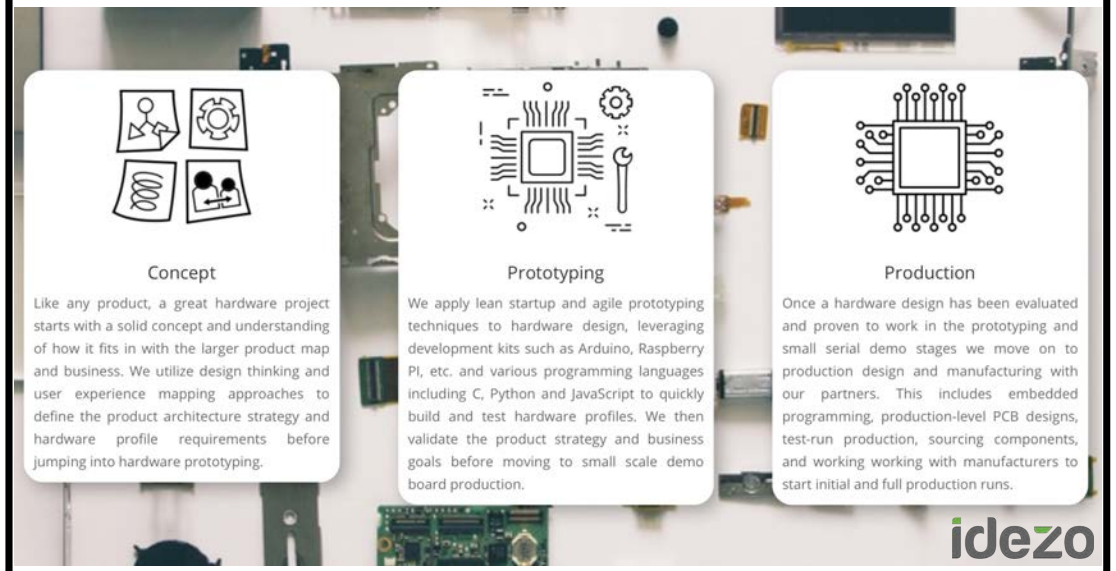
HANDS ON APPROACH TO

# Data Science

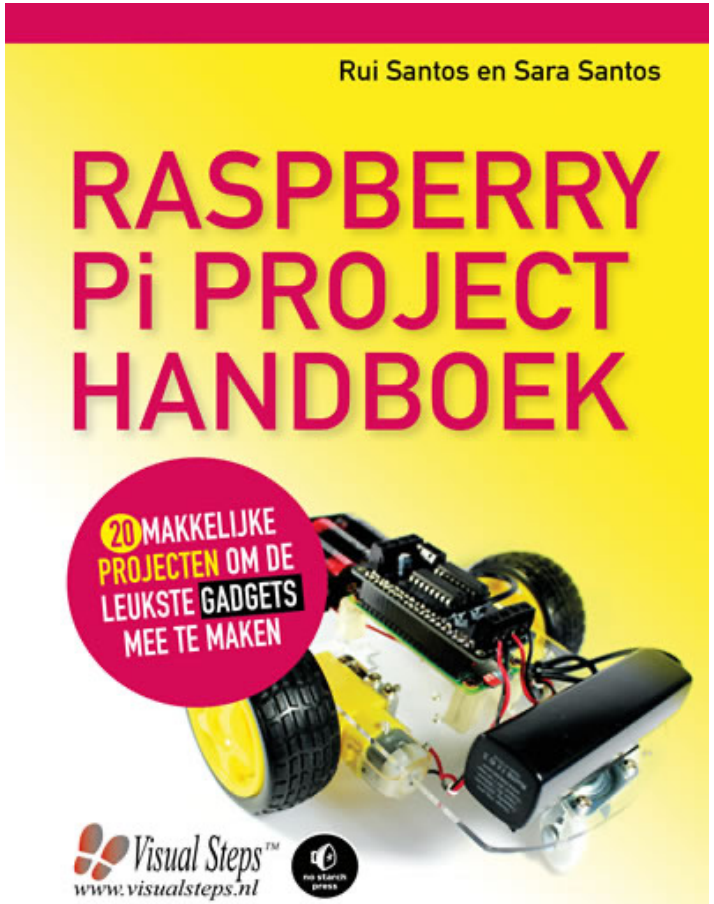
## for (the)

# IoT




**Rob van der Willigen**

https://github.com/robvdw/CMIDAT01K-DATA-SCIENCE-for-IOT

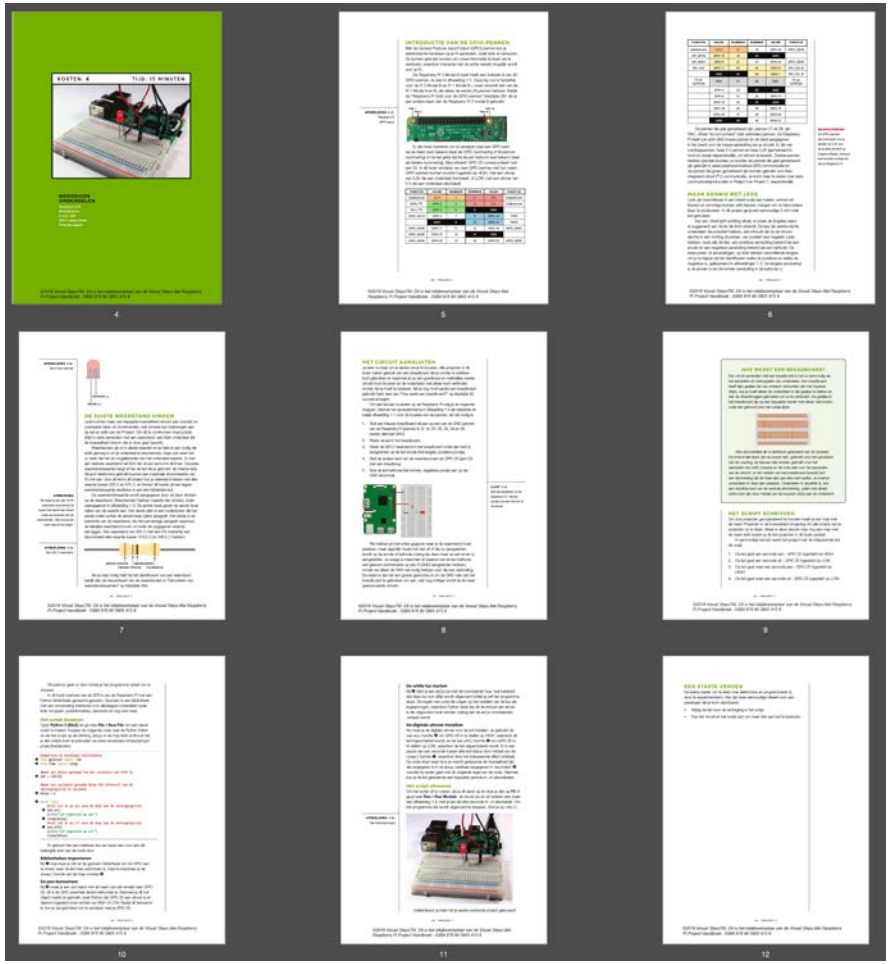



https://www.visualsteps.nl/raspberrypi/
https://www.visualsteps.nl/inkijkexemplaar/957.pdf

# 1 EEN LED LATEN KNIPPEREN

IN DIT EERSTE PROJECT GA JE EEN LED AANSLUITEN OP JE PI EN DEZE LATEN KNIPPEREN MET EEN PYTHON SCRIPT. HET LEREN HOE JE EEN LED LAAT KNIPPEREN MET DE GPIO-PENNEN IS EEN BELANGRIJKE STAP IN HET LEREN OVER JE PI. ALS JE EENMAAL WEET HOE JE EEN LED KUNT BESTUREN, KUN JE BIJNA ELKE UITVOER BESTUREN, OF HET NOU EEN MOTOR, EEN LAMP OF ZELFS EEN BROODROOSTER IS.






Learning O'Reilly is een Engelstalig online leerplatform gericht op ICT vaardigheden. Het biedt tienduizenden e-books en vele video's, case studies en "learning paths", waarbij je zelf je voortgang kunt monitoren.

## Toegang

### Via de website

De Learning O'Reilly website is gekoppeld aan de Hogeschool Rotterdam login. Kom je toch een O'Reilly inlogscherm tegen, vul dan alleen je @hr.nl e-mailadres in en klik op **Sign In with Single Sign On**. Krijg je een blanco pagina te zien? Schakel eventuele adblockers uit!

### Via de O'Reilly app

Leer wat je wil, waar je wil. Met de O'Reilly app download je e-books en ga je verder met een playlist waar je op een ander apparaat gebleven was.

1 Google play Download on the App Store

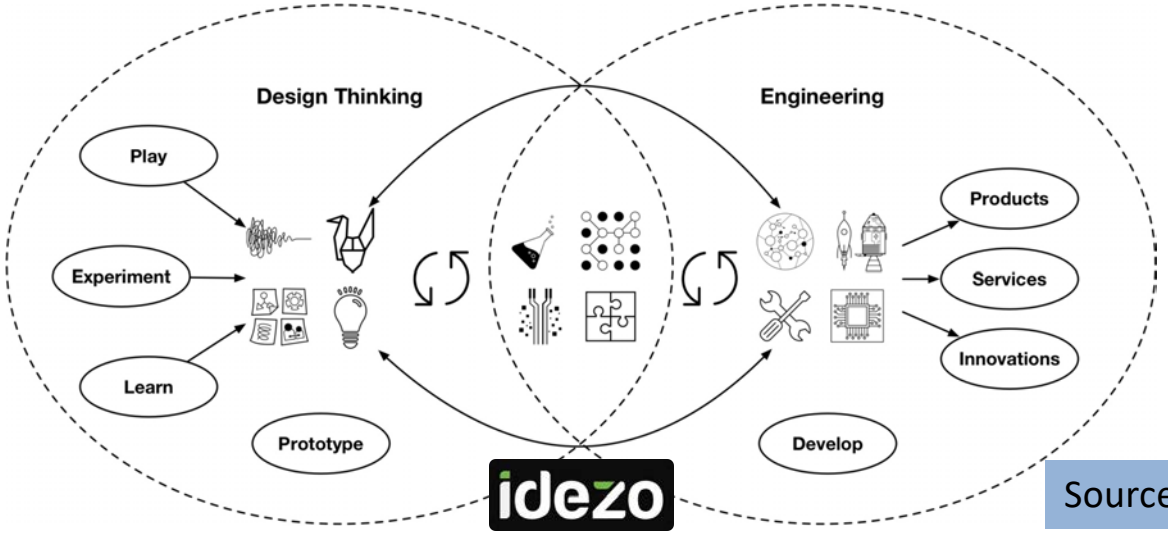
2 Gebruik je @hr.nl e-mailadres om in te loggen.

## Inhoud

- Meer dan 8000 instructievideo's en 42.000 (hoofdzakelijk Engelstalige) e-books.

https://www.hogeschoolrotterdam.nl/voorlichting/voorzieningen/mediatheek/informatiebronnen/databases/mtdb/learning_oreilly/184698/

# HANDS ON APPROACH TO DATA SCIENCE for (the) IoT




Source: https://idezo.ch



This Data Science Course was developed for Hogeschool Rotterdam **(Rotterdam University of Applied Sciences, RUAS) through the Program for AI & ethics (SPAiCE)**.
If you find errors or omissions, please contact the author, Rob van der Willigen, at r.f.van.der.willigen@hr.nl. Materials of this course and code examples used will become available at:
https://github.com/robvdw/CMIDAT01K-DATA-SCIENCE-for-IOT

# Course Setup

HOGESCHOOL ROTTERDAM

| | |
|---|---|
| **Lesson 01:** week 02 | **Discovering the IoT Data Science Domain** |
| **Lesson 02:** week 03 | **Defining project requirements** |
| **Lesson 03:** week 04 | **Learn to write code** |
| **Lesson 04:** week 05 | **Data Science: How to start your own IoT Project** |
| **Lesson 05:** week 06 | **IoT Platforms & MiddleWare** |
| **Lesson 06:** week 07 | **Core IoT Concepts + Code-testing via IoT** middelware |
| **Lesson 07:** week 08 | **Explaining Grading + Summary + Q & A** |

| | |
|---|---|
| **Week 08:** | **FEEDBACK** |
| **Week 10:** | **Submit your IoT-Project via LMS** |

# lesson three

HOGESCHOOL ROTTERDAM

HOGESCHOOL ROTTERDAM

# LES: 03

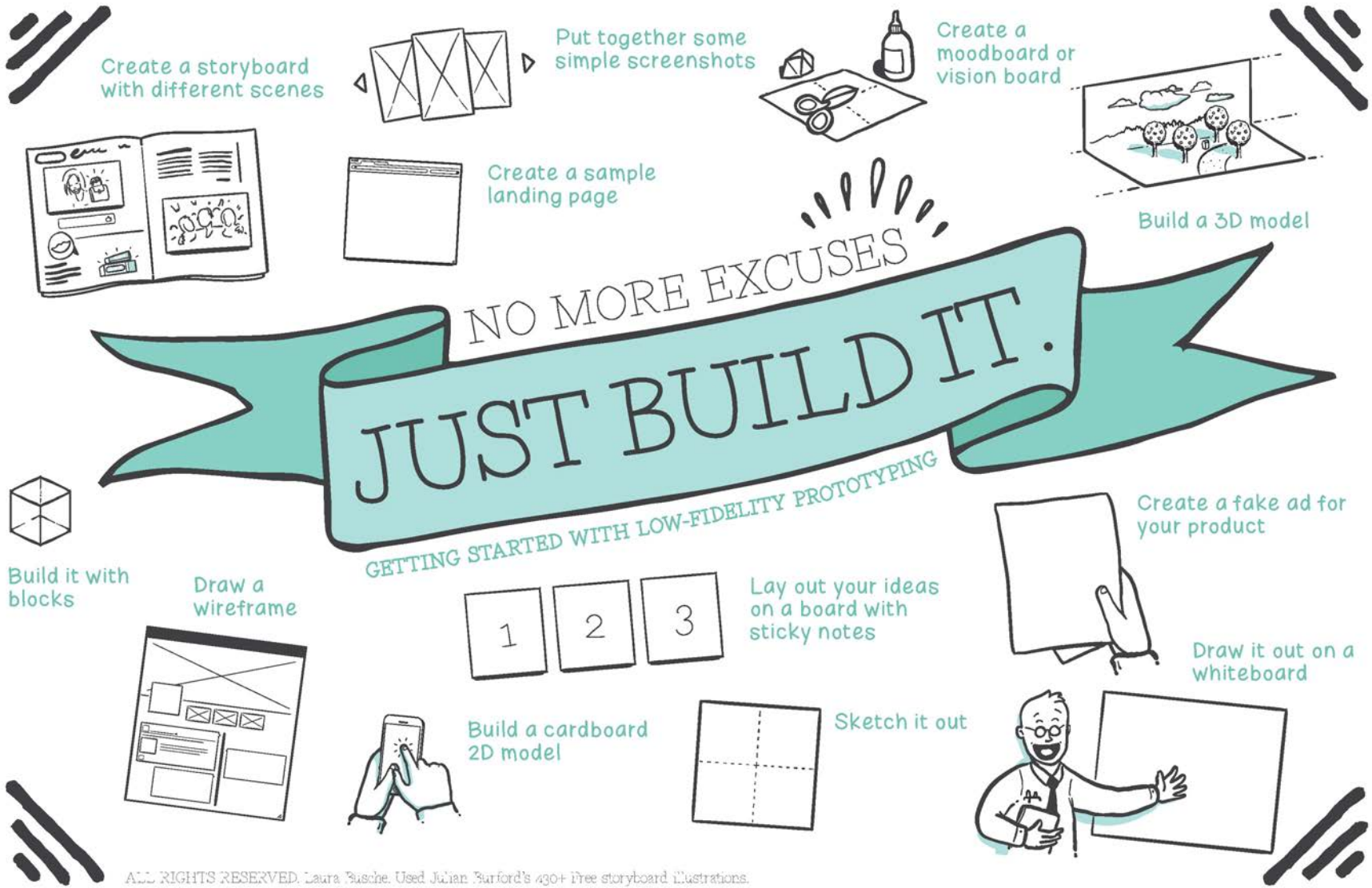
**Think like a programmer & Rapid prototyping**

**Preview Les 04**

**Write your first line of code (Python)**

# Rapid Prototyping

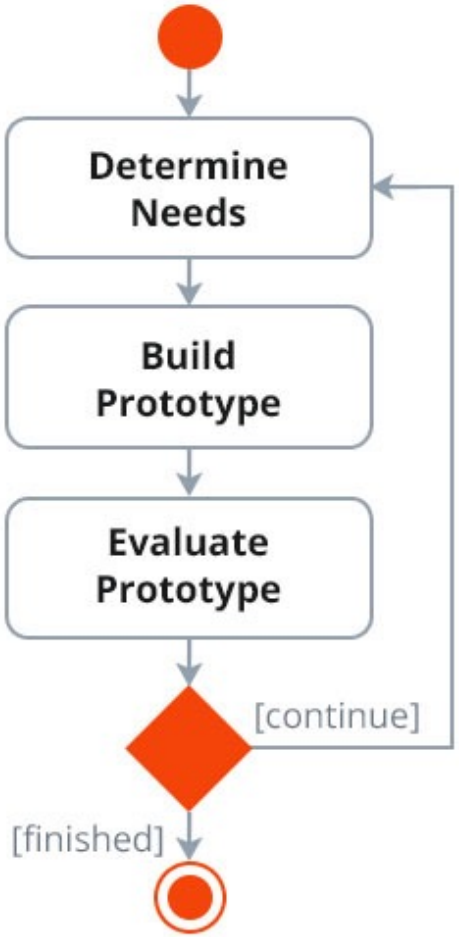
HOGESCHOOL ROTTERDAM


Create a storyboard with different scenes
Put together some simple screenshots
Create a moodboard or vision board
Create a sample landing page
Build a 3D model
NO MORE EXCUSES
JUST BUILD IT.
GETTING STARTED WITH LOW-FIDELITY PROTOTYPING
Build it with blocks
Draw a wireframe
1
2
3
Lay out your ideas on a board with sticky notes
Create a fake ad for your product
Draw it out on a whiteboard
Build a cardboard 2D model
Sketch it out
ALL RIGHTS RESERVED. Laura Busche. Used Julian Burford's 430+ Free storyboard Illustrations.

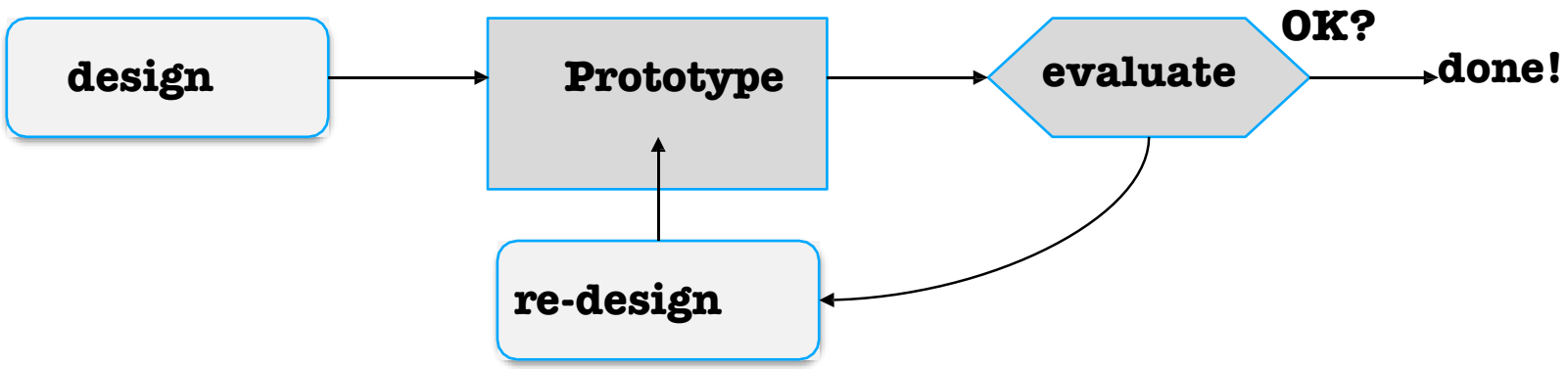

https://www.youtube.com/watch?v=JMjozqJS44M

# Rapid Prototyping

- You never get it right first time
- If at first you don"t succeed ...

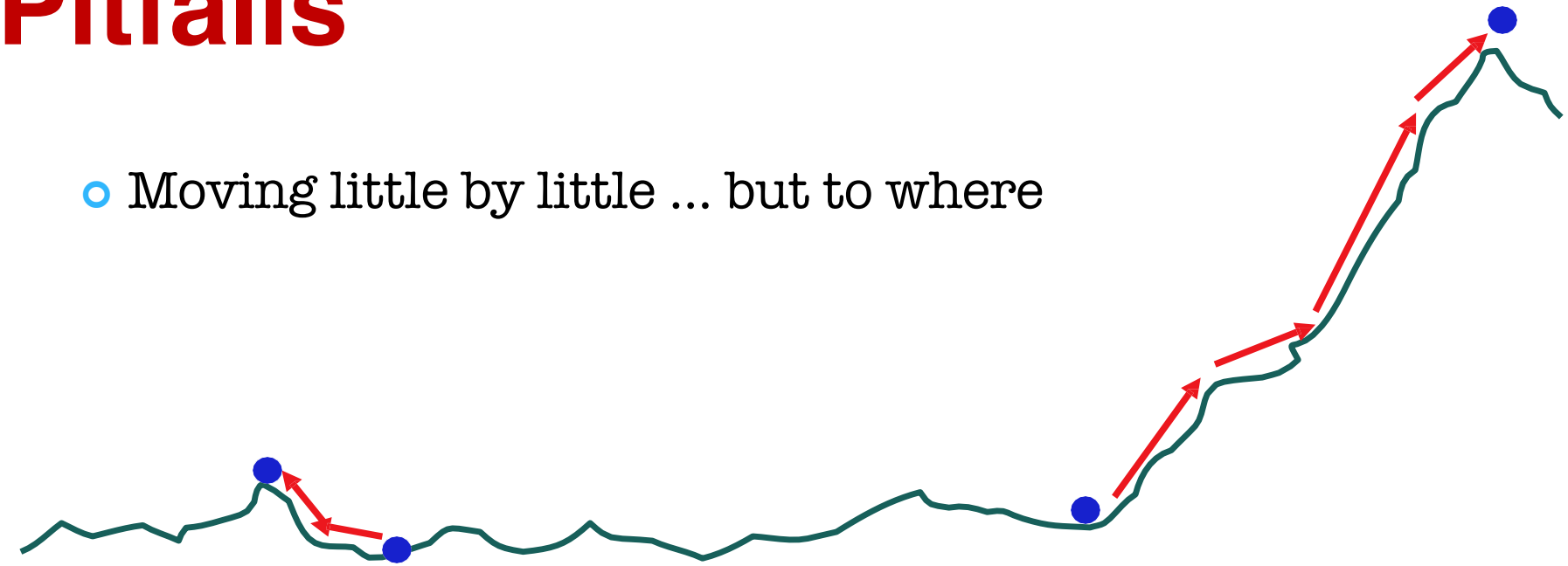


# Rapid Prototyping



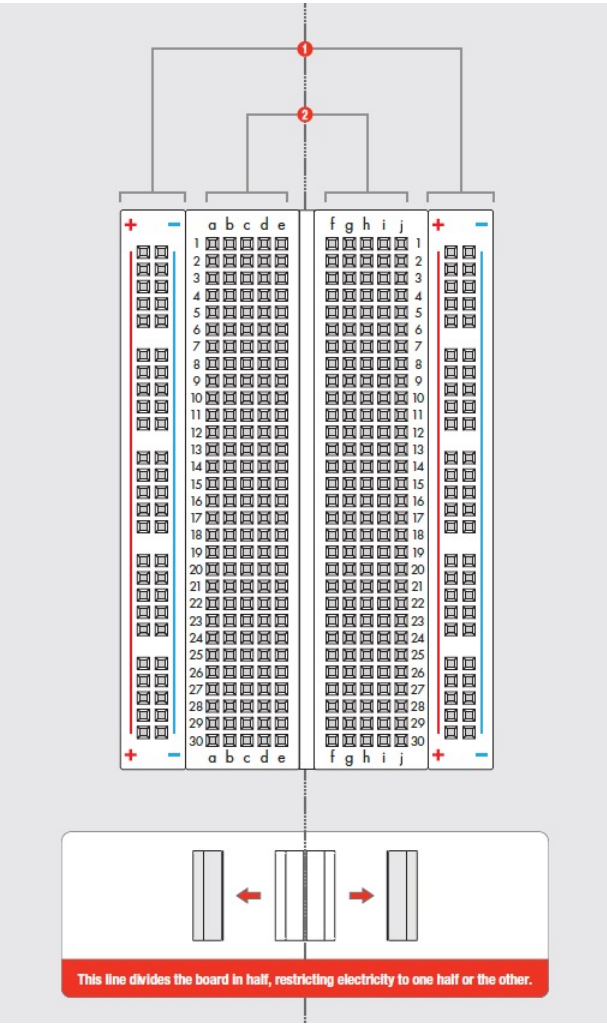
# Rapid Prototyping Pitfalls

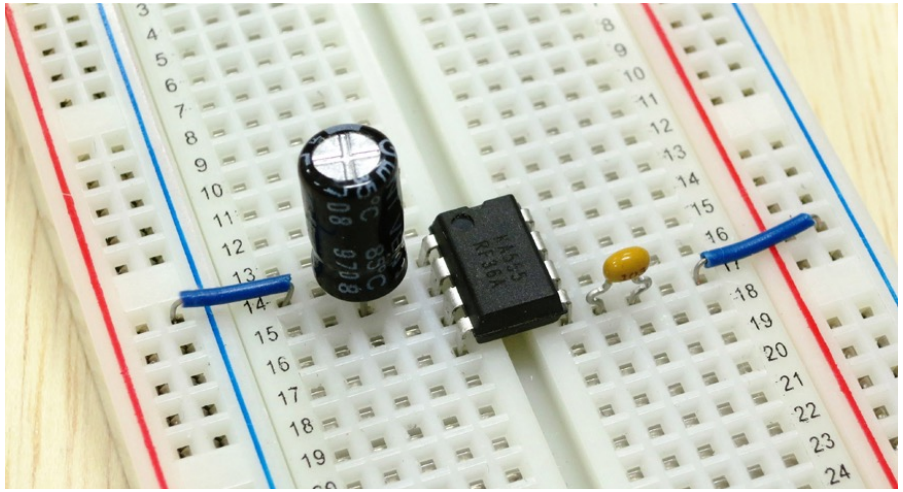
- Moving little by little … but to where

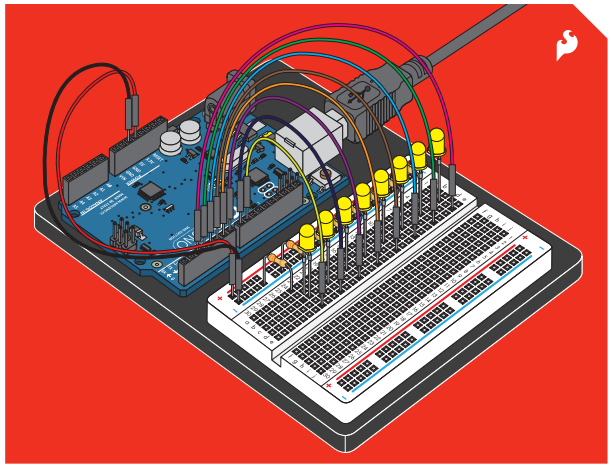


1. need a good start point
2. need to understand what is wrong

# Breadboard

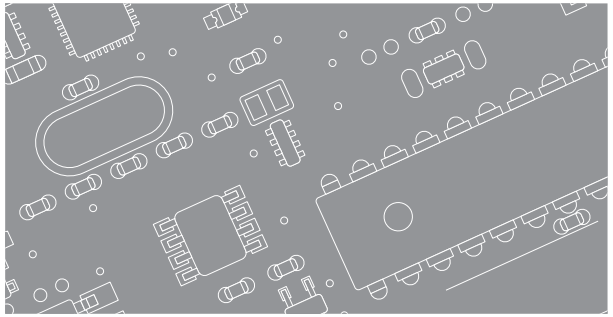




SIK GUIDE

Your Guide to the SparkFun Inventor's Kit for Arduino

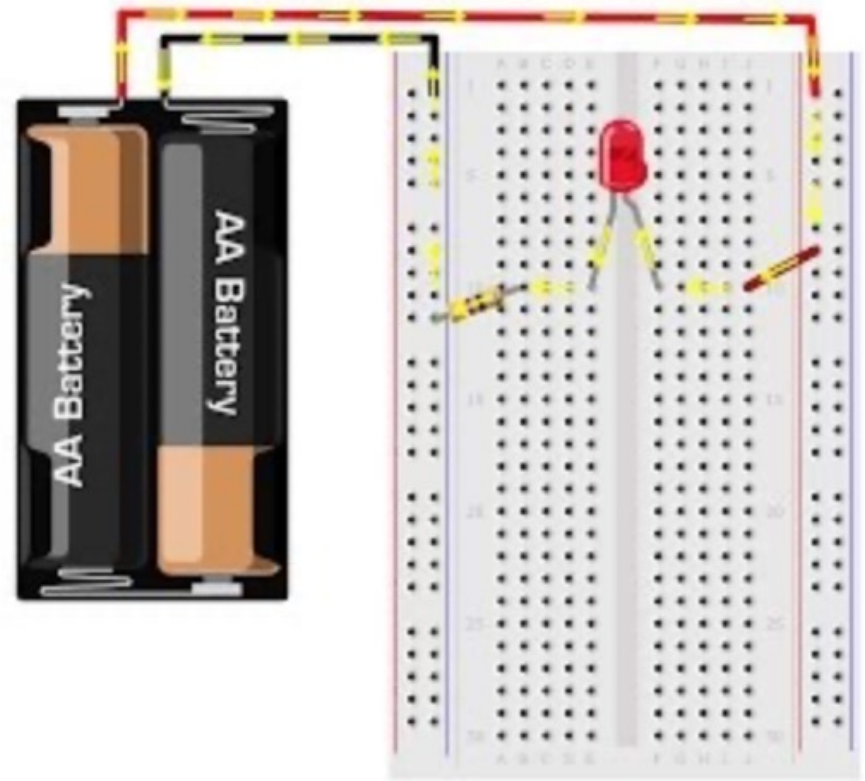


https://cdn.sparkfun.com/datasheets/Kits/RedBoard_SIK_3.2.pdf

https://learn.sparkfun.com/tutorials/how-to-use-a-breadboard/

# Breadboard

## How to use it for Rapid Prototyping




https://www.youtube.com/watch?v=6WReFkfrUIk

# ALGORITHMS

HOGESCHOOL ROTTERDAM

HOGESCHOOL ROTTERDAM

**Thinking like a computer scientist means more than being able to program a computer. It requires thinking at multiple levels of abstraction.**

---

**JEANNETTE M. WING** (wing@cs.cmu.edu) is the President's Professor of Computer Science in and head of the Computer Science Department at Carnegie Mellon University, Pittsburgh, PA.

---

HOGESCHOOL ROTTERDAM

# Computational Thinking

It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use.

" Computational thinking
is a way of solving problems,
designing systems, and
understanding human behavior
that draws on concepts
fundamental to computer science."

COMMUNICATIONS OF THE ACM March 2006/Vol. 49, No. 3

# Wat is an algorithm?

**A tool for solving a well-defined (computational) problem by manipulating symbols**

The word is derived from the name of the Persian mathematician al-Khwārizmī, who lived in the 9th century.

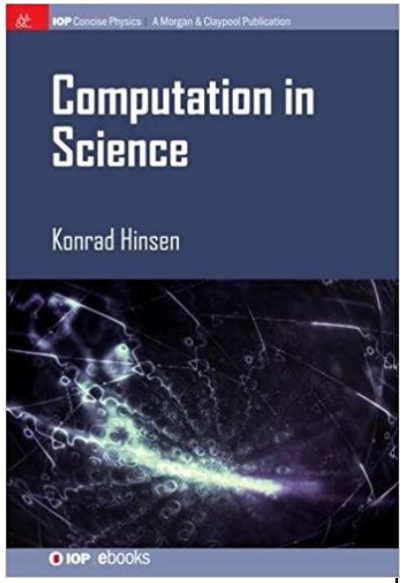
His book describing the 'Indian numbers', which today we call Arabic numerals, introduced our modern decimal notation and its rules for arithmetic into Europe.

The use of this system was called 'algorism' in the late middle ages and is transformed into today's 'algorithm'.

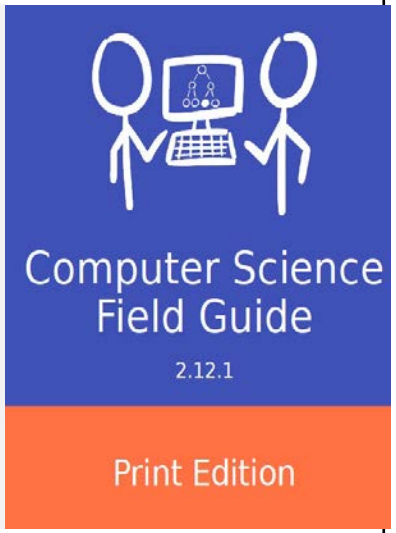
Crossley J.N., & Henry A.S. (1990)Thus spake al-Khwārizmī:
a translation of the text of Cambridge University Library Ms. Ii.vi.5 *Hist. Math.* 17(2) 103–31

# Wat is computation?

$$\begin{array}{rrr} 1 & 7 & 3 \\ 0 & 5 & 1 \\ & 0 & \\ \hline & & 4 \end{array} \quad \rightarrow \quad \begin{array}{rrr} 1 & 7 & 3 \\ 0 & 5 & 1 \\ 1 & & \\ \hline & 2 & 4 \end{array} \quad \rightarrow \quad \begin{array}{rrr} 1 & 7 & 3 \\ 0 & 5 & 1 \\ & & \\ \hline 2 & 2 & 4 \end{array}$$




Computations are arithmetic operations or numerical calculations that we all do in everyday life: adding up, multiplying, dividing etc.



# Wat is computation?

When thinking about computation,
it is important to know that
**symbols** & **meanings** are separate.

This is known as the distinction between
**syntax** and **semantics**.

# NUMBERS vs SYMBOLS

Computations do not work on numbers alone,
but on **Symbols:**
a specific representation for numbers

.

# NUMBERS vs SYMBOLS

Different representations lead to different methods for doing arithmetic.

# NUMBERS vs SYMBOLS

Even though the decimal character string '42', the roman-numeral character string 'XLII', and the English language character string 'forty-two' refer to the same number, they cannot be manipulated in the same way.

# SYNTAX vs SEMANTICS

**Syntax** defines **which** sequences of **symbols** a particular algorithm deals with:

'a sequence of any number of the digits 0 to 9'

0 … 9 (maths)
X=0:1:9 (matlab)

# SYNTAX vs SEMANTICS

**Semantics** defines
**how** such sequences of **symbols** are **interpreted**,
such as "natural numbers"
2  3 4 but not 0.00123 or 89/234

# Non-numerical computation

$$y + 2x = z \quad \rightarrow \quad x = \tfrac{1}{2}(z - y)$$

Once we get rid of the idea that computation is about numbers, we can easily identify other operations that qualify as computations.

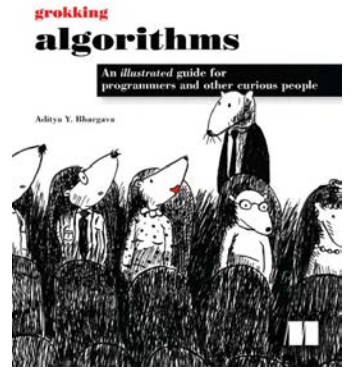
# Wat is an algorithm?

No knowledge of *semantics* is needed to apply an algorithm for adding two natural numbers.

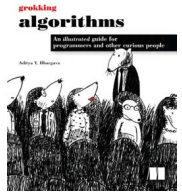
It is essential to understand what **an algorithm** does, and in particular which problems it is ment to solve.

# Algorithm

**Step by step process or recipe describing *how to solve a problem* and/or *complete a task*, which will always give the correct result**



# Algorithm 3:
# The Fox, the Goose, and the Corn




## PROBLEM: HOW TO CROSS THE RIVER?

A farmer with a fox, a goose, and a sack of corn needs to cross a river. The farmer has a rowboat, but there is room for only the farmer and one of his three items. Unfortunately, both the fox and the goose are hungry. The fox cannot be left alone with the goose, or the fox will eat the goose. Likewise, the goose cannot be left alone with the sack of corn, or the goose will eat the corn. How does the farmer get everything across the river?

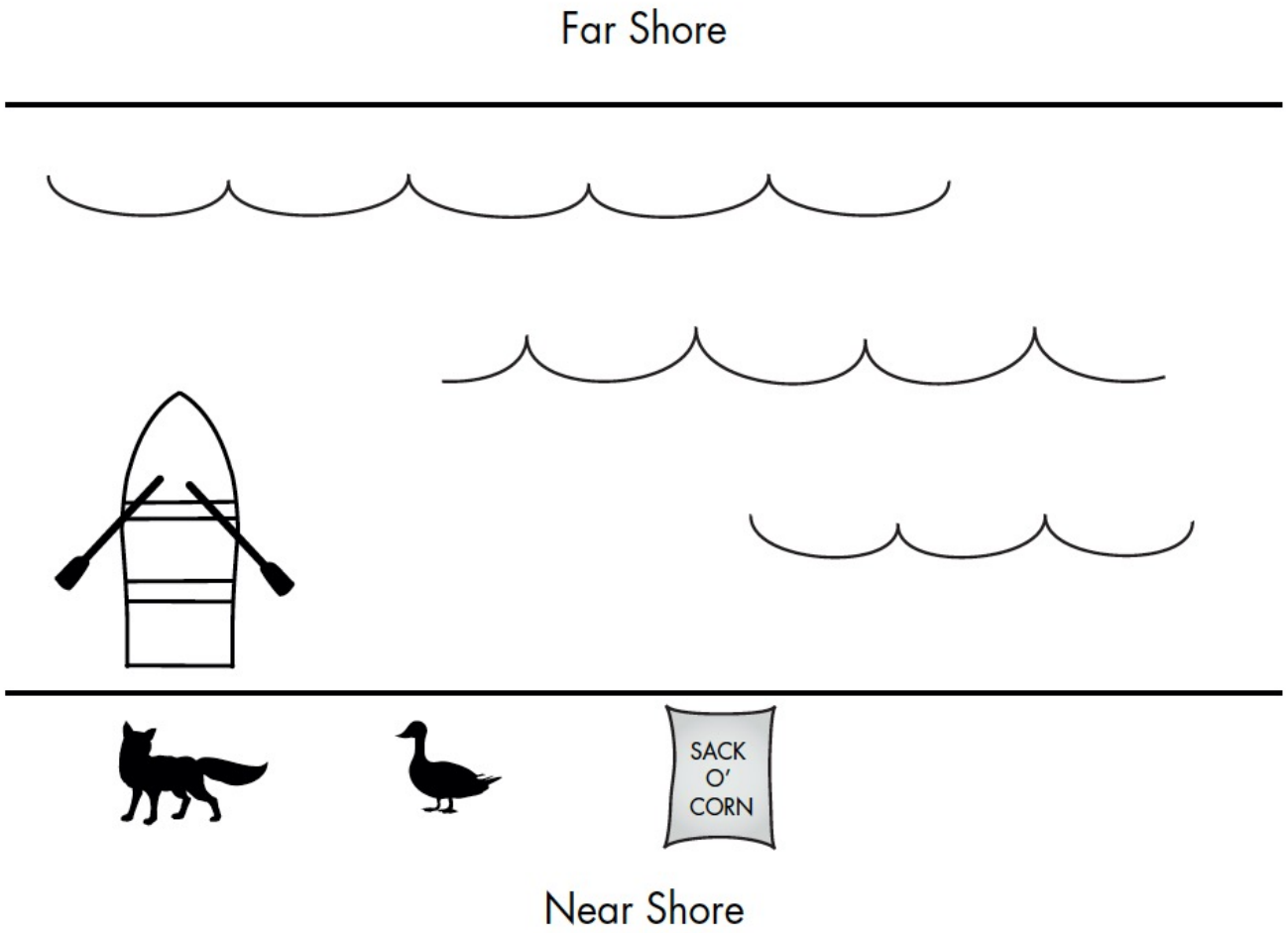
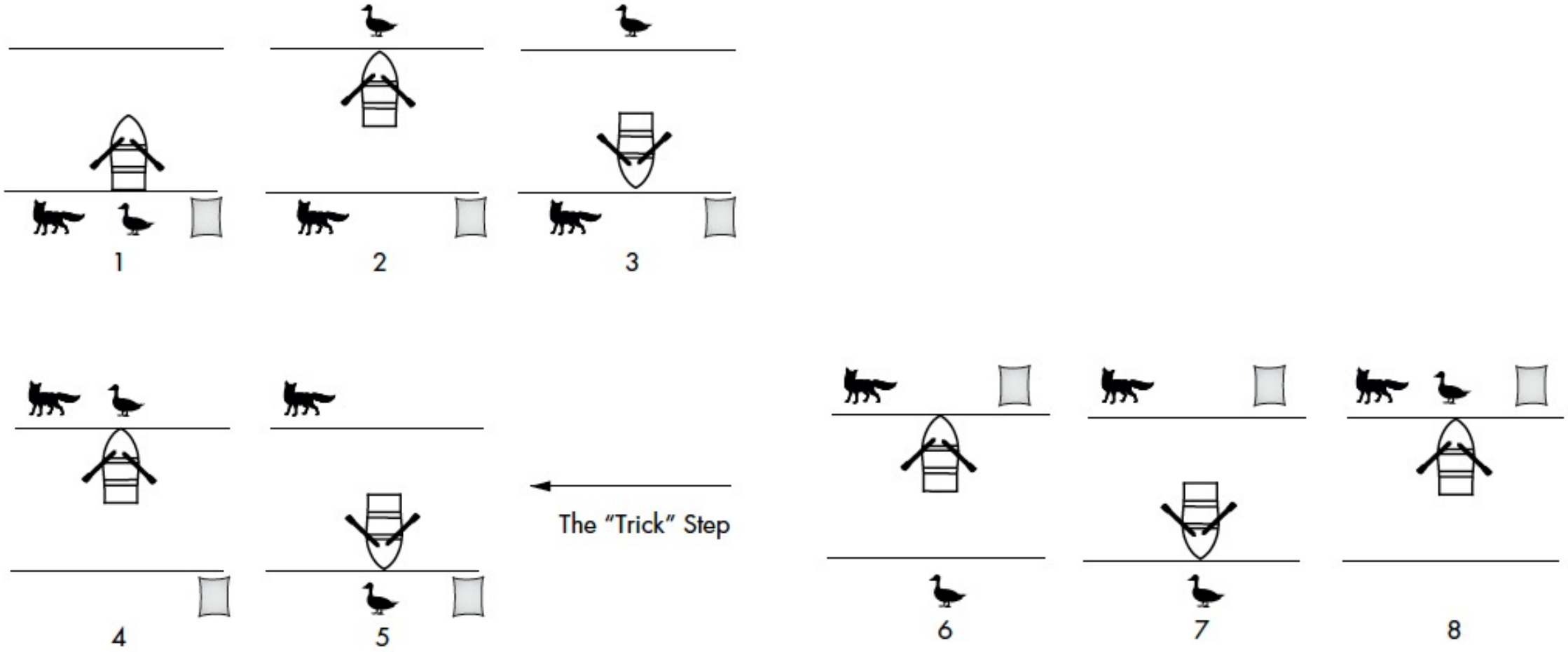


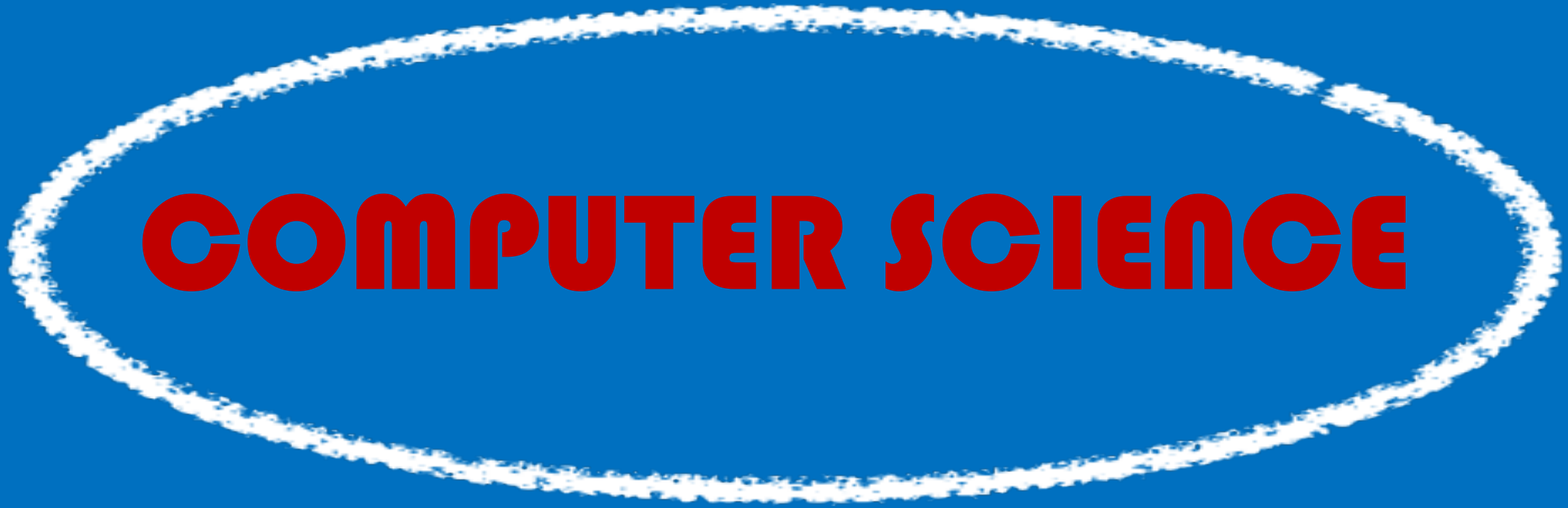

*Figure 1-1: The fox, the goose, and the sack of corn. The boat can carry one item at a time. The fox cannot be left on the same shore as the goose, and the goose cannot be left on the same shore as the sack of corn.*

# Algorithm 3:
## Step-by-step solution to the fox, goose, and corn problem



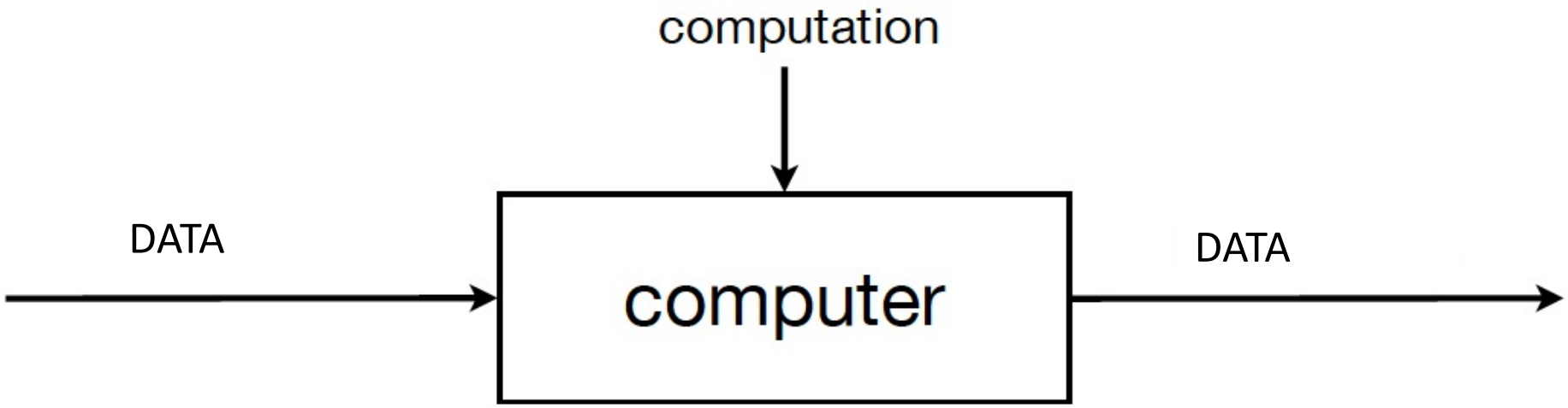


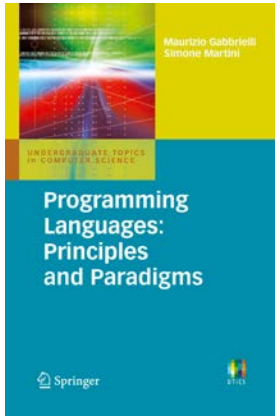
32




HOGESCHOOL ROTTERDAM

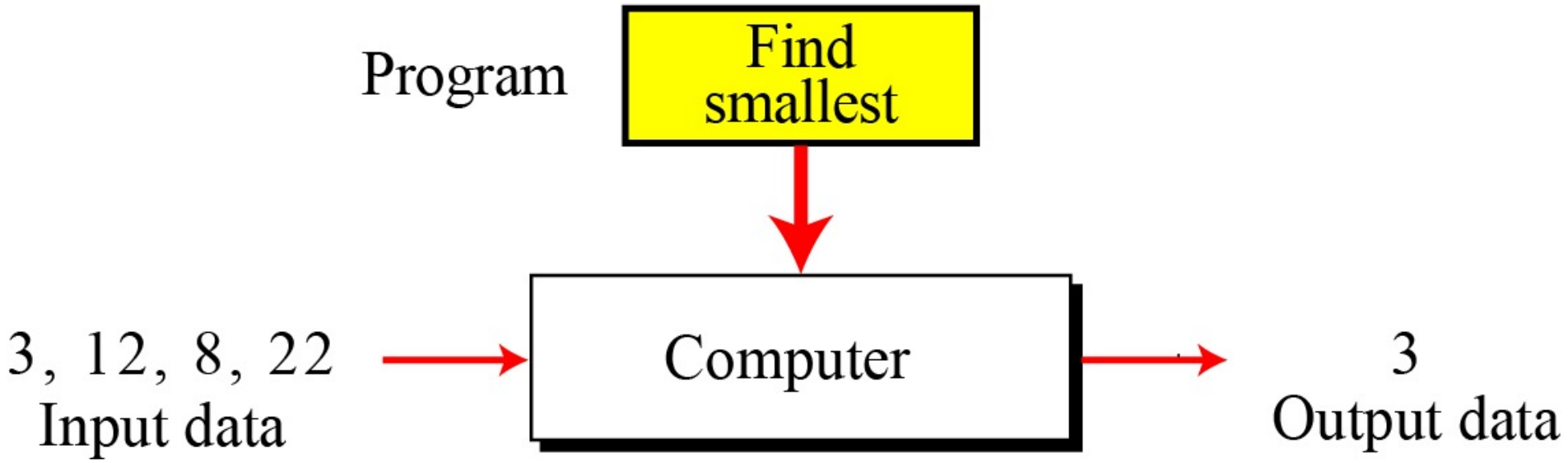
# What is a Computer?




A computer is a programmable machine that receives input, stores and manipulates data, and provides output in a useful format.

# Computers act as a Computational Data Processor




Modern computer acts as a *black box* that accepts input data, processes the data, and creates output data.

# Software

Computersoftware, of gewoon software,
maakt deel uit van een computersysteem dat bestaat uit gegevens of computerinstructies,
in tegenstelling tot de fysieke hardware waaruit het systeem is opgebouwd.

https://www.codeguild.nl/software-engineering-development-ontwikkeling-wikipedia-begrippenlijst-terminologieen/wat-is-software/

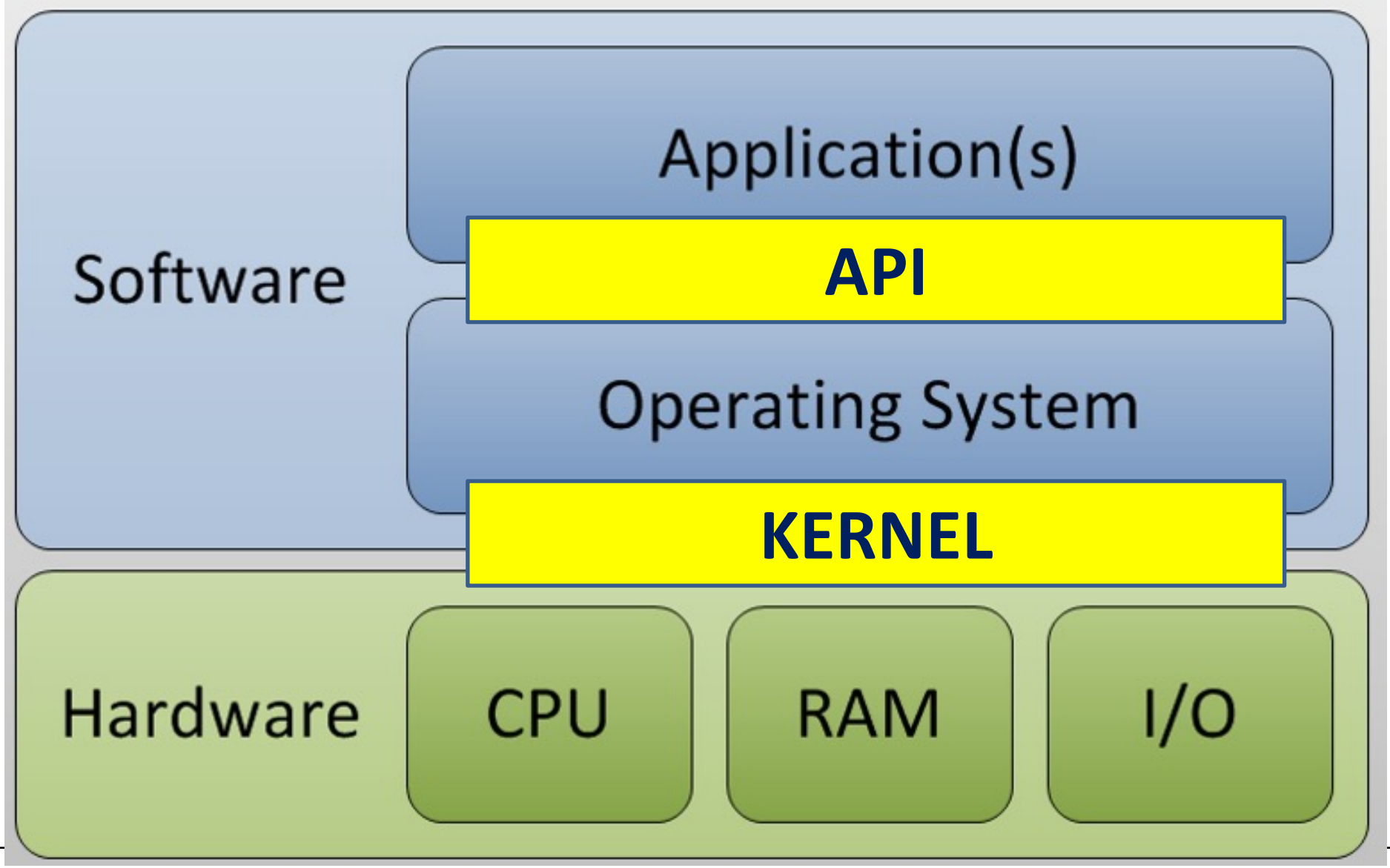
In computerwetenschap en softwareontwikkeling is computersoftware alle informatie die wordt verwerkt door computersystemen, programma's en gegevens.

# Software

Software is a generic term, which is used to describe a group of computer programs & procedures which perform some task on a computer (system).

It is an ordered sequence of instructions given for changing the state of the computer hardware, in a certain predefined manner.

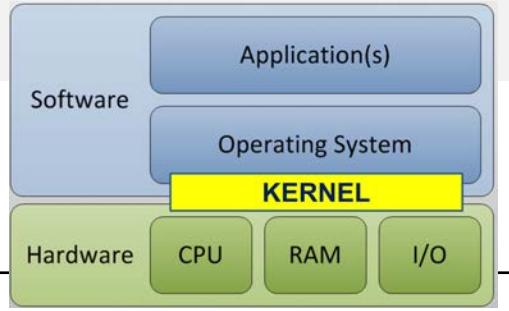
# Software vs Hardware



# Application software: Perfoms User Tasks

Application software is a computer software which is designed to help the user in performing single or multiple related tasks.

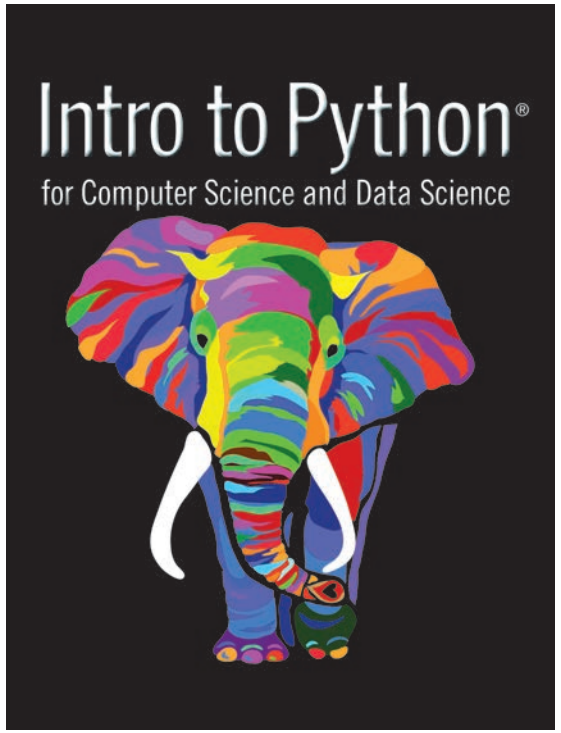
In other words, application software is actually a subclass of computer software,
which employs the capabilities of a computer directly to a task that the user wishes it to perform.



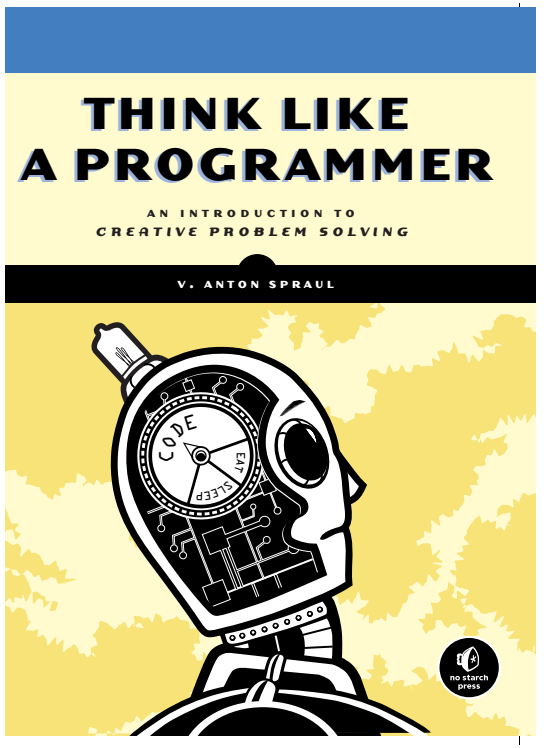
# How to start Coding

HOGESCHOOL ROTTERDAM

# Coding


Intro to Python®
for Computer Science and Data Science


GET CODING!
Learn HTML, CSS & JavaScript & build a website, app & game
Supported by
Young Rewired State
the expert community of coders


THINK LIKE A PROGRAMMER
AN INTRODUCTION TO CREATIVE PROBLEM SOLVING
V. ANTON SPRAUL
CODE
EAT
SLEEP
no starch press

# Types of programming languages

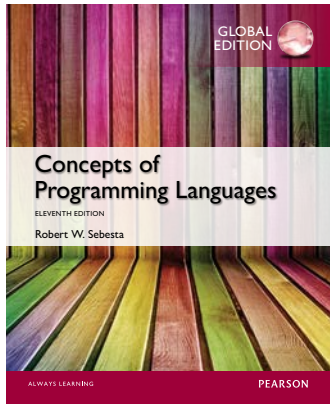
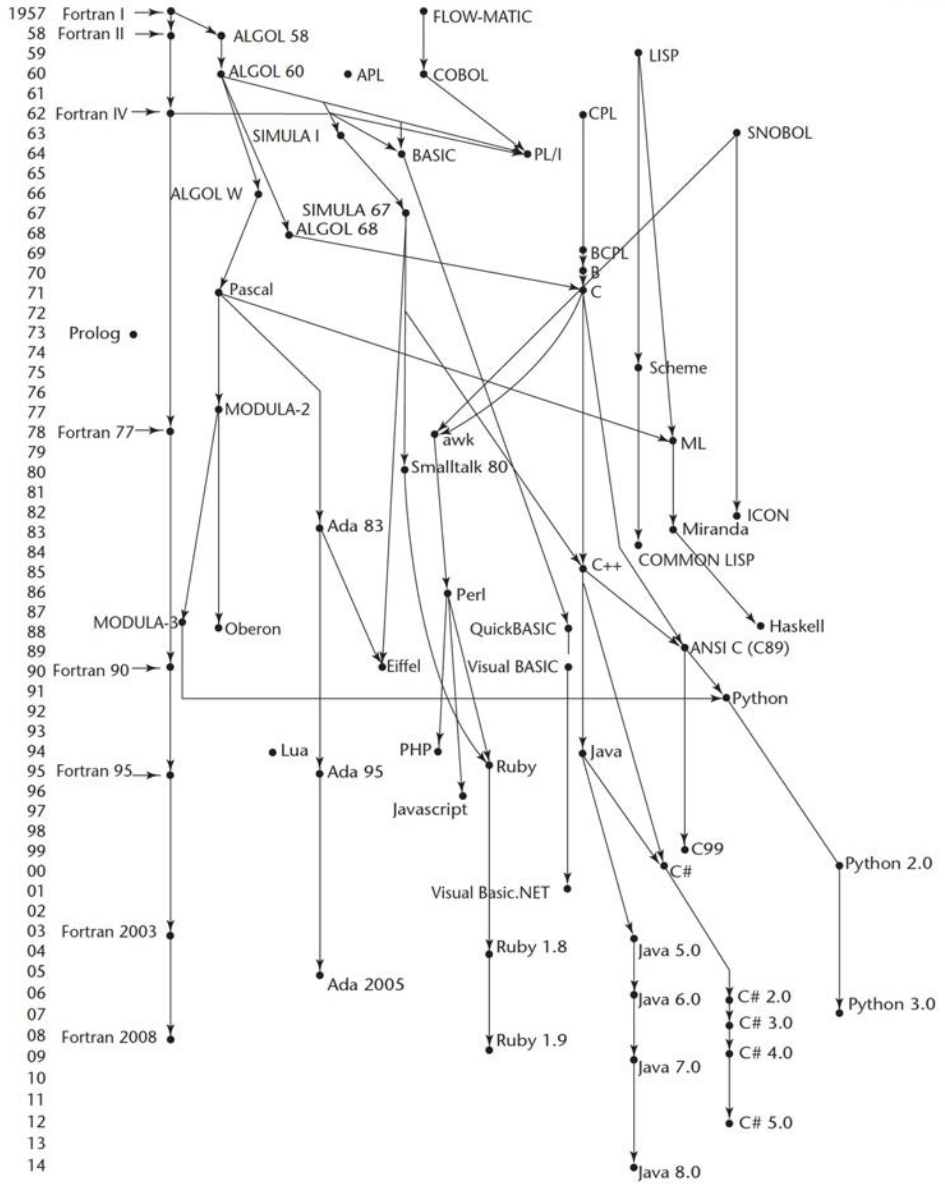



**Table 1.1** Language evaluation criteria and the characteristics that affect them

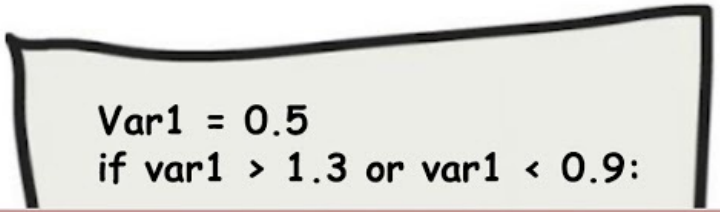
| | CRITERIA | | |
|---|---|---|---|
| Characteristic | READABILITY | WRITABILITY | RELIABILITY |
| Simplicity | • | • | • |
| Orthogonality | • | • | • |
| Data types | • | • | • |
| Syntax design | • | • | • |
| Support for abstraction | | • | • |
| Expressivity | | • | • |
| Type checking | | | • |
| Exception handling | | | • |
| Restricted aliasing | | | • |

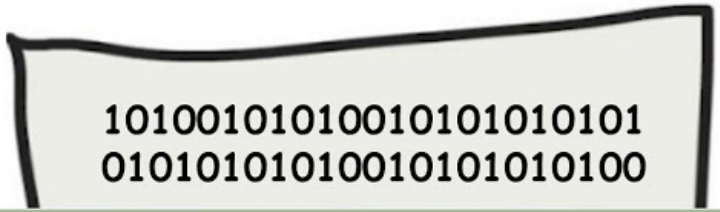


# Types of programming languages


https://www.youtube.com/watch?v=fbZ4ZhzCeR4

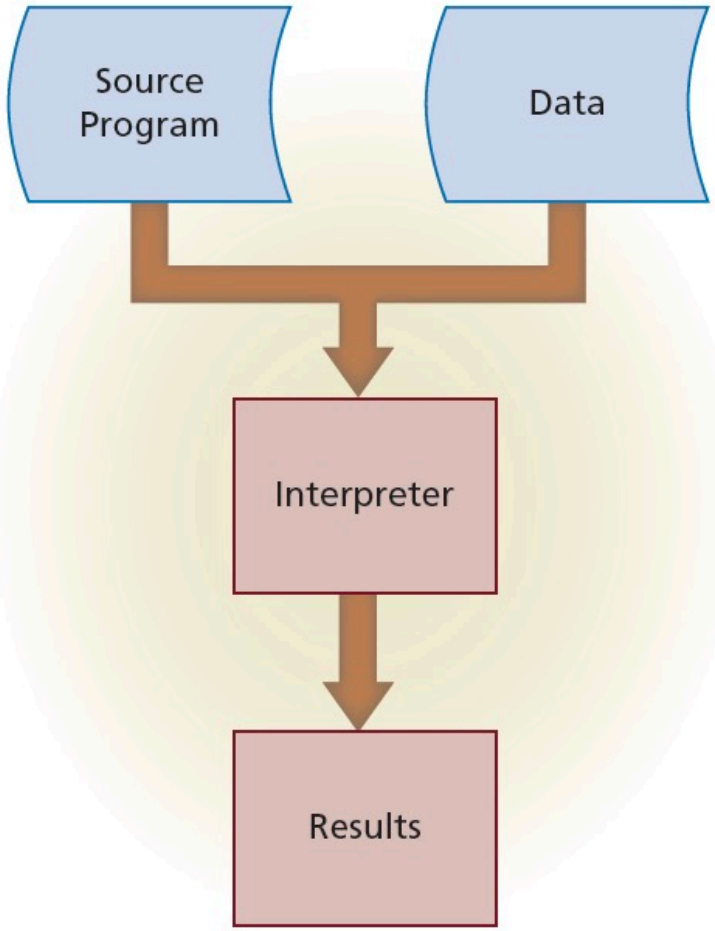
# High vs low level Languages



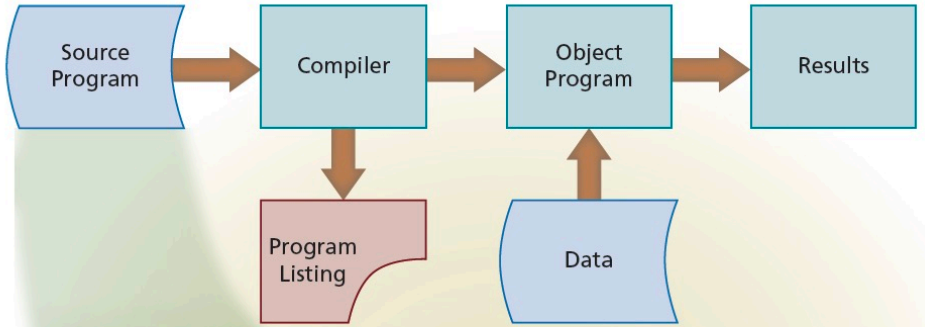




| High-level language | Low-level language |
|---|---|
| Easy for humans to read, write and modify | Hard for humans to read, write and modify |
| Programs run slower as they make worse use of the CPU and are not very memory efficient | Direct control over the CPU means memory use is more efficient and programs run faster |
| No understanding of how hardware components work is needed | Good understanding of how the different hardware components work is needed |
| Examples are python, java, C#, C++ etc. | Examples are machine code and assembly language |

# Script(ing) languages
# Do NOT need to be Compiled




# High-level Program languag
# need to be Compiled

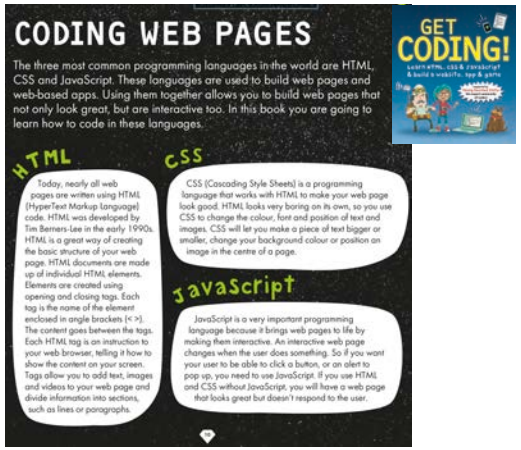



```
/* Compute Regular Time Pay                    */
rt_pay = rt_hrs * pay_rate;

/* Compute Overtime Pay                        */
if (ot_hrs > 0)
    ot_pay = ot_hrs * 1.5 * pay_rate;
else
    ot_pay = 0;

/* Compute Gross Pay                           */
gross = rt_pay + ot_pay;

/* Display Gross Pay                           */
printf("The gross pay is %d\n", gross);
```

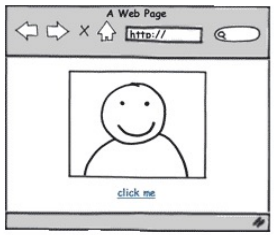
# Mark-up vs Programing Languages

## Markup Languages

HTML
CSS
XML

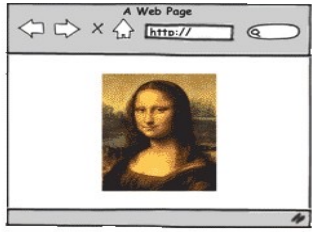



HTML defines content




CSS defines presentation


JavaScript defines behaviour




## Programming languages

C
C++
C#
Java

### Scripting languages

JavaScript
Php
Perl
VBScript

# High-level Program languages need to be Compiled

Compile is the process of creating (compilation) an executable program from code written in a compiled programming language.

Compiling allows the computer to run software without the need of the source-code itself on your computers.

Is Platform/Operating System [OS] dependent:
Desktop: Mac OS / Windows / UNIX / Linux
Mobile: iOS / Android / Chromium / Tizen

# Script(ing) languages do NOT need to be Compiled

A script or scripting language is a computer language with a series of commands within a file that is capable of being executed without being compiled.

Examples are Perl, PHP, Python & JavaScript.

# Interpreted vs Compiled Languages

An interpreted programming language does **not** need to be compiled before its programs are executed.

Instead, another program, called an interpreter, reads the program and executes it on the fly.

Often called scripting languages Perl, PhP & Python

# Interpreted vs Compiled Languages

A compiled program generally performs faster (higher Big O) for the end user, because its machine code can be highly optimized during the compilation process.

In contrast, interpreted languages can offer unique benefits to the programmer. One example is a REPL, which allows the programmer to interact with the program while it is being written.

# API v.s. SDK v.s. IDE

## Application Programming Interface (API)

A library of functions and methods.

You don't need to know how it works, but you have to know how to call them.

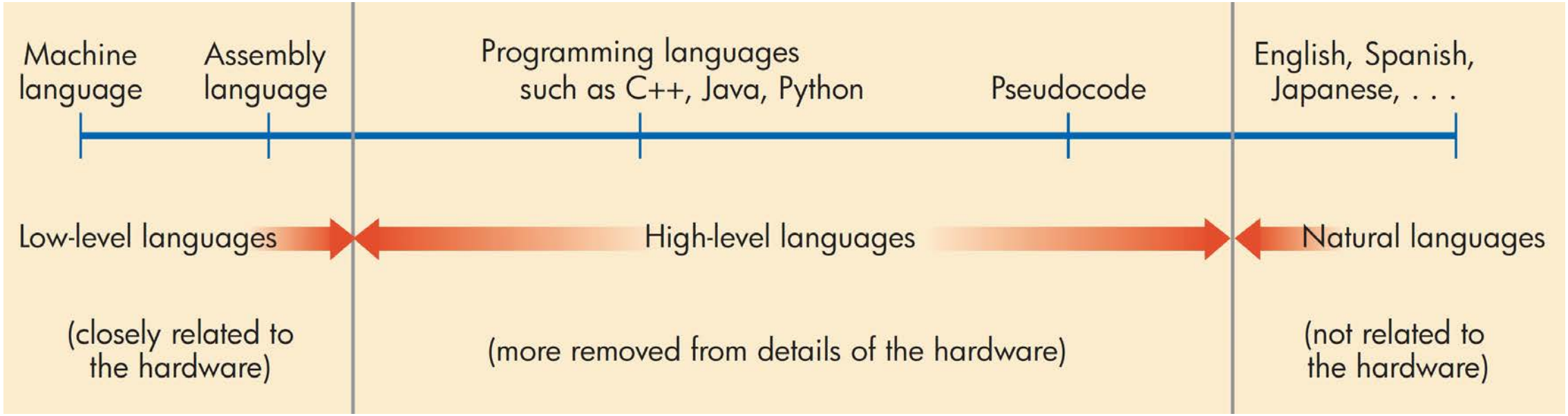
## Software Development Kit (SDK)

Many tools are included in SDK. Different platform, different SDK.

## Integrated Development Environment (IDE)

Usually includes code editor, debugger, compiler.

# Coding-paradigma's



# Coding-paradigma's

| | Scripting | Compiled |
|---|---|---|
| Feature | Can run without compilation | Need to compile to machine code before running |
| Examples | Python, Perl, PHP, Ruby, Javascript | C/C++, Fortran, Pascal, Java |
| Advantage | Easy to understand and extend<br>Portable | Optimized and run faster,<br>Source code protected |
| Running and Debugging model | Use an interpreter to understand and run the code on-the-fly | Codes are compiled to executables using compiler |

# Coding: a definition

Anyone interested in developing software, such as a source-code, game, or online service, must start by learning a programming language & decide which (source-code) Editor/Debugger to use.

https://www.computerhope.com/issues/ch000675.htm

# Command Line Terminal (interface)

HOGESCHOOL ROTTERDAM

What Is the Command Line?

The command-line interface, sometimes referred to as the CLI,

is a tool into which you can type text commands to perform specific tasks—in contrast to the mouse's pointing and clicking on menus and buttons.

Since you can directly control the computer by typing, many tasks can be performed more quickly, and some tasks can be automated with special commands that loop through and perform the same action on many files—saving you, potentially, loads of time in the process.

# Script(ing) languages
# Do NOT need to be Compiled

A script or scripting language is a computer language with a series of commands within a file that is capable of being executed without being compiled.

Examples are Perl, PHP, Python & JavaScript.

# Interpreted vs Compiled Languages

An interpreted programming language does **not** need to be compiled before its programs are executed.

Instead, another program, called an interpreter, reads the program and executes it on the fly.

Often called scripting languages Perl, PhP & Python

# Interpreted vs Compiled Languages

A compiled program generally performs faster (higher Big O) for the end user, because its machine code can be highly optimized during the compilation process.

In contrast, interpreted languages can offer unique benefits to the programmer. One example is a REPL, which allows the programmer to interact with the program while it is being written.

# API v.s. SDK v.s. IDE

## Application Programming Interface (API)

A library of functions and methods.

You don't need to know how it works, but you have to know how to call them.

## Software Development Kit (SDK)

Many tools are included in SDK. Different platform, different SDK.

## Integrated Development Environment (IDE)

Usually includes code editor, debugger, compiler.

# CLT vs GUI

## Command line Terminal

## vs

## Graphical User Interface

HOGESCHOOL ROTTERDAM

# Command Line Terminal (interface)

HOGESCHOOL ROTTERDAM

What Is the Command Line?

The command-line interface, sometimes referred to as the CLI,

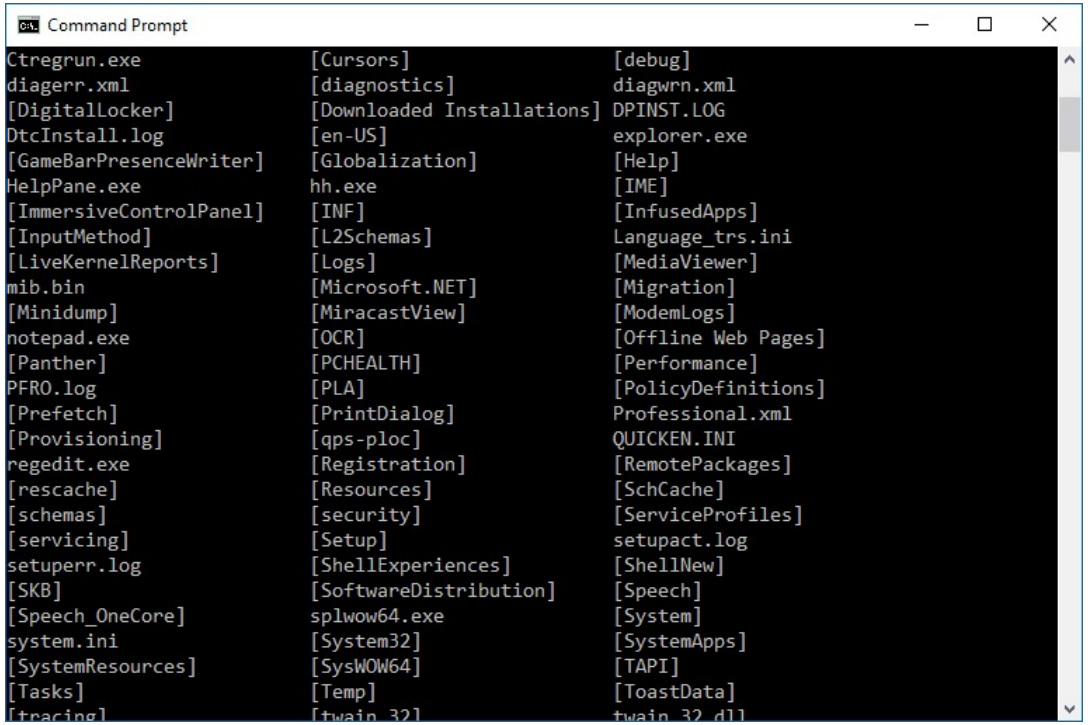
is a tool into which you can type text commands to perform specific tasks—in contrast to the mouse's pointing and clicking on menus and buttons.

Since you can directly control the computer by typing, many tasks can be performed more quickly, and some tasks can be automated with special commands that loop through and perform the same action on many files—saving you, potentially, loads of time in the process.

# Command Line Terminal (windows)

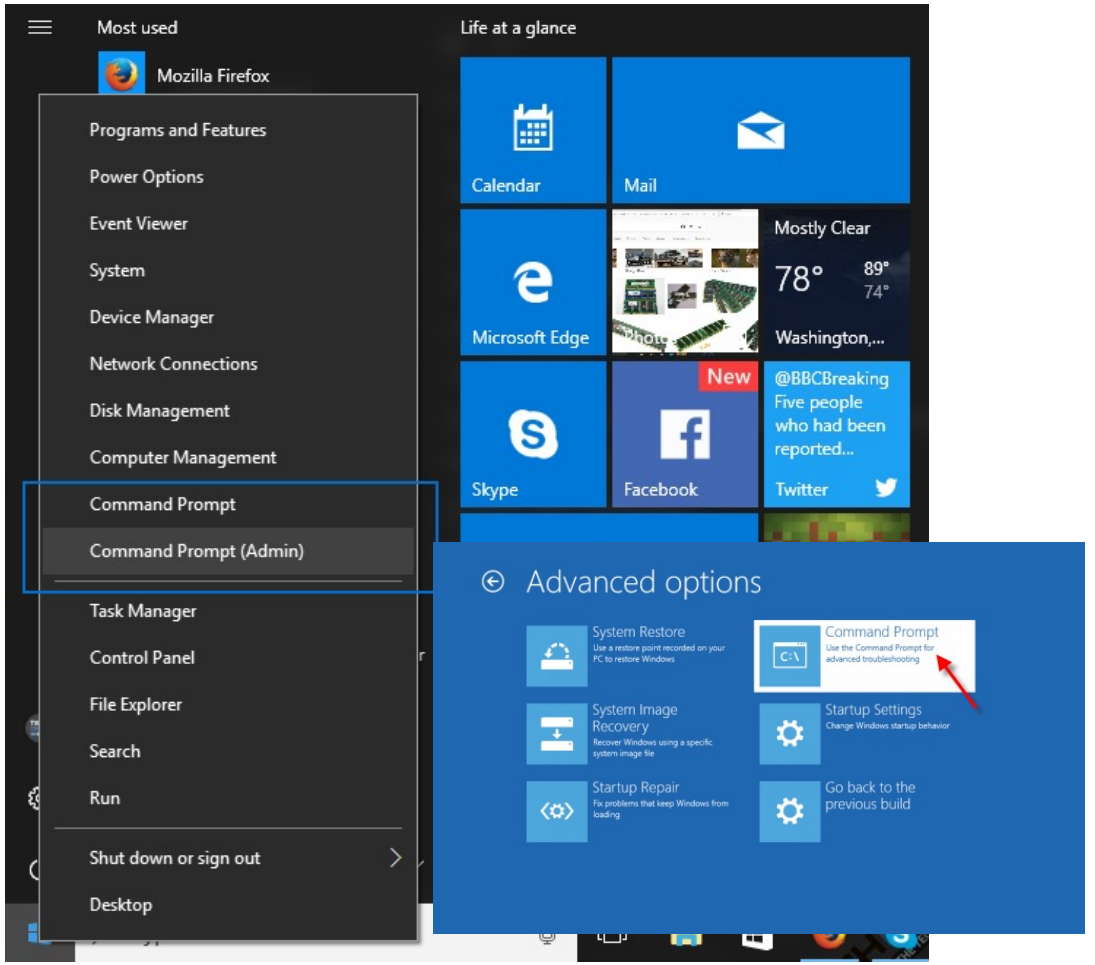
## List of Command Prompt Commands

Complete list of CMD commands for Windows




The Command Prompt in Windows provides access to over 280 commands! These commands are used to do certain operating system tasks from a command line interface instead of the graphical Windows interface we use most of the time.

**Note:** It's important to know that the commands in Windows 10, 8, 7, Vista, and XP are called *CMD commands* or *Command Prompt commands*, and the commands in Windows 98/95 and MS-DOS are called *DOS commands*.




https://www.lifewire.com/command-prompt-tricks-and-hacks-2618104

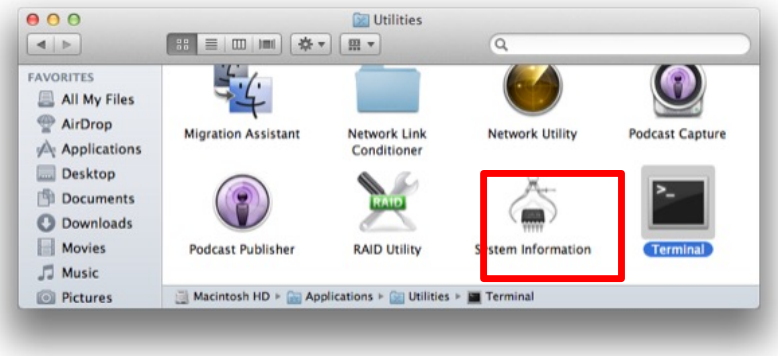
# Command Line Terminal MacOS

## Introduction to the Mac OS X Command Line

### How to open the command line.

Before you can use it, you need to be able to find it.

So what we need to do is open the terminal. On OS X, open your Applications folder, then open the Utilities folder. Open the Terminal application. You may want to add this to your dock. I like to launch terminal by using Spotlight search in OS X, searching for "terminal".




http://blog.teamtreehouse.com/introduction-to-the-mac-os-x-command-line

### Anatomy of the Console

First let's clarify a few terms.

**Console:** This is the system as a whole. This is both the command line as well as the output from previous commands.

**Command Line:** This is the actual line in a console where you type your command.

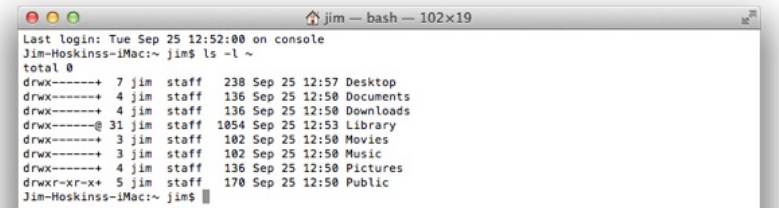
**Prompt:** This is the beginning of the command line. It usually provides some contextual information like who you are, where you are and other useful info. It typically ends in a **$** . After the prompt is where you will be typing commands.

**Terminal:** This is the actual interface to the console. The program we use to interact with the console is actually a "terminal emulator", providing us the experience of typing into an old school terminal from the convenience of our modern graphical operating system.

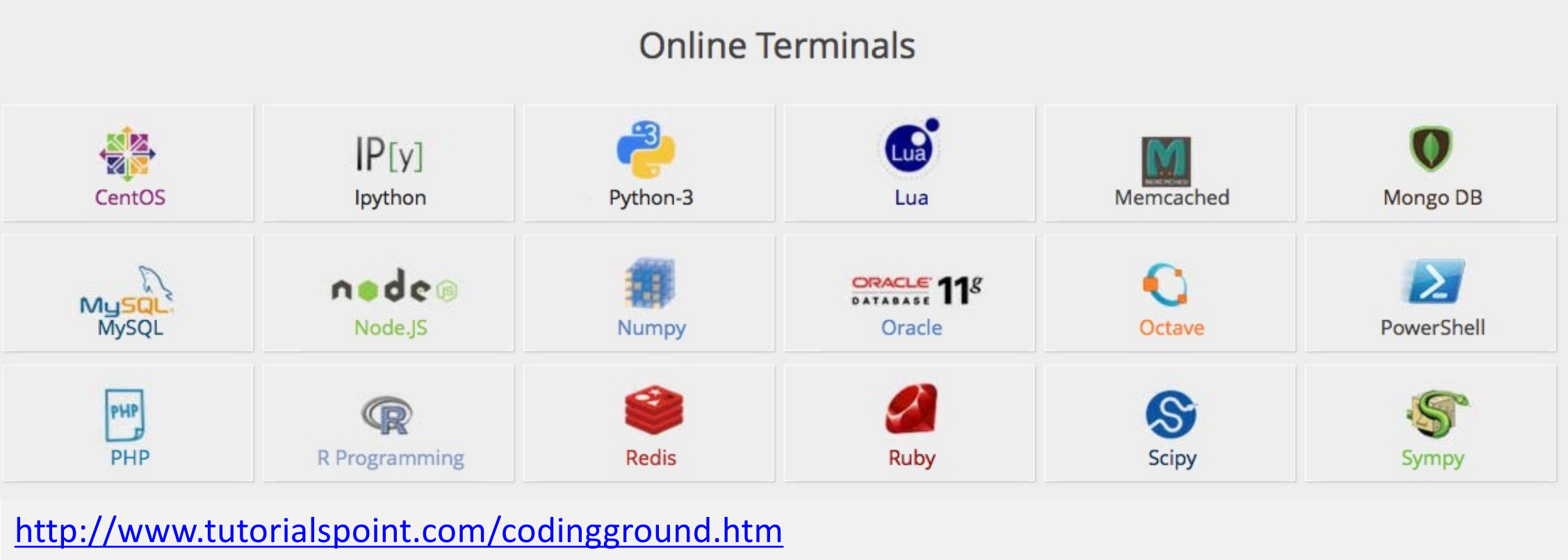
### Running a Command.

Nearly all commands follow a common pattern with 3 main parts. The program, the options, and the arguments. Let's see an example.

```
$ ls -l ~
```

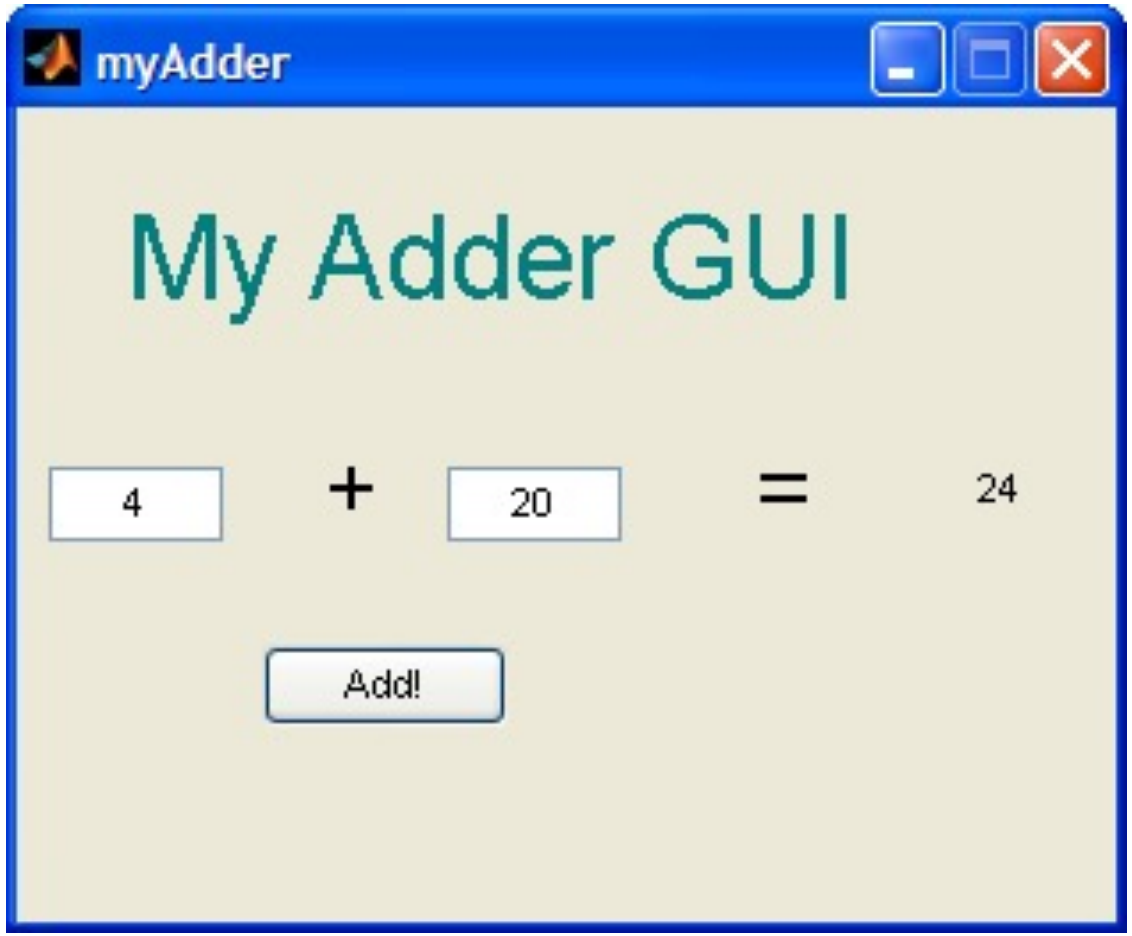


# Command Line Terminal



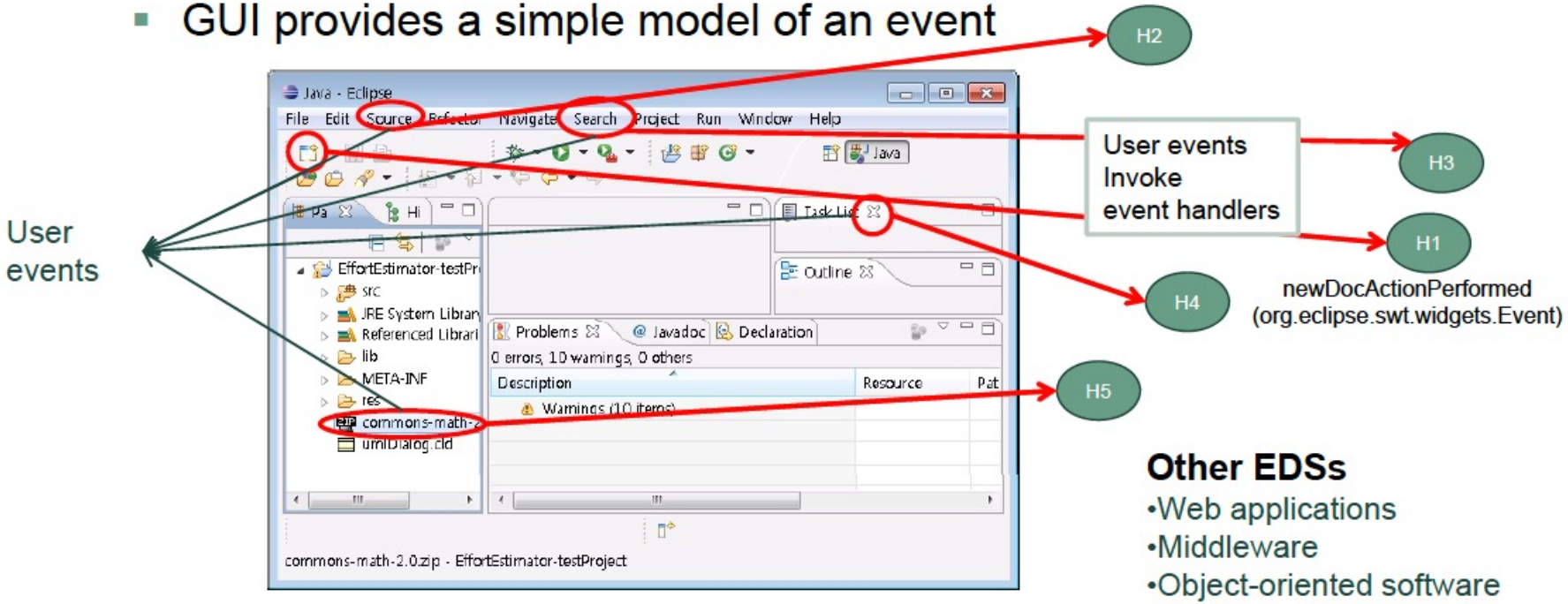

http://www.tutorialspoint.com/codingground.htm

# Graphical User Interface (GUI)



# Graphical User Interface (GUI)

- GUI provides a simple model of an event




**Other EDSs**

- Web applications
- Middleware
- Object-oriented software
- Multi-agent based systems

GUI is a type of user interface item that allows people to interact with programs in more ways than typing.
GUI software is an Event-driven Software(EDS)

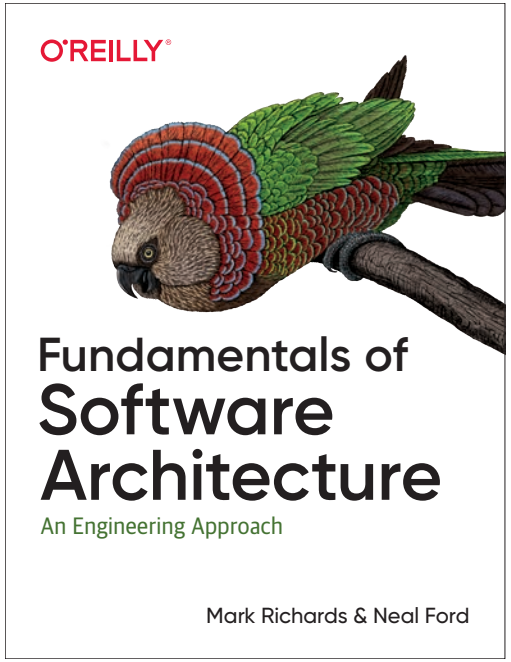
# CLI vs GUI

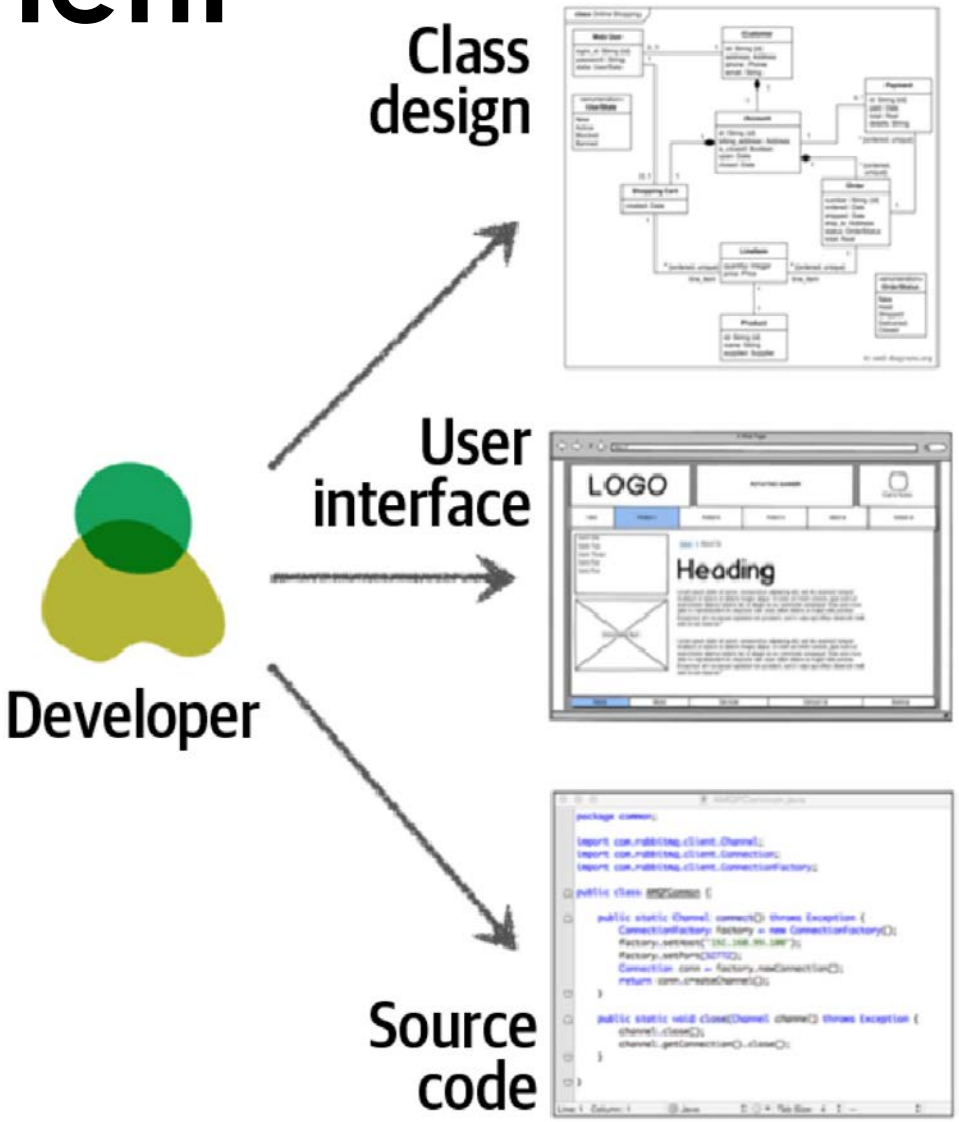
| Topic | Command line (CLI) | GUI |
|---|---|---|
| **Ease** | Due to a higher degree of memorization and familiarity needed for operation and navigation, new users find operating a command line interface more difficult than a GUI. | Because a GUI is much more visually intuitive, users typically pick up on how to use a GUI faster than a command line interface. |
| **Control** | Users have a good bit of control over both the file and operating systems in a command line interface. However, for new or novice users, it is not as user friendly as a GUI. | A GUI offers a lot of access to files, software features, and the operating system as a whole. Being more user friendly than a command line, especially for new or novice users, a GUI is utilized by more users. |
| **Multitasking** | Although many command line environments are capable of multitasking, they do not offer the same ease and ability to view multiple things at once on one screen. | GUI users have windows that enable a user to view, control, manipulate, and toggle through multiple programs and folders at same time. |
| **Speed** | Command line users only need to utilize a keyboard to navigate the interface, often resulting in faster performance. | While newer technology is making a GUI faster and more efficient than ever before, using both a mouse and keyboard to navigate and control the GUI is still a bit slower than a command line interface. |
| **Resources** | A computer that is only using the command line takes a lot less of the computer's system resources than a GUI. | A GUI requires more system resources because of the elements that require loading, such as icons and fonts. Video, mouse, and other drivers need to be loaded, taking up additional system resources. |
| **Scripting** | A command line interface mostly requires users to already know scripting commands and syntax, making it difficult for new or novice users to create scripts. | Creating scripts using a GUI has beco much easier with the help of programming software, which allows users to write the scripts without having to know all the commands and syntax. Programming software provides guides and tips for how to code specific functions, as well as preview options to see if and how the script will work. |
| **Remote access** | When accessing another computer or device over a network, a user can manipulate the device or its files with a command line interface. However, you must know the commands to do so and is not as easy for new or novice users. | Remotely access another computer or server is possible in a GUI and easy to navigate with little experience. IT professionals typically use a GUI for remote access, including the management of servers and user computers. |
| **Diversity** | After you've learned how to navigate and use a command line, it's not going to change as much as a new GUI. Although new commands may be introduced, the original commands almost always remain the same. | Each GUI has a different design and structure when it comes to performing different tasks. Even different iterations of the same GUI, such as Windows, can have hundreds of different changes between each version. |
| **Strain** | A command line interface is often very basic and can be more of a strain on a user's vision. Carpal Tunnel Syndrome can also be a bit of a risk when using a command line interface because users are only using a keyboard. There is little need to change hand positions and strain to the wrists or even fingers can occur. | The use of shortcut keys and more frequent movement of hand positions, due to switching between a keyboard and a mouse, strain may be reduced. Visual strain can still be a risk, but a GUI has more colors and is more visually appealing, leading to a potential reduction in visual strain. |

https://www.computerhope.com/issues/ch000619.htm

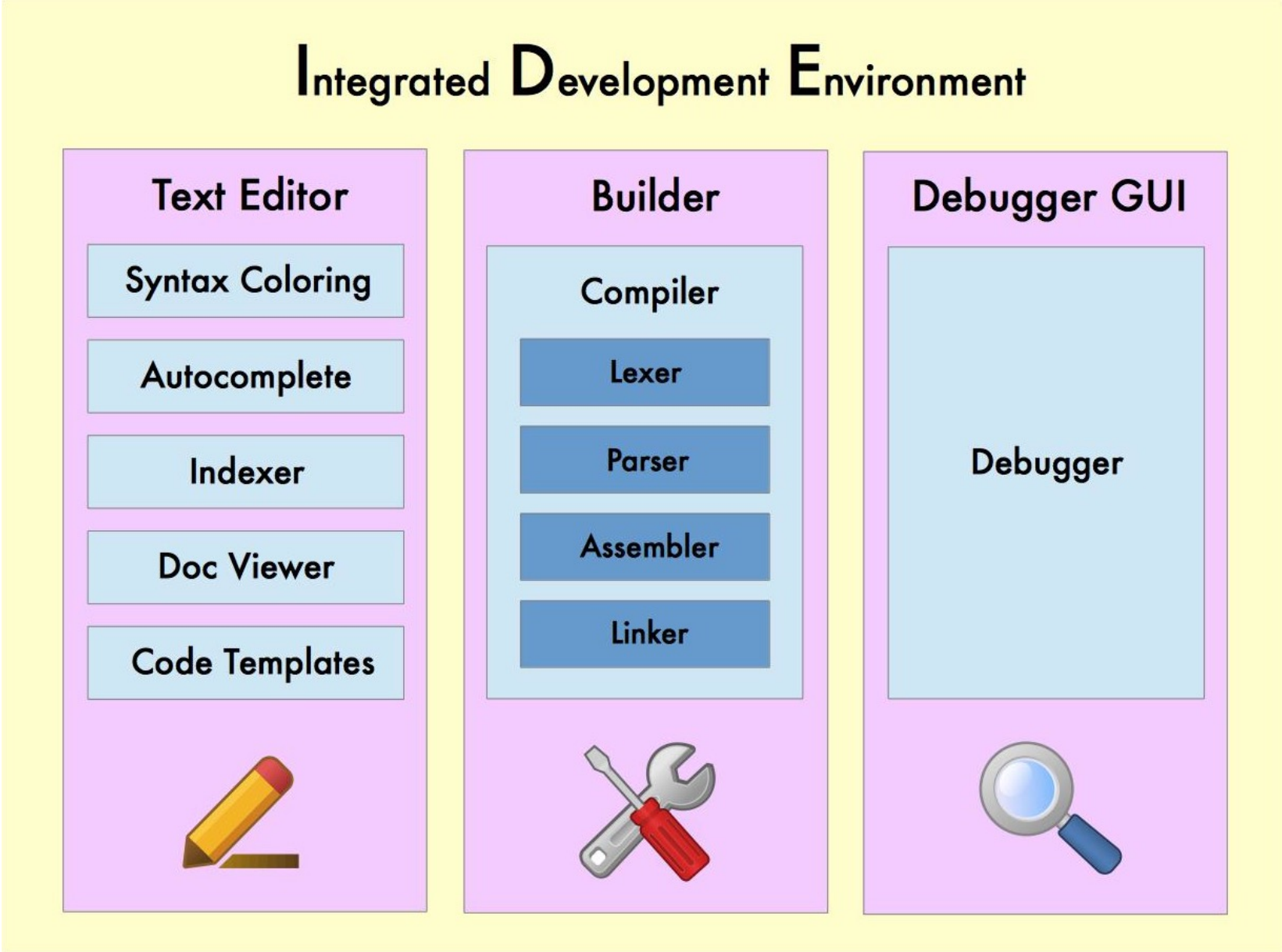
# Integrated Development Environment {IDE}

# Code development



# IDE


Integrated Development Environment
Text Editor
Syntax Coloring
Autocomplete
Indexer
Doc Viewer
Code Templates
Builder
Compiler
Lexer
Parser
Assembler
Linker
Debugger GUI
Debugger

# What is (source) Program/DEV Software: Integrated Development Environment (IDE)?

IDEs are designed to encompass
all programming tasks in one application.

IDE is a software application that provides comprehensive facilities or graphical user interface [GUI] to programmers for software development in addition to CLI (command line interfacing).
An IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs have intelligent code completion.

# What is (source-code/DEV) Program Software: Integrated Development Environment (IDE)?

Code editor: This feature is a text editor designed for writing and editing source code.

Source code editors are distinguished from text editors because they enhance or simplify the writing and editing of code.

Compiler/Interpreter: This tool transforms source code written in a human readable/writable language into a form executable by a computer.

Debugger: This tool is used during testing to help debug application programs.

Build automation tools: These tools automate common developer tasks.

# Source-Code Editor

A source code editor is a text editor program designed specifically for editing source code of computer programs by programmers.

It may be a Standalone Application or it may be built into an Integrated Development Environment (IDE) or Web Browser.

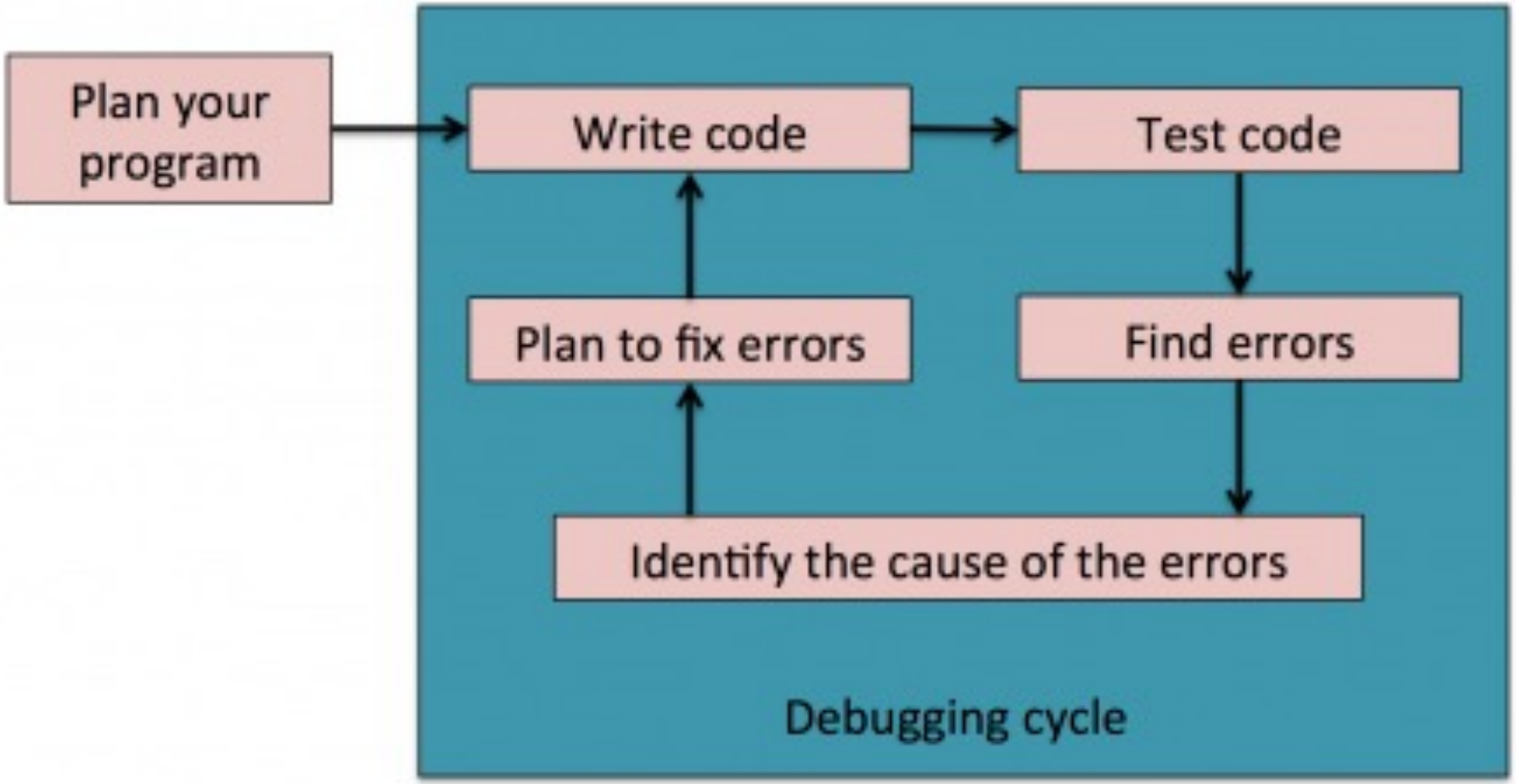
Source code editors are the most fundamental programming tool, as the fundamental job of programmers is to write and edit source code.

# Debugging

Debugging is a process of analyzing a computer program source-code and removing its logical or syntactical errors.

Software which assists this process is a debugger.

Using a debugger, a software developer can step through a program's code and analyze its variable values, searching for errors.

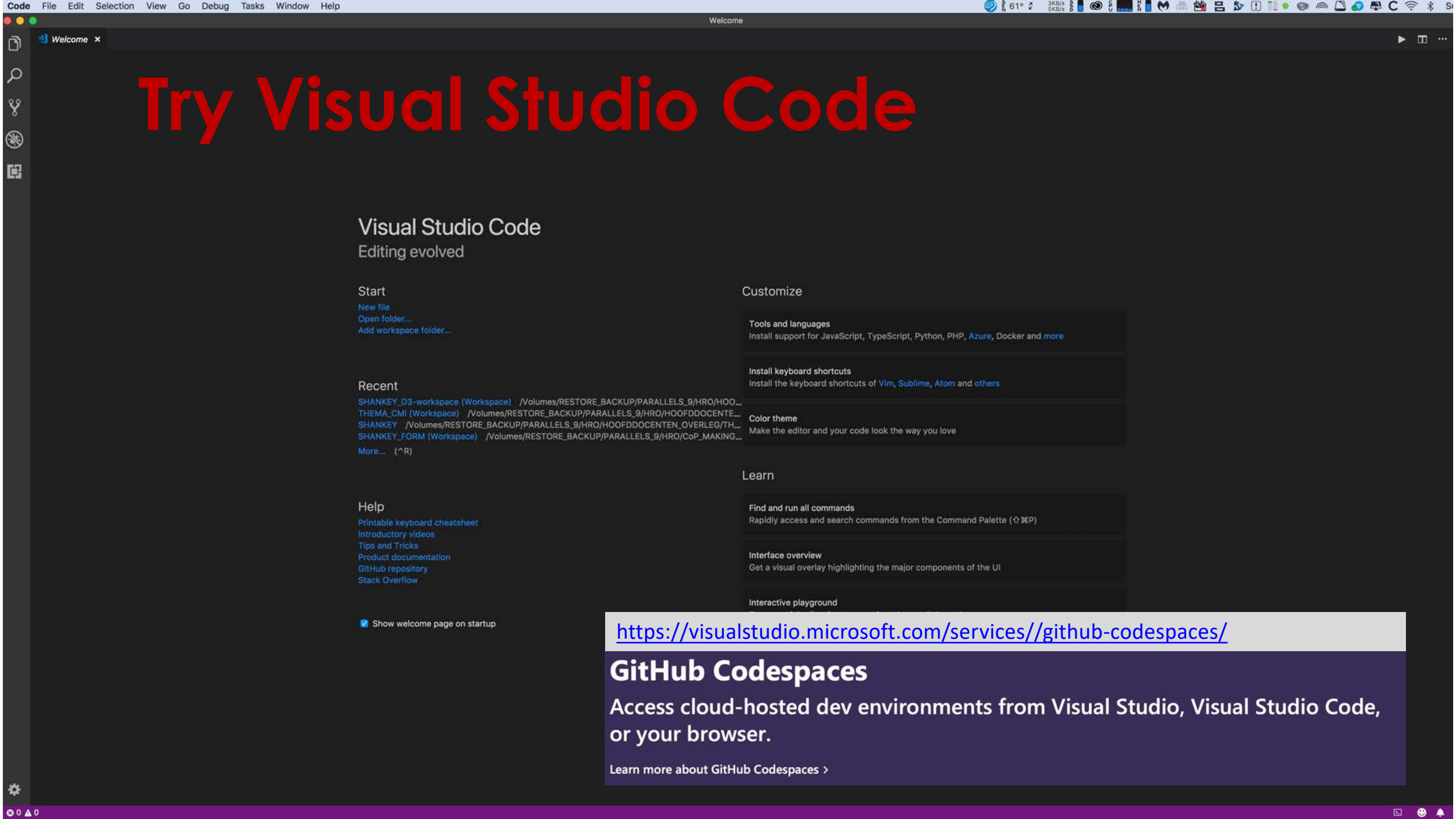

Plan your program
Write code
Test code
Plan to fix errors
Find errors
Identify the cause of the errors
Debugging cycle

# Things you should Do

HOGESCHOOL ROTTERDAM

# To Do's

# Try Visual Studio Code



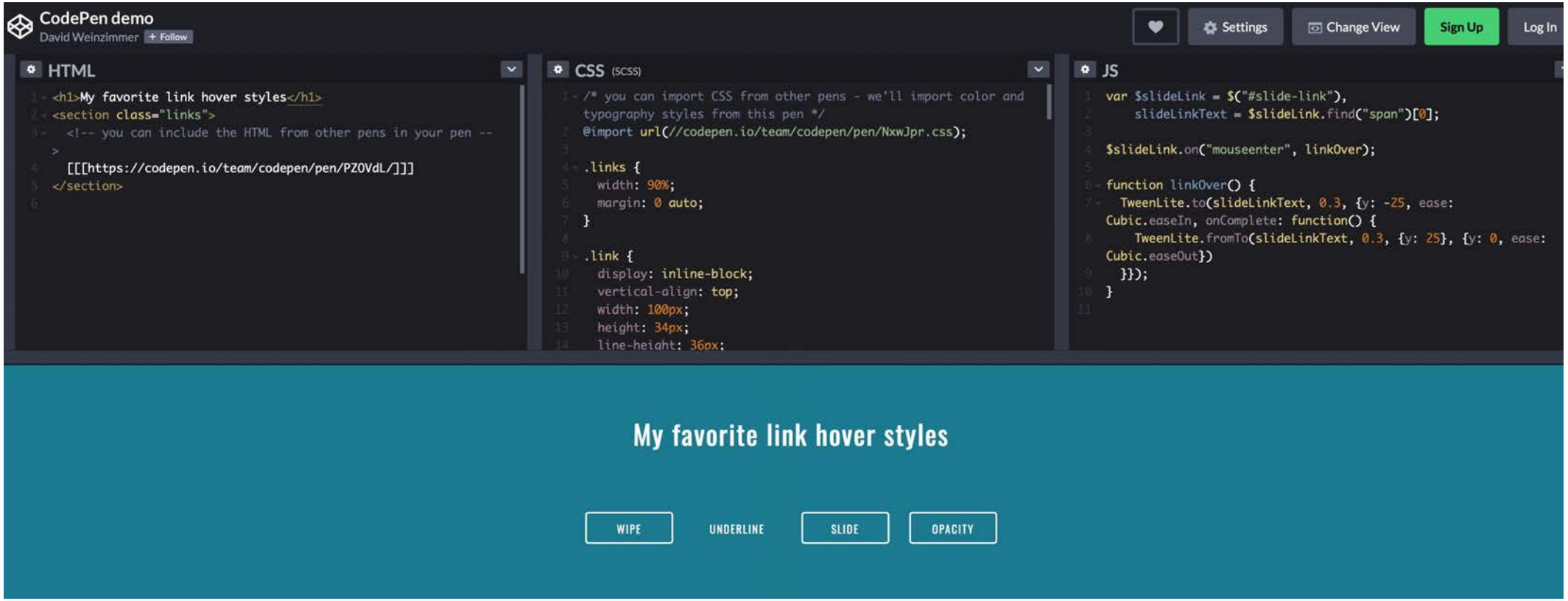

https://visualstudio.microsoft.com/services//github-codespaces/

## GitHub Codespaces

**Access cloud-hosted dev environments from Visual Studio, Visual Studio Code, or your browser.**

Learn more about GitHub Codespaces ›

# Try CODEPEN.IO



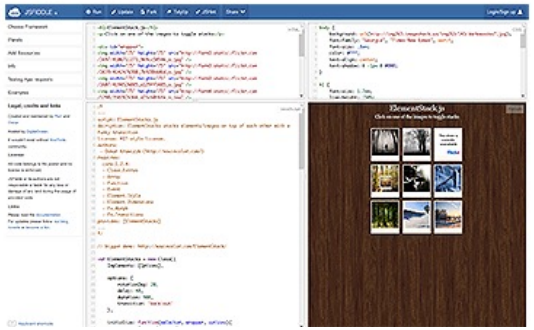

https://codepen.io/dweinz/pen/GdMbPz

# Source-Code Play-Grounds

https://www.sitepoint.com/7-code-playgrounds/

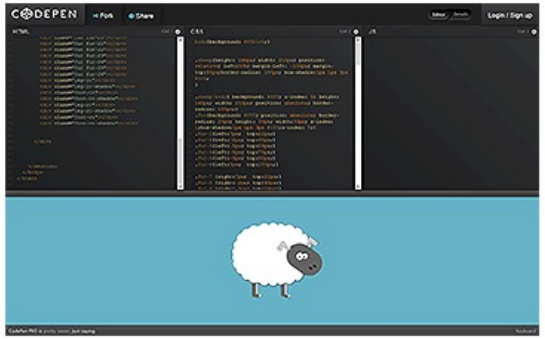
## JSFiddle

JSFiddle was one of the earliest code playgrounds and a major influence for all which followed. Despite the name, it can be used for any combination of HTML, CSS and JavaScript testing. It's looking a little basic today, but still offers advanced functionality such as Ajax simulation.



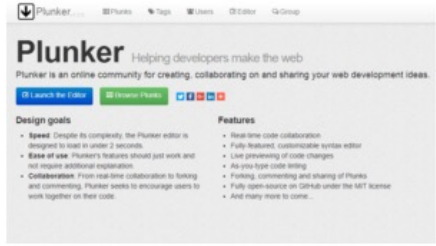
## CodePen

The prize for the best-looking feature-packed playground goes to CodePen. The service highlights popular demonstrations ("Pens") and Projects, which is an online Integrated Development Environment you can use to build and deploy web projects, a feature only added in March 2017. It offers advanced functionality such as sharing and embedding of Pens, adding external JS and CSS libraries, popular preprocessors, and tons more. The PRO service provides cross-browser testing, pair-programming and teaching options starting from just $9 per month.



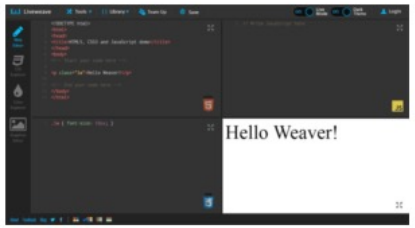

## Plunker

Plunker lets you add multiple files, including community generated templates, to kick-start your project. Just like CodePen, with Plunker you can create working demos, also in collaboration with other devs, and share your work. Plunker's source code is free and lives on its GitHub repository.




## Liveweave

Liveweave is one more online HTML5, CSS3 & JavaScript editor with live preview capabilities. It offers code-hinting for HTML5, CSS3, JavaScript and jQuery and lets you download your project as a zip file.




You can also add external libraries such as jQuery, AndgularJS, Bootstrap etc. quite easily in your workspace. Furthermore, Liveweave offers a ruler to help you code responsive designs and a "Team Up" feature which has the same features as JSFiddle's collaborative editing.

https://en.wikipedia.org/wiki/Comparison_of_online_source_code_playgrounds

# Watch: "How to think like a programmer"

https://www.youtube.com/watch?v=azcrPFhaY9k

# DIY Blinking an LED on the Raspberry Pi

https://embeddedcode.wordpress.com/2017/01/18/blinking-an-led-on-the-raspberry-pi/

Do it yourself (DIY) : Make a LED blinking

Raspberry Pi, Power Supply,

microSD Card with Raspbian Installed and Setup,

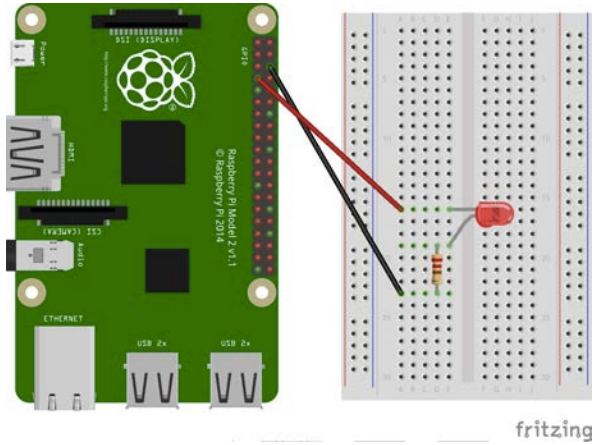
Internet connectivity (Optional),

Bread board,

connecting wires,

an LED 5mm of preferred colour

and a resistor of 220Ohms




## LEARNING OBJECTIVES

In this Rapid prototyping experiment,
You will learn the essentials of Raspberry Pi GPIO control by toggling an LED at predefined intervals of time.

## KEY WORDS, CONCEPTS, & PRACTICES

+ LED
+ Resistor
+ GPIO
+ Rapid Prototyping
+ Coding (Python)

# Reminder lesson 01

## 1st STEP Managing Data Science for IoT Projects

## ➔➔➔Getting started with GitHub➔➔➔

A Computational Thinking culture has an intellectual dimension, engaging with a set of creative concepts and practices. It has a physical dimension, encouraging interactions with others through the placement of desks, chairs, and computers. Most importantly, it has an affective dimension, cultivating a sense of confidence and fearlessness.

### LEARNING OBJECTIVES

Students will:

+ be introduced to the concept of computational Thinking, in the context of GitHub
+ be able to imagine possibilities for their own GitHub-based computational thinking
+ become familiar with resources that support their computational thinking
+ prepare for creating GitHub projects by establishing a GitHub account, exploring GitHub, creating design journals

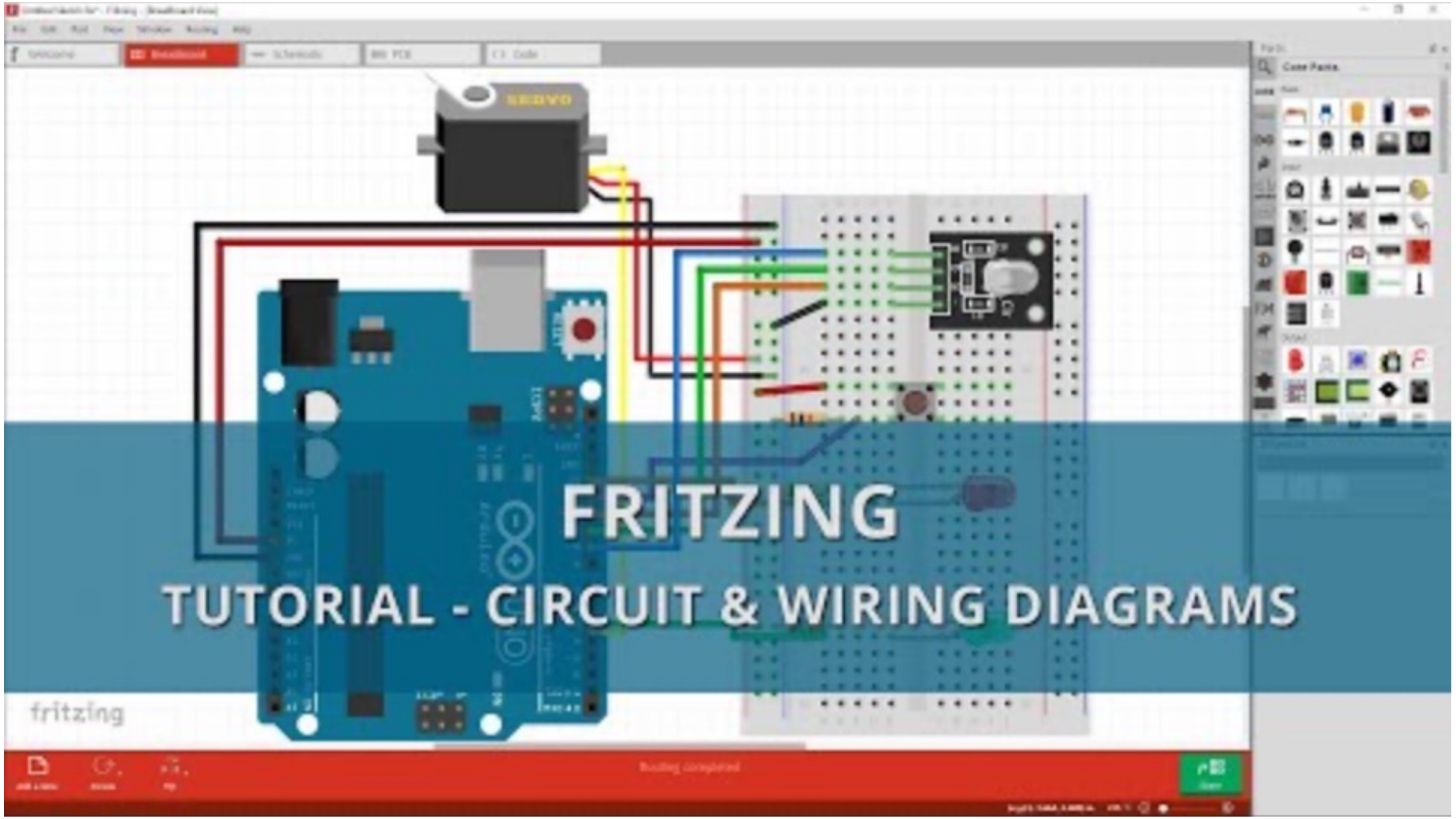
### KEY WORDS, CONCEPTS, & PRACTICES

+ GitHub
+ Computational Thinking

# Preview
# lesson four

HOGESCHOOL ROTTERDAM

# Fritzing beginners guide

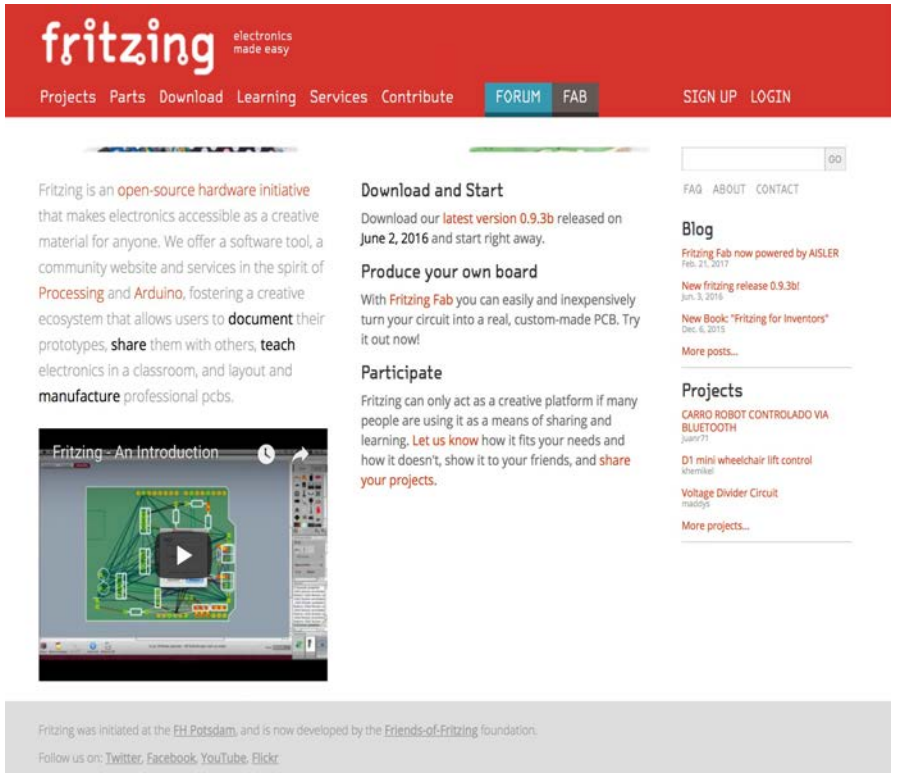



https://www.youtube.com/watch?v=-saXw1EipX0

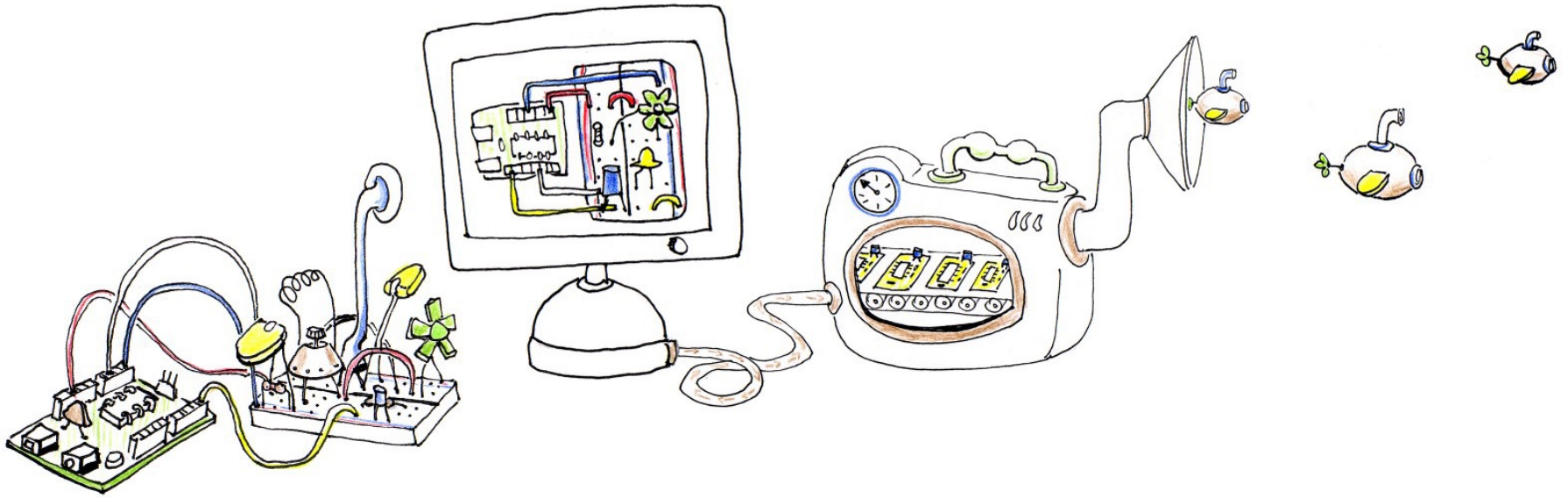
# Fritzing

A tool for advancing
electronic prototyping for designers

Today a growing community of DIY-practitioners, artists and designers are using microcontroller-based toolkits to express their concepts for digital artifacts by building them.

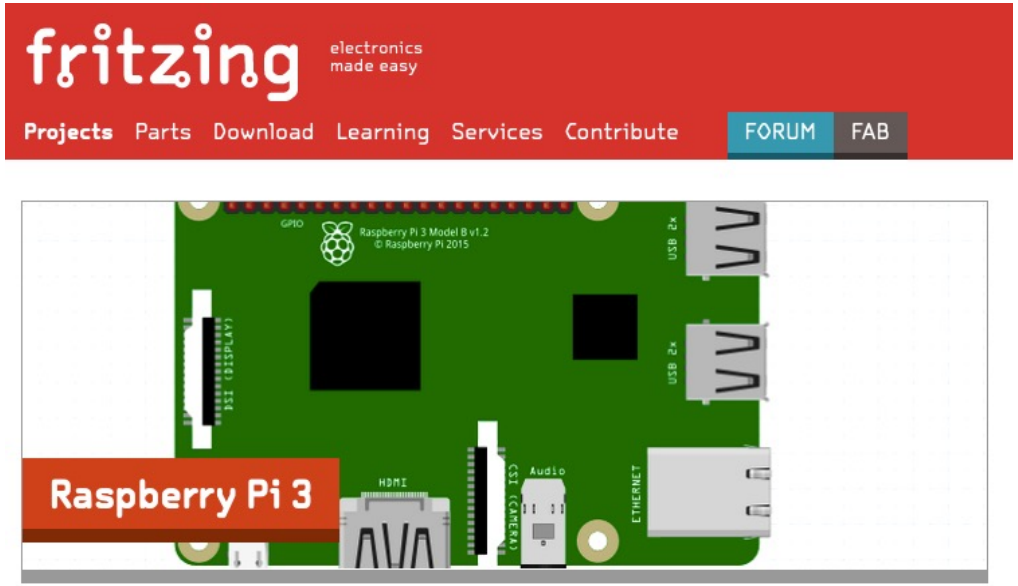


# Concept of Fritzing

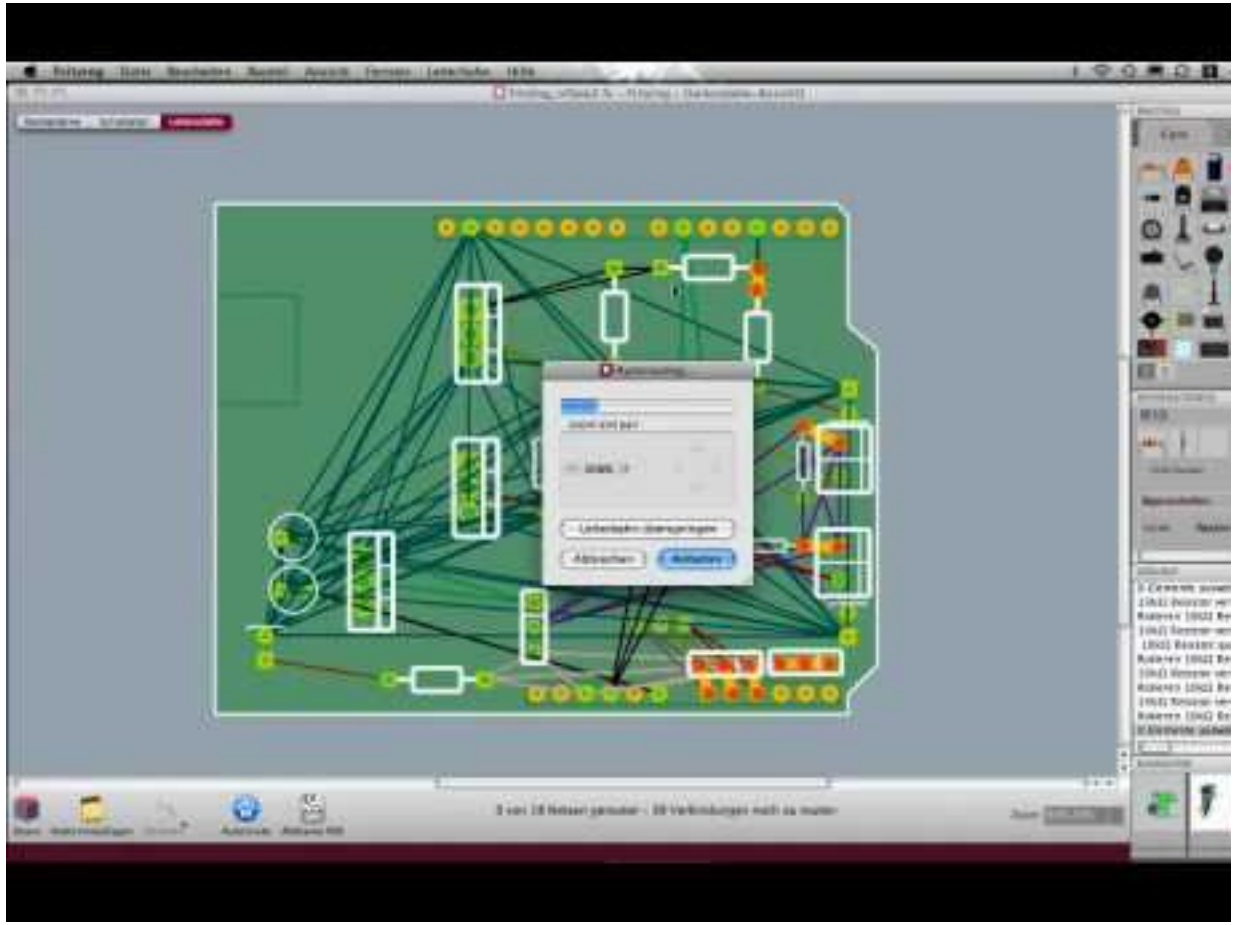


The recreation of their breadboard prototype in the Fritzing software enables designers to produce professional PCBs and also to share their designs.

# Installing Fritzing




https://fritzing.org/projects/raspberry-pi-3



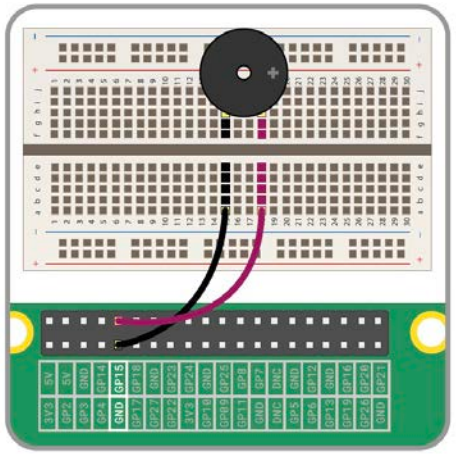
# Rapid Prototyping
## with Raspberry Pi

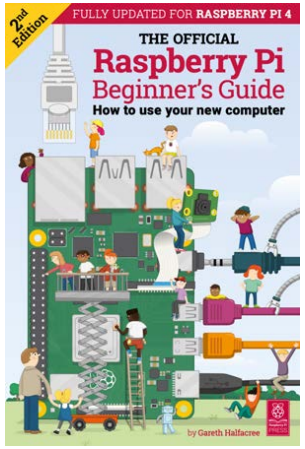
How It Works:

1 ASSEMBLE

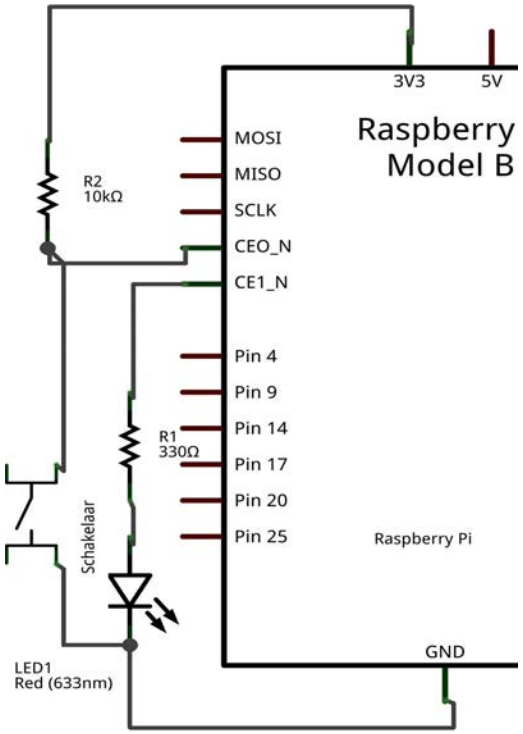
2 WRITE

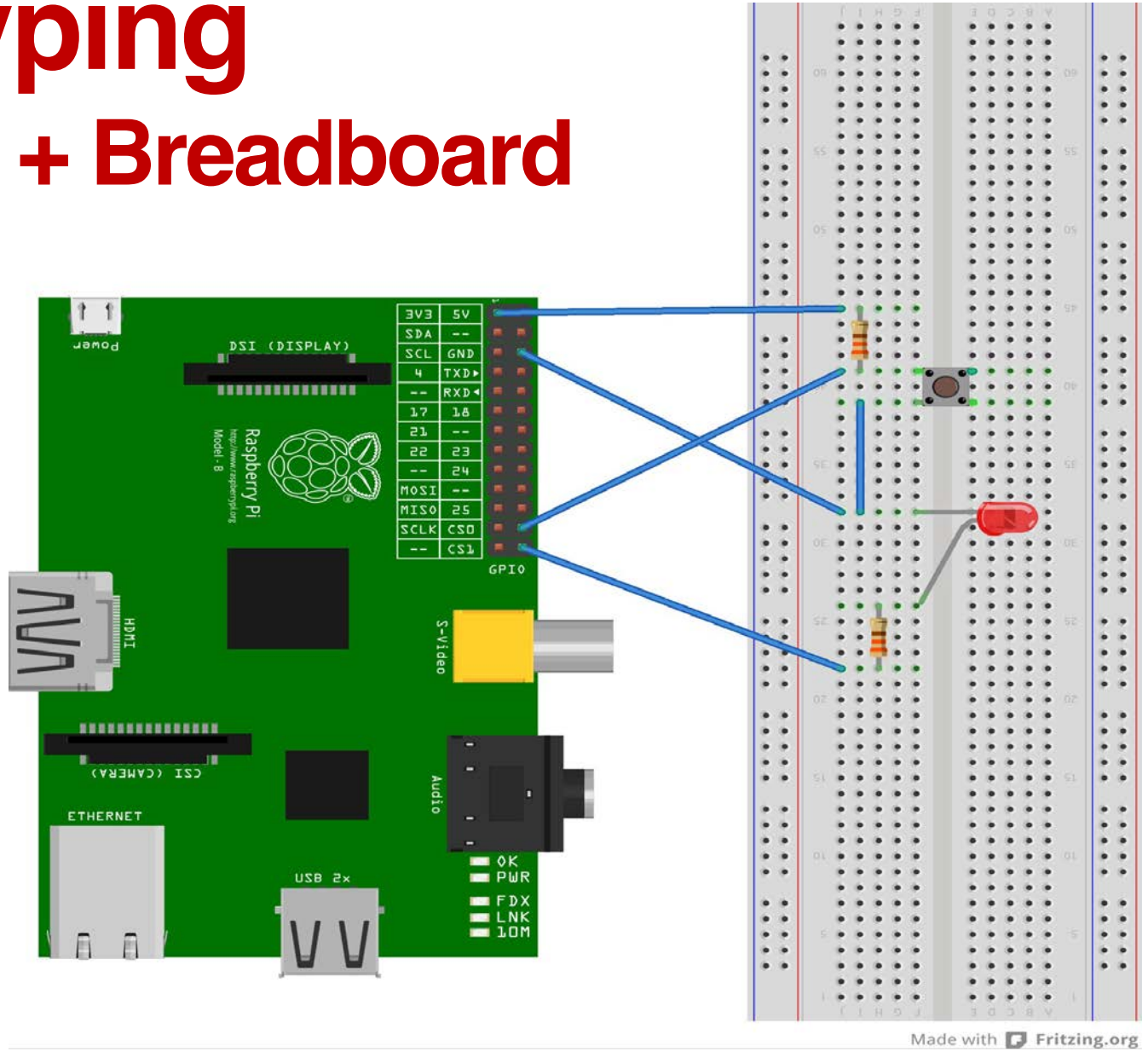
3 UPLOAD



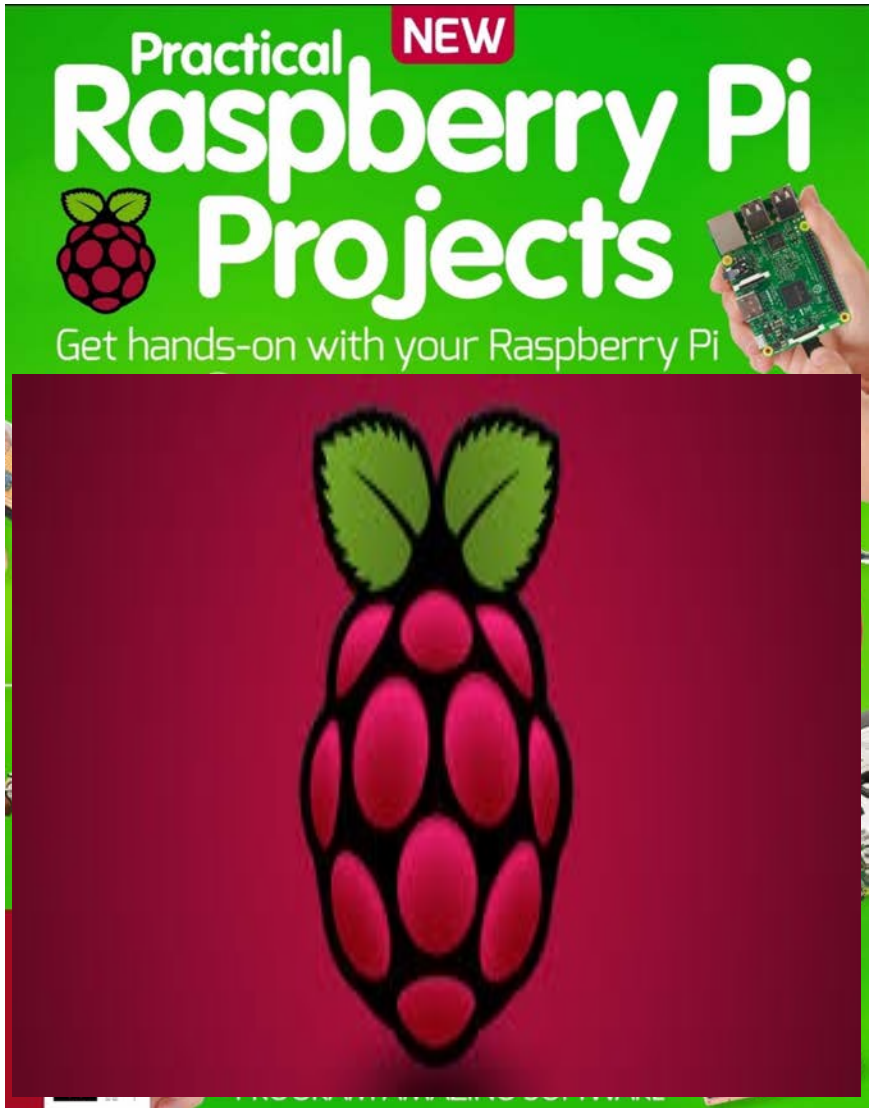


# Rapid Prototyping
## with Raspberry Pi + Breadboard



# Raspberry Pi




## Get interactive with Scratch

Experiment with physical computing by using Scratch to interact with buttons and lights on your Pi

**What you'll need**

- Breadboard
- LEDs
- Buttons
- Resistors
- Jumper wires
- ScratchGPIO3

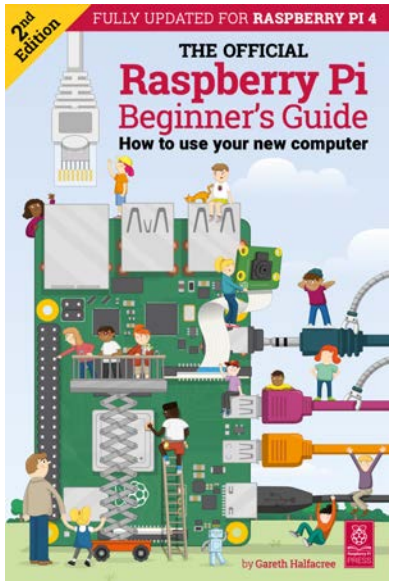
**Scratch is a very simple visual programming language, commonly used to teach basic programming concepts to learners of any age.** In this project we'll learn how to light up an LED when a button is pressed in Scratch, and then change a character's colour when a physical button is pressed. With these techniques you can make all manner of fun and engaging projects, from musical keyboards to controllers for your Scratch games and animations.

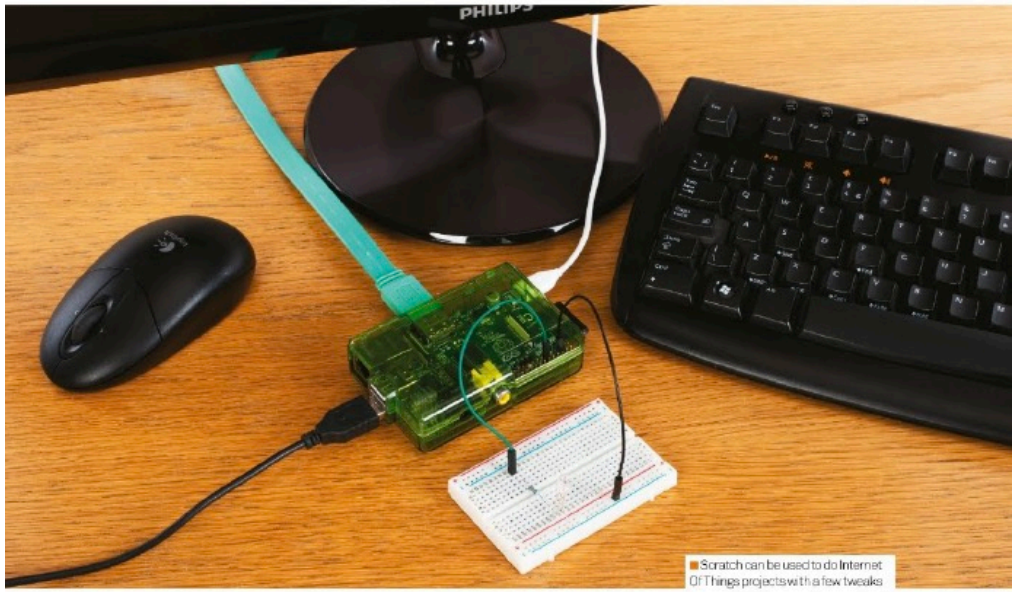
**01 Installing the required software**

Log into the Raspbian system with the username **Pi** and the password **raspberry**. Start the LXDE desktop environment using the command **startx**. Then open LXTerminal and type the following commands:

```
wget http://liamfraser.co.uk/lud/install_scratchgpio3.sh
chmod +x install_scratchgpio3.sh
sudo bash install_scratchgpio3.sh
```

This will create a special version of Scratch on your desktop called ScratchGPIO3. This is a normal version of Scratch with a Python script that handles communications between Scratch and the GPIO. ScratchGPIO was created by simplesi (cymplecy.wordpress.com).



Scratch can be used to do Internet Of Things projects with a few tweaks

# Get interactive with Scratch

## Experiment with physical computing by using Scratch to interact with buttons and lights on your Pi

### What you'll need

- Breadboard
- LEDs
- Buttons
- Resistors
- Jumper wires
- ScratchGPIO3

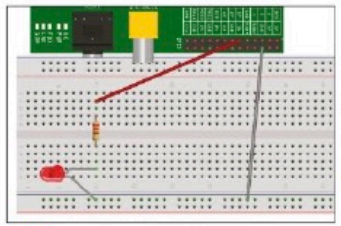
**Scratch is a very simple visual programming language, commonly used to teach basic programming concepts to learners of any age.** In this project we'll learn how to light up an LED when a button is pressed in Scratch, and then change a character's colour when a physical button is pressed. With these techniques you can make all manner of fun and engaging projects, from musical keyboards to controllers for your Scratch games and animations.

### 01 Installing the required software

Log into the Raspbian system with the username **Pi** and the password **raspberry**. Start the LXDE desktop environment using the command **startx**. Then open LXTerminal and type the following commands:

```
wget http://liamfraser.co.uk/lud/install_scratchgpio3.sh
chmod +x install_scratchgpio3.sh
sudo bash install_scratchgpio3.sh
```

This will create a special version of Scratch on your desktop called ScratchGPIO3. This is a normal version of Scratch with a Python script that handles communications between Scratch and the GPIO. ScratchGPIO was created by simplesi (pymplecy.wordpress.com).



### 02 Connecting the breadboard

Power off your Pi and disconnect the power cable. Get your breadboard, an LED, a 330-ohm resistor and two GPIO cables ready. You'll want to connect the 3.3V pin (top-right pin, closest to the SD card) to one end of the 330-ohm resistor, and then connect the positive terminal of the LED (the longer leg is positive) to the other end. The resistor is used to limit the amount of current that can flow to the LED.

Then put the negative terminal of the LED into the negative rail of the breadboard. Connect one of the GROUND pins (for example, the third pin from the right on the bottom row of pins) to the negative lane. Now connect the power to your Pi. The LED should light up. If it doesn't, then it's likely that you've got it the wrong way round, so disconnect the power, swap the legs around and then try again.

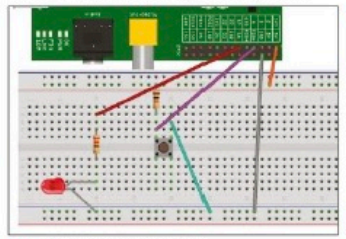
### 03 Switching the LED on and off

At the moment, the LED is connected to a pin that constantly provides 3.3V. This isn't very useful if we want to be able to turn it on and off, so let's connect it to GPIO 17, which we can turn on and off. GPIO 17 is the sixth pin from the right, on the top row of pins. Power the Pi back on. We can turn the LED on by exporting the GPIO pin, setting it to an output pin and then setting its value to 1. Setting the value to 0 turns the LED back off:

```
echo 17 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio17/direction
echo 1 > /sys/class/gpio/gpio17/value
echo 0 > /sys/class/gpio/gpio17/value
```


### 04 Controlling the LED from Scratch

Start the LXDE desktop environment and open ScratchGPIO3. Go to the control section and create a simple script that broadcasts **pin11on** when Sprite1 is clicked. Then click the sprite. The LED should light up. Then add to the script to wait 1 second and then broadcast **pin11off**. If you click the sprite again, the LED will come on for a second and then go off. ScratchGPIO3 uses pin numbers rather than GPIO numbers to identify pins. The top-right pin (the 3.3V we first connected our LED to) is pin number 1, the pin underneath that is pin number 2, and so on.



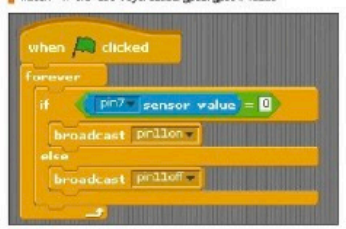
### 05 Wiring up our push button

Power off the Pi again. This circuit is a little bit more complicated than the LED one we created previously. The first thing we need to do is connect 3.3V (the top-right pin we used to test our LED) to the positive rail of the breadboard. Then we need to connect a 10Kohm resistor to the positive rail, and the other end to an empty track on the breadboard. Then on the same track, add a wire that has one end connected to GPIO 4. This is two pins to the right of GPIO 17. Then, on the same track again, connect one pin of the push button. Finally, connect the other pin of the push button to ground by adding a wire that is connected to the same negative rails that ground is connected to.

When the button is not pressed, GPIO 4 will be receiving 3.3V. However, when the button is pressed, the circuit to ground will be completed and GPIO 4 will be receiving 0V (and have a value of 0), because there is much less resistance on the path to ground.

We can see this in action by watching the pin's value and then pressing the button to make it change:

```
echo 4 > /sys/class/gpio/export
echo in > /sys/class/gpio/gpio4/direction
watch -n 0.5 cat /sys/class/gpio/gpio4/value
```




### 06 Let there be light!

Boot up the Pi and start ScratchGPIO3 as before. Go to the control section and add **when green flag clicked**, then attach a **forever** loop, and inside that an if **else** statement. Go to the operators section and add an if [] = [] operator to the if statement. Then go to the sensing section and add a value sensor to the left side of the equality statement, and set the value to pin7. On the right side of the equality statement, enter 0. Broadcast **pin11on** if the sensor value is 0, and broadcast **pin11off** otherwise. Click the green flag. If you push the button, the LED will light up!

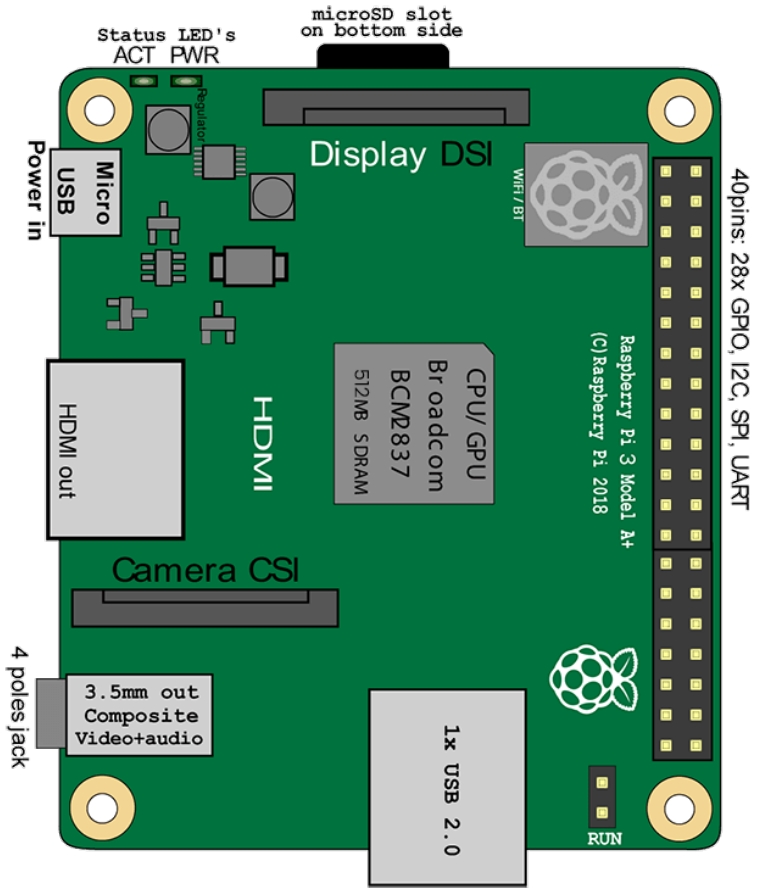
### You'll learn...

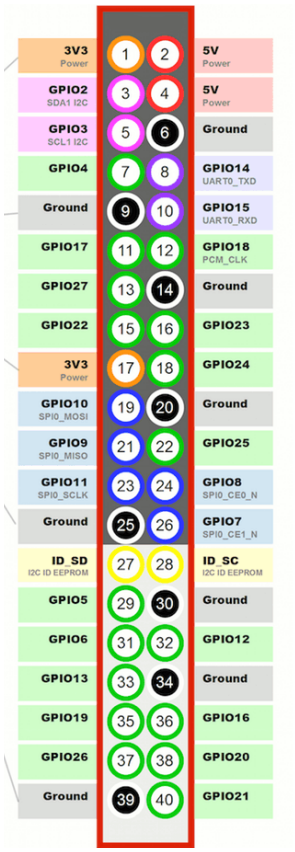
**1. Simple circuits**
While these are very simple circuits, you'll get a great feel of how the Raspberry Pi interfaces with basic prototyping kit. If you need to buy the bits and pieces, we recommend you check out: shop.pimoroni.com

**2. Coding principles**
If you're new to programming, Scratch is the perfect place to learn the same programming principles employed by all programming languages.

**3. Physical computing**
There's nothing more **magical** than taking code from your computer screen and turning it into a **real-life effect.** Your first project might just turn a light on and off, but with that skill banked, the sky is the limit.

# Raspberry Py 3A+



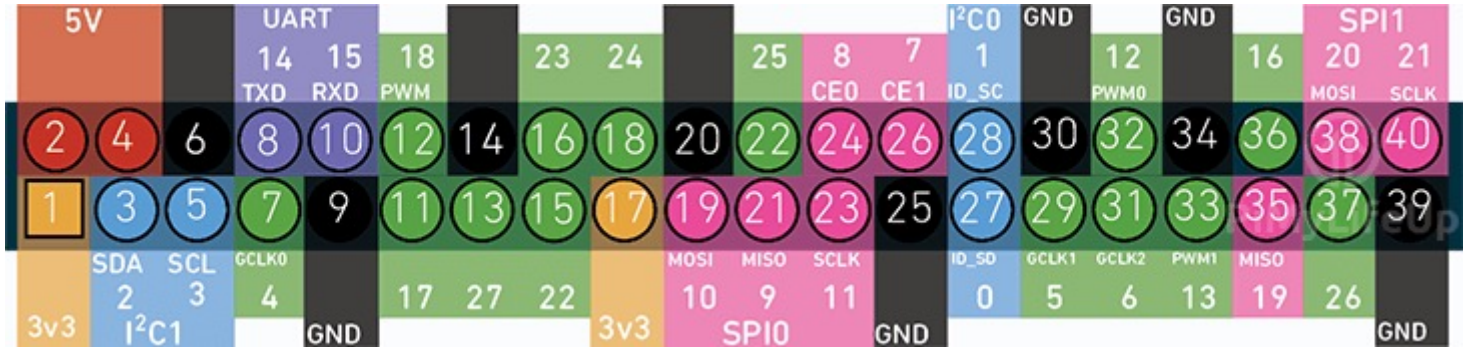



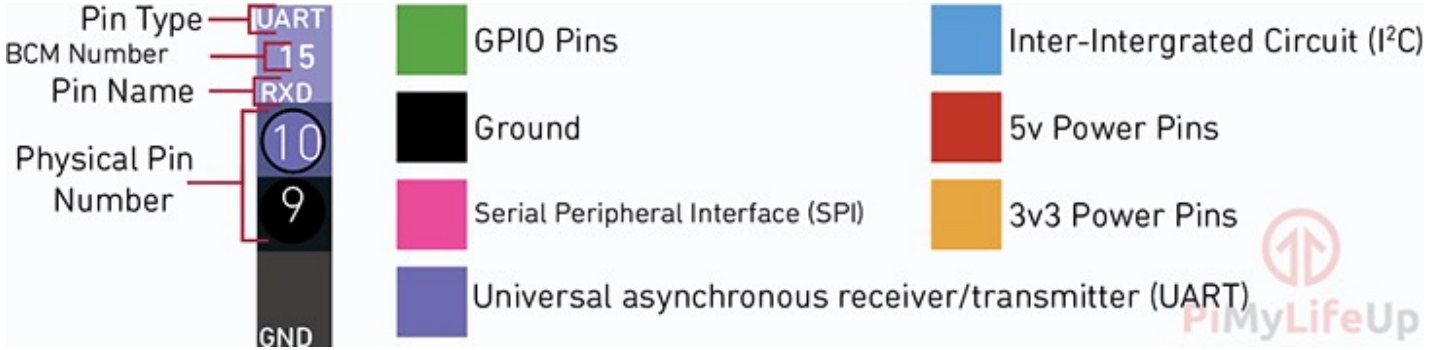

Here we have included some schematics of the Raspberry Pi 3 A Plus hardware.
These schematics show the general positioning of all the vital circuitry on the board.
The Raspberry Pi 3 A+ is a cut down version of the Pi 3B. As you can tell by its diagram, it features a single USB 2.0 Port. It's only means of network connectivity is the inbuilt Wi-Fi.

**CPU:** 1.4 GHz quad core ARM Cortex-A53
**GPU:** 250MHz Broadcom VideoCore IV
**RAM:** 512mb (Shared with GPU)
**Storage:** Micro SD
**USB 2.0 Ports:** 1
**USB 3.0 Ports:** 0
**Networking:** 802.11b/g/n/ac dual band 2.4/5 GHz wireless, Bluetooth 4.2 LS BLE
**Video Input:** 15-pin MIPI camera interface (CSI) connector
**Video Ouputs:** HDMI 1.3, MIPI display interface, DSI
**Audio Inputs:** Audio over I2S
**Audio Outputs:** 3.5mm phone jack, Digital Audio via HDMI
**Low-Level peripherals:** 17 x GPIO, +3.3v, +5v, ground, Plus the following that can be used as GPIO: UART, I2C Bus, SPI bus with two chip select, I2S audio
**Power Source:** 5v via MicroUSB or GPIO header
**Size:** 65.00mm x 56.50mm x 17mm
**Weight:** 23 g (0.81 oz)

# GPIO (general-purpose input/output)

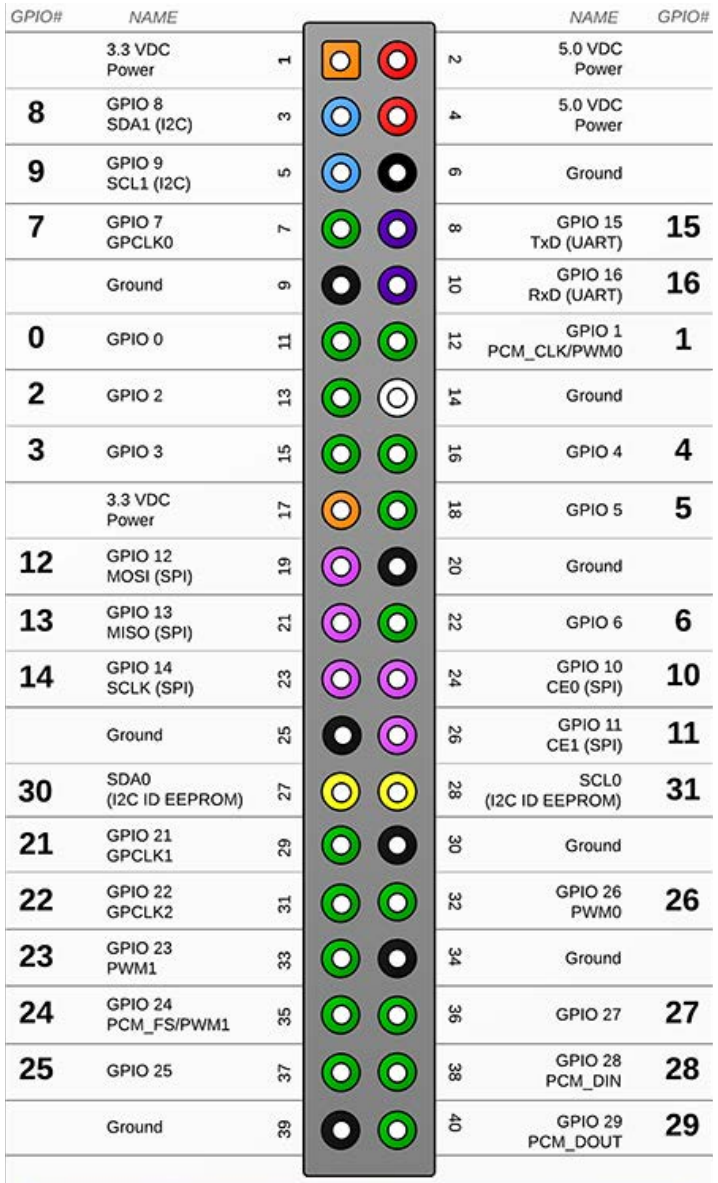
## connecting to the outside world




https://pimylifeup.com/raspberry-pi-3-a-plus/

# GPIO pins groups on the Raspberry 3



| GPIO# | NAME | | | | NAME | GPIO# |
|---|---|---|---|---|---|---|
| | 3.3 VDC Power | 1 | 2 | | 5.0 VDC Power | |
| 8 | GPIO 8 SDA1 (I2C) | 3 | 4 | | 5.0 VDC Power | |
| 9 | GPIO 9 SCL1 (I2C) | 5 | 6 | | Ground | |
| 7 | GPIO 7 GPCLK0 | 7 | 8 | | GPIO 15 TxD (UART) | 15 |
| | Ground | 9 | 10 | | GPIO 16 RxD (UART) | 16 |
| 0 | GPIO 0 | 11 | 12 | | GPIO 1 PCM_CLK/PWM0 | 1 |
| 2 | GPIO 2 | 13 | 14 | | Ground | |
| 3 | GPIO 3 | 15 | 16 | | GPIO 4 | 4 |
| | 3.3 VDC Power | 17 | 18 | | GPIO 5 | 5 |
| 12 | GPIO 12 MOSI (SPI) | 19 | 20 | | Ground | |
| 13 | GPIO 13 MISO (SPI) | 21 | 22 | | GPIO 6 | 6 |
| 14 | GPIO 14 SCLK (SPI) | 23 | 24 | | GPIO 10 CE0 (SPI) | 10 |
| | Ground | 25 | 26 | | GPIO 11 CE1 (SPI) | 11 |
| 30 | SDA0 (I2C ID EEPROM) | 27 | 28 | | SCL0 (I2C ID EEPROM) | 31 |
| 21 | GPIO 21 GPCLK1 | 29 | 30 | | Ground | |
| 22 | GPIO 22 GPCLK2 | 31 | 32 | | GPIO 26 PWM0 | 26 |
| 23 | GPIO 23 PWM1 | 33 | 34 | | Ground | |
| 24 | GPIO 24 PCM_FS/PWM1 | 35 | 36 | | GPIO 27 | 27 |
| 25 | GPIO 25 | 37 | 38 | | GPIO 28 PCM_DIN | 28 |
| | Ground | 39 | 40 | | GPIO 29 PCM_DOUT | 29 |

https://pi4j.com/1.1/pins/model-3b-rev1.html

•**Power:** Pins that are labeled 5.0v supply 5 volts of power and those labeled 3V3 supply 3.3 volts of power. There are two 5V pins and two 3V3 pins.
•**GND:** These are the ground pins. There are eight ground pins.

•**Input/Outputpins**: These are the pins labeled with the # sign, for example, #17, #27, #22, etc. These pins can be used for input or output.

•**I2C:** I2C is a serial protocol for a two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces, and other similar peripherals in embedded systems. These pins are labeled **SDA** and **SCL**.

•**UART:** The **Universal Asynchronous Receiver/Transmitter** allows your Raspberry Pi to be connected to serial peripherals. The UART pins are labeled **TXD**and **RXD**.

•**SPI:** The **Serial Peripheral Interface** is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems. The SPI pins are labeled **MOSI**, **MISO**, **SCLK**, **CE0**, and **CE1**.

•**ID EEPROM:Electrically Erasable Programmable Read-Only Memory** is a user-modifiable read-only memory that can be erased and written to repeatedly through the application of higher than normal electrical voltage. The two EEPROM pins on the Raspberry Pi (**EED** and **EEC**) are also secondary I2C ports that primarily facilitate the identification of Pi Plates (e.g., Raspberry Pi Shields/Add-On Boards) that are directly attached to the Raspberry Pi.

# GPIO general purpose IO

Voorgedefinieerde pennen

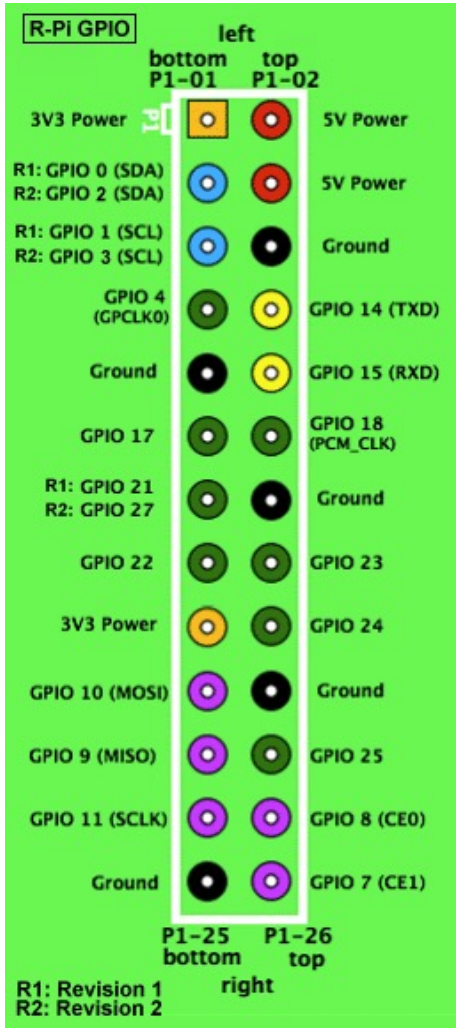
- 0V, 3.3V, 5V, Transmit, Receive
- I2C, 1-Wire

Vrijbeschikbare pennen

- GPIO 4, 17, 18, 8, 7

Naamgeving

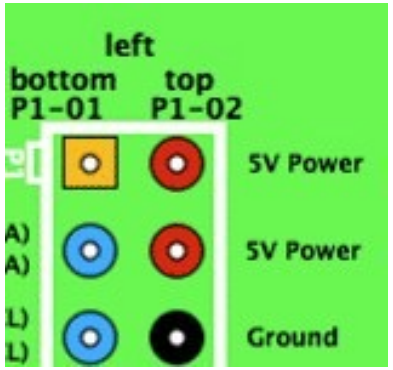
- Pinnummer van connector P1
- Broadcom documentatie van de SoC (System on Chip)



# GPIO

Voordat je iets aansluit

- input maximaal 3.3V
  - let met name op de 5V van pin 2 en 4
- output maximaal 16 mA
  - LED over 330 Ω weerstand mag
  - motortje heeft een buffer nodig (ULN2003)
- totale output maximaal 50 mA



# Programmeren

Schrijven (led laten knipperen)

- Eerst: GPIO pennetje definieren als output
- Dan: herhaaldelijk schrijven

Pennetje 26 = GPIO 7

Lezen (schakelaar uitlezen)

- Eerst: GPIO pennetje definieren als input
- Dan: herhaaldelijk lezen

Pennetje 24 = GPIO 8

# Programmeren (bash shell)

- bash is de Linux command line interpreter
- als root in directory /sys/class/gpio werken
- GPIO 7 voor uitvoer

```
echo "7" >/sys/class/gpio/export
echo "out" >/sys/class/gpio/gpio7/direction
```

- led aan (1), led uit (0)

```
echo   "1"      > /sys/class/gpio/gpio7/value
echo   "0"      > /sys/class/gpio/gpio7/value
```

# Programmeren (bash shell )

Knipperen

```
while sleep 0.5
do echo "1" > /sys/class/gpio/gpio7/value  sleep  0.5
   echo "0" > /sys/class/gpio/gpio7/value  done
```

GPIO 8 voor invoer

```
echo "8" > /sys/class/gpio/export
echo "in" > /sys/class/gpio/gpio8/direction
```

eenmalig lezen

```
cat /sys/class/gpio/gpio8/value
```

# Programmeren (bash shell)

herhaald lezen

```
while sleep 0.1
do cat /sys/class/gpio/gpio8/value
done
```

opruimen

```
echo "7"      > /sys/class/gpio/unexport
echo "8"      > /sys/class/gpio/unexport
```

# Programmeren (python)

Pi in Raspberry Pi staat voor Python

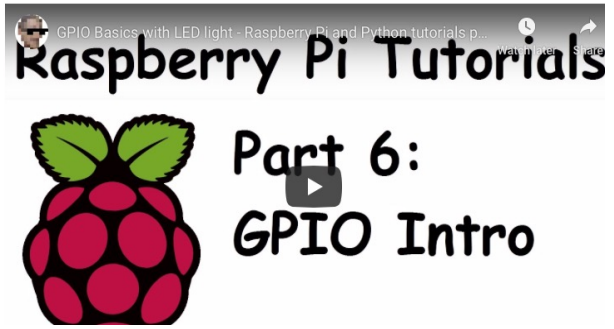
Standaard modules en bibliotheken

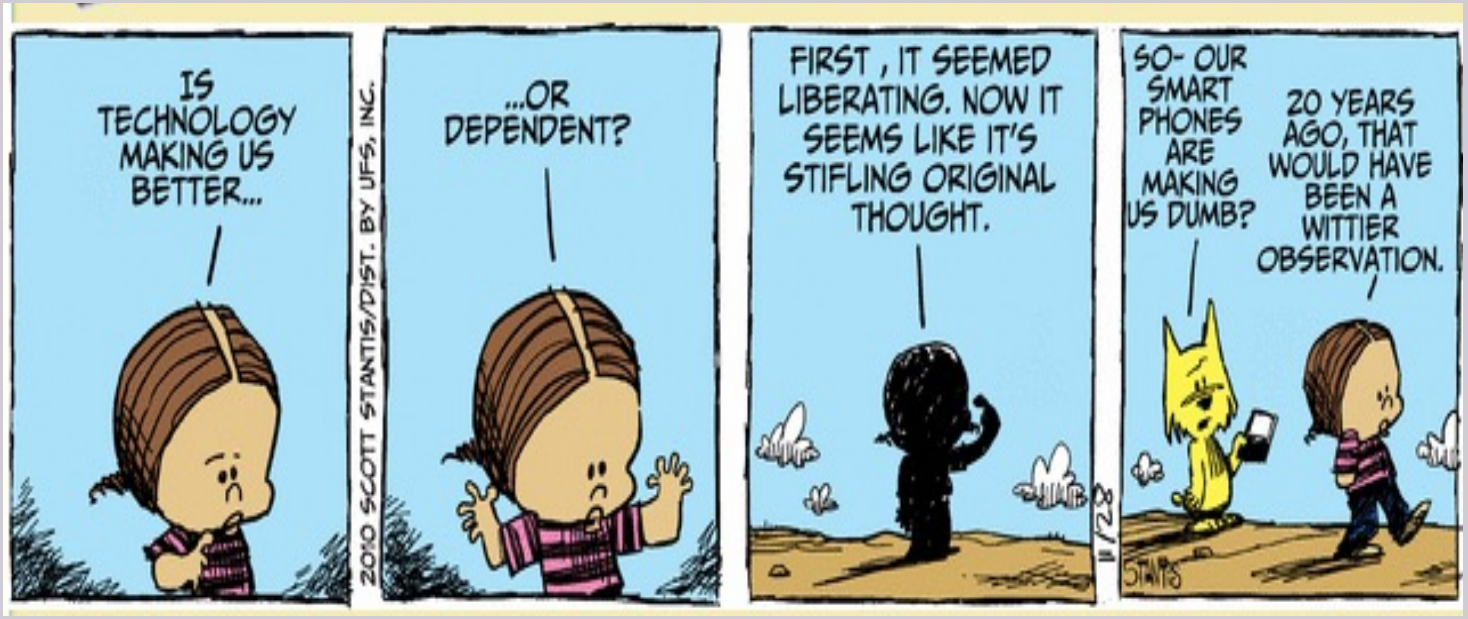
- keuze in naamgeving pennetjes / meer functies
- https://pythonprogramming.net/gpio-raspberry-pi-tutorials/

RPi.GPIO

- standaard onderdeel van Raspbian
- https://pypi.org/project/RPi.GPI

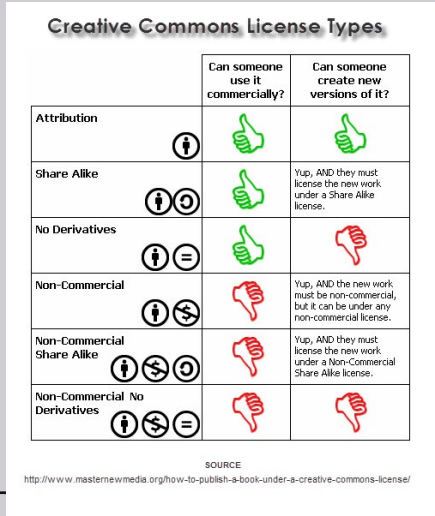
GPIO (General Purpose Input Output) Pins - Raspberry Pi tutorial












**This seminar was developed by:**
Rob van der Willigen
LivingLab AI & Ethics
Hogeschool Rotterdam
November 2021

HOGESCHOOL ROTTERDAM




HOGESCHOOL ROTTERDAM