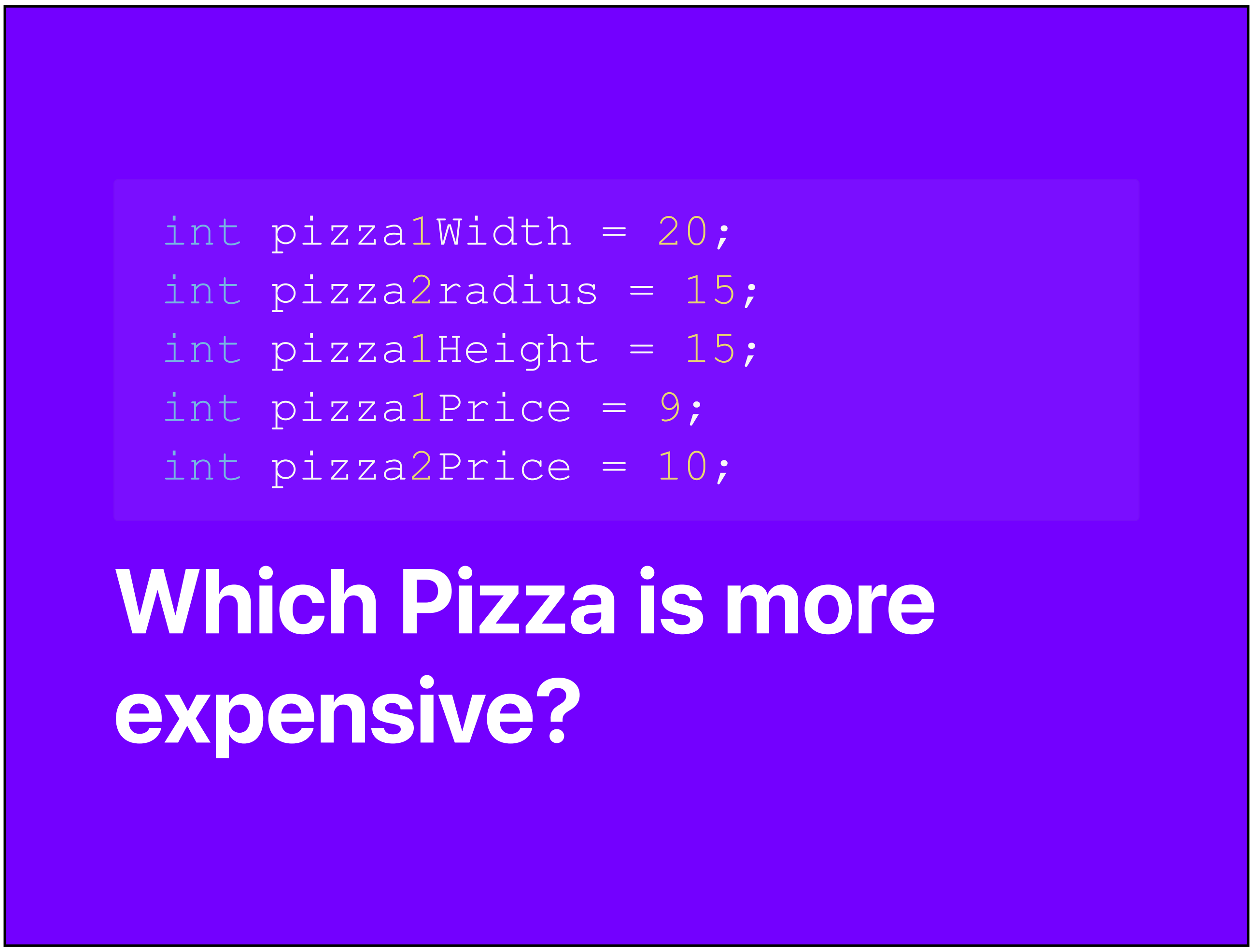
Saving and Loading Information

- Use some sort of (semi)permanent record to store and load data
- Perform calculation on it

Level 1

```
int pizza_radius =
15;
int pizza_price =
6;
```

If you just store information, you will manage to achieve something, but you will quickly be stuck in a bottomless tar pit, as the information is there, but it is not yet connected. The German name for computer science (Informatik) better describes this branch of science, as it is less about computers and more about the art of information. Only by transcribing the data in such a way that information that is linked together stays together through the structure.



```
int pizza1Width = 20;
int pizza2radius = 15;
int pizza1Height = 15;
int pizza1Price = 9;
int pizza2Price = 10;
```

Which Pizza is more expensive?

The ability to store data without further technique loses the implicit links between them.

Level 2

Grouping Information

- Information is seldom in a vacuum
- Often, there is information that belongs together

Level 2

```
struct RoundPizza{
    int radius;
    int price;
}

struct
RectanglePizza{
    int width;
    int height;
    int price;
}
```

By grouping information that belongs together to its respective contexts, we achieve more clarity! Easy to expand or modify

C is at Level 2

Level 3 Generalize Specialize

- Groups of Data can be generalised
- Generalisation allows to share common data between similar types
- E.g. UI system, where you generalise the closing of windows
- E.g. Videogame, where different enemies share a health bar etc.
- Specialisation is the reverse direction
- You specialise something generic

Level 3

```
class Pizza{
     int getArea();
     int price;
}

class RoundPizza extends
Pizza{
     int radius;
}

class RectanglePizza
extends Pizza{
     int width;
     int height;
}
```

Instead of duplicating the concept of a Pizza that has some area and a price for the round and rectangle pizza, we generalise the concepts of an Area and a price and specialise into a round or rectangle pizza!

Level 4
Behaviour (Polymorphism)

- Data is just the manifestation of a behaviour.
- What is it your application really requires? The data or a specific method?
- Idea: Separate data from behaviour

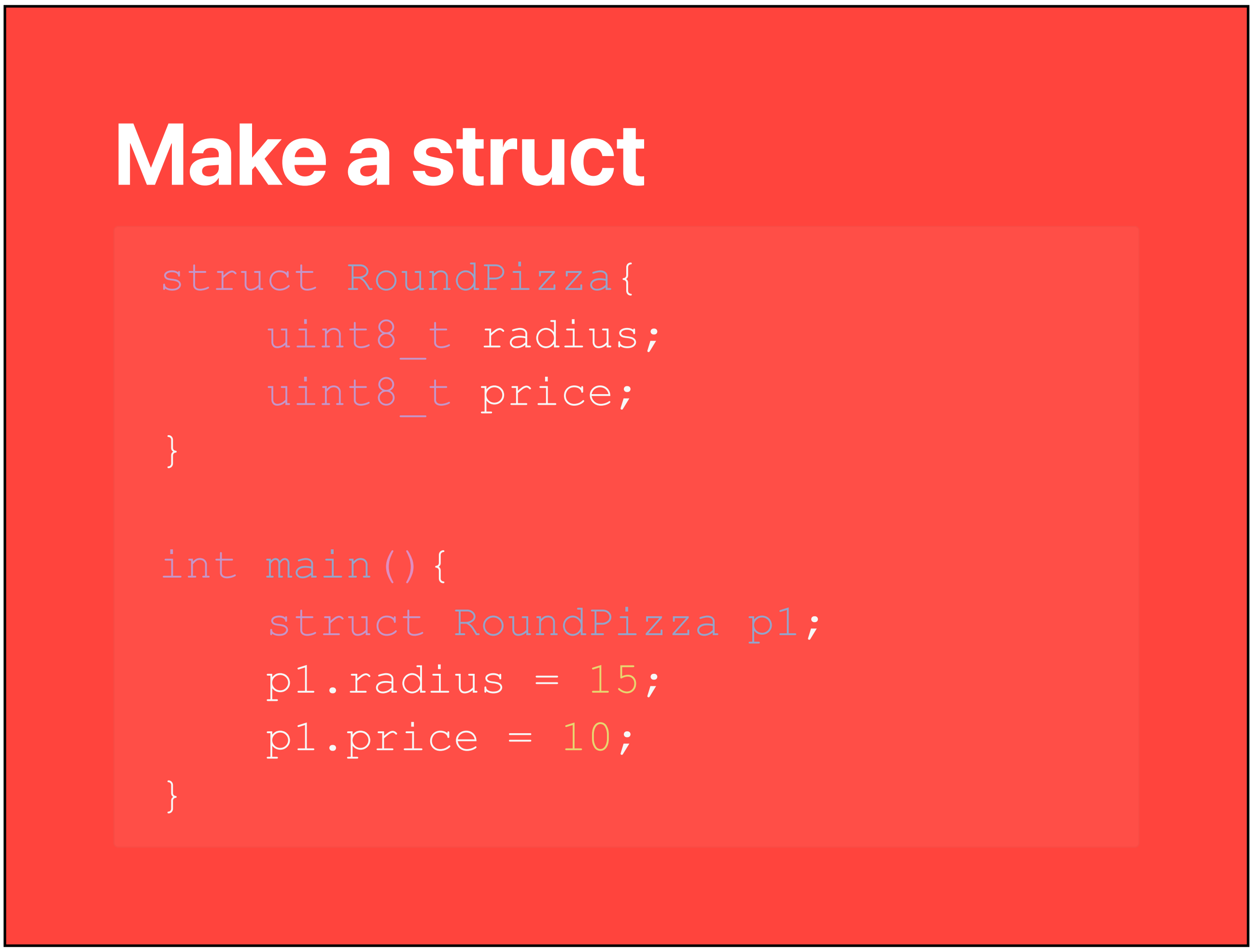
Level 4

```
interface Price{
    int getPrice();
}
interface Size{
    int getSize();
}

virtual class Pizza
implements Price, Size{}

class RoundPizza extends
Pizza{
    int getPrice(){}
    int getSize(){}
}
```

- By defining the behaviour, you reduce the dependancy on specific implementations. Who says that a pizza cannot be stored in a database?
- If somebody uses a radius in a pizza, you don't actually care. What you care about is the getSize!

Albeit being stuck at Level 2, C does the trick

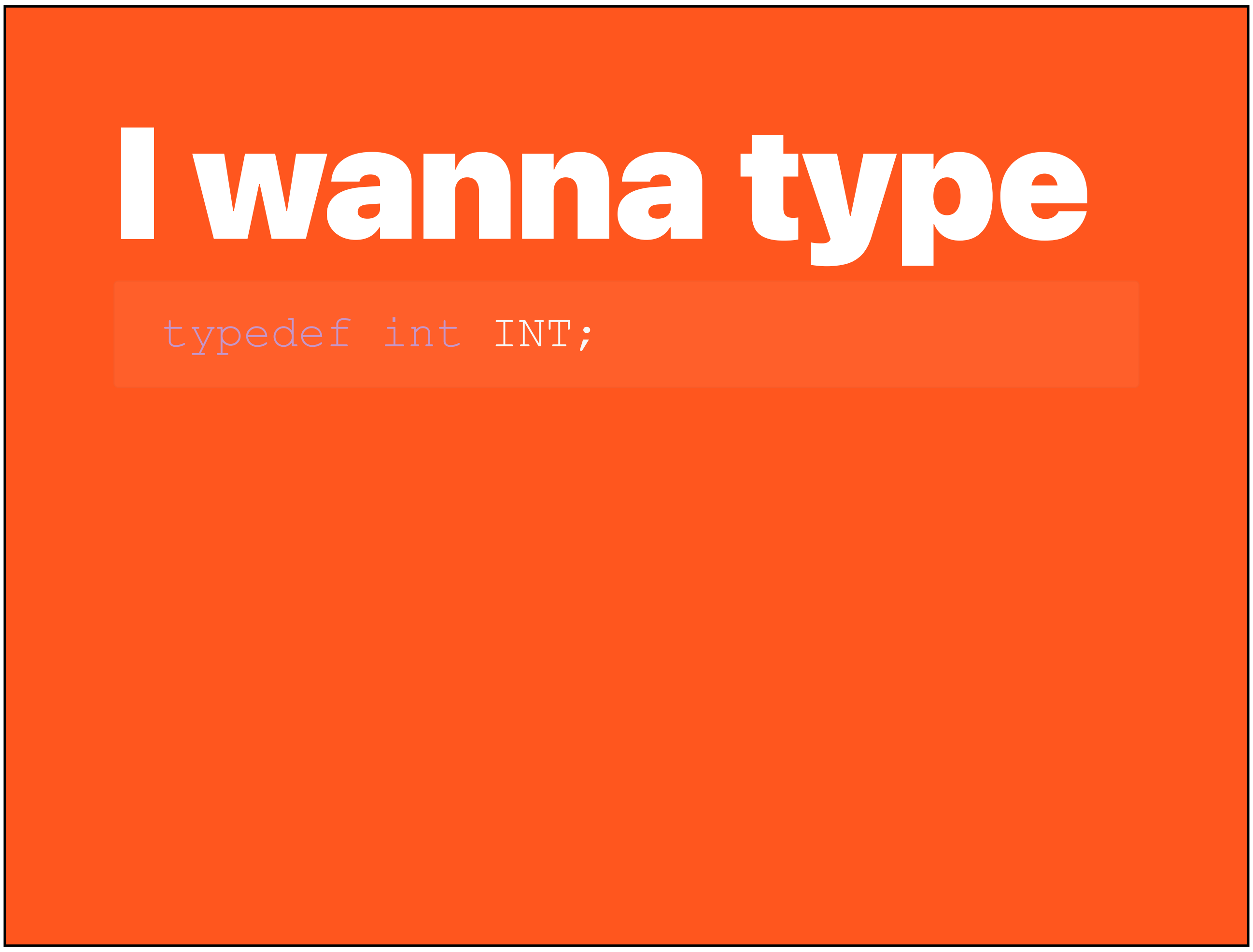
- Group data? Struct!
- Generalize? Struct object in a struct!



Make a struct

```
struct RoundPizza{
    uint8_t radius;
    uint8_t price;
}

int main(){
    struct RoundPizza p1;
    p1.radius = 15;
    p1.price = 10;
}
```

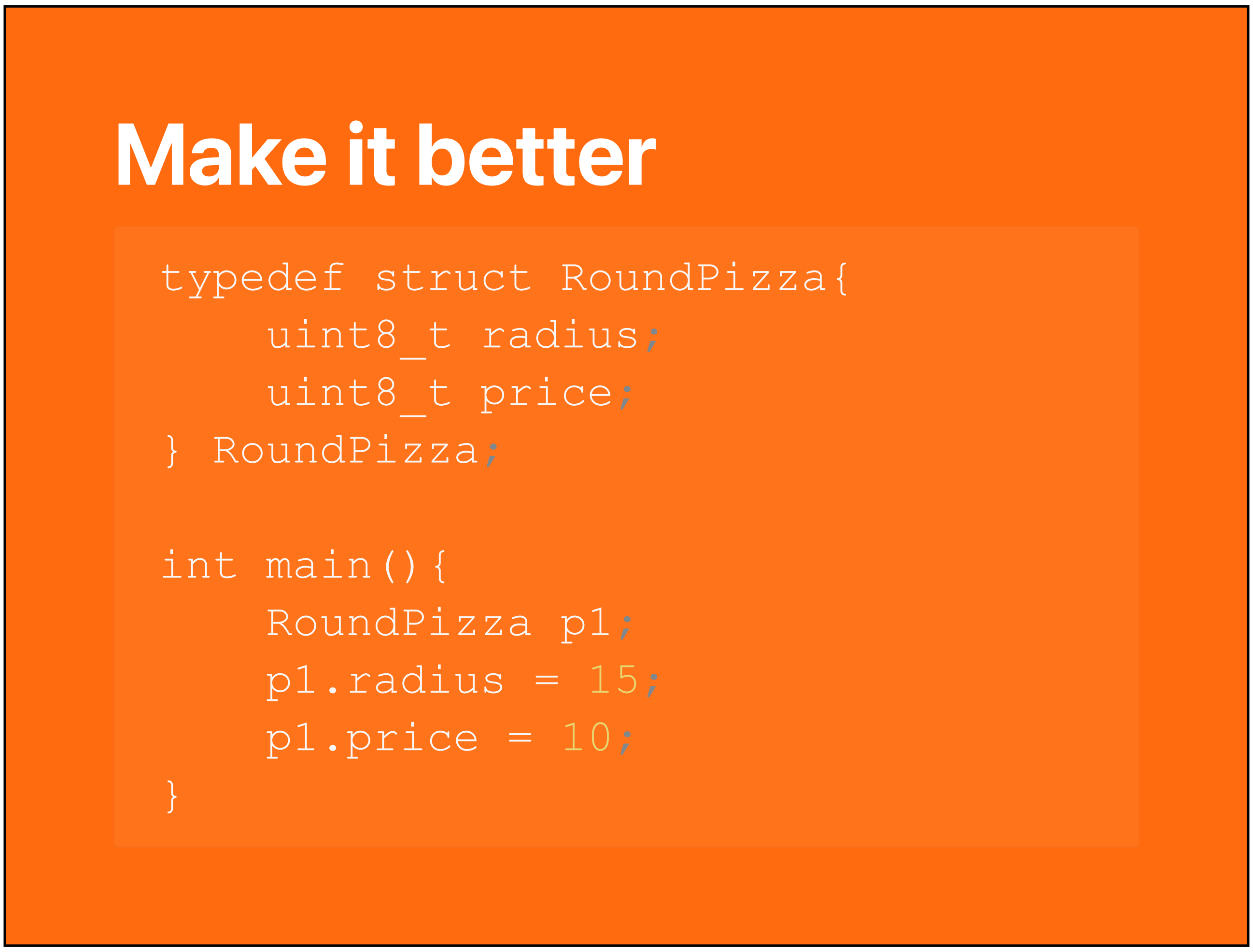
Here you create a struct that contains two members. To create an instance of this struct, you need to write `struct RoundPizza p1;`, which creates p1 as an instance of a RoundPizza. Funfact! The content of this struct is at this time undefined. It can be zero. Be careful whenever you assume "I'm sure no idiot would allow this to be something besides zero. Right???"



I wanna type

```
typedef int INT;
```

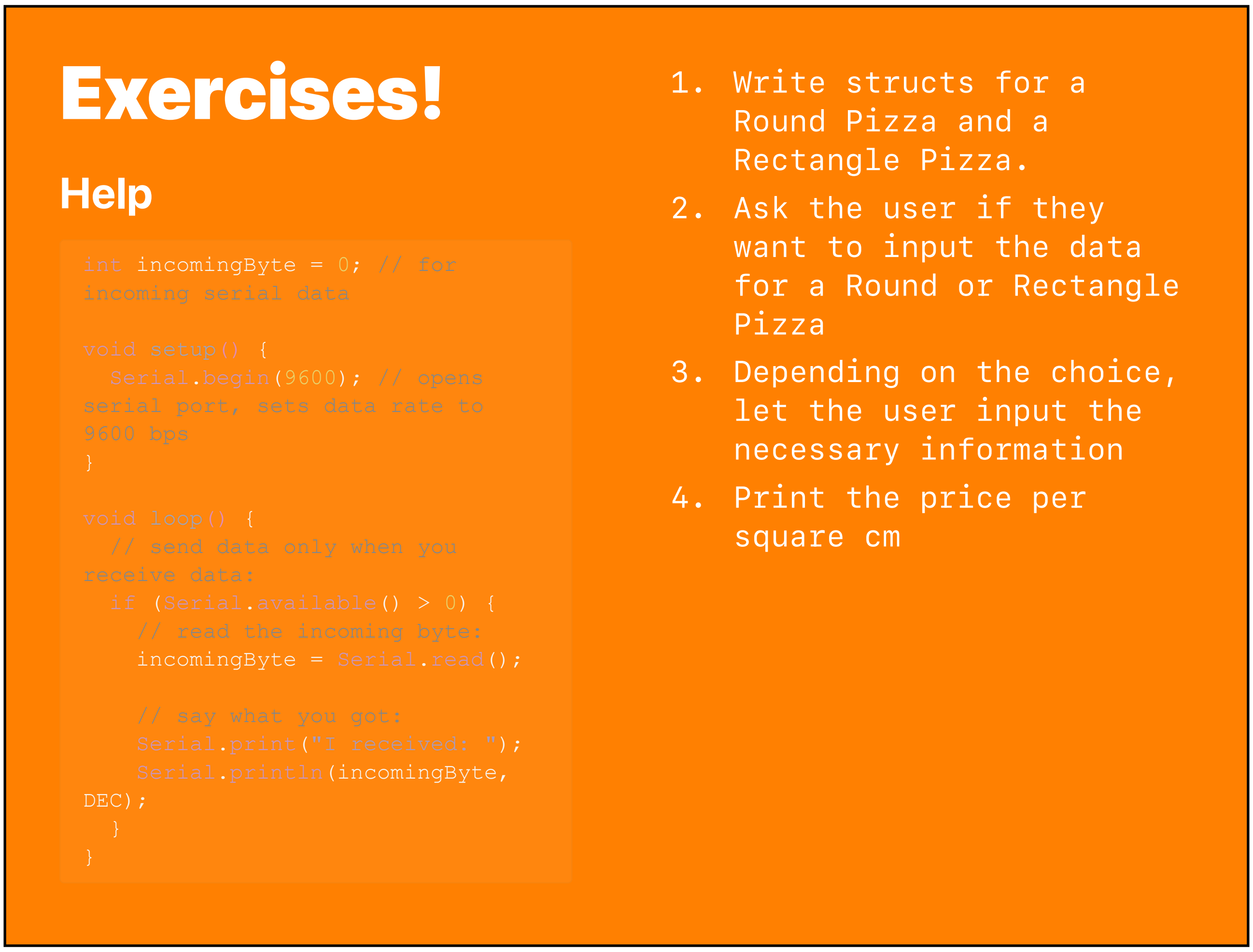
With typedef you say that the first operant shall now also be known as the second operant



Make it better

```c
typedef struct RoundPizza{
    uint8_t radius;
    uint8_t price;
} RoundPizza;

int main(){
    RoundPizza p1;
    p1.radius = 15;
    p1.price = 10;
}
```

As typing the struct before each instance feels weird, I like to use typedef to make it a little bit easier to create an instance of this struct.



Exercises!

Help

```
int incomingByte = 0; // for
incoming serial data

void setup() {
  Serial.begin(9600); // opens
serial port, sets data rate to
9600 bps
}

void loop() {
  // send data only when you
receive data:
  if (Serial.available() > 0) {
    // read the incoming byte:
    incomingByte = Serial.read();

    // say what you got:
    Serial.print("I received: ");
    Serial.println(incomingByte,
DEC);
  }
}
```

1. Write structs for a Round Pizza and a Rectangle Pizza.
2. Ask the user if they want to input the data for a Round or Rectangle Pizza
3. Depending on the choice, let the user input the necessary information
4. Print the price per square cm

Serial.begin(9600); start serial communication with 9600 bauds per second Serial.available() tells you whether incoming data is available Serial.read() reads a single byte Serial.println(incomingByte, DEC); prints the incoming byte formatted as a decimal number

Have Fun!