

A night landscape with a starry sky, dark hills, and a red tent on a beach. The tent is illuminated from within, casting a warm glow. The background shows a dark sea or lake under a vast, star-filled sky. The overall mood is serene and quiet.

# Summer Camp

Embedded Programming Clinic

Summer 2003

Dr. Robbeloth

Assistant Professor of Computer Science @ MVNU

A close-up, artistic photograph of a green printed circuit board (PCB) with various electronic components and glowing light effects. The image is dominated by a vibrant green color palette, with bright, out-of-focus light spots scattered across the surface, creating a sense of depth and technological complexity. The text "Why do we need programming?" is overlaid in the center in a clean, white, sans-serif font.

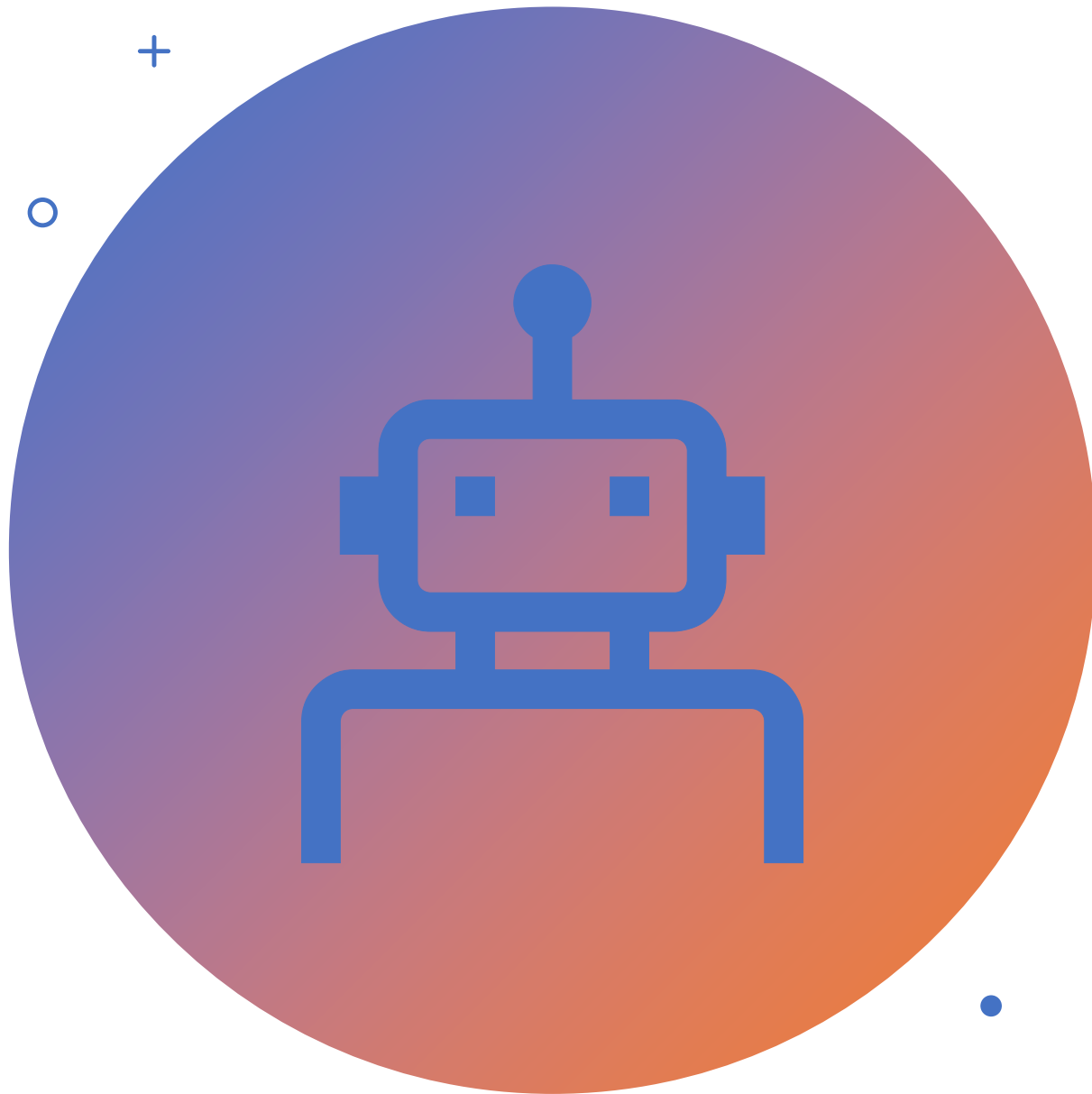
Why do we need programming?



# Software Development

---

- Apps to perform tasks efficiently

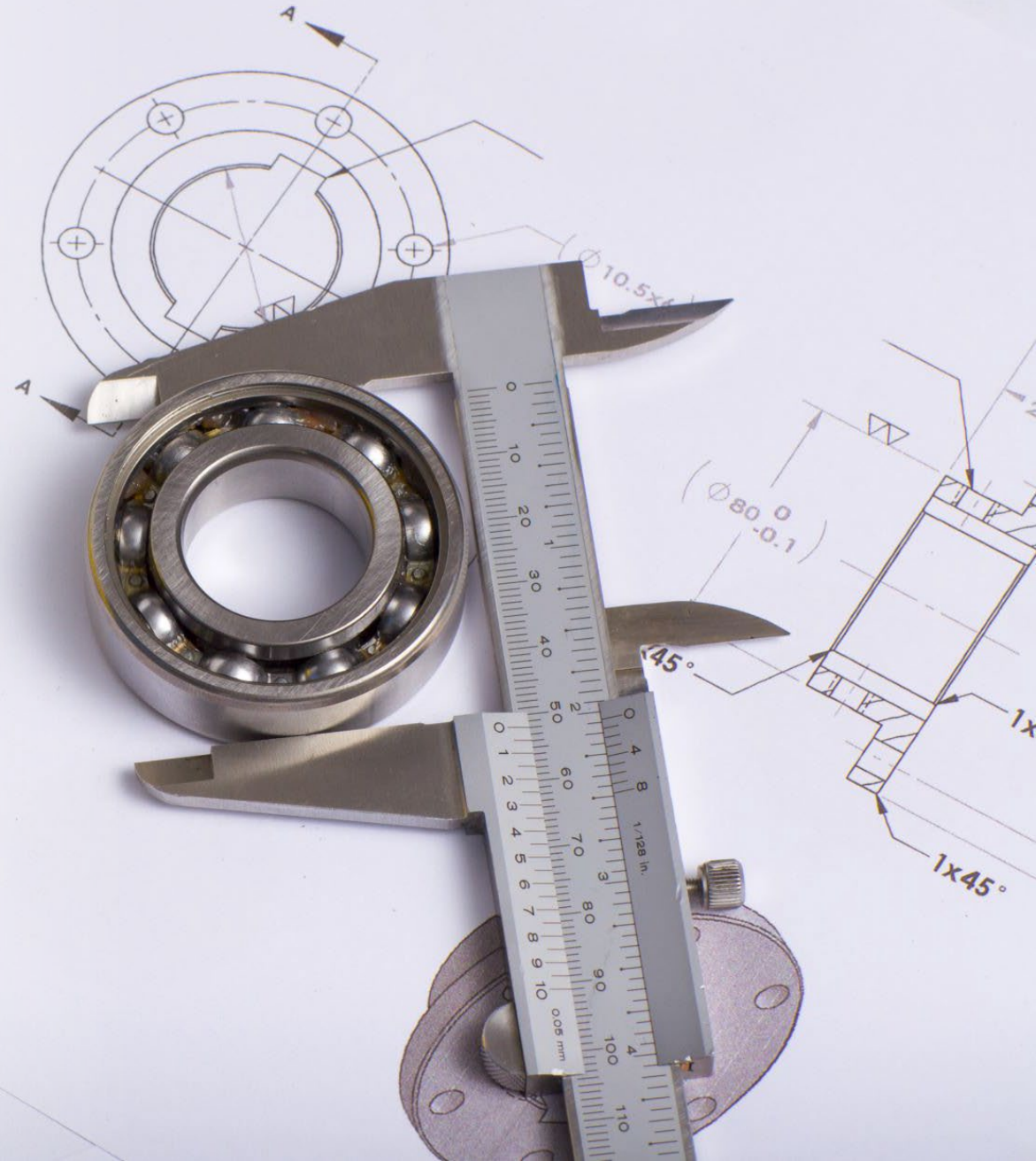


# Automation

- Automate repetitive tasks or processes
- Save time/effort
- Remove manual interventions

# Problem Solving

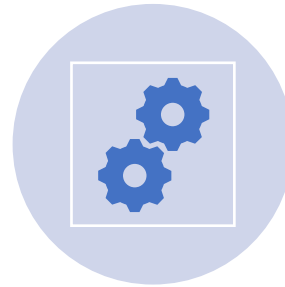
- Break down complex problems into small, manageable parts
- Develop algorithms (step-by-step instructions)



# Scientific and Mathematical Modeling



Programming needed in scientific research and mathematical modeling



Simulate complex phenomena



Analyze data



Make predictions



---

## Innovation and Creativity

- Bring new ideas to life
- Develop new tech, products, and services



# Make silicon smarter than a dumb rock

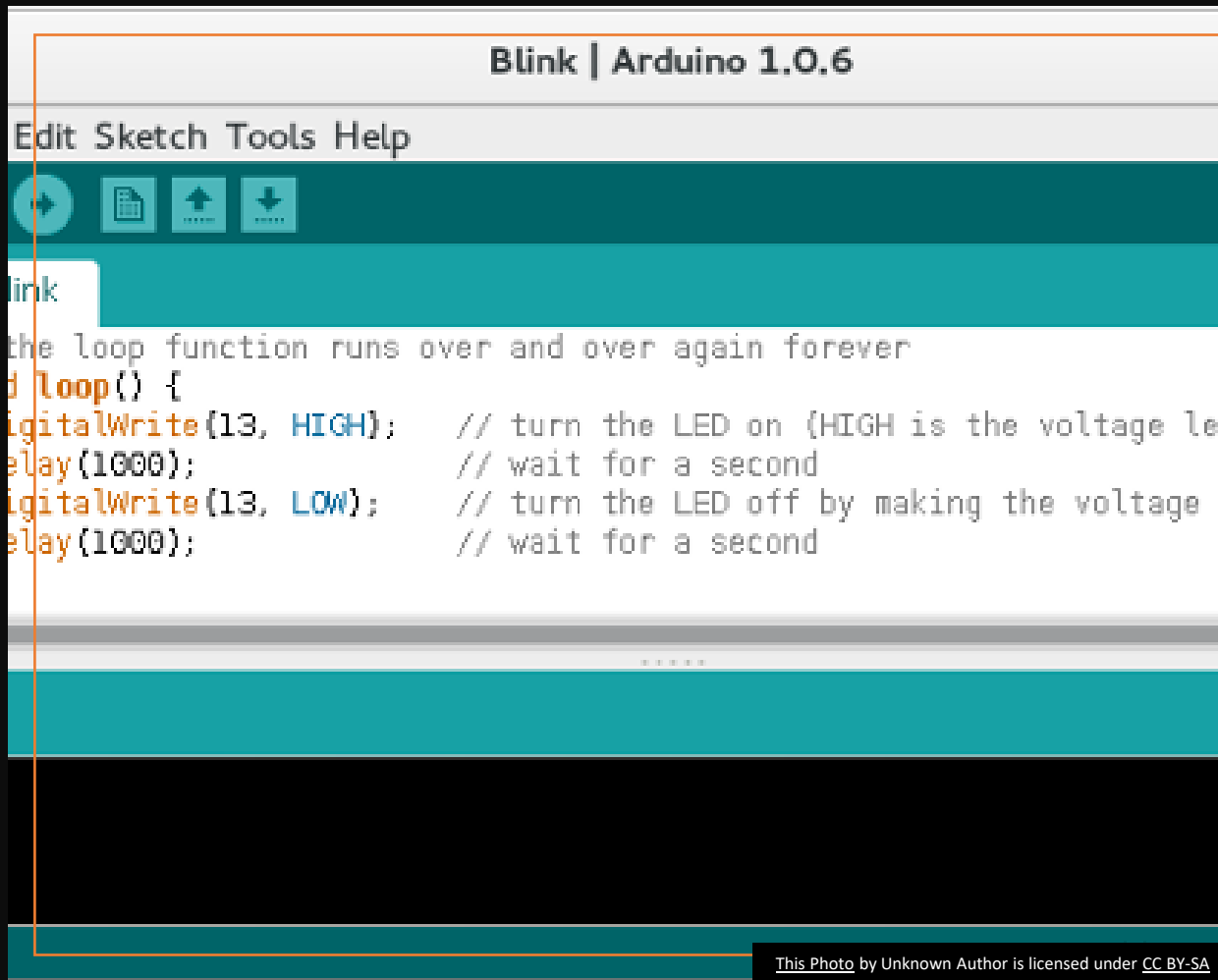
Tell devices capable of receiving instructions, how to work





# What do I need to start programming?

Notepad/Notebook

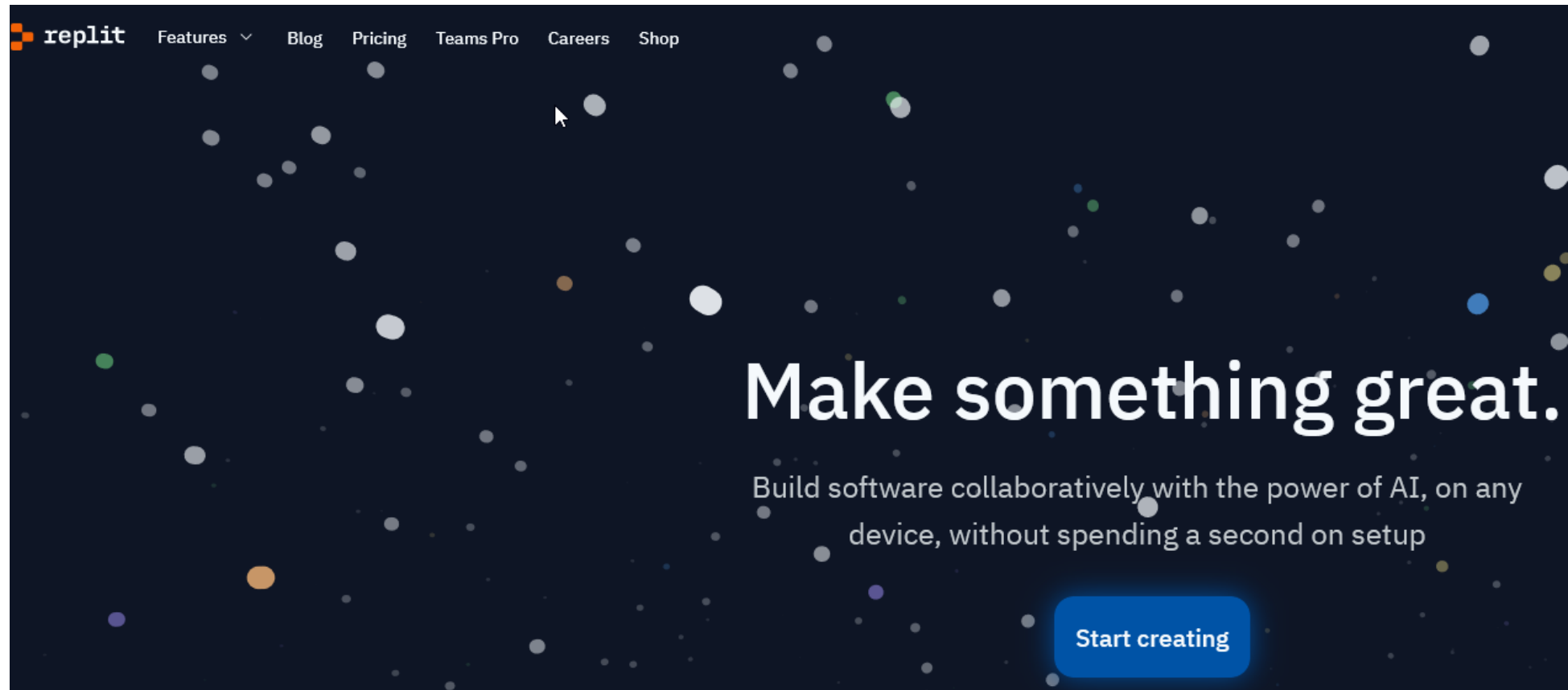


# What do I need to start programming?

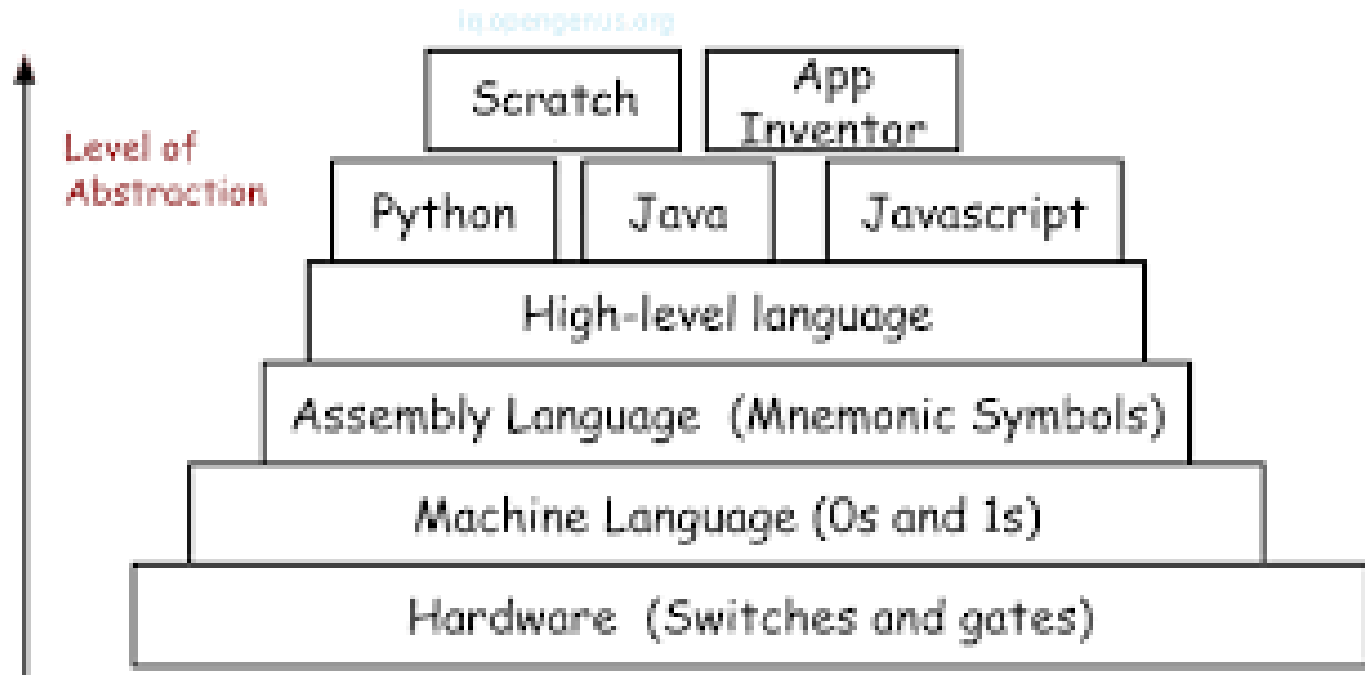
Development Environment

# What do I need to start programming?

- Could use a website to do your coding
- Replit.com



# Computer Abstractions



## Categories of Programming Languages

- Procedural
- C, Pascal, Ada, Fortran, Cobol, etc.
- Sequential, Conditional, and Iterative statements
- Code organized into procedures or functions and blocks

```
Free Pascal 3.2
File Edit Search Run Compile Debug Tools Options Window Help
[ ] SOL4.PAS 1-[↑]
var F,L:real;
    i,j,n:integer;
    x:array[1..10] of real;
    y:array[1..10] of real;

begin
  write('n=');readln(n);
  FOR i:=1 TO n DO
  begin
    write('x[' ,i,']=');readln(x[i]);
    write('y[' ,i,']=');readln(y[i]);
  end;
  begin
    write('x[' ,n+1,']=');readln(x[n+1]);
  end;
  y[n+1]:=0;
  F:=0;
  FOR j:=1 TO n DO begin
  L:=1;
  FOR i:=1 TO n DO
  begin
    IF i<>j THEN
    begin
      L:=L*(x[n+1]-x[i])/(x[j]-x[i]);
    end;
  end;
  y[n+1]:=y[n+1]+y[j]*L;end;
  writeln('y[' ,n+1,']=',y[n+1]:1:0);
  FOR i:=1 TO n DO
  begin
    writeln('x[' ,i,']=',x[i]:10:10, ' y[' ,i,']=',y[i]:10:10);
  end;
  begin
    writeln('x[' ,n+1,']=',x[n+1]:10:10, 'y[' ,n+1,']=',y[n+1]:10:10);
  end;
  readln;
end.
37:23
F1 Help F2 Save F3 Open Alt+F9 Compile F9 Make Alt+F10 Local menu
```

## Categories of Programming Languages

- Object-Oriented Purpose
- Java (w/ JIT), C++, Delphi, C#, etc.
- Source Code either compiled into machine code
- Machine Code runs on a particular computing device (x86-64 or ARM)

```
//returning the results in array
int[] res=new int[4];
res[0]=min;
res[1]=freq1;
res[2]=max;
res[3]=freq2;
//return the array
return(res);
}
//main function
public static void main(String[] args) {
    //scanner object to take input
    Scanner in=new Scanner(System.in);
    //test cases
    int n=in.nextInt();
    //to point the index of resultant array
    int x=0;
    //allocating memory for resultant array
    int size=(n*4)+1;
    int[] res=new int[size];

    for(int i=0;i<n;i++)
    {
        //size of the input array
```

```

import turtle
from turtle import TurtleGraphicsError

def programSetup():
    # Get a turtle object
    t = turtle.Turtle()
    daScreen = t.getscreen()

    # Ask for color of turtle using command line prompt
    try:
        prompt = "Turtle Color: "
        colorChoice = input(prompt)
        t.color(colorChoice)
    except TurtleGraphicsError as tge:
        print("Bad color choice, using purple ", tge)
        t.color("Purple")

    # set the background the turtle will draw on, use the dialog prompt
    try:
        backgroundColor = daScreen.textinput("Background Color", "Color:")
        if (backgroundColor == ""):
            daScreen.bgcolor("black")
        else:
            daScreen.bgcolor(backgroundColor)
    except ValueError as ve:
        print("Value ", ve)

    # what shape will do the drawing, a turtle or something else
    try:
        prompt = "Turtle Shape (" + str.join(', ', daScreen.getshapes()) + "): "
        shape = input(prompt)
        t.shape(shape)
    except TurtleGraphicsError as tge:
        print("Bad color choice, using purple ", tge)
        t.shape("arrow")

    # Return the turtle
    return t, daScreen

```

# Categories of Programming Languages

---

- Scripting Languages
  - PHP, Python, Node.js (JavaScript), Bash, Perl, etc.
  - Interpreted line at a time by interpreter (slower)
  - Tools to convert to machine code blob in some cases
  - Programming Languages often fall into multiple categories
-

# Categories of Programming Languages

- Functional
- Mathematics functions and evaluations, lots of reuse
- Scala, Erlang, Haskell, Elixir, F#, etc.

```
% This is file 'listsort.erl' (the compiler is made this way)
-module(listsort).
% Export 'by_length' with 1 parameter (don't care about the type and name)
-export([by_length/1]).

by_length(Lists) -> % Use 'qsort/2' and provides an anonymous function as a parameter
    qsort(Lists, fun(A,B) -> length(A) < length(B) end).

qsort([], _) -> []; % If list is empty, return an empty list (ignore the second parameter)
qsort([Pivot|Rest], Smaller) ->
    % Partition list with 'Smaller' elements in front of 'Pivot' and not-'Smaller' elements
    % after 'Pivot' and sort the sublists.
    qsort([X || X <- Rest, Smaller(X,Pivot)], Smaller)
    ++ [Pivot] ++
    qsort([Y || Y <- Rest, not(Smaller(Y, Pivot))], Smaller).
```



# Categories of Programming Languages

---

```
mother_child(trude, sally).  
  
father_child(tom, sally).  
father_child(tom, erica).  
father_child(mike, tom).  
  
sibling(X, Y)      :- parent_child(Z, X), parent_child(Z, Y).  
  
parent_child(X, Y) :- father_child(X, Y).  
parent_child(X, Y) :- mother_child(X, Y).
```

This results in the following query being evaluated as true:

```
?- sibling(sally, erica).  
Yes
```

- Logic
  - Expresses series of facts and rules to tell computer how to make decisions
  - Prolog, Absys, Datalog, Alma-0 (all very esoteric languages)
-

# Categories of Programming Languages

---

- Web
- Front-end vs. Backend
- Front-end: HTML/CSS/JavaScript, React
- Back-end: Node.js, PHP, JavaScript, TypeScript, etc.

```
ws.on("message", m => {  
  let a = m.split(" ")  
  switch(a[0]){  
    case "connect":  
      if(a[1]){  
        if(clients.has(a[1])){  
          ws.send("connected");  
          ws.id = a[1];  
        }else{  
          ws.id = a[1]  
          clients.set(a[1], {client: {position: {x: 0, y: 0}, id: (1)}, conn})  
          ws.send("connected")  
        }  
      }else{  
        let id = Math.random().toString().slice(2, 8)  
        ws.id = id;  
        clients.set(id, {client: {position: {x: 0, y: 0}, id: (1)}, conn})  
      }  
    }  
  }  
})
```

# Categories of Programming Languages

---

```
fn main() {  
    // Defining a mutable variable with 'let mut'  
    // Using the macro vec! to create a vector  
    let mut values = vec![1, 2, 3, 4];  
  
    for value in &values {  
        println!("value = {}", value);  
    }  
  
    if values.len() > 5 {  
        println!("List is longer than five items");  
    }  
  
    // Pattern matching  
    match values.len() {  
        0 => println!("Empty"),  
        1 => println!("One value"),  
        2..=10 => println!("Between two and ten values"),  
        11 => println!("Eleven values"),  
        _ => println!("Many values"),  
    };  
  
    // while loop with predicate and pattern matching using let  
    while let Some(value) = values.pop() {  
        println!("value = {value}"); // using curly braces to format a local variable  
    }  
}
```

- System Programming Languages
  - C, C++, Go, Rust, etc.
  - Tradeoff: compatibility vs speed and ease of hardware access
-

# Categories of Programming Languages

---

- Assembly
  - Low-level, usually direct one-to-one correspondence with machine instructions
  - There are pseudoinstructions/macros
  - Code is assembled into machine code
- 

```
1          .data
2 A:        .word 10      # change value to desired number for A, which is a0
3 B:        .word 4       # change value to desired number for B, which is a1
4 array:    .word 0:50    # size must be changed to accommodate A and B
5          .text
6
7 main:
8     lw $s0, A           # $s0 = A
9     addi $s0, $s0, -1   # to accommodate for loop condition
10    lw $s1, B           # $s1 = B lowercase b turns blue... WHY
11    addi $s1, $s1, -1   # see line 9
12    la $s2, array       # "look at" address of array
13    li $s3, 0           # set i = 0
14    li $s4, 0           # set j = 0
15
16 For1:
17    blt $s0, $s3, Exit  # for(i = 0; i < A; i++)
18    addi $s3, $s3, 1    # i++
19    li $s4, 0           # resets j to 0 after each iteration of the for loop
20    j For2              # executes the nested for loop
21
22 For2:
23    blt $s1, $s4, For1  # for(j = 0; j < B; j++)
24    addi $t2, $s3, -1   # cancel line 18 from interfering with calculations
25    sub $t1, $t2, $s4   # $t1 = i - j
26    sll $t3, $t2, 3     # $t3 = i * 8 (provides offset)
27    add $t3, $t3, $s2   # $t3 = address of save + offset
28    sw $t1, 0($t3)     # store result of i - j in array
29    addi $s4, $s4, 1    # j++
30    j For2              # loop
31
32 Exit:
33    li $v0, 10          # load exit opcode
34    syscall             # execute exit
35
```

# Categories of programming languages

---

- Hardware Description Languages
- Verilog and VHDL (on right)
- Used with Field Programming Gate Arrays (FPGAs)
- Devices that you can write program code for to transform the silicon into different computing devices
- Code is synthesized into electronic design resulting in logic gates being wired up on the board

```
use ieee;
use std_logic_1164.all;
use numeric_std.all;

entity signed_adder is
port
(
    aclr : in    std_logic;
    clk  : in    std_logic;
    a    : in    std_logic_vector;
    b    : in    std_logic_vector;
    q    : out   std_logic_vector
);
end signed_adder;

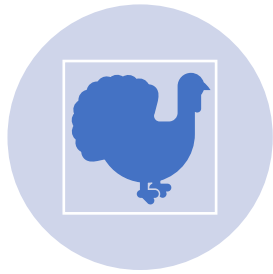
architecture signed_adder_arch of signed_adder is
    signal q_s : signed(a'high+1 downto 0); -- extra bit wide
begin -- architecture
    assert(a'length >= b'length)
        report "Port A must be the longer vector if different sizes!"
        severity FAILURE;
    q <= std_logic_vector(q_s);
    adding_proc:
        process (aclr, clk)
        begin
            if (aclr = '1') then
                q_s <= (others => '0');
            elsif rising_edge(clk) then
                q_s <= ('0'&signed(a)) + ('0'&signed(b));
            end if; -- clk'd
        end process;
end architecture;
```



# FPGA

```
library IEEE;
use IEEE.std_logic_1164.all;
entity mux4 is
    port(
        a1      : in  std_logic_vector(2 downto 0);
        a2      : in  std_logic_vector(2 downto 0);
        a3      : in  std_logic_vector(2 downto 0);
        a4      : in  std_logic_vector(2 downto 0);
        sel     : in  std_logic_vector(1 downto 0);
        b       : out std_logic_vector(2 downto 0)
    );
end mux4;
architecture rtl of mux4 is
    -- declarative part: empty
begin
    p_mux : process(a1,a2,a3,a4,sel)
    begin
        case sel is
            when "00" => b <= a1 ;
            when "01" => b <= a2 ;
            when "10" => b <= a3 ;
            when others => b <= a4 ;
        end case;
    end process p_mux;
end rtl;
```

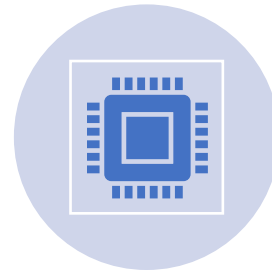
# Basics:



USING THE I.P.O.  
PATTERN



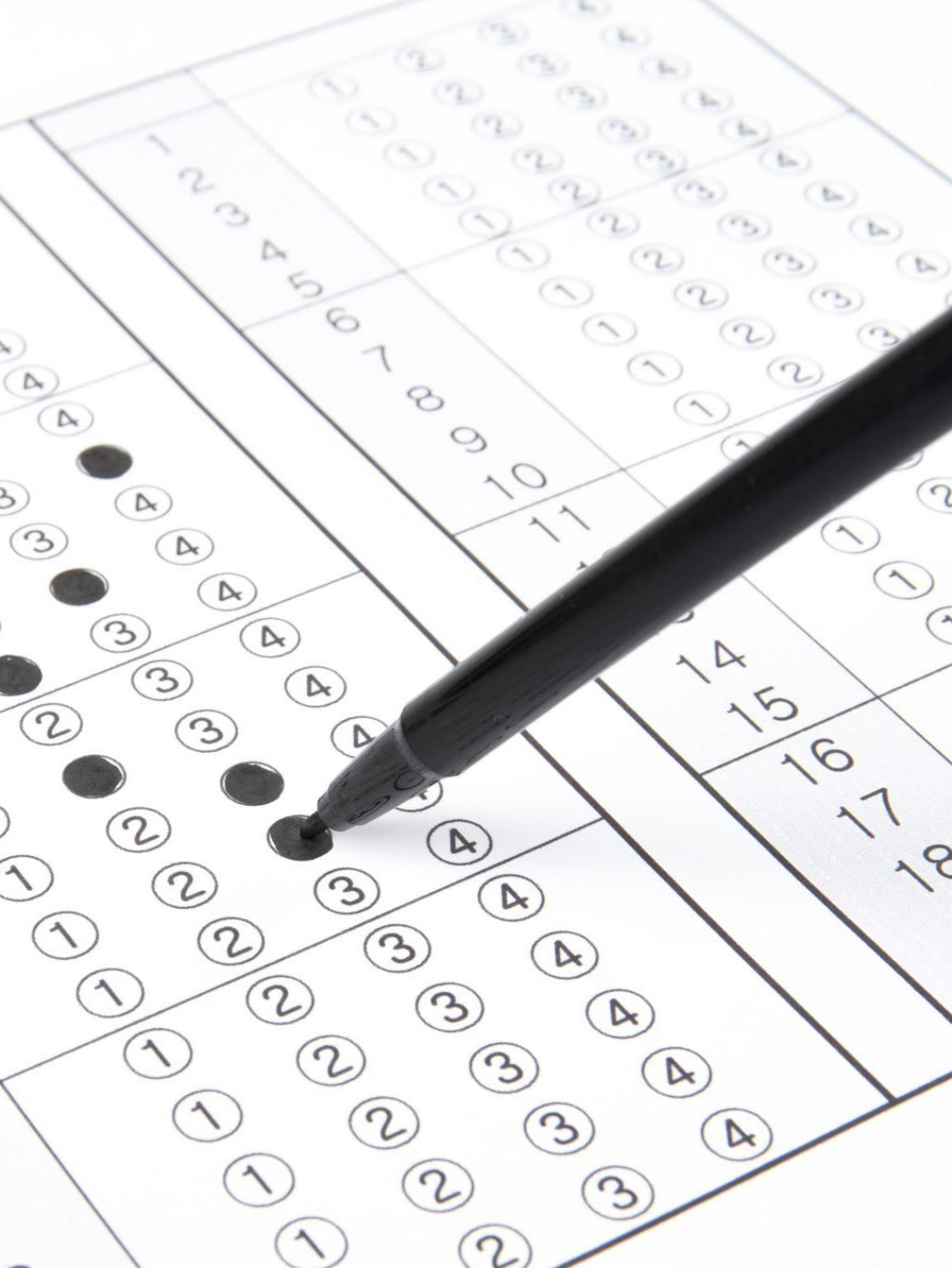
INPUTS



PROCESSING



OUTPUTS



---

## Types of Statements

### Sequential

- Ask for input from the console (***cin >> variable***)
- Perform a mathematical calculation (***int a = b + c***)
- Call a procedure/function/routine (***sqrt(16)***)
- Display a result (e.g., output) on the console (***cout << "I like Peppa Pig " << endl;***)



# Types of Statements

- Conditionals
- Evaluate an expression and choose a code path base on its results

**if (this is true)**

**take the first course of action**

**else if (something else is true)**

**take that second course of action**

**else**

**default to that final course of action**



# Types of statements

- Iterative statements
- Perform a sequence of actions repeatedly
- Can repeat based on something becoming true (zero, one, multiple)

***while (something is true)***

***statement 1***

***statement 2***

...

***statement n***

- Can repeat until something becomes false

***do***

***statement 1***

***statement 2***

...

***statement n***

***while (something is true) // do this at least one time***



# Variables

- Need to hold our inputs and intermediate values somewhere
- Use variables
- Not like math variables representing an unknown
- Take up memory, computers don't have unlimited resources

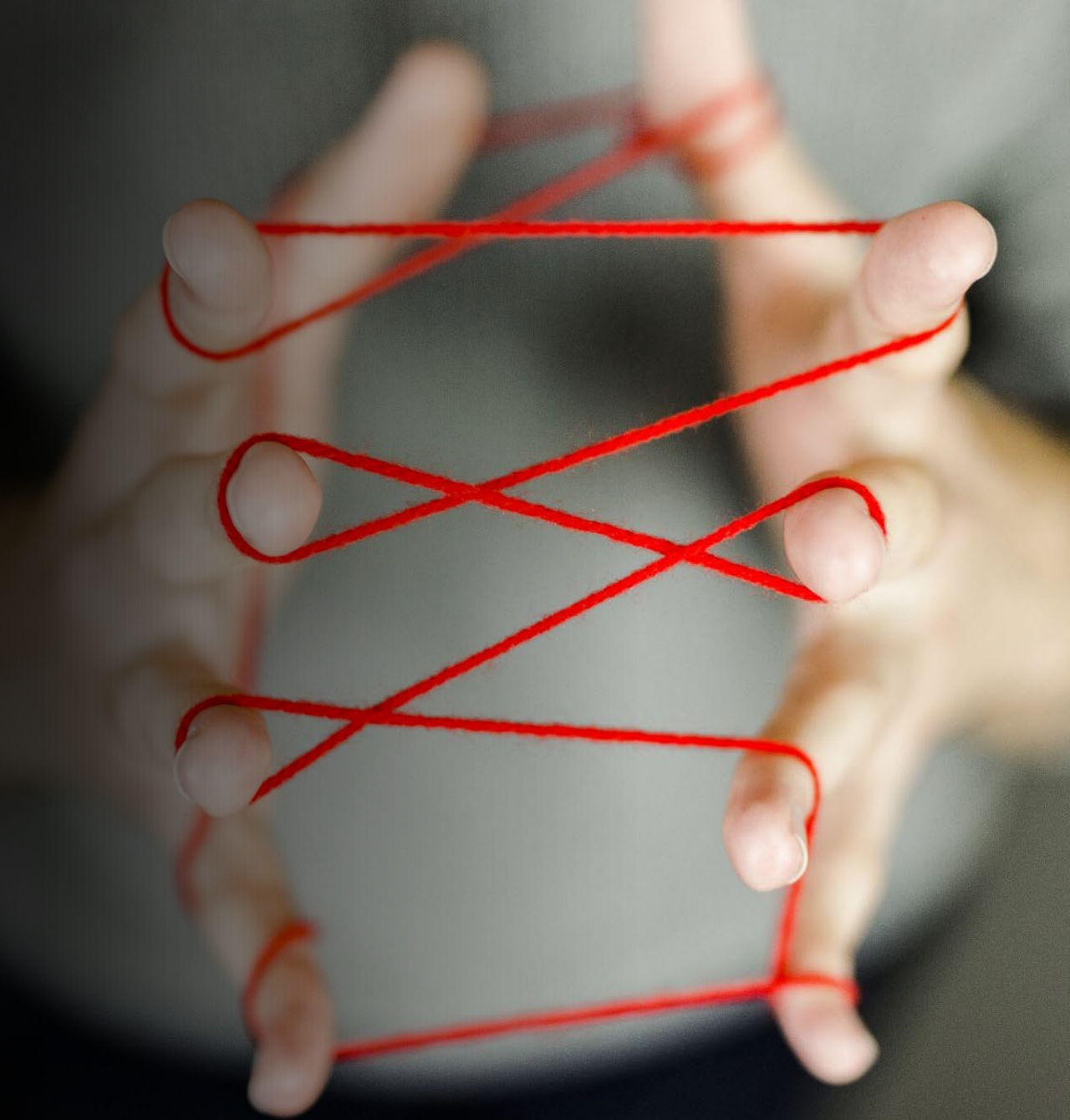
# Datatypes

---

Each variable has to be represented by a datatype

- Integers → Use byte/int/long
- Decimals → Use float/double
- Characters → Use chars
- Strings → Use string

Tradeoff: range vs memory



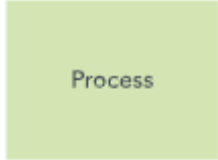
# Flow Chart Model

- Flow charts let's us visually map out each step, decision, and repeated sequence of instructions

Terminal/Terminator



Process



Decision



Document



Data, or Input/Output



# Flow Chart Model

- Flow charts let's us visually map out each step, decision, and repeated sequence of instructions



Stored Data



Flow Arrow



Comment or Annotation



Predefined process



On-page connector/reference

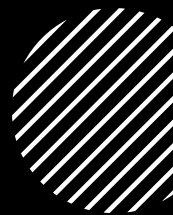


Off-page connector/reference





# Let's try a quick exercise



Imagine we go to the local carnival.



One ride costs one ticket



Do we have money?



If so, buy some ticket, otherwise go home



Wait in line, get on ride. Wonder is it over yet.




Ride it again if we still want to AND have at least one ticket.



Go to **draw.io**



Select Local Device, New Diagram, Basic Flowchart

A hand with red nail polish holds a lit sparkler, which is emitting bright white sparks. The background is dark with numerous out-of-focus, colorful bokeh lights in shades of yellow, orange, blue, and green. The overall scene suggests a festive or celebratory atmosphere.

How would we welcome a new person to camp and display their approximate age?



# First Example Solution

```
/* To do any real work, you need to use programmer libraries like this one
   called input-output stream

   An stream is just the flow of data into your program from some device
   say a keyboard, through the code of your program and out some other device
   say a monitor/display
*/
#include <iostream>

// make code more modular, avoid name collisions in different libraries
using namespace std;

void helloAgeExample() {
    // Need to hold data in variables, which are stored in memory
    // C and C++ are strongly typed languages, each variable indicates data type
    string name; // store user's name in a string variable
    int yob;     // store user's yob in integer

    // Use the input, processing, and output pattern, start with a prompt
    cout << "Name: ";

    // gather the name
    cin >> name;

    // welcome this user and ask for yob
    cout << "Hello " << name << " What year were you born?: " << endl;


    // get yob of birth and gustomer age with two examples
    cin >> yob;
    cout << "Oh, so you must be " << 2023 - yob << " years old or " << 2022 - yob
        << " years old. " << endl;
}
```



# Example Running on Replit.com

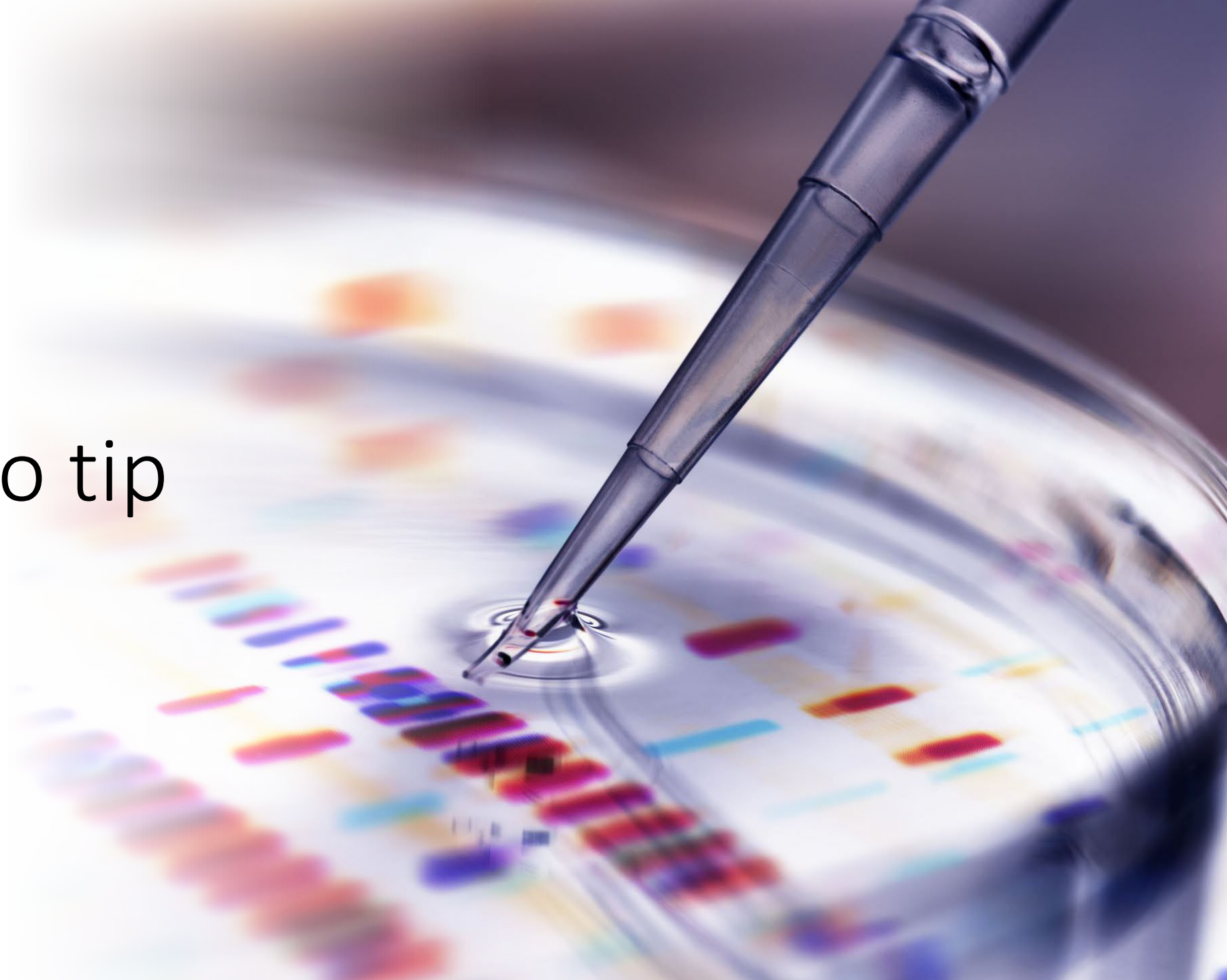
## How would you calculate the tip?

- Use draw.io to map out steps
- Let's write the pseudocode
- Ask below ('B') expectations (10%), meet ('M') expectations (18%), or exceeded ('E') expectations (20%)
- Keep using the IPO pattern
- Next, translate each pseudocode instruction into a line of C++ code
- How do you differentiate the service you got?



Tip: use `cout << setprecision(2);` and `cout << fixed;` to round amounts to two places.  
What datatypes do we need to use?

Solution to tip  
program



# Let's work with loops

---

- What would it take to print from one number to another, say 1 to 10, and then give a sum of those numbers?
- IPO
- Inputs:
- Processing:
- Outputs:
  
- Later, we'll try to print the odds or evens from some start value to some end value based on user inputs.

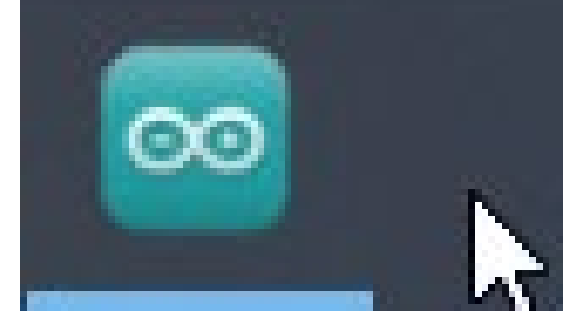


# Infinity Mirror

---

- 2+ mirrors/mirror-like surfaces in parallel or angled arrangement
- LED strip (60 lights total) will line the perimeter
- Create series of ever smaller reflections toward infinity
- Recursive due to Droste effect
- Front-mirror must be one-way
- Software code and electronics creating art

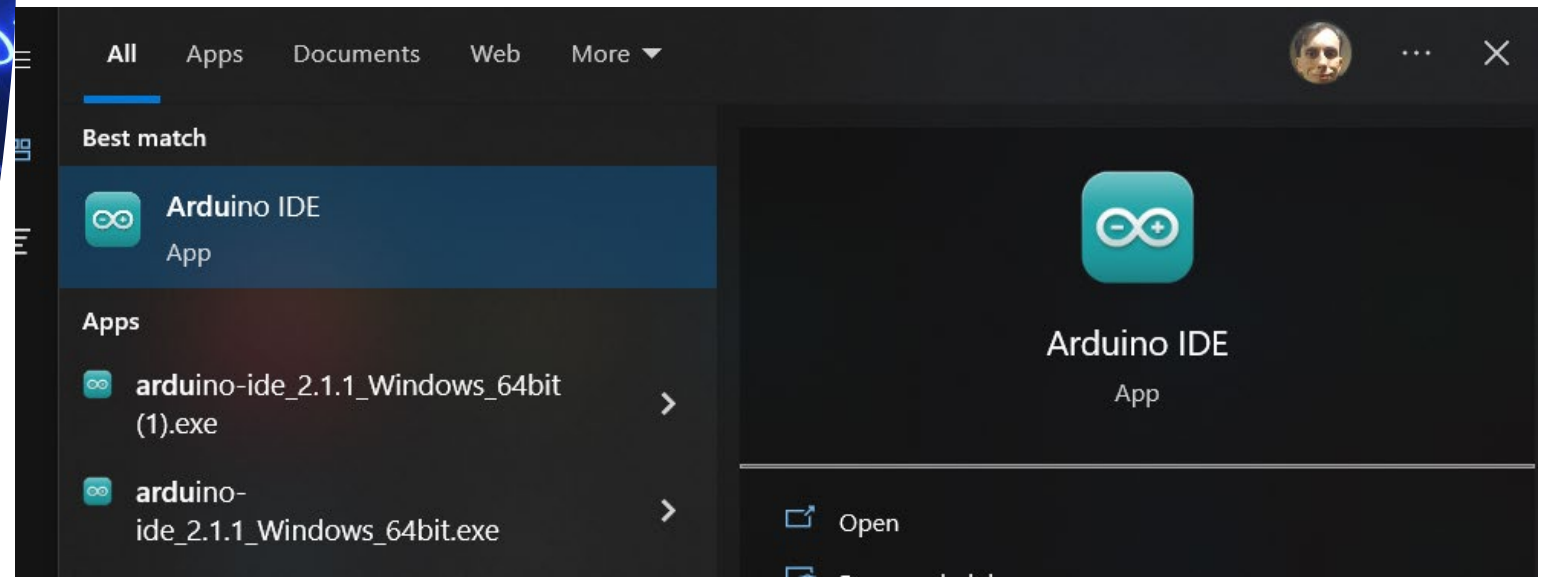




# Arduino Overview

---

- Click Start button and type Arduino and select Arduino result or look for the icon



sketch\_jul13a | Arduino IDE 2.1.1

File Edit Sketch Tools Help



Select Board



sketch\_jul13a.ino

```
1 void setup() {  
2   // put your setup code here, to run once:  
3  
4 }  
5  
6 void loop() {  
7   // put your main code here, to run repeatedly:  
8  
9 }  
10
```



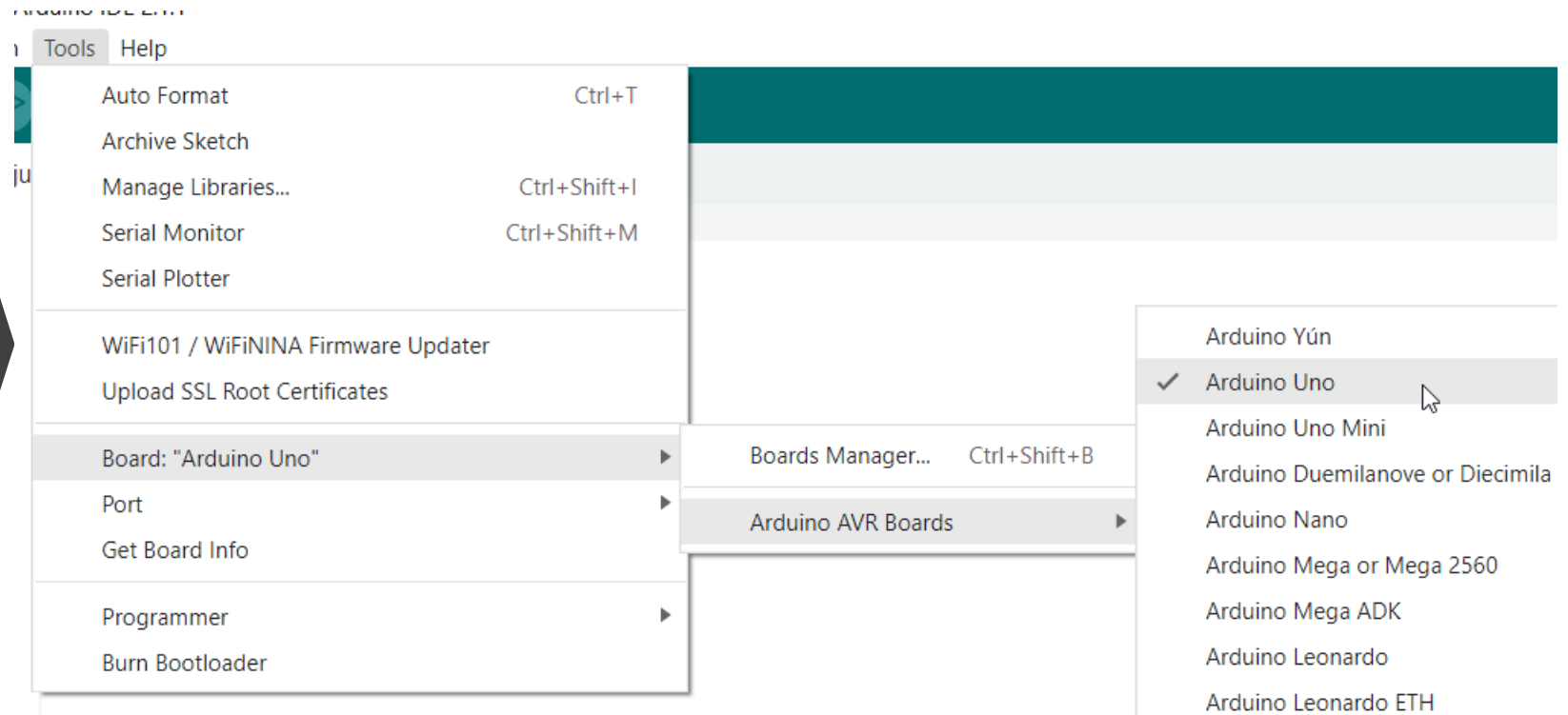
Arduino  
Environment

Classic  
version

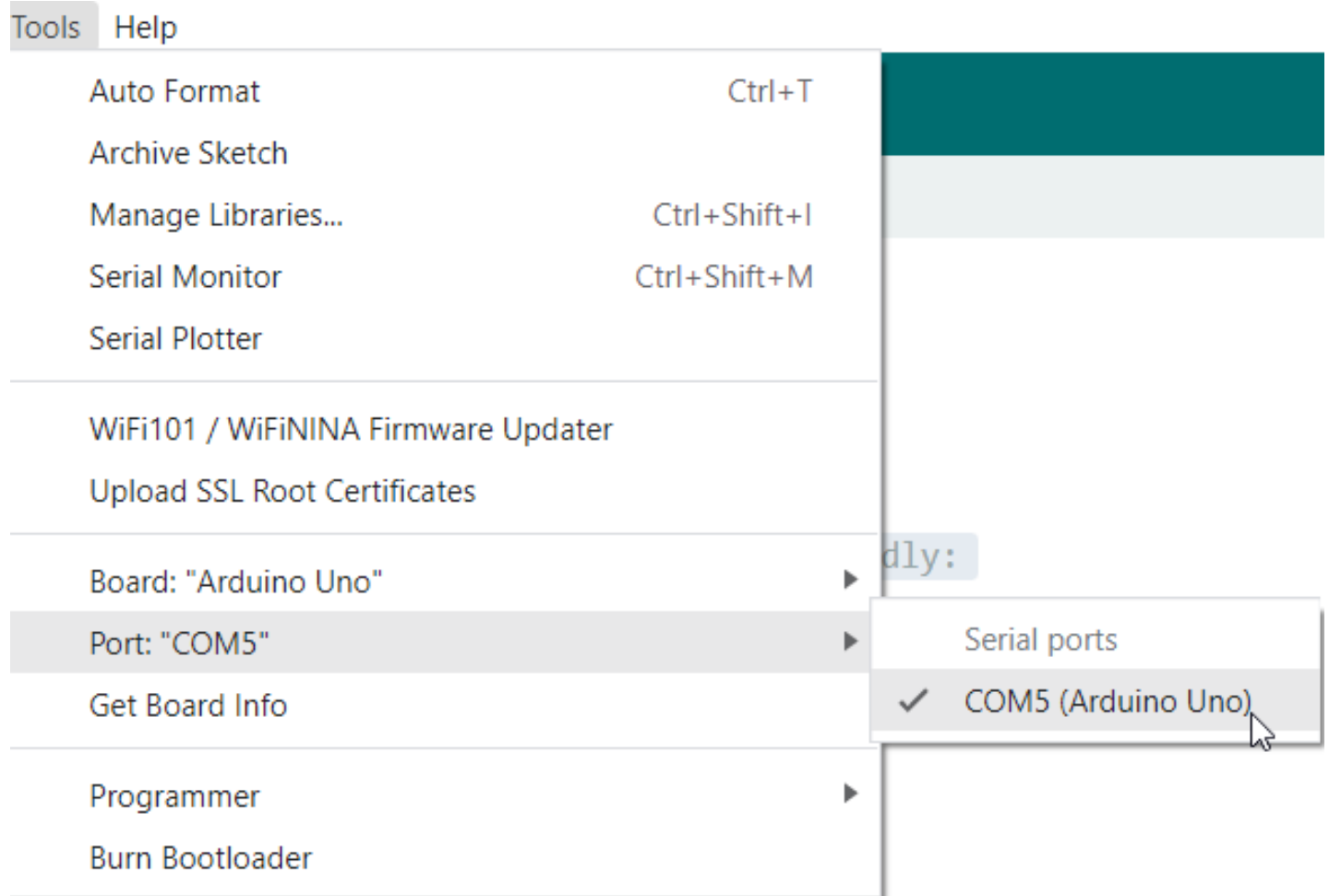
Syntax is C++  
compatible



Make sure  
this board is  
selected



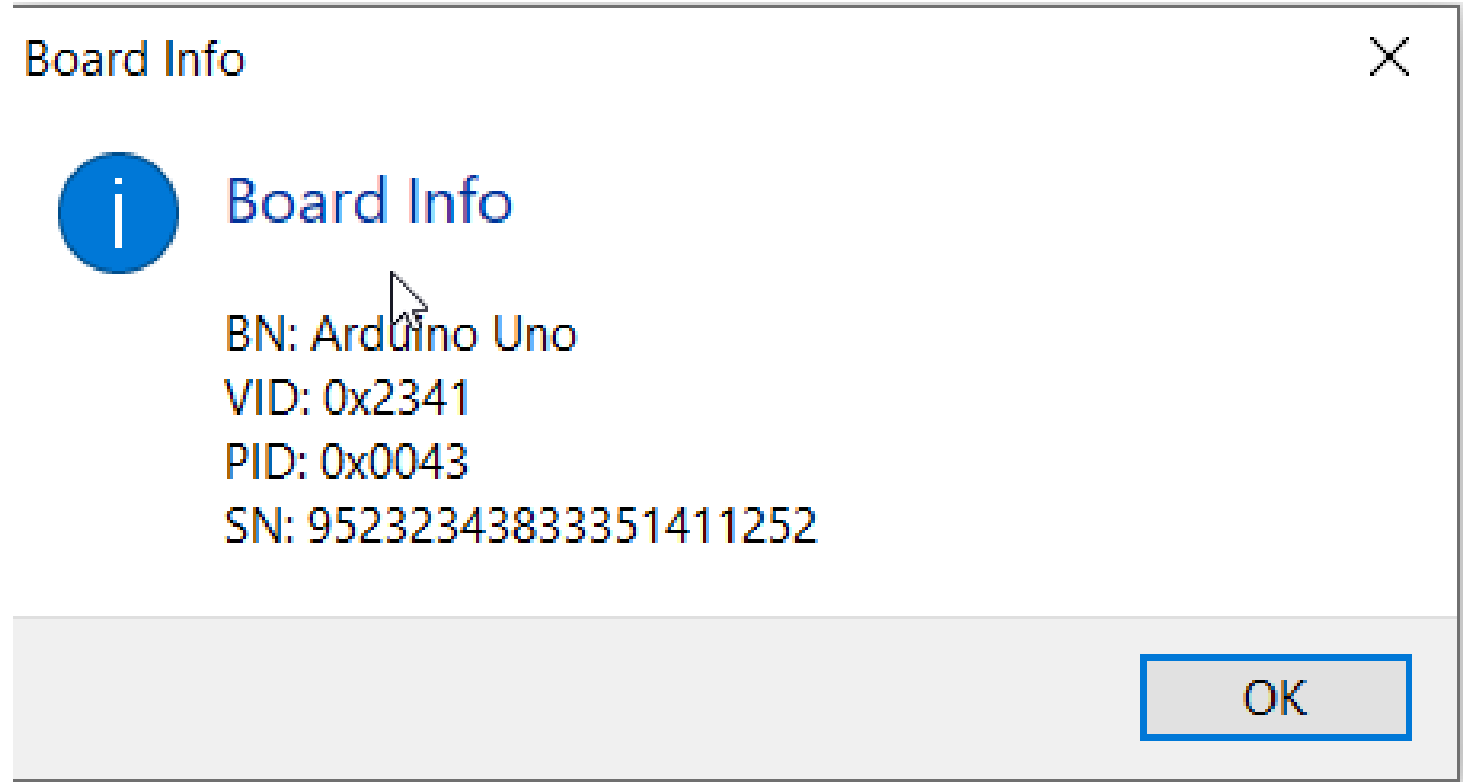
Make sure  
the correct  
serial port is  
selected



If the correct board and port is selected

---

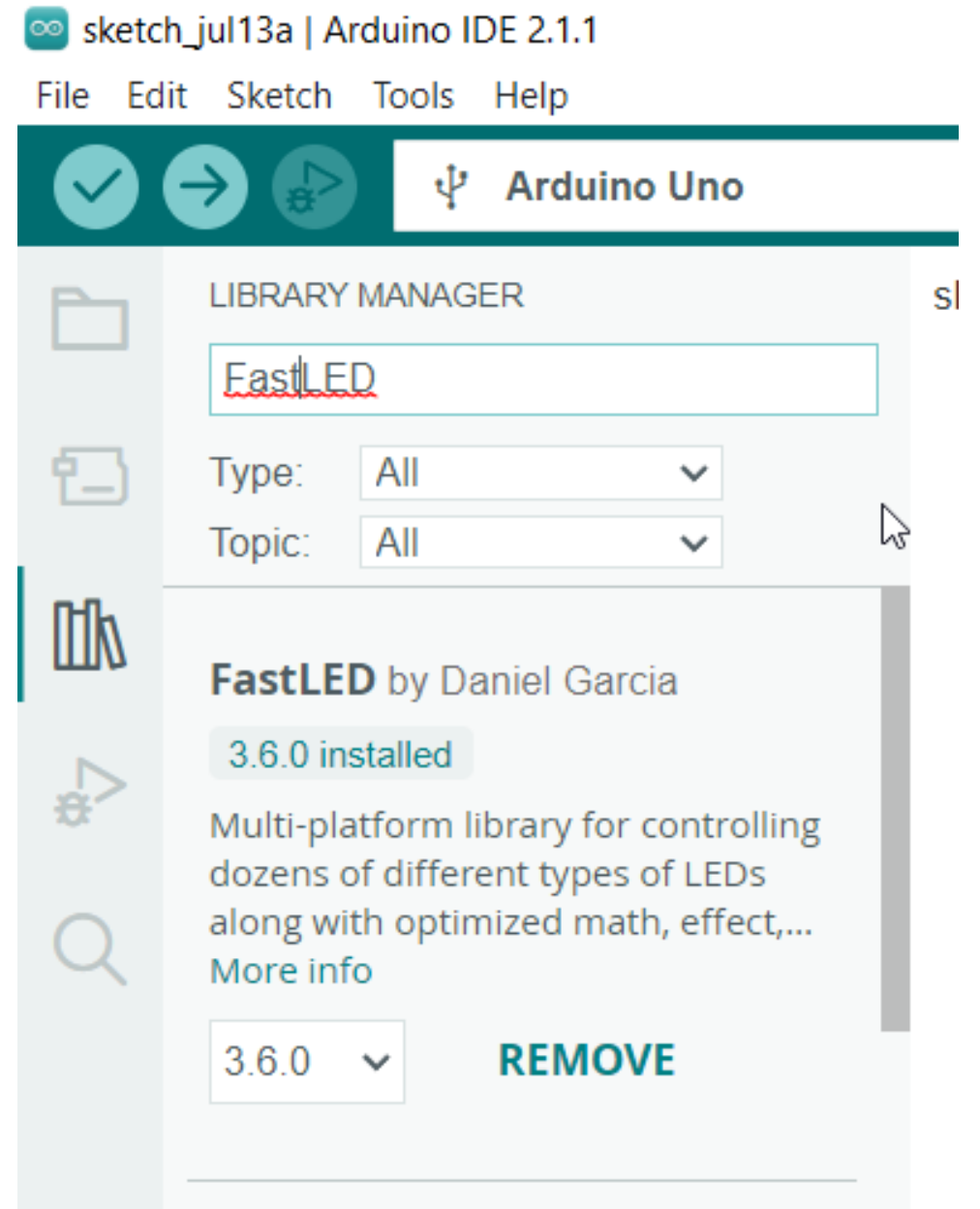
- **Tools**→**Get Board Info**
- BN and maybe PID should be the same, but other values may vary. It's okay



# Library for LED Strips

- Select 3.6 or later version
- **Tools** → **Manage Libraries**
- Enter **FastLED** into search box
- Pick the one by Daniel Garcia.
- Click Install
- <https://github.com/FastLED/FastLED/wiki/Basic-usage>
- In code, select new sketch and add as 1<sup>st</sup> line before **setup()** block:

```
#include <FastLED.h>
```



sketch\_jul13a | Arduino IDE 2.1.1

File Edit Sketch Tools Help

Arduino Uno

LIBRARY MANAGER

FastLED

Type: All

Topic: All

**FastLED** by Daniel Garcia

3.6.0 installed

Multi-platform library for controlling dozens of different types of LEDs along with optimized math, effect,...

More info

3.6.0 REMOVE

# Next steps

Better to change code in one place than many places  
Constants help us to do this

In C/C++, we can use `#define`

```
#define NUMBER_LEDS 60  
#define DATA_PIN 6
```

Let's do this after `#include <FastLED.h>` on line 1

Wherever `NUMBER_LEDS` appears in our code, the compiler will replace it with 60.



# Now, we work on setup(), We need to add:

---

Need to declare *globally* an array of LED objects using the variable declaration:

```
CRGB objects leds[NUMBER_LEDS];
```

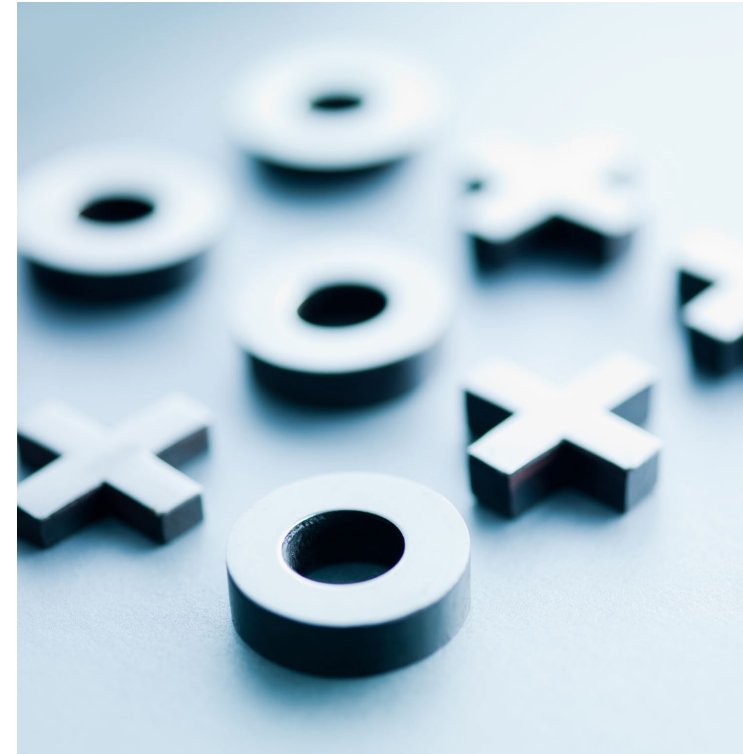
// Next add the following line in setup:

```
FastLED.addLeds<NEOPIXEL, DATA_PIN>(leds, NUMBER_LEDS);
```

FastLED is an CFastLED class object, addLeds is a method (specifically a factory method).

Yes, the period, angle brackets, parenthesis, and the semicolon are all important – part of what we call syntax

It says we have 60 LEDs we would like to use that are NeoPixel compatible (WS2812s) and are connected to pin 6.





# Finally, let's work on loop()

---

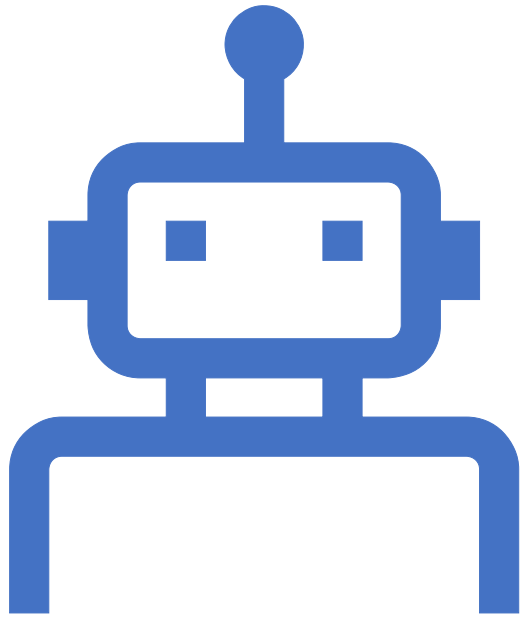
We need to use a construct that repeats a certain number of times

We call this a loop

In this case, we'll use a special type of loop called the **for** loop

```
for(myVariable = start_value; myVariable < something_else;  
start_value++) {  
    // do something over and over again until myVariable equals something  
    else  
}
```

**=, ==, <, ++, {, and }** all mean something different, do you know what it is?



To get an LED to actually do something

- We need to call a method, **FastLED.show()**
- We may want to add **delay(some milliseconds)** to get interesting effects.



# Verify the code

- Click the checkmark icon
- **Sketch**→**Verify** or CTRL+R
- If you don't see any red error text, that's good
- You can now upload the compiled sketch to the board

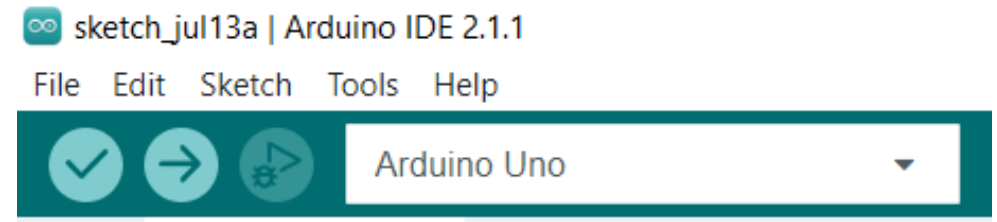


Output

```
Sketch uses 444 bytes (1%) of program storage space. Maximum is 32256 bytes.  
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables. Maximum is 2048 bytes.
```

# Upload the compiled sketch

- Click the upload (*play*) arrow icon
- Board should be selected
- **Sketch**→**Upload** or CTRL+U
- If you don't see any red error text, that's good
- You can now upload the compiled sketch to the board



# Q&A Break

- Next two exercises, time permitting
- Make  $\frac{1}{2}$  of the lights blue and  $\frac{1}{2}$  red
- Finally, let's animate all the lights

If there is extra time beyond that, let's try something more challenging like different colors for odd/even lights or generate different random colors.